UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Grado en Ingeniería del Software

Order Frame: Desarrollo de un videojuego 3D de acción centrado en elecciones morales en Unreal Engine 5.

Order Frame: Development of a 3D action video game focused on moral choices in Unreal Engine 5.

Realizado por
Marta Granado Rodríguez

Tutorizado por
David Bueno Vallejo

Departamento
Lenguajes y Ciencias de la Computación
UNIVERSIDAD DE MÁLAGA

MÁLAGA, febrero de 2026

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
INGENIERÍA DEL SOFTWARE

Order Frame: Desarrollo de un videojuego 3D de acción centrado en elecciones morales en Unreal Engine 5.

Order Frame: Development of a 3D action video game focused on moral choices in Unreal Engine 5.

Realizado por
Marta Granado Rodríguez

Tutorizado por
David Bueno Vallejo

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, FEBRERO DE 2026

Fecha defensa: febrero de 2026

Resumen

Este proyecto surgió de la motivación por integrar la ingeniería del software con el desarrollo creativo de videojuegos, buscando materializar un concepto de mundo abierto centrado en la toma de decisiones. El objetivo principal consistió en el diseño e implementación de ***Order Frame***, un videojuego en tercera persona que utiliza un sistema de **alineamiento dinámico** (Cielo, Tierra e Infierno) para modificar la experiencia del usuario y las estadísticas del personaje según sus acciones morales.

La metodología se basó en un enfoque iterativo empleando **Unreal Engine 5.6** como motor principal. La lógica se desarrolló íntegramente mediante el sistema de programación visual ***Blueprints***, estructurando el proyecto de forma modular mediante el uso de interfaces y despachadores de eventos para asegurar la escalabilidad de los sistemas. Ante la limitación de recursos técnicos, se emplearon herramientas de inteligencia artificial y plataformas de animación para la creación de activos 3D funcionales.

Como resultado, se obtuvo un prototipo funcional que integra mecánicas de combate, gestión de inventario, misiones y una narrativa reactiva. Las conclusiones del trabajo subrayaron la eficacia de la arquitectura modular para gestionar sistemas complejos y facilitar el mantenimiento del software. Se demostró que la reducción controlada del alcance y la optimización técnica resultaron fundamentales para la viabilidad de un proyecto individual de esta envergadura.

Palabras clave: Unreal Engine 5, Blueprints, Videojuegos, Mundo abierto, Sistema de alineamiento, Ingeniería del Software.

Abstract

This project was motivated by the desire to integrate software engineering with the creative development of video games, aiming to materialize a concept of an open world focused on decision-making. The main objective was the design and implementation of *Order Frame*, a third-person video game that uses a dynamic alignment system (Heaven, Earth, and Hell) to modify the user experience and the character's statistics according to their moral actions.

The methodology was based on an iterative approach, using Unreal Engine 5.6 as the main engine. The logic was fully developed through the visual scripting system Blueprints, structuring the project modularly through the use of interfaces and event dispatchers to ensure system scalability. Due to limited technical resources, artificial intelligence tools and animation platforms were employed to create functional 3D assets.

As a result, a functional prototype was achieved, integrating combat mechanics, inventory management, quests, and a reactive narrative. The conclusions of the work highlighted the effectiveness of a modular architecture for managing complex systems and facilitating software maintenance. It was demonstrated that controlled scope reduction and technical optimization were crucial for the feasibility of an individual project of this magnitude.

Keywords: Unreal Engine 5, Blueprints, Video Games, Open World, Alignment System, Software Engineering.

Índice

Índice de figuras

Índice de tablas

1

Introducción

1.1. Motivación

El desarrollo de este Trabajo de Fin de Grado se sitúa dentro del ámbito del desarrollo de videojuegos, área relacionada directamente con los contenidos y competencias adquiridas a lo largo del Grado en Ingeniería del Software. El proyecto parte de una idea concebida y desarrollada progresivamente, encontrando en el TFG el marco adecuado para su implementación como un sistema interactivo funcional. Con el objetivo de contextualizar la propuesta, resulta relevante situarla en relación con las tendencias actuales del mercado del videojuego.

En los últimos años, el mercado ha mostrado una clara preferencia por los videojuegos de mundo abierto en tercera persona, caracterizados por ofrecer libertad de exploración y flexibilidad en la experiencia del jugador. Order Frame se alinea con esta tendencia, tomando como referencia títulos como *Skyrim [3]* u *Horizon Zero Dawn [19][19]*, de los que adopta estructuras no lineales y sistemas de combate dinámicos. El diseño evita recorridos estrictamente lineales, favoreciendo la agencia del jugador y permitiéndole decidir el orden de las misiones y su forma de interactuar con el entorno.

Asimismo, existe una creciente demanda de narrativas reactivas, en las que las decisiones del jugador tengan consecuencias reales más allá de lo estético. La principal aportación del proyecto es la implementación de un sistema de alineamiento dinámico (Cielo, Tierra e Infierno), en el que la moralidad del personaje no se muestra mediante indicadores explícitos, sino que se construye de forma orgánica a partir de las acciones realizadas durante la partida. Este sistema influye en la relación con facciones, personajes y el entorno, generando consecuencias visibles tanto a nivel narrativo como jugable. Además, determinadas estadísticas del personaje, como la velocidad, el daño o la salud, se ven modificadas de manera coherente con dicho alineamiento, dando lugar a una experiencia moral emergente.

El diseño del proyecto también incorpora elementos ocultos y consecuencias que no son evidentes de forma inmediata, de modo que ciertas mecánicas o ramificaciones narrativas solo se descubren mediante la experimentación o la rejugabilidad. Este

enfoque refuerza el componente de descubrimiento y genera un mayor impacto en el jugador, al introducir sorpresas que favorecen el recuerdo de la experiencia. Esta filosofía de diseño presenta similitudes con juegos como *Undertale [18]*, donde las decisiones adoptadas influyen directamente tanto en el desarrollo narrativo como en el estilo de juego, dando lugar a rutas diferenciadas.

Por otro lado, el mercado valora positivamente los sistemas de progresión complejos y el contenido secundario que contribuye a la retención del jugador. En este contexto, Order Frame incorpora mecánicas como la domesticación de animales, las cuales afectan directamente a la jugabilidad y a la obtención de distintos finales. Estas se complementan con sistemas de creación de pociones e inventarios especializados, aportando profundidad y variedad a la experiencia de juego.

Finalmente, desde el punto de vista de la ingeniería del software, el proyecto se apoya en una arquitectura modular basada en el uso de Blueprints [8], interfaces [11] y despachadores de eventos [10]. Esta estructura facilita la organización del código, mejora la mantenibilidad del sistema y permite la incorporación de nuevas funcionalidades o ampliaciones de forma eficiente.

1.2. Objetivos

El proyecto se fundamenta en la convergencia de la ingeniería del software y el diseño narrativo interactivo. A continuación, se detallan los objetivos que rigen el desarrollo de esta propuesta técnica:

- Objetivo Principal
  El propósito central de este trabajo es el diseño e implementación de una arquitectura de software modular en Unreal Engine 5.6 que sustente una experiencia de juego no lineal y una narrativa reactiva. Se busca que las decisiones del usuario no solo afecten el curso de la historia, sino que modifiquen dinámicamente los sistemas lógicos y las estadísticas del personaje a través de un sistema de alineamiento original.

- Objetivos Específicos
  - Implementación de un Sistema de Alineamiento Dinámico: Desarrollar una lógica de juego basada en tres categorías simbólicas (Cielo, Tierra e Infierno) que altere de forma orgánica las capacidades del protagonista (salud, daño y velocidad) y su relación con el entorno, sin depender de indicadores morales explícitos.
  - Desarrollo de una Arquitectura Modular y Escalable: Utilizar exclusivamente el sistema de Blueprints para crear una estructura basada en la separación de responsabilidades mediante el uso de Interfaces, Event Dispatchers y Funciones parametrizadas. Esto garantiza la consistencia de las interacciones y facilita la depuración y futura expansión del código.
  - Integración de un Sistema de Combate con Pociones: Programar mecánicas que combinen el uso simultáneo de armas cuerpo a cuerpo con un sistema de creación de pociones e ingredientes recolectables,

permitiendo al jugador experimentar con diversas estrategias de combate.

- Diseño de una Narrativa Reactiva mediante NPCs Funcionales: Crear un sistema de interacción con personajes no jugables que emplee contratos de interacción genéricos (Interfaces) para la asignación de misiones y la toma de decisiones, asegurando que cada NPC aporte valor directo a la progresión del mundo.
- Gestión de Persistencia y Estado del Mundo: Implementar un sistema de guardado mediante Save Game Blueprints que almacene el inventario, el progreso de las misiones y el estado dinámico del mapa, permitiendo una experiencia de usuario fluida y coherente.
- Aseguramiento de la Viabilidad Técnica y Calidad: Ejecutar una decisión estratégica de ingeniería consistente en la creación de una "vertical slice" de alta fidelidad. Este enfoque permite validar todas las mecánicas críticas en una zona jugable completa, optimizando el rendimiento y garantizando la estabilidad del software frente a las limitaciones de tiempo y hardware.
- Optimización de Recursos mediante IA Generativa: Investigar y aplicar flujos de trabajo innovadores que integren herramientas de Inteligencia Artificial (como Rodin AI y Craiyon) para la obtención de recursos 3D funcionales, superando las limitaciones de tiempo asociadas al modelado tradicional.

1.3. Estructura del documento

El trabajo se organiza de manera que guía al lector desde la motivación inicial hasta los resultados obtenidos y posibles desarrollos futuros. En primer lugar, se presenta la introducción, donde se explica la motivación que originó el proyecto, los problemas que se encontraron a lo largo de su desarrollo, los objetivos que se perseguían y la metodología seguida. También se incluye una descripción de la estructura del documento para orientar al lector.

A continuación, se aborda el estado del arte, proporcionando los conceptos y fundamentos necesarios para comprender el proyecto. Se analizan los videojuegos de mundo abierto, los sistemas de elecciones morales, las mecánicas de combate y gestión de inventario, así como los sistemas de alineamiento y narrativa reactiva, con el objetivo de contextualizar el trabajo dentro del panorama actual de desarrollo de videojuegos.

El siguiente apartado se centra en las tecnologías utilizadas en el proyecto. Se explican tanto los motores de desarrollo y herramientas de modelado 3D, como Unreal Engine, Blender y Mixamo, así como diversas soluciones basadas en inteligencia artificial, control de versiones y gestión de proyectos, proporcionando al lector la información necesaria para entender su aplicación en el prototipo.

Tras esto, se presentan el análisis de requisitos y diseño, detallando los requisitos funcionales y no funcionales, los casos de uso y los diagramas que describen la lógica y la estructura del sistema, sirviendo como puente hacia la fase de implementación.

En la sección de implementación se describe cómo se llevó a cabo la creación de personajes y assets, la construcción del mundo, la implementación del sistema de alineamiento, las mecánicas de combate, la gestión de inventario, el desarrollo de misiones y narrativa reactiva, así como la interfaz de usuario y las pruebas realizadas para garantizar el correcto funcionamiento del prototipo.

Posteriormente, se exponen los resultados y su evaluación, mostrando el prototipo funcional, evaluando las mecánicas de juego, la escalabilidad y el mantenimiento, y señalando las limitaciones detectadas durante el desarrollo.

En la discusión se analizan los resultados obtenidos, se comparan con otros videojuegos similares y se explican las dificultades técnicas encontradas, así como las soluciones adoptadas para superarlas.

Finalmente, en conclusiones y líneas futuras, se resumen las conclusiones extraídas del proyecto y se proponen posibles vías de desarrollo futuro que podrían ampliar o mejorar el prototipo.

El documento se completa con referencias bibliográficas y diversos apéndices, que incluyen manuales de instalación y uso, recursos adicionales y un glosario de términos relevantes para facilitar la comprensión del proyecto.

2

Contexto

2.1. Conceptos habituales

Dentro del sector de los videojuegos y el desarrollo con Unreal Engine, se emplean diversos términos técnicos fundamentales para comprender la arquitectura de Order Frame. A continuación, se presenta una tabla ordenada alfabéticamente con los términos más frecuentes:

Término	Significado
Alineamiento (Alignment)	Sistema central del juego donde las decisiones del jugador suman puntos invisibles a tres categorías (Cielo, Tierra e Infierno), modificando permanentemente las estadísticas y habilidades del personaje.
Asset	Recurso digital utilizado en el desarrollo, como modelos 3D, texturas, sonidos o animaciones.
Binding	Vinculación directa entre las variables lógicas del jugador y los elementos visuales de la interfaz para actualizaciones en tiempo real.
Blueprint	Sistema de scripting visual de Unreal Engine que permite crear lógica compleja mediante nodos y conexiones sin necesidad de programación textual.
Buff / Debuff	Modificadores temporales aplicados a las estadísticas. Un *buff* mejora atributos (daño, velocidad), mientras que un *debuff* aplica efectos negativos.
Character Blueprint	Clase especializada de Blueprint que gestiona todas las mecánicas del jugador, incluyendo movimiento, combate y estadísticas.
Event Dispatcher	Mecanismo de comunicación que permite emitir eventos desde un Blueprint y notificar a otros suscritos, facilitando una arquitectura desacoplada.
Frames	Cada una de las imágenes individuales y consecutivas que, al ser mostradas rápidamente, componen la ilusión de movimiento en un

	videojuego, siendo la métrica de **Frames por segundo (FPS)** la medida de cuántas de estas imágenes se ven cada segundo.
HUD (Heads-Up Display)	Interfaz de usuario visible durante la partida que muestra información crítica como la vida (corazones) o equipo.
Interface	Contrato genérico que permite que diferentes Blueprints (como NPCs y objetos) respondan de manera consistente a una misma acción, como "hablar".
Journal	Pestaña especializada del inventario para el seguimiento de la historia, misiones y gestión de animales acompañantes.
Lock-on	Mecánica que permite fijar la cámara y los ataques sobre un enemigo concreto para mejorar la precisión en combate.
Nav Mesh	Malla de navegación que define las áreas por las que la Inteligencia Artificial puede desplazarse de forma autónoma.
NPC	Personaje No Jugable (*Non-Playable Character*), diseñado para interactuar con el jugador mediante diálogos, misiones o comercio.
Save Game Blueprint	Sistema destinado al almacenamiento persistente de datos (posición, inventario, progreso) para poder restaurar la partida.
Sprite	Ilustración bidimensional (2D) utilizada como recurso gráfico dentro de un videojuego para representar visualmente personajes, objetos, iconos, efectos o elementos de interfaz.
Vertical Slice	Estrategia de desarrollo consistente en crear una sección completa y funcional que incluye todas las mecánicas principales para validar la viabilidad del proyecto.
Widget Blueprint	Componente utilizado para el diseño y creación de interfaces de usuario interactivas como menús e inventarios.

Tabla 1: Conceptos habituales en el desarrollo de *Order Frame.*

2.2. Videojuegos ARPG en mundo abierto con elecciones

Los videojuegos de rol de acción (ARPG) en mundo abierto representan una evolución del género que combina la libertad de exploración, el combate en tiempo real y sistemas complejos de progresión de personajes. Esta categoría se caracteriza por ofrecer al usuario un entorno extenso y dinámico, donde la toma de decisiones no solo afecta el desarrollo narrativo, sino también la estructura misma del mundo de juego.

En la actualidad, títulos como The Elder Scrolls V: Skyrim [3] se han consolidado como referentes por su estructura de mundo abierto y flexibilidad, mientras que obras como Horizon Zero Dawn destacan por la implementación de sistemas de combate fluidos en tercera persona. El proyecto Order Frame se alinea con estas tendencias, adoptando una experiencia no dirigida donde la agencia del jugador es el eje central del diseño.

Un componente crítico en los RPG modernos es la narrativa reactiva, influenciada por sistemas de elecciones morales. A diferencia de los modelos lineales, juegos como Undertale [18] o The Witcher 3: Wild Hunt [5] demuestran cómo las acciones del jugador (ya sea atacar, perdonar o interactuar) alteran profundamente los finales posibles y las relaciones con los personajes no jugables (NPCs). En este sentido, el uso de un sistema de alineamiento dinámico permite que la moralidad del protagonista se construya de

forma orgánica, impactando directamente en sus estadísticas de salud, daño y velocidad.

Figura 1: Representación de un entorno de mundo abierto dinámico con libertad de exploración.

Videojuego	Tipo de Mundo	Sistema de Elección	Consecuencia Principal
Skyrim	Abierto Masivo	Facciones y Diálogos	Cambio en el estado del mundo
Undertale	Cerrado/Lineal	Acciones Morales (Perdonar/Matar)	Rutas y finales únicos
The Witcher 3	Abierto	Diálogos con tiempo limitado	Impacto en el destino de los personajes
Order Frame	**Mundo de grandes dimensiones**	**Sistema de alineamiento dinámico a través de diálogos y acciones.**	**Modificación sistémica de estadísticas.**

Tabla 2: Comparativa de mecánicas de decisión en referentes del género RPG de acción.

2.2.1. Origen de los RPG

El género RPG (Role-Playing Game) se caracteriza por permitir al jugador asumir el rol de uno o varios personajes dentro de mundos ficticios, donde la progresión del avatar y el desarrollo narrativo adquieren un papel central en la experiencia de juego. Sus orígenes se sitúan a mediados de la década de 1970, con la popularización del juego de mesa Dungeons & Dragons, que sentó las bases de la ambientación de fantasía medieval y de los sistemas de progresión sustentados en estadísticas y atributos.

En el ámbito digital, el sistema informático PLATO constituyó uno de los primeros entornos donde se desarrollaron videojuegos de rol. Entre ellos destaca The Game of Dungeons (dnd), considerado pionero al incorporar elementos que posteriormente se convertirían en estándares del género, como la exploración de mazmorras, la obtención de objetos mágicos y la gestión de atributos fundamentales tales como fuerza, destreza, inteligencia y sabiduría.

A finales de la década de 1980, el género experimentó una notable diversificación con la aparición de sagas emblemáticas como Dragon Quest, Final Fantasy y The Legend of Zelda. Este proceso propició, además, una diferenciación cultural y de diseño entre los

denominados JRPG (desarrollados en Japón) y WRPG (de tradición occidental), atendiendo a sus particularidades mecánicas, estéticas y narrativas.

Como evolución dentro de este marco surge el subgénero de los RPG de acción (ARPG), que integra la profundidad de los sistemas de progresión propios del rol con mecánicas de combate en tiempo real y dinámicas de exploración más inmediatas. Los ARPG contemporáneos se caracterizan por la presencia de mundos abiertos dinámicos y por ofrecer un elevado grado de libertad al jugador, alejándose de estructuras estrictamente lineales para potenciar su capacidad de decisión e interacción con el entorno.

Referentes actuales del subgénero, como The Elder Scrolls V: Skyrim [3] o Horizon Zero Dawn [19], destacan por la flexibilidad de sus sistemas, la construcción de ecosistemas verosímiles y la integración de narrativas que evolucionan de forma orgánica según la actuación del usuario. En este contexto técnico y de mercado se enmarca Order Frame, un ARPG en tercera persona que emplea una arquitectura modular orientada a sustentar una narrativa reactiva, así como un sistema de alineamiento dinámico capaz de modificar parámetros como la salud, el daño o la velocidad en función de las decisiones morales adoptadas por el jugador.

2.3. Referentes

El diseño de *Order Frame* no pretende reproducir de forma directa las mecánicas de títulos comerciales, sino **reinterpretar elementos comunes del género ARPG** para crear una experiencia propia. El foco se sitúa en la agencia del jugador y en cómo sus decisiones generan un impacto tanto narrativo como sistémico.

2.3.1. The Elder Scrolls V: Skyrim

Este título se toma como referencia fundamental por su **estructura de mundo abierto y libertad de exploración**. De Skyrim [3] se adopta la flexibilidad en la experiencia de juego, permitiendo que el usuario navegue por vastos paisajes con una gran variedad de entornos y sistemas de facciones sin seguir un recorrido estrictamente lineal. En el proyecto, esta influencia se traduce en un **mundo de grandes dimensiones** diseñado para ofrecer una alta sensación de apertura.

Figura 2: Captura de pantalla de The Elder Scrolls V: Skyrim [3] ilustrando la navegación en mundo abierto y el sistema de brújula/marcadores.

2.3.2. Horizon Zero Dawn

La obra de Guerrilla Games [19] sirve como referente para la jugabilidad en tercera persona y el sistema de combate. Se ha analizado cómo este título combina un mundo visualmente atractivo con ecosistemas vivos y una narrativa que se desarrolla de manera orgánica a medida que el jugador interactúa con el entorno. Order Frame emula este enfoque mediante la implementación de mecánicas de combate dinámicas y el uso de acompañantes animales domesticados que participan en los enfrentamientos.

Figura 3: Captura de Horizon Zero Dawn [19] mostrando el combate contra criaturas y la perspectiva de cámara en tercera persona.

2.3.3. Undertale

La principal influencia de *Undertale [18]* radica en su sistema de **elecciones morales con consecuencias reales**. En este referente, las acciones del jugador, como atacar o perdonar, alteran profundamente el transcurso de la historia y los finales posibles, dando lugar a rutas diferenciadas. *Order Frame* traslada esta filosofía a un entorno 3D mediante el **sistema de alineamiento dinámico (Cielo, Tierra e Infierno)**, donde la moralidad se construye orgánicamente a través de las acciones y no mediante indicadores explícitos.

Figura 4: Imagen de la interfaz de decisión en Undertale [18] (Mercy/Fight).

2.3.4. The Witcher 3: Wild Hunt

Se toma como referencia por su maestría en implementar **elecciones con consecuencias a corto y largo plazo** que afectan las relaciones con personajes y la evolución del mundo. Al igual que en la obra de CD Projekt Red [5], en *Order Frame* los NPCs no son meros elementos decorativos, sino que cumplen funciones críticas como la **asignación de misiones y la influencia en el alineamiento** moral del protagonista. La narrativa reactiva del proyecto busca que el mundo refleje la personalidad del jugador de manera natural, similar a la madurez narrativa observada en este título.

Figura 5: Captura de un diálogo con opciones temporales o ramificadas en The Witcher 3: Wild Hunt [5].

2.4. Metodología de trabajo

Para el desarrollo del proyecto Order Frame, se ha diseñado una metodología estructurada en fases que combina la agilidad de un marco de trabajo basado en Scrum con técnicas tradicionales de ingeniería de software, asegurando un avance ordenado y la optimización de recursos. Este enfoque híbrido permite una iteración constante y la flexibilidad necesaria para adaptar el diseño ante descubrimientos técnicos o limitaciones de hardware identificadas durante el proceso.

La implementación se organiza bajo los siguientes pilares metodológicos:

1. **Marco de Trabajo Ágil** (Scrum e Iteración)

    El desarrollo del proyecto se ha articulado siguiendo un marco de trabajo ágil basado en Scrum, adoptando una estructura iterativa e incremental especialmente adecuada para la complejidad inherente al desarrollo de un videojuego 3D con sistemas de decisiones dinámicas. Este enfoque ha permitido una evolución progresiva del prototipo, facilitando la adaptación del diseño y la incorporación de mejoras técnicas a lo largo del tiempo.

    El trabajo se ha organizado en sprints con una duración comprendida entre 7 y 14 días, en los que se han definido objetivos concretos y alcanzables en función del estado del proyecto. La planificación, gestión y priorización de tareas se ha realizado mediante herramientas visuales como Trello [2] y Milanote [22], mientras que el control de versiones y la integridad del código se han garantizado mediante el uso de GitHub [35], principalmente para consolidar funcionalidades relevantes una vez verificadas como estables, el desarrollo se apoyó mayoritariamente en la realización de copias de seguridad manuales del

proyecto, como medida adicional para prevenir la pérdida de datos y facilitar la recuperación ante posibles incidencias técnicas.

Dentro de este marco iterativo, el desarrollo se ha estructurado en una serie de iteraciones claramente diferenciadas, cada una de ellas centrada en un conjunto específico de objetivos técnicos y de diseño:

- **Iteración 1: Cimiento técnico.**
  Fase inicial orientada a la formación técnica autónoma en Unreal Engine 5.6 [16] y al dominio del sistema de scripting visual Blueprints [8]. Esta etapa permitió establecer una base sólida de conocimientos antes de abordar el desarrollo funcional del prototipo.

- **Iteración 2: Núcleo jugable (Core Gameplay).**
  Implementación del personaje jugable mediante el Character Blueprint, incluyendo las mecánicas de movilidad avanzada (doble salto y rodar para esquivar), así como el sistema de combate base necesario para validar la jugabilidad fundamental.

- **Iteración 3: Obtención y evaluación de assets.**
  Fase dedicada a la búsqueda, generación y evaluación de recursos visuales, explorando tanto repositorios de modelos gratuitos como herramientas de inteligencia artificial para la creación de assets 3D funcionales.

- **Iteración 4: Sistemas dinámicos y narrativa reactiva.**
  Desarrollo del sistema de alineamiento dinámico (Cielo, Tierra e Infierno), incluyendo la lógica de acumulación de puntos invisibles y la modificación de estadísticas del personaje en función de las decisiones tomadas durante la partida.

- **Iteración 5: Entorno, IA y expansión del mundo.**
  Construcción de un mundo de grandes dimensiones y programación del comportamiento de enemigos y NPCs, implementando sistemas de patrullaje y navegación mediante Nav Mesh [12], así como la integración de interacciones narrativas y misiones.

Este enfoque iterativo ha permitido validar de forma temprana las mecánicas principales mediante una “vertical slice” funcional, priorizando la estabilidad, la mantenibilidad y la viabilidad técnica del proyecto frente a la ampliación descontrolada del alcance.

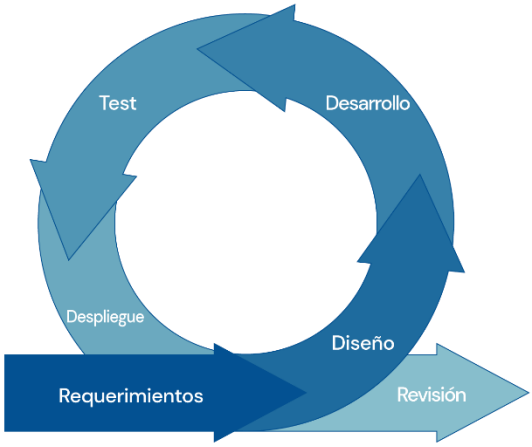


Figura 6: Metodología ágil tomada de [7] para fines ilustrativos.

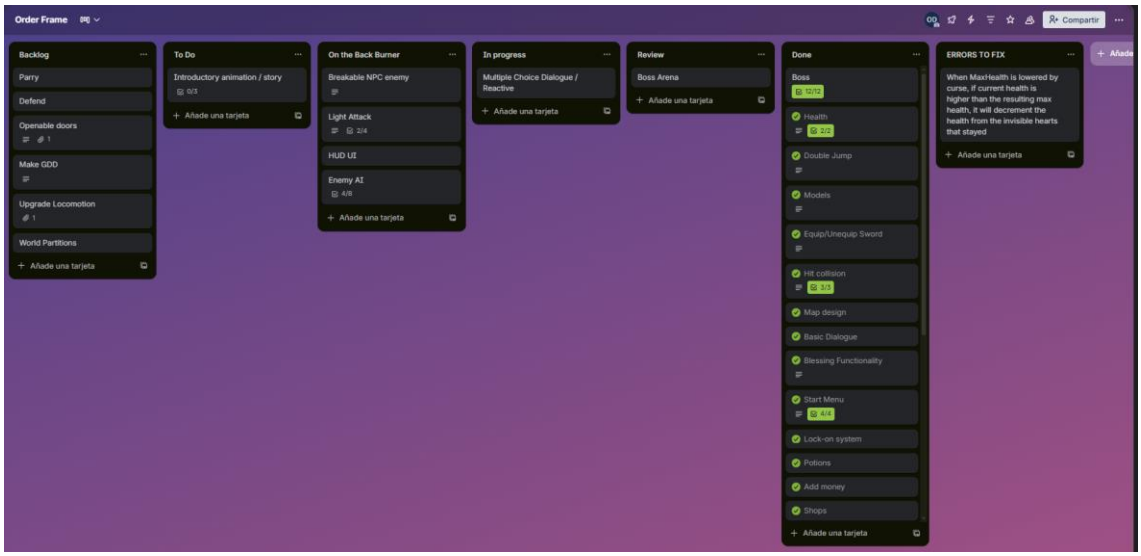


Figura 7: Organización de sprints con Trello [2].

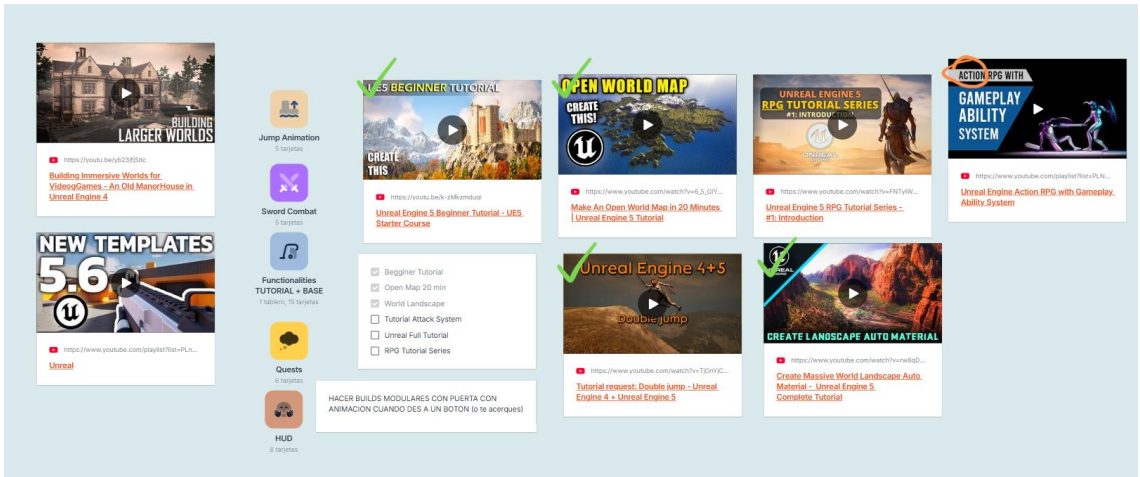


Figura 8: Milanote [22]. Toda la organización.

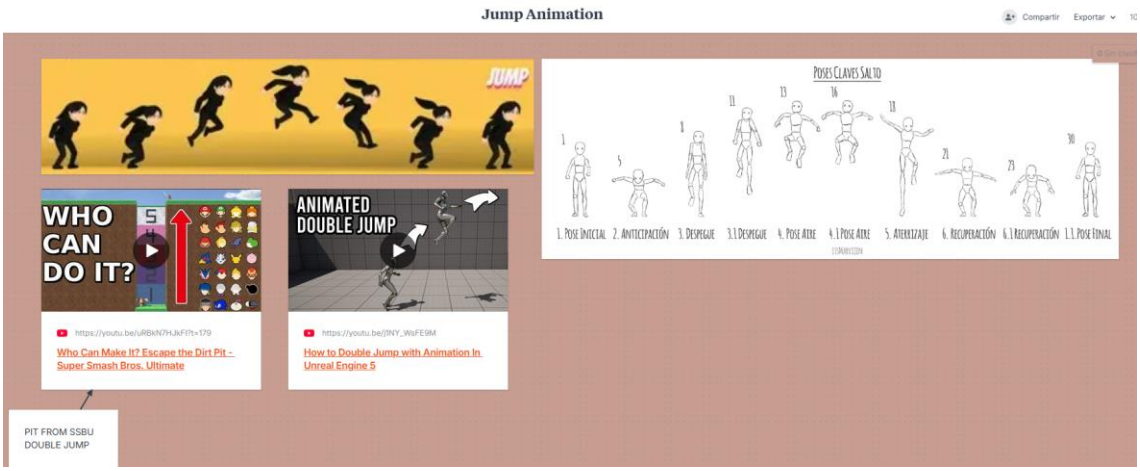


Figura 9: Milanote [22]. Apartado de animación de salto.

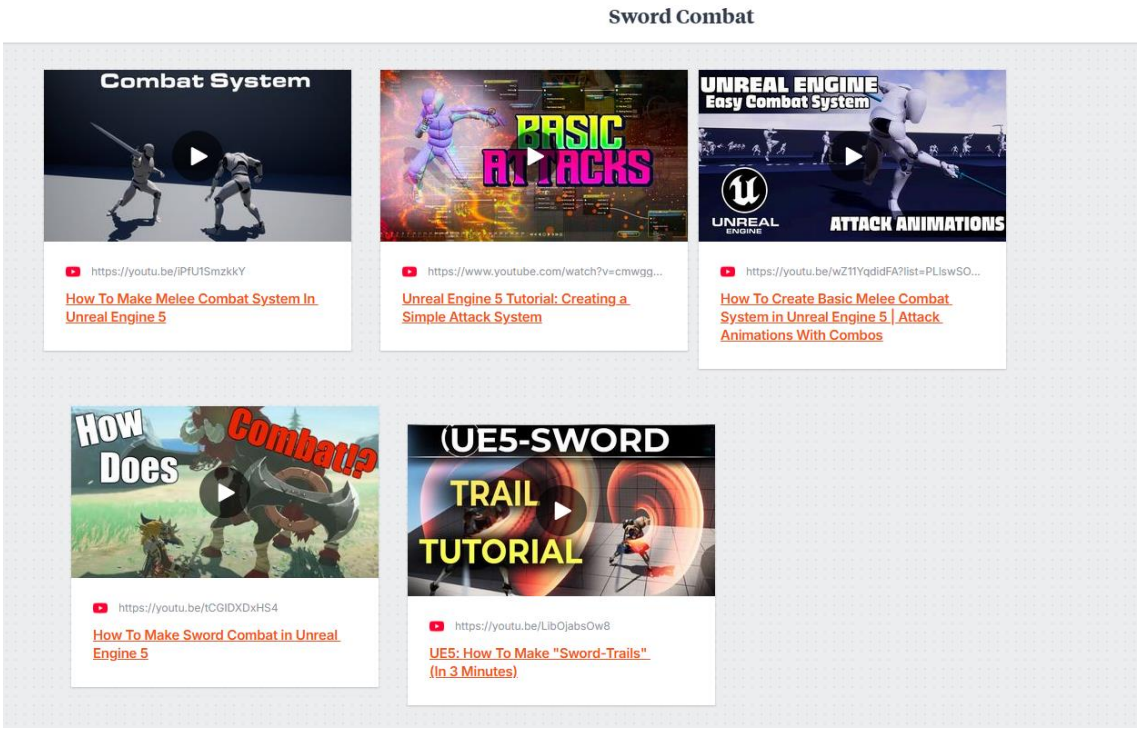


Figura 10: Milanote [22]. Información combate con espada.

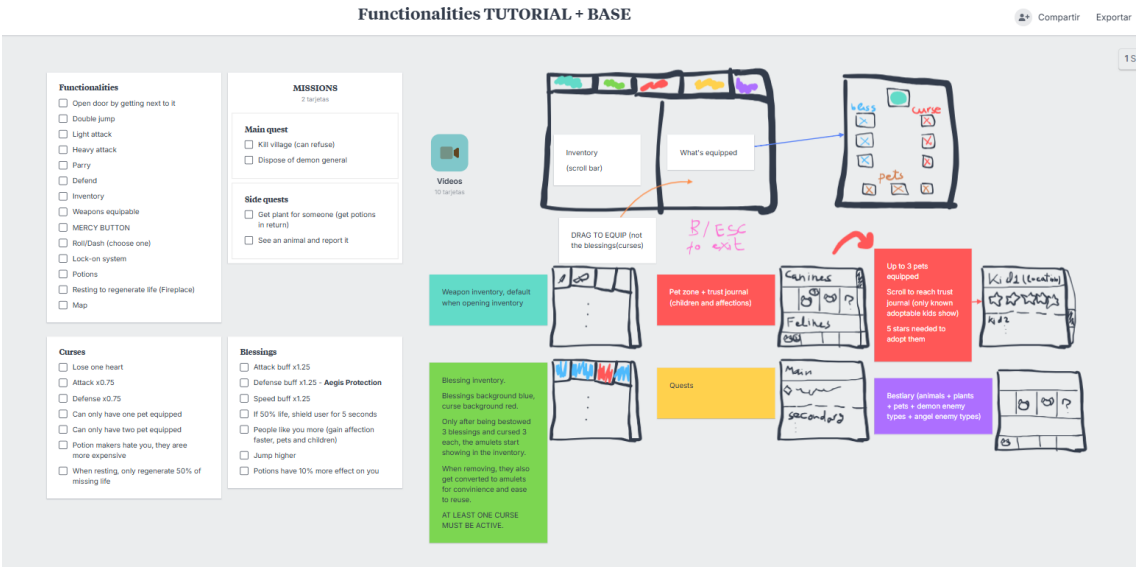


Figura 11: Milanote [22]. Funcionales bases planteadas en un inicio.

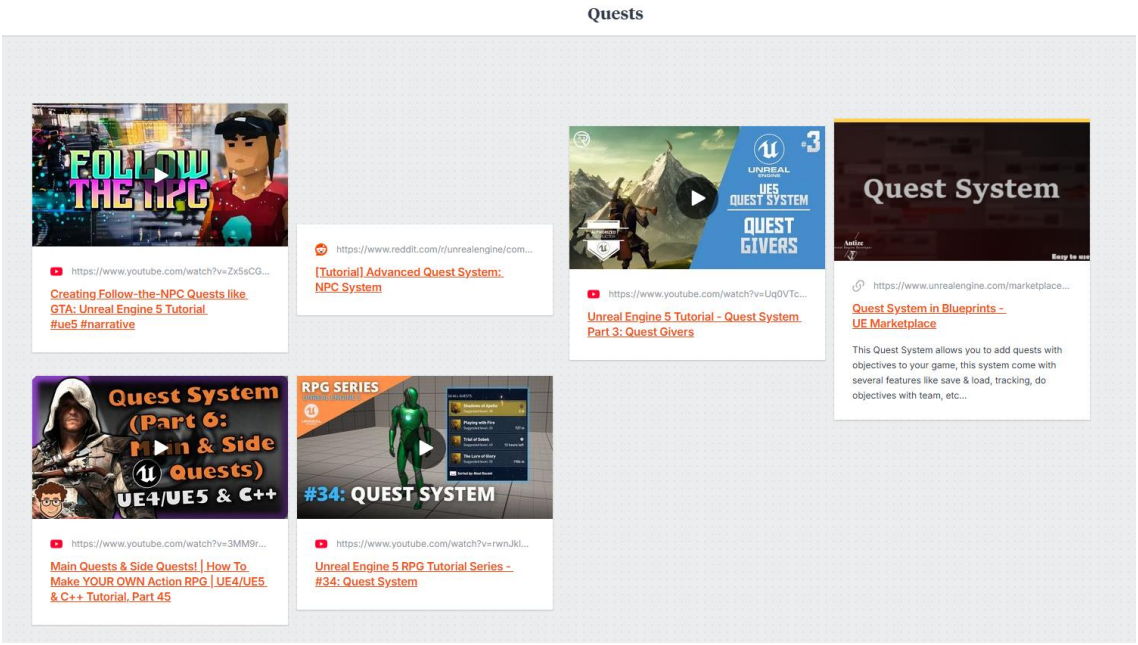


Figura 12: Milanote [22]. Información sistema de misiones.

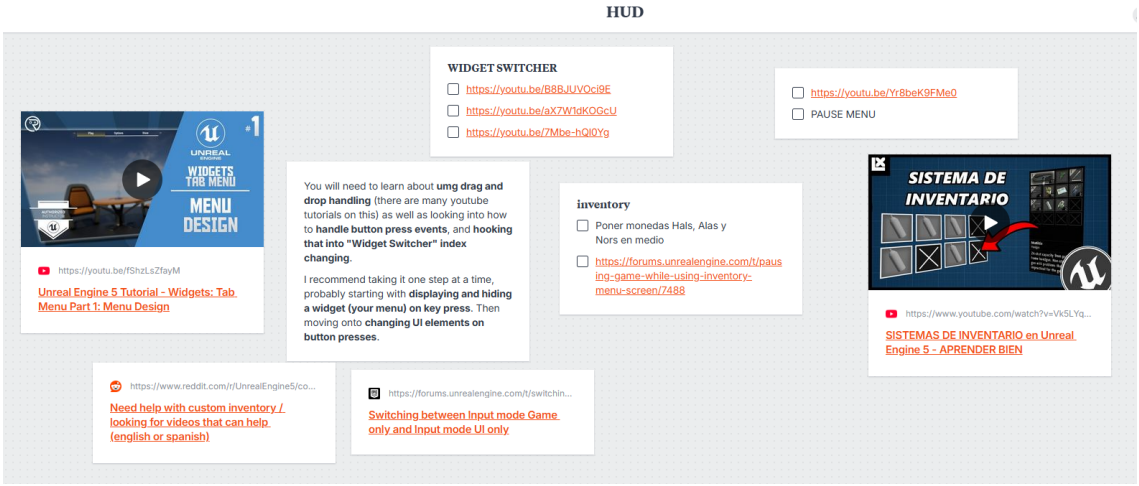


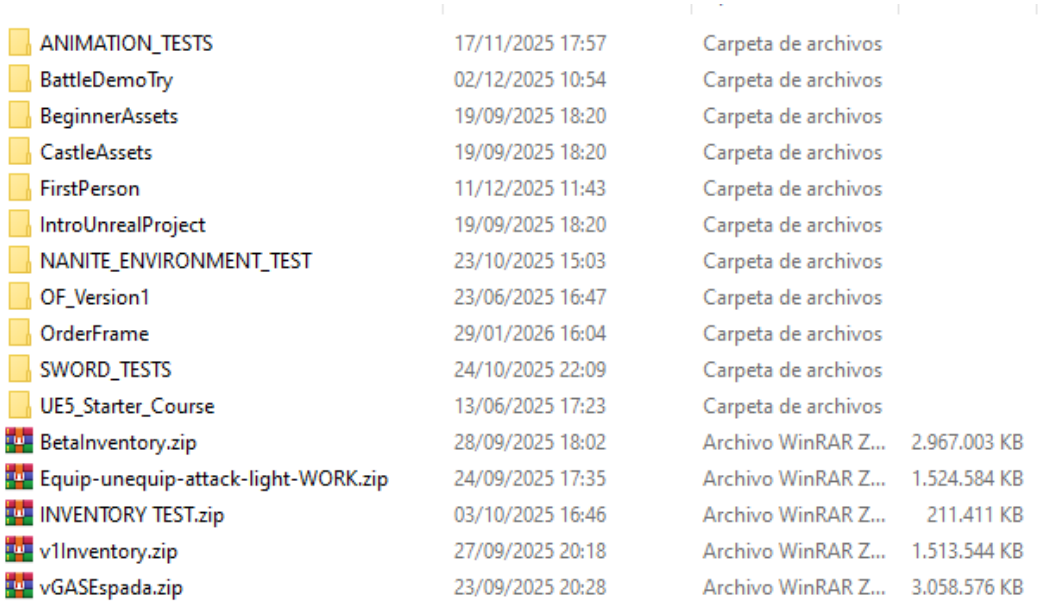
Figura 13: Milanote [22]. Información interfaz usuario.

2. **Desarrollo Modular e Independiente**

Dada la naturaleza multifacética del proyecto (mecánicas, IA, narrativa y sistemas de alineación), se ha optado por una arquitectura modular basada en Blueprints [8]. Esto permite trabajar de forma independiente en componentes clave como los Character Blueprints, Actor Blueprints y Widget Blueprints [17], para integrarlos progresivamente sin generar dependencias rígidas.

3. **Aplicación del Método Científicoy Resolución de Retos**

Se ha empleado el método científico para investigar soluciones a problemas técnicos críticos, siguiendo un ciclo de recopilación de información, formulación de hipótesis y contrastación mediante experimentos. Un ejemplo destacado fue la obtención de modelos 3D: tras descartar la búsqueda en repositorios y el modelado manual en Blender por su alta carga temporal, se experimentó con diversas IA generativas (Meshy AI [21], Rodin AI [29]) hasta validar un flujo de trabajo funcional que integra imágenes conceptuales y *auto-rigging* en Mixamo [24].



ANIMATION_TESTS	17/11/2025 17:57	Carpeta de archivos	
BattleDemoTry	02/12/2025 10:54	Carpeta de archivos	
BeginnerAssets	19/09/2025 18:20	Carpeta de archivos	
CastleAssets	19/09/2025 18:20	Carpeta de archivos	
FirstPerson	11/12/2025 11:43	Carpeta de archivos	
IntroUnrealProject	19/09/2025 18:20	Carpeta de archivos	
NANITE_ENVIRONMENT_TEST	23/10/2025 15:03	Carpeta de archivos	
OF_Version1	23/06/2025 16:47	Carpeta de archivos	
OrderFrame	29/01/2026 16:04	Carpeta de archivos	
SWORD_TESTS	24/10/2025 22:09	Carpeta de archivos	
UE5_Starter_Course	13/06/2025 17:23	Carpeta de archivos	
BetaInventory.zip	28/09/2025 18:02	Archivo WinRAR Z...	2.967.003 KB
Equip-unequip-attack-light-WORK.zip	24/09/2025 17:35	Archivo WinRAR Z...	1.524.584 KB
INVENTORY TEST.zip	03/10/2025 16:46	Archivo WinRAR Z...	211.411 KB
v1Inventory.zip	27/09/2025 20:18	Archivo WinRAR Z...	1.513.544 KB
vGASEspada.zip	23/09/2025 20:28	Archivo WinRAR Z...	3.058.576 KB

Figura 14: Ejemplos de versiones guardadas manualmente.

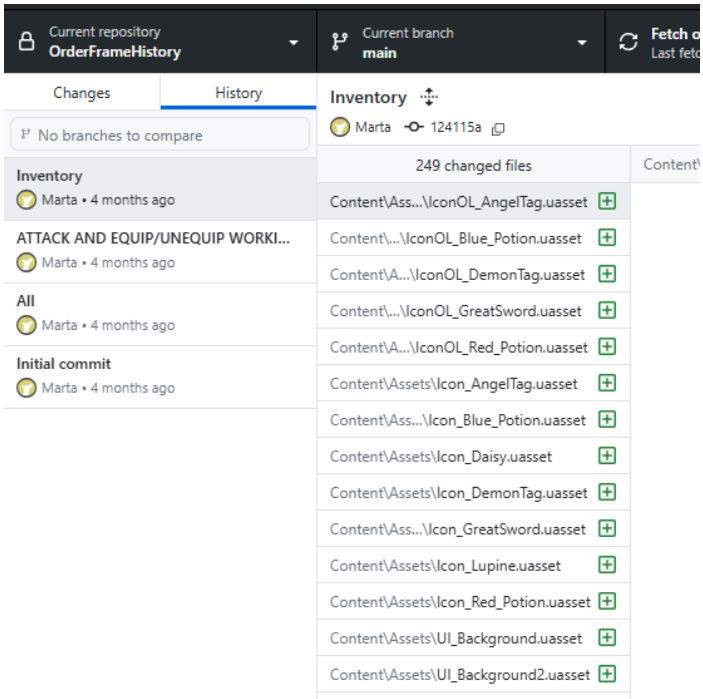


Figura 15: Uno de los repositorios utilizados en GitHub [35].

2.5. Tecnologías

2.5.1. Unreal Engine 5.6

Unreal Engine 5.6 [16] es un motor de desarrollo de videojuegos de alto rendimiento utilizado para la creación de entornos interactivos en tiempo real. Destaca por su capacidad para gestionar gráficos avanzados, físicas, inteligencia artificial y sistemas complejos de juego, permitiendo el desarrollo tanto de prototipos como de productos finales de gran escala. Su arquitectura facilita la integración de múltiples sistemas, así como el desarrollo modular y escalable de videojuegos.

2.5.1.1. Blueprints

Blueprints [8] es el sistema de scripting visual integrado en Unreal Engine [16] que permite implementar lógica de juego sin necesidad de programación textual. Mediante nodos y conexiones visuales, se pueden definir comportamientos, eventos y flujos de ejecución, facilitando el desarrollo rápido, la depuración y la iteración de mecánicas de juego, especialmente en prototipos y sistemas complejos.

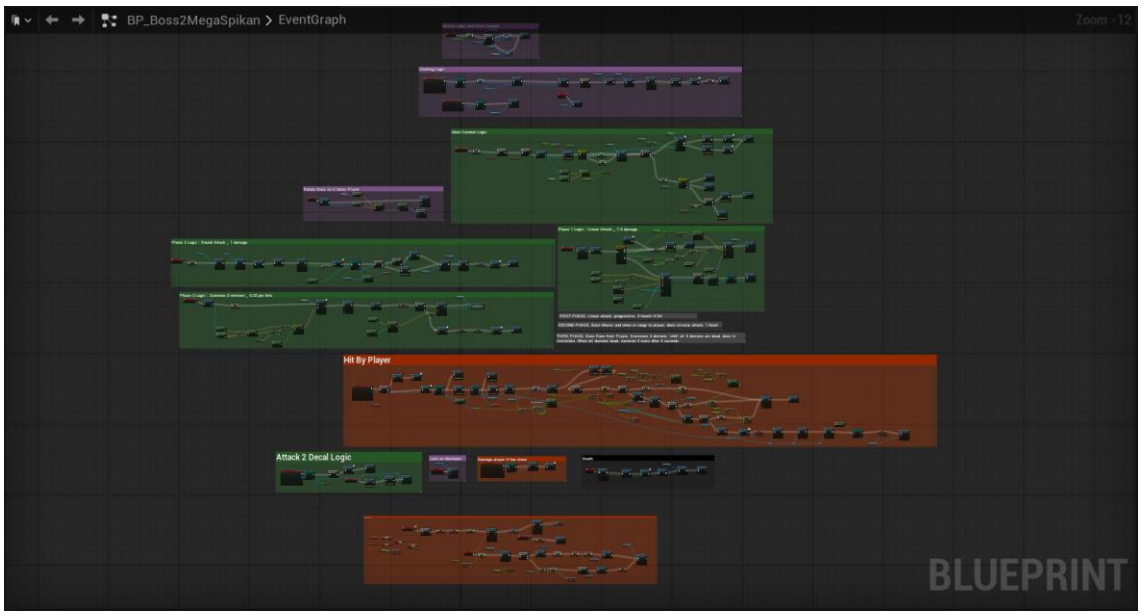


Figura 16: Ejemplo de interfaz de Blueprints [8].

2.5.1.2. Plugins

Los Plugins [14] de Unreal Engine [16] son módulos adicionales que amplían las funcionalidades nativas del motor mediante la incorporación de herramientas, sistemas o integraciones específicas. Pueden estar orientados a múltiples ámbitos del desarrollo, como gráficos avanzados, inteligencia artificial, físicas, audio, conectividad en red o compatibilidad con software externo. Su arquitectura modular permite activarlos o desactivarlos según las necesidades del proyecto, favoreciendo la optimización del rendimiento y la escalabilidad del desarrollo. Asimismo, facilitan la reutilización de sistemas y la incorporación de soluciones desarrolladas por terceros o por la propia comunidad.

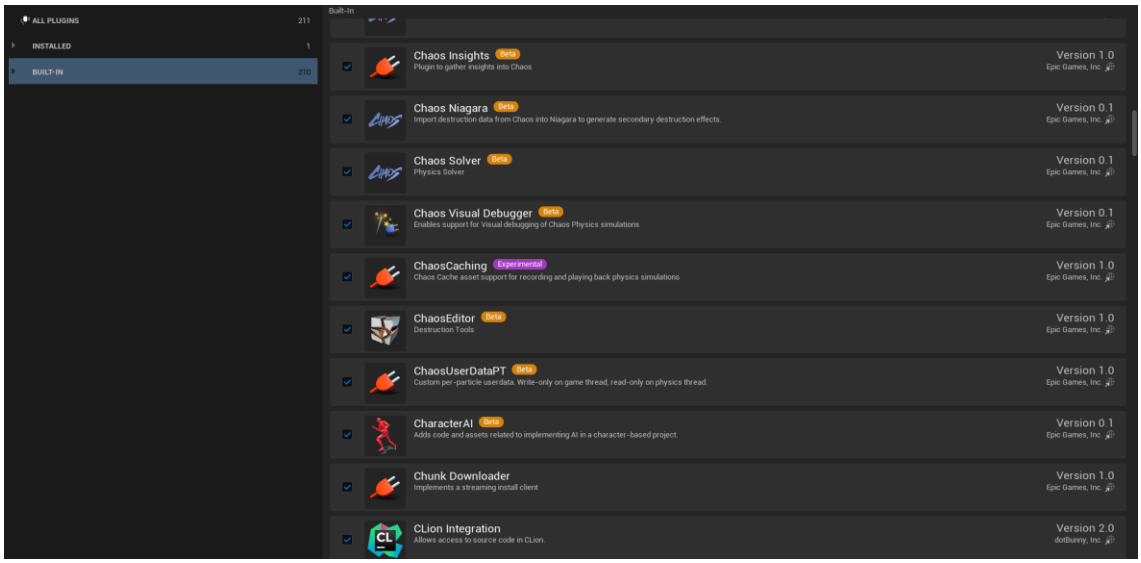


Figura 17: Ejemplo de Plugins [14] activos.

2.5.1.3. Widget Blueprints

Los *Widget Blueprints [17]* son componentes utilizados para la creación de interfaces de usuario dentro de Unreal Engine [16]. Permiten diseñar elementos visuales interactivos como menús, HUDs e inventarios, facilitando la comunicación entre el jugador y el sistema de juego.

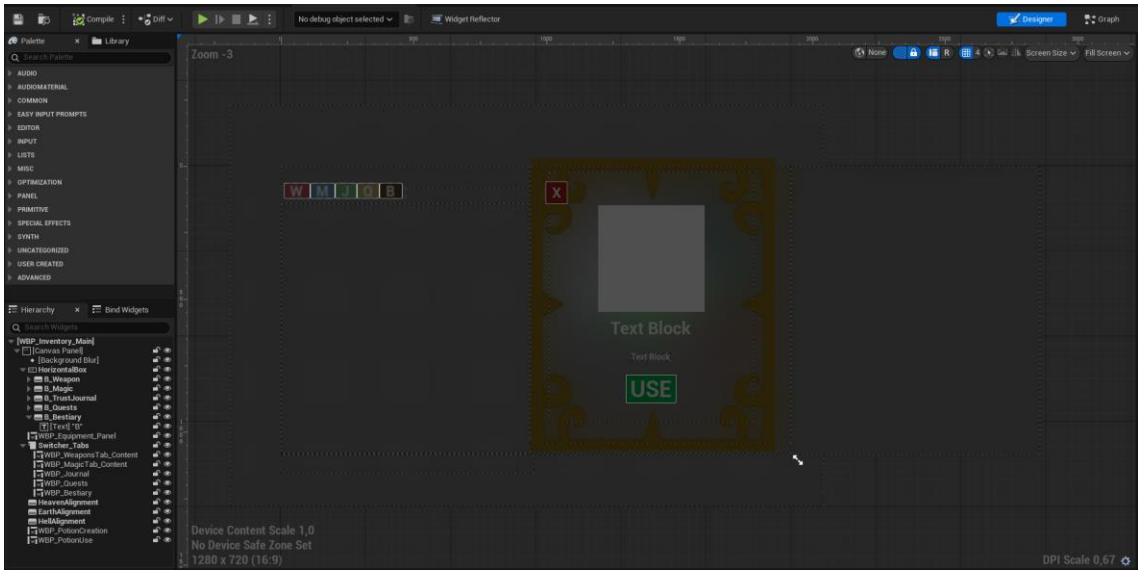


Figura 18: Ejemplo de Designer en un Widget Blueprint [17].

2.5.1.4. Niagara System

Niagara System [13] es el sistema avanzado de creación y gestión de efectos visuales (VFX) de Unreal Engine [16]. Permite diseñar simulaciones complejas de partículas como fuego, humo, chispas, fluidos, energía o efectos atmosféricos mediante un enfoque modular basado en nodos. A diferencia de sistemas anteriores, Niagara ofrece un mayor control sobre el comportamiento físico, la interacción con el entorno y la optimización del rendimiento en tiempo real. Su uso resulta fundamental para reforzar la inmersión

visual del jugador y para la representación de habilidades, impactos, entornos dinámicos y elementos narrativos dentro del juego.

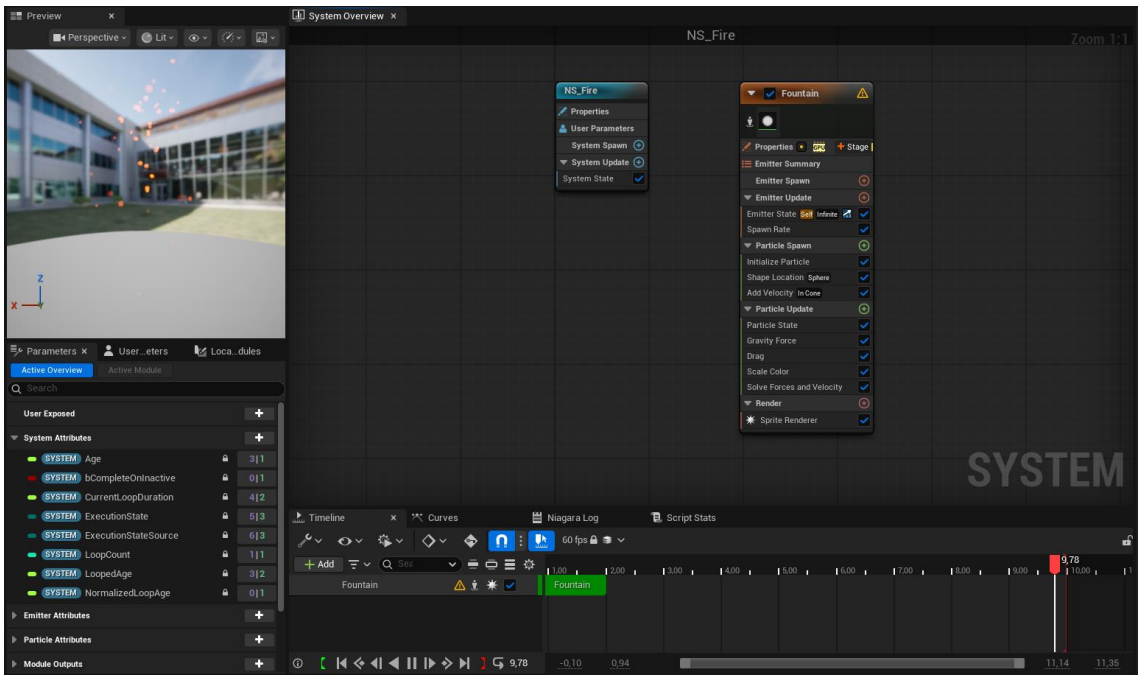


Figura 19: Ejemplo de Niagara System [13].

2.5.1.5. Save Game Blueprints

Los *Save Game Blueprints [15]* son un sistema de Unreal Engine [16] destinado al almacenamiento persistente del estado del juego. Permiten guardar y cargar información relevante, como progreso del jugador, decisiones tomadas o configuraciones del mundo, garantizando la continuidad de la experiencia de juego.

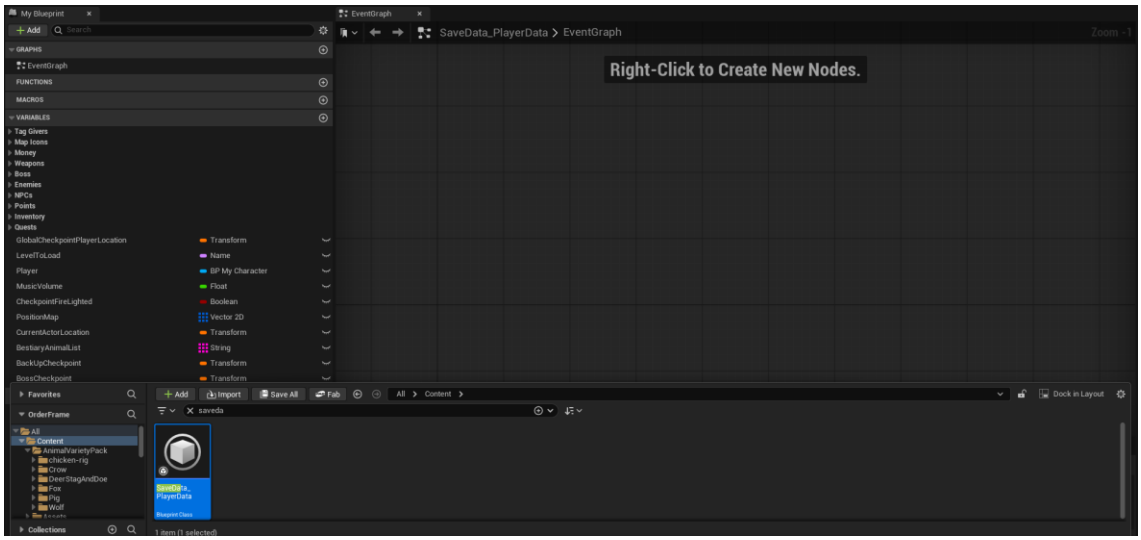


Figura 20: Ejemplo de Save Game blueprint [15].

2.5.1.6. Nav Mesh

El sistema *Nav Mesh* (Navigation Mesh) [12] se utiliza para definir áreas navegables dentro del entorno del juego, permitiendo a los personajes controlados por inteligencia artificial desplazarse de forma coherente y eficiente. Es fundamental para el comportamiento de enemigos y NPCs en mundos abiertos.

Figura 21: Ejemplo de Nav Mesh [12] activado.

2.5.1.7. Sistemas de control de eventos e interfaces

Los sistemas de control de eventos [10] e interfaces [11] permiten gestionar la comunicación entre distintos componentes del juego, coordinando acciones, respuestas y flujos de información. Estos sistemas son esenciales para la implementación de mecánicas complejas, narrativa reactiva e interacción entre el jugador y el entorno.

2.5.2. Blender

Blender [4] es una herramienta de código abierto destinada al modelado, animación y renderizado en 3D. Se utiliza ampliamente en el desarrollo de videojuegos para la creación y edición de modelos, texturas y animaciones, ofreciendo un flujo de trabajo flexible y compatible con motores de juego modernos.

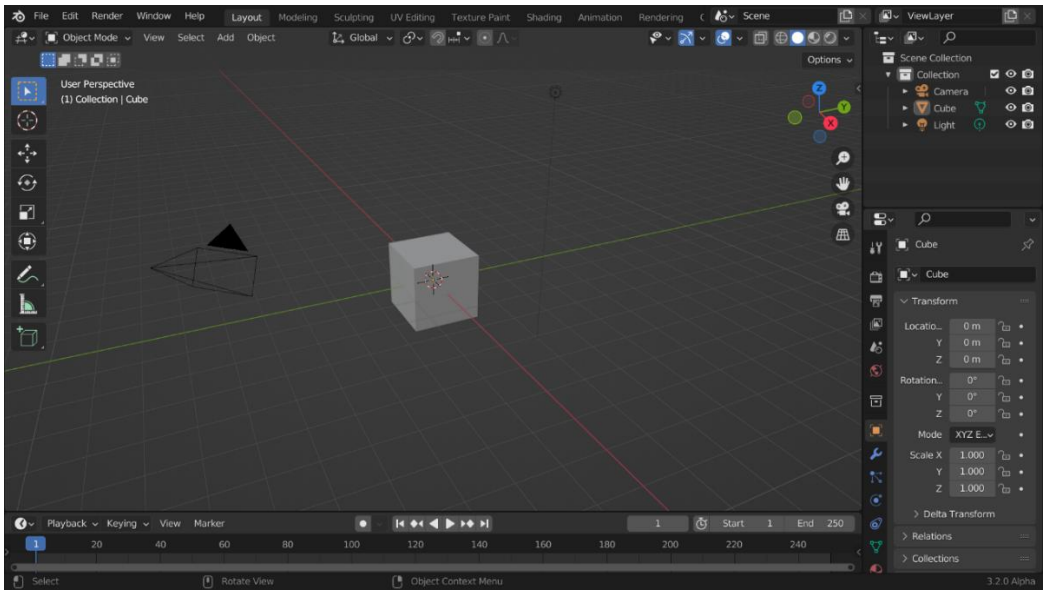


Figura 22: UI de Blender [4].

2.5.3. Mixamo

Mixamo [24] es una plataforma que permite la generación automática de animaciones para modelos 3D, especialmente personajes humanoides. Facilita la aplicación de esqueletos (*rigging*) y animaciones predefinidas, reduciendo el tiempo necesario para integrar personajes animados en un motor de juego.

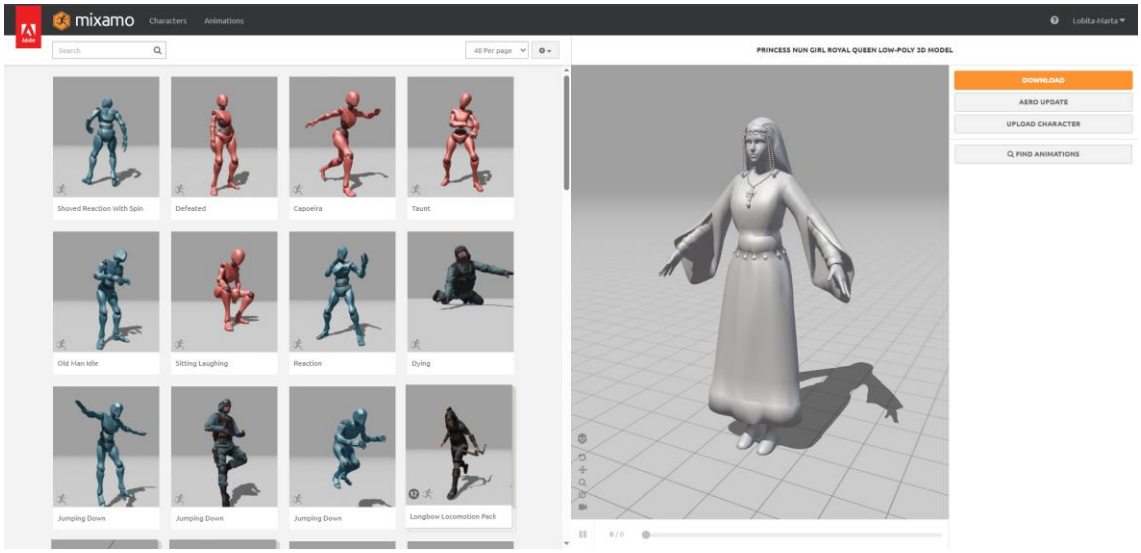


Figura 23: UI de Mixamo [24].

2.5.4. Rodin AI (Hyper3D)

Rodin AI [29], también conocida como Hyper3D, es una herramienta basada en inteligencia artificial que permite la generación de modelos tridimensionales a partir de descripciones o referencias visuales. Su uso resulta especialmente útil en la creación rápida de assets 3D durante fases tempranas de desarrollo o prototipado.

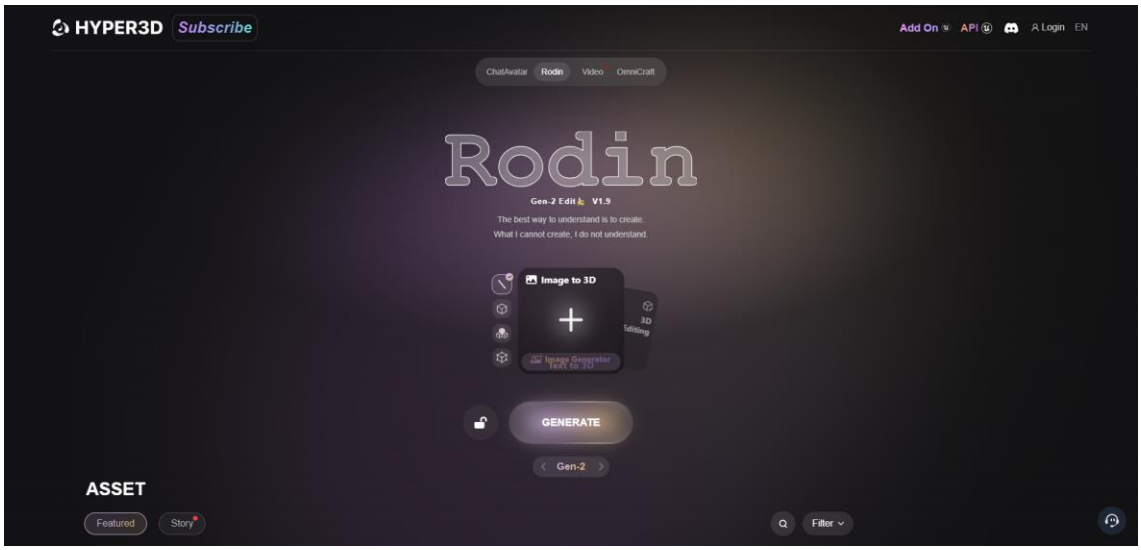


Figura 24: UI de inicio de Rodin AI [29].

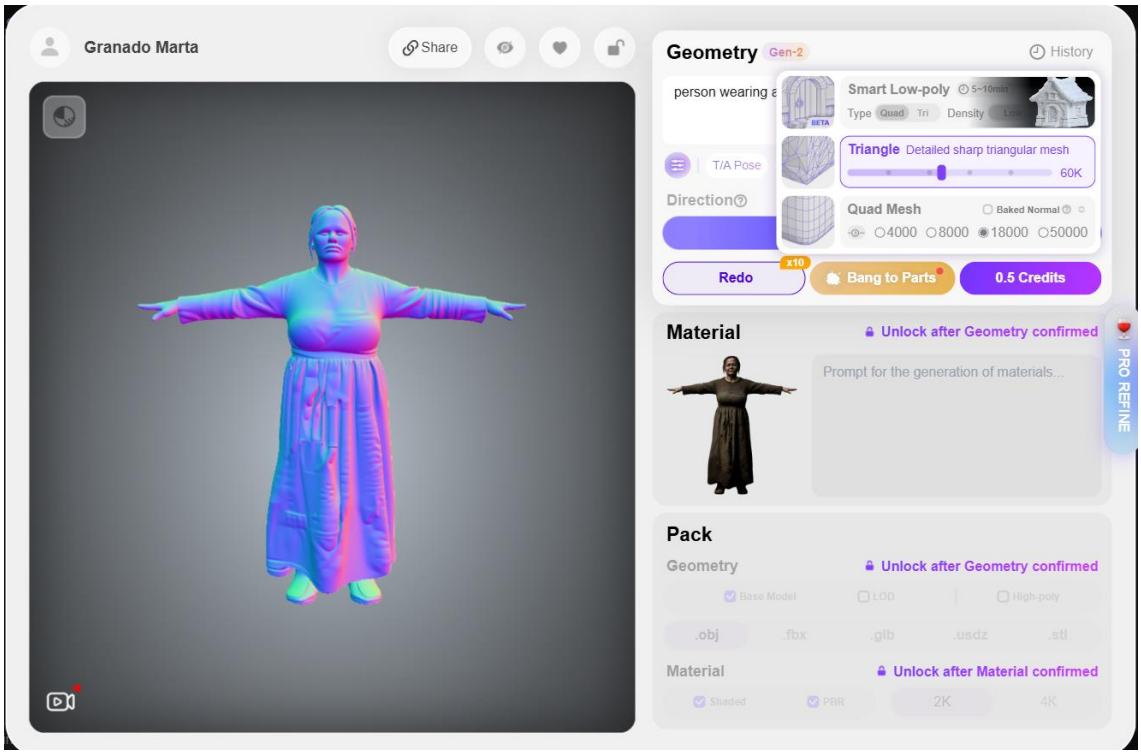


Figura 25: UI funcional de Rodin AI [29].

2.5.5. Meshy AI

Meshy AI [21] es una plataforma de generación de modelos 3D mediante inteligencia artificial, orientada a la creación rápida de mallas y objetos tridimensionales. Esta herramienta facilita la producción de contenido visual sin requerir conocimientos avanzados de modelado tradicional.

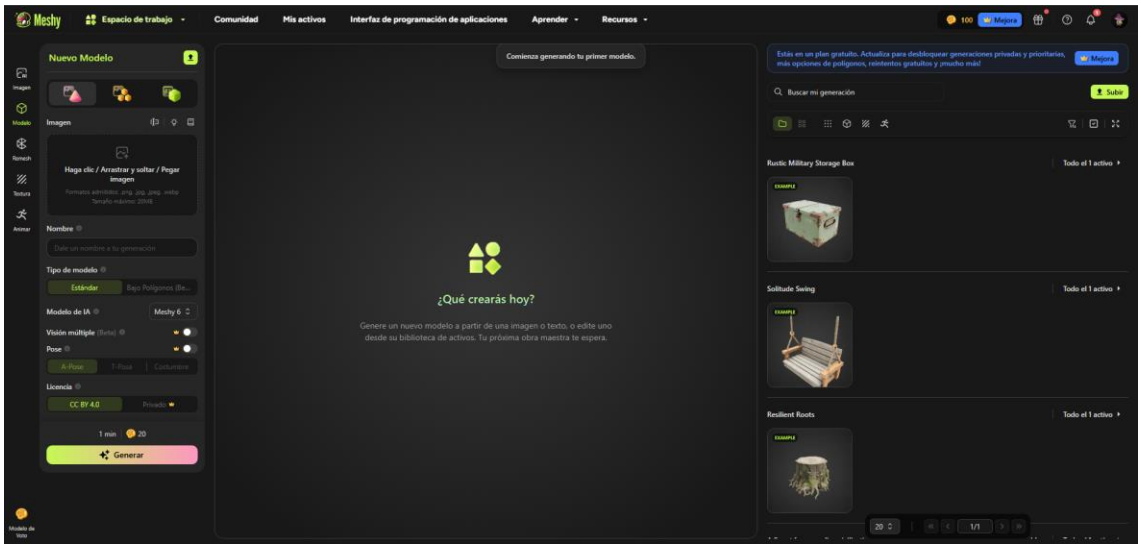


Figura 26: UI de Meshy AI [21].

2.5.6. Craiyon AI

Craiyon AI [6] es una herramienta de generación de imágenes a partir de texto que emplea modelos de inteligencia artificial. En el contexto del desarrollo de videojuegos, puede utilizarse para la conceptualización visual, creación de referencias artísticas o diseño preliminar de personajes y entornos.

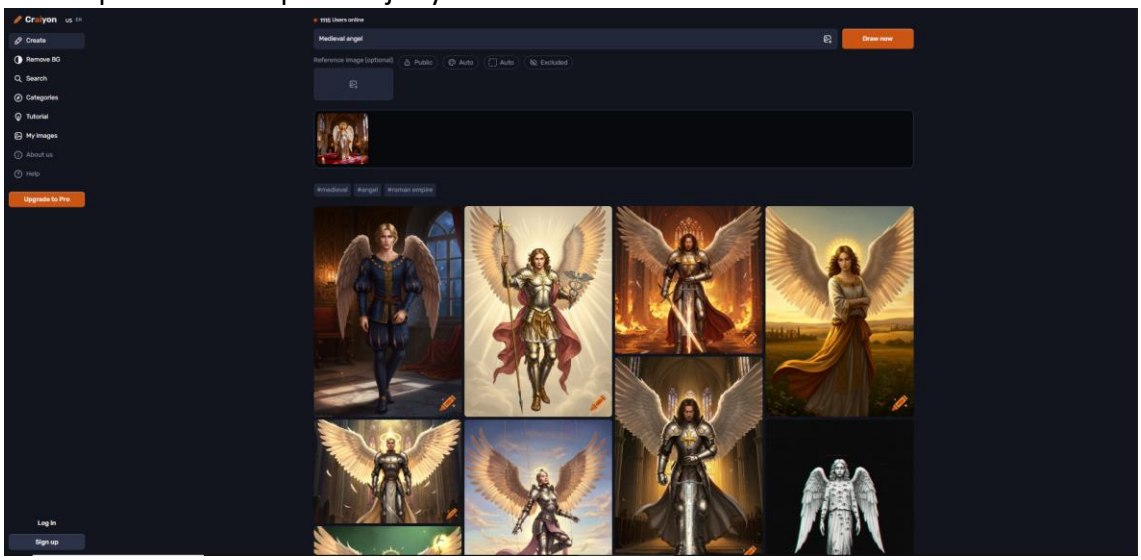


Figura 27: UI de Craiyon AI [6].

2.5.7. Nano Banana

Nano Banana [26] es un modelo de generación de imágenes basado en inteligencia artificial, diseñado para producir contenido visual a partir de descripciones textuales. Su aplicación resulta útil en la fase creativa del proyecto, especialmente para la generación de ideas visuales y apoyo al diseño artístico.

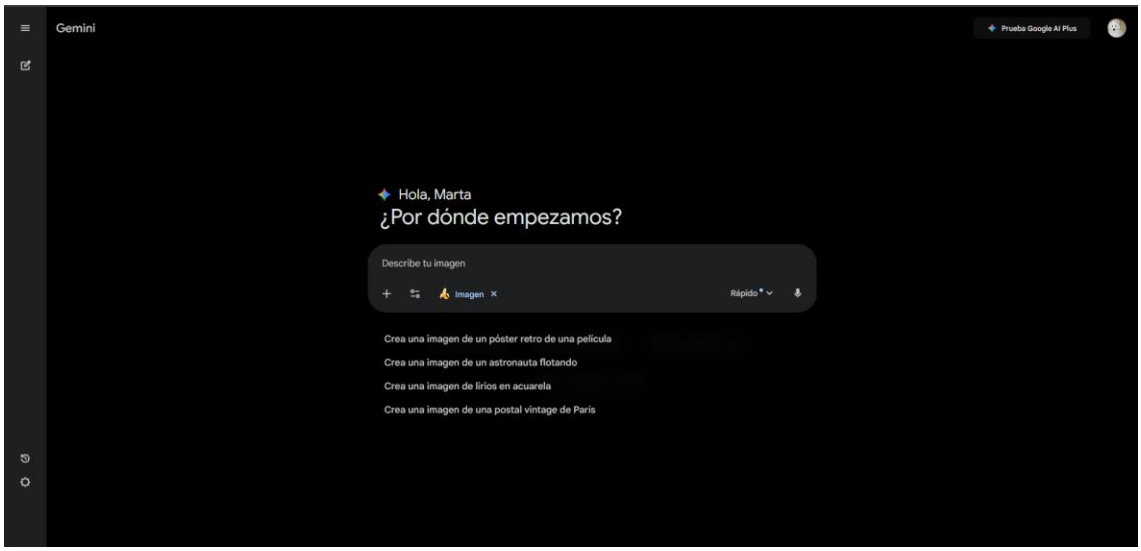


Figura 28: UI de Nano Banana [26].

2.5.8. VisualGPT

VisualGPT [33] es una herramienta basada en inteligencia artificial que permite la edición y modificación de imágenes a partir de instrucciones en lenguaje natural. En el contexto del proyecto, se ha utilizado principalmente para refinar, ajustar y transformar imágenes previamente generadas, facilitando la iteración visual y el ajuste de elementos gráficos durante las fases de diseño y prototipado.

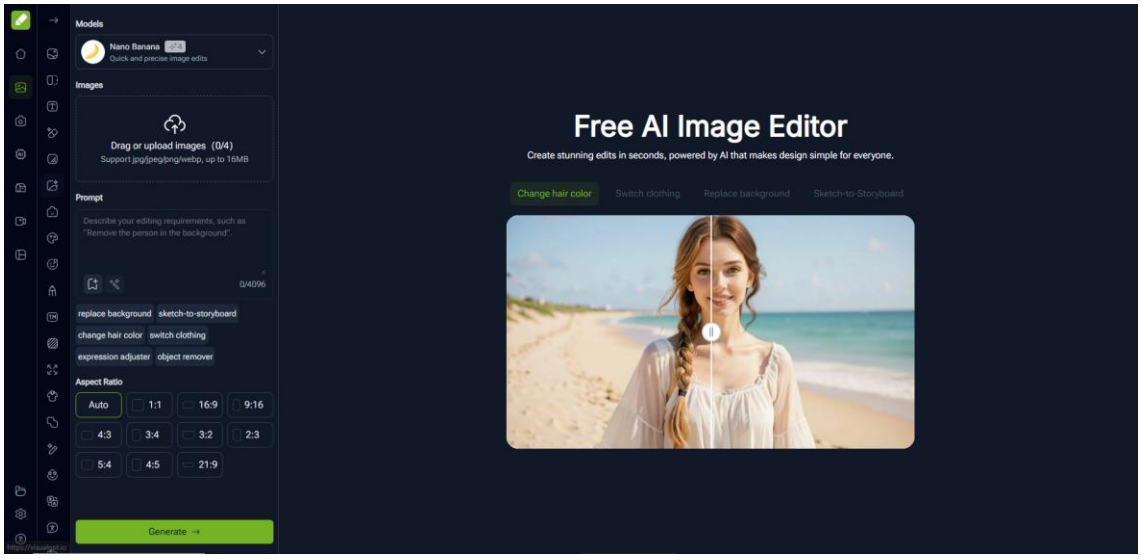


Figura 29: UI de VisualGPT [33].

2.5.9. Git y GitHub

Git [32] es un sistema de control de versiones distribuido que permite gestionar y registrar los cambios realizados en archivos a lo largo del tiempo. Su funcionamiento facilita el seguimiento del historial del proyecto, la recuperación de versiones anteriores y el trabajo simultáneo sobre el mismo código sin pérdida de información. En el contexto del desarrollo de software y videojuegos, Git resulta fundamental para mantener la integridad del proyecto y gestionar de forma eficiente la evolución del código fuente.

GitHub [35] es una plataforma basada en la web que utiliza Git como sistema de control de versiones y proporciona servicios adicionales orientados a la colaboración y gestión de proyectos. Permite alojar repositorios de código, coordinar el trabajo entre desarrolladores, documentar el proyecto y realizar un seguimiento de incidencias y mejoras. En este proyecto, GitHub se emplea como herramienta central para el almacenamiento del código y la organización del desarrollo, facilitando el control y mantenimiento del trabajo realizado.

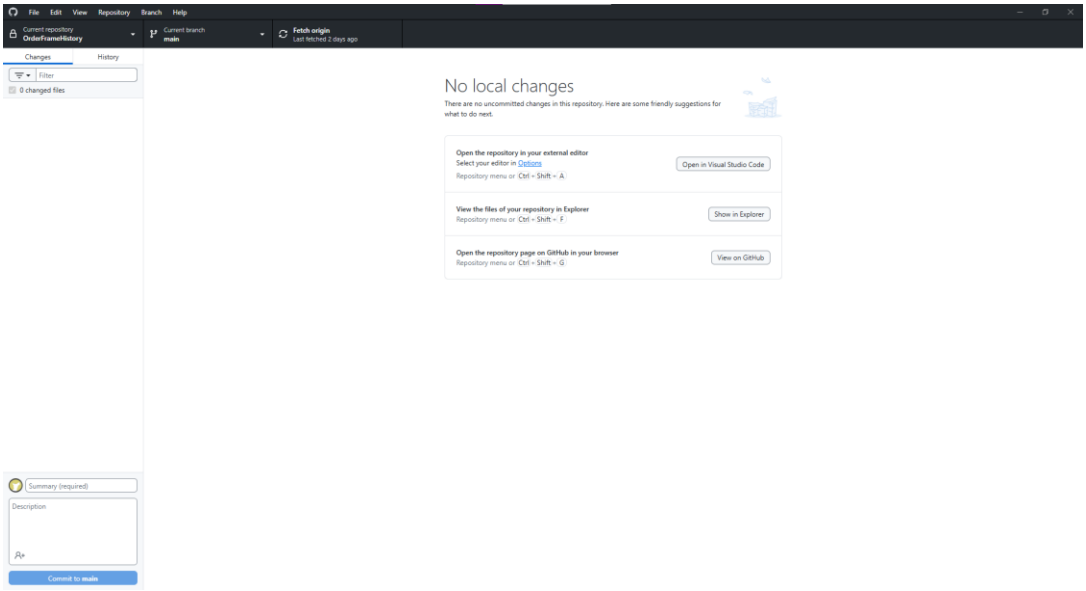


Figura 30: UI de Github Desktop [35].

2.5.10. Trello

Trello [2] es una herramienta de gestión de proyectos basada en tableros visuales que permite organizar tareas, establecer prioridades y realizar un seguimiento del progreso. Se utiliza para planificar y coordinar las diferentes fases del desarrollo, especialmente en entornos de trabajo iterativos.

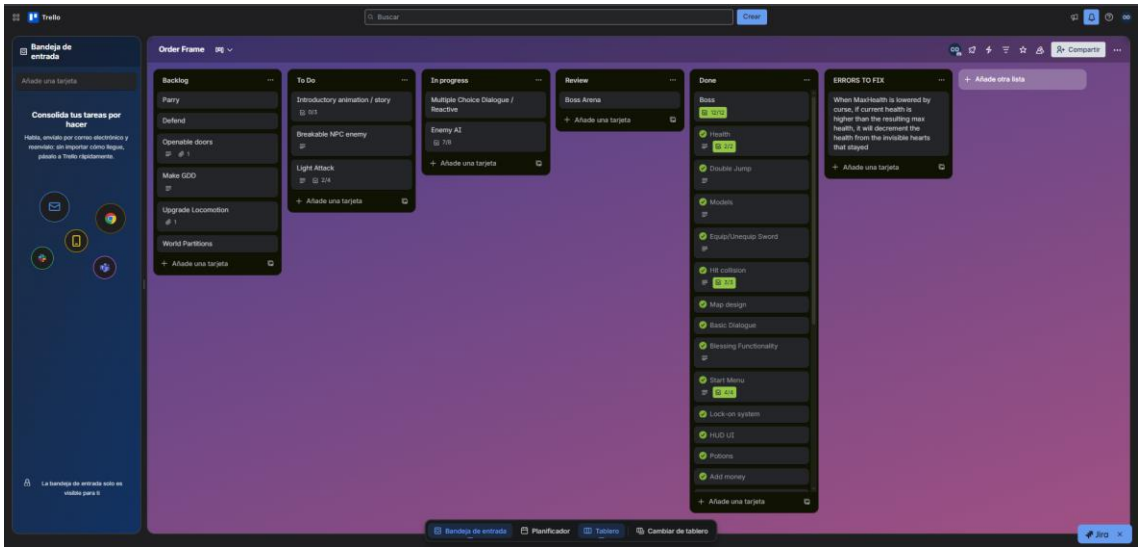


Figura 31: UI de Trello [2].

2.5.11. Milanote

Milanote [22] es una herramienta digital de organización visual orientada a la planificación creativa y al diseño conceptual de proyectos. Funciona mediante tableros flexibles donde es posible integrar notas, imágenes, esquemas, enlaces y diagramas, facilitando la estructuración de ideas y la construcción de moodboards. En el contexto del desarrollo de videojuegos, se emplea para la conceptualización artística, el diseño de narrativa, la planificación de mecánicas y la organización de referencias visuales. Su enfoque visual e intuitivo favorece la ideación, la coherencia estética del proyecto y la comunicación de conceptos dentro del flujo de trabajo.

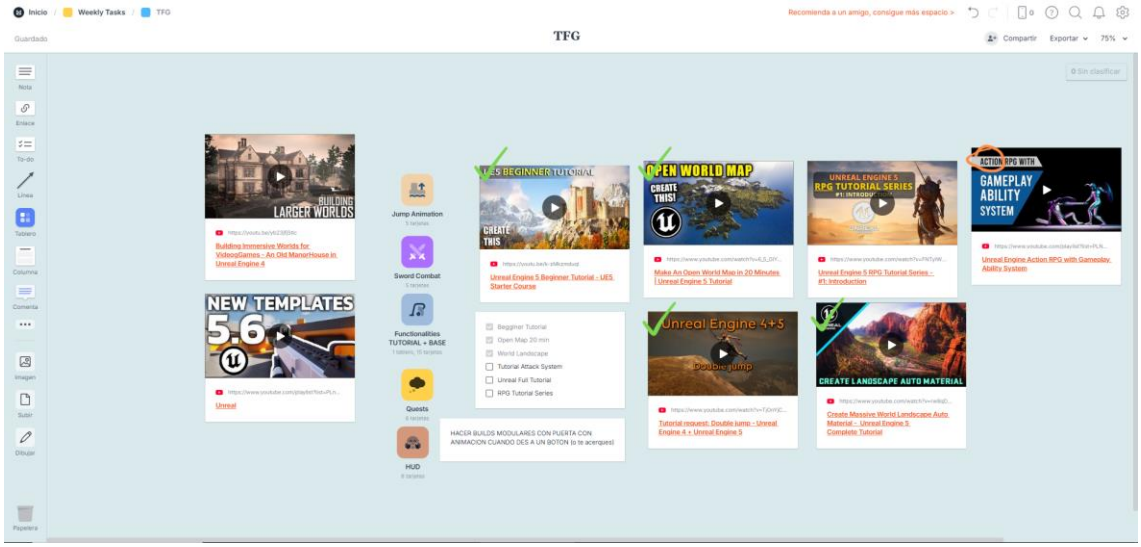


Figura 32: UI de Milanote [22].

2.5.12. Hojas de cálculo y notas

Las hojas de cálculo y herramientas de toma de notas se emplean para la organización de datos, el seguimiento de sistemas de juego, el diseño de mecánicas y la planificación del proyecto. Estas herramientas permiten estructurar información compleja de forma clara y accesible durante el desarrollo.

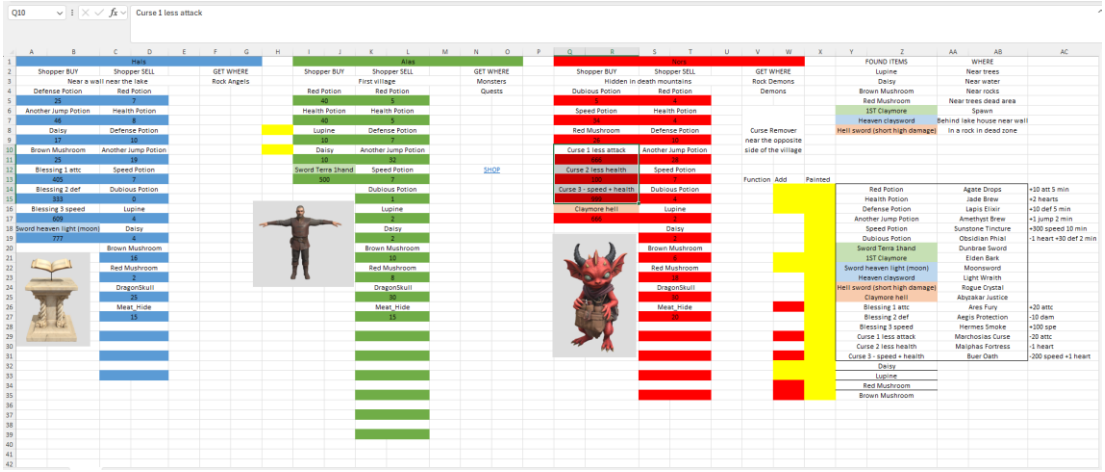


Figura 33: Ejemplo Excel organización de tiendas.

2.6. Arte

La dirección artística de Order Frame se fundamenta en la construcción de una atmósfera inmersiva de fantasía oscura, estructurada conceptualmente en tres planos existenciales: Cielo, Tierra e Infierno. Cada uno de estos dominios no solo se diferencia a nivel narrativo y jugable, sino también mediante un lenguaje visual propio que refuerza la identidad simbólica del alineamiento moral del jugador.

Con el fin de preservar la inmersión, la interfaz de usuario adopta un enfoque minimalista, reduciendo la sobrecarga de información en pantalla y priorizando la claridad visual, la legibilidad y la integración estética con el mundo de juego.

En lo relativo a la producción de assets, la mayoría de los modelos tridimensionales han sido obtenidos a través de repositorios gratuitos o generados mediante herramientas de inteligencia artificial, en línea con el flujo de trabajo híbrido adoptado para optimizar tiempos de desarrollo.
En contraste, la totalidad de los elementos gráficos bidimensionales (incluyendo iconografía, ilustraciones de interfaz y recursos visuales complementarios) han sido elaborados manualmente mediante Adobe Photoshop [1], garantizando un mayor control estilístico y coherencia estética con la dirección artística general del proyecto.

2.6.1. Modelos 3D

La creación de modelos tridimensionales constituye uno de los núcleos técnicos de la dirección artística del proyecto. Dada la naturaleza individual del desarrollo, uno de los principales retos ha sido alcanzar un nivel de calidad visual elevado manteniendo, al mismo tiempo, la viabilidad productiva y temporal del trabajo.

En una fase inicial, se planteó la creación de los modelos desde cero mediante Blender, empleando técnicas tradicionales de modelado, texturizado y preparación para motor. No obstante, este enfoque resultó altamente demandante en términos de tiempo y carga de trabajo, comprometiendo el avance en otros ámbitos considerados prioritarios dentro del proyecto, especialmente el desarrollo de sistemas jugables, arquitectura modular y lógica de juego. Debido a ello, se determinó que la producción íntegra de *assets* 3D de forma manual no era viable dentro de los plazos establecidos.

Paralelamente, se intentó recurrir a repositorios de modelos gratuitos como Sketchfab [30]. Sin embargo, la temática específica del juego, dificultó la localización de *assets* que cumplieran simultáneamente los requisitos de calidad, coherencia estética y adaptabilidad técnica necesarios para su integración en el proyecto.

Ante estas limitaciones, se optó por implementar un **flujo de trabajo híbrido asistido por inteligencia artificial**, combinando herramientas de generación automática con procesos tradicionales de preparación e integración en motor. Inicialmente, se empleó la plataforma Meshy AI [21] para la generación de modelos tridimensionales; no obstante, la transición de la herramienta a un modelo de suscripción exclusivamente de pago obligó a replantear el pipeline de producción y a buscar alternativas viables.

El pipeline adoptado finalmente es el siguiente:

1. **Generación conceptual 2D** mediante Craiyon AI [6] y Nano Banana [26], utilizadas para producir referencias visuales iniciales de personajes, criaturas y objetos.
2. **Refinamiento y limpieza** de las imágenes generadas a través de VisualGPT [33], ajustando proporciones, eliminando artefactos y homogeneizando el estilo artístico.
3. **Conversión a modelo 3D** mediante Rodin AI (Hyper3D) [29], que permite transformar las referencias bidimensionales en mallas tridimensionales funcionales.
4. **Auto-rigging y animación** en Mixamo [24], facilitando la incorporación de esqueletos y animaciones base listas para su implementación en Unreal Engine [16].

Este flujo de trabajo ha permitido acelerar significativamente la producción de *assets*, manteniendo al mismo tiempo coherencia estética, funcionalidad técnica y compatibilidad con los sistemas de animación e integración del motor gráfico.

2.6.2. Personajes

2.6.2.1. Jugador

El diseño del protagonista (denominado Caelvorn, Elior o Azrakar según su evolución narrativa) se concibe como un reflejo visual directo de su facilidad de transición moral dentro del sistema de alineamiento del juego. Su concepción estética parte de una dualidad física explícita: una mitad de su cuerpo presenta rasgos angelicales, acompañados de un ala de naturaleza celestial, mientras que la otra mitad adopta características demoníacas, incluyendo un ala de morfología infernal. Esta división corporal actúa como metáfora visual permanente del conflicto ético que define su progresión narrativa.

Durante las fases iniciales de diseño conceptual, el personaje incorporaba además dos cuernos demoníacos que sostenían un halo en su interior, reforzando simbólicamente la coexistencia de ambas naturalezas. No obstante, este elemento fue finalmente descartado en la versión definitiva del modelo debido a criterios de simplicidad visual, legibilidad a media distancia y optimización estética dentro del entorno de juego.

En la Figura 34 puede observarse el primer boceto conceptual del protagonista, donde se aprecian los elementos originales planteados en su diseño preliminar.

Además del personaje principal, el mundo de juego se encuentra poblado por una amplia variedad de personajes no jugables (NPCs), tanto de carácter amigable como hostil. Estos cumplen funciones sistémicas, narrativas y comerciales dentro de la experiencia. En apartados posteriores se analizarán con mayor profundidad tanto los NPCs funcionales como las entidades enemigas presentes en el juego.

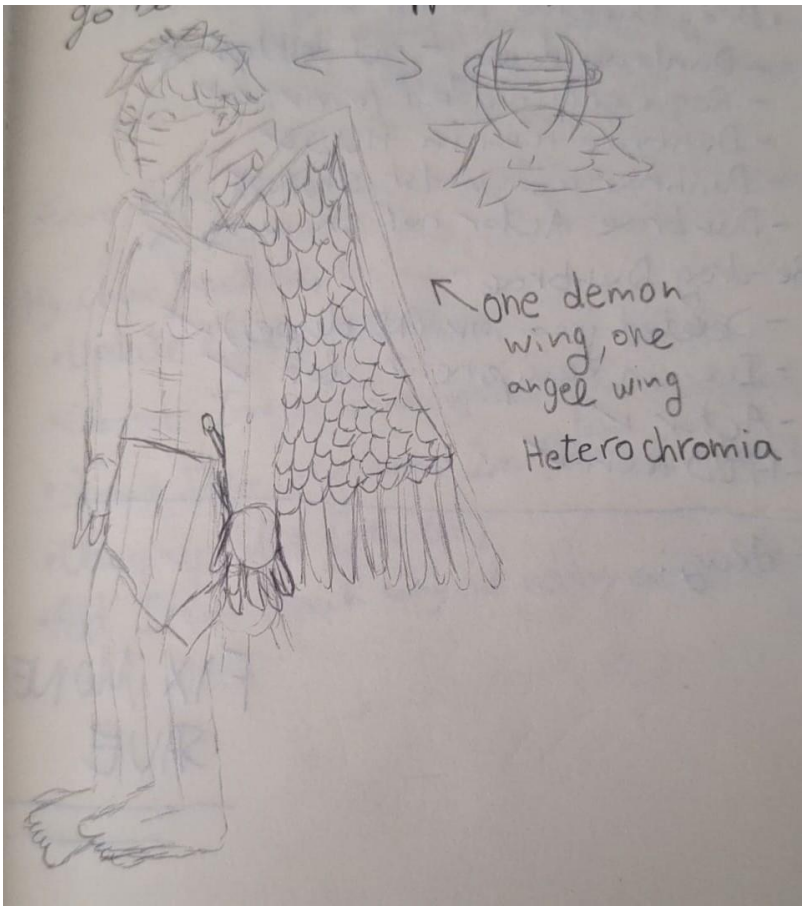


Figura 34: Boceto inicial concepción protagonista.


Figura 35: Modelo actual del protagonista.

2.6.2.2. Modelos NPCs

En el presente apartado se recogen la totalidad de los modelos de NPC empleados a lo largo del desarrollo del juego. La mayor parte de estos recursos ha sido generada mediante el uso de herramientas de inteligencia artificial, siguiendo el pipeline de producción previamente descrito en el apartado 2.6.1.

Figura 36: NPC. Chica de las flores.

Figura 37: NPC. Chico genérico.

Figura 38: NPC. Granjero.

Figura 39**:** NPC. Hombre pobre.

Figura 40**:** NPC. Niño rico.

Figura 41**:** NPC. Hombre genérico, misión fuerza.

Figura 42**:** NPC. Guardia Seraphen.

Figura 43**:** NPC. Caballero Seraphen.

Figura 44**:** NPC. Niño atacado.

Figura 45**:** NPC. Niña genérica.

Figura 46**:** NPC. Hombre genérico.

Figura 47**:** NPC. Bruja.

Figura 48**:** NPC. Niña manzanas.

Figura 49**:** NPC. Mujer pobre.

Figura 50**:** NPC. Hombre genérico 2, guía, comerciante tierra.

Figura 51**:** NPC. Cuervo.

Figura 52**:** NPC. Ciervo hembra.

Figura 53**:** NPC. Ciervo macho.

Figura 54**:** NPC. Cerdo.

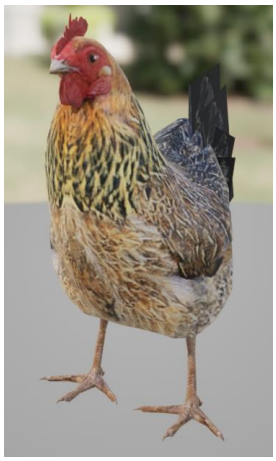


Figura 55: NPC. Gallina.

Figura 56: NPC. Pollito.

Figura 57: NPC. Lobo, enemigo tierra, acompañantes.

Figura 58: NPC. Gato.

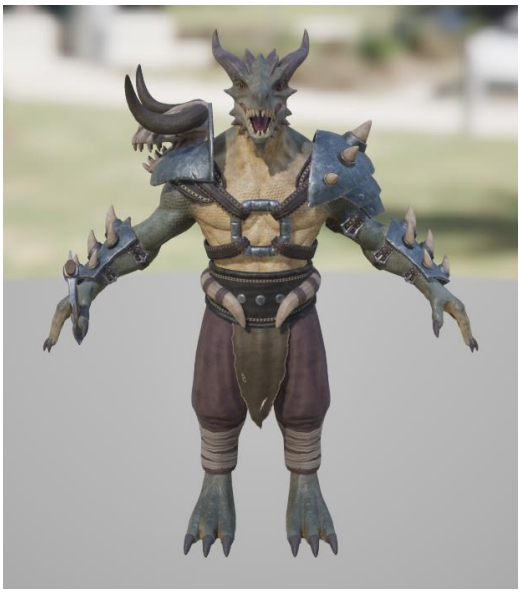


Figura 59**:** NPC. Enemigo infierno. Lizard.

Figura 60**:** NPC. Enemigo cielo. Esqueleto.

Figura 61**:** NPC. Jefe final.

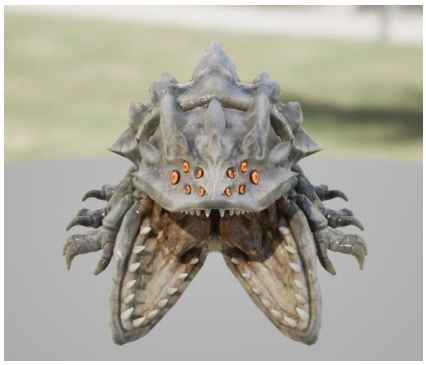


Figura 62**:** NPC. Súbdito jefe final.

Figura 63**:** NPC. Entregador de bendiciones.

Figura 64**:** NPC. Entregador de maldiciones.

Figura 65: NPC. Entregador de bestiario.

Figura 66: NPC. Tienda cielo.

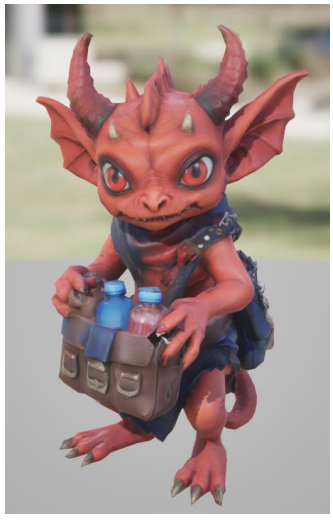


Figura 67: NPC. Tienda infierno.

Figura 68: NPC. Princesa Seraphen.

2.6.3. Entorno y mundo

El mundo de juego se ha construido empleando Unreal Engine 5.6, priorizando la creación de mapas de gran escala con alta densidad ambiental.

Para ello se han utilizado principalmente los siguientes sistemas del motor:

- Landscape Mode: empleado para el esculpido del terreno, permitiendo generar montañas, valles, desiertos y llanuras con control sobre materiales y capas de textura.

- Foliage System: utilizado para la dispersión masiva de vegetación (árboles, arbustos, hierba), optimizado mediante técnicas de HLOD (Hierarchical Level of Detail) para mantener el rendimiento a larga distancia.

- Post-processing: aplicado para reforzar la atmósfera, incluyendo efectos de distorsión, niebla volumétrica e iluminación cromática. Destaca el uso de tinte azul y distorsión óptica en entornos submarinos.

- Modeling Mode: edición directa de mallas dentro del propio motor. Esta herramienta ha resultado especialmente útil para la adaptación de assets gratuitos, permitiendo la separación de sub-mallas, el centrado de pivotes y la simplificación estructural de modelos (como en el caso de formaciones de cristales) con el objetivo de optimizar su colocación, reutilización e integración funcional dentro del entorno de juego.

Para la gestión eficiente de escenarios de gran extensión, se ha implementado el sistema de **World Partition / Level Streaming**, que permite segmentar el mundo en celdas cargadas y descargadas dinámicamente en función de la posición del jugador. Este enfoque optimiza el uso de memoria y rendimiento, posibilitando la construcción de mapas continuos sin pantallas de carga perceptibles, al tiempo que facilita la organización modular del entorno durante el desarrollo.

Regiones como Eldhollow o Death Valley ejemplifican la integración de iluminación fija, composición paisajística y densidad vegetal para generar biomas diferenciados.

Figura 69: Captura de entorno mostrando iluminación y vegetación.

Figura 70: Tintado de la pantalla con postprocesado bajo el agua.

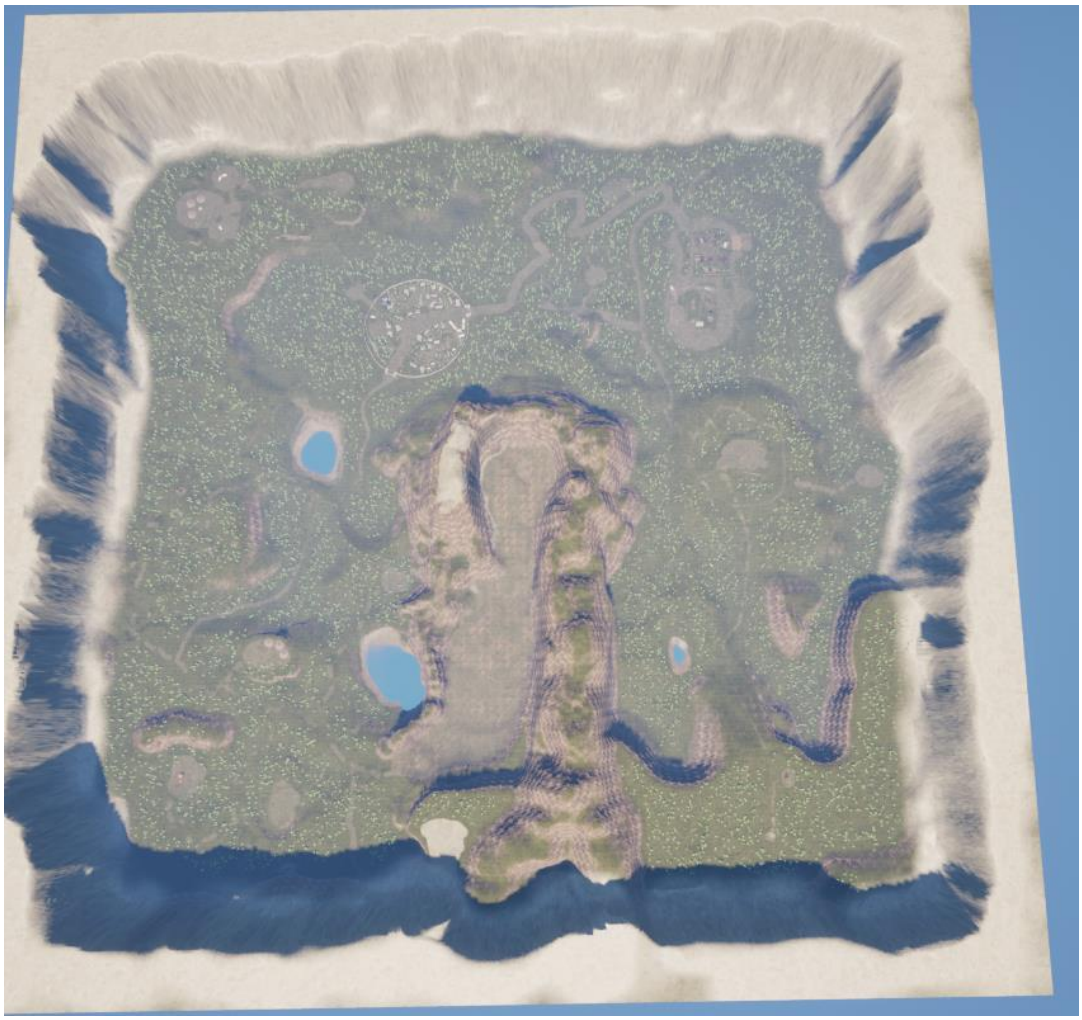


Figura 71**:** Mapa completo visto desde arriba.

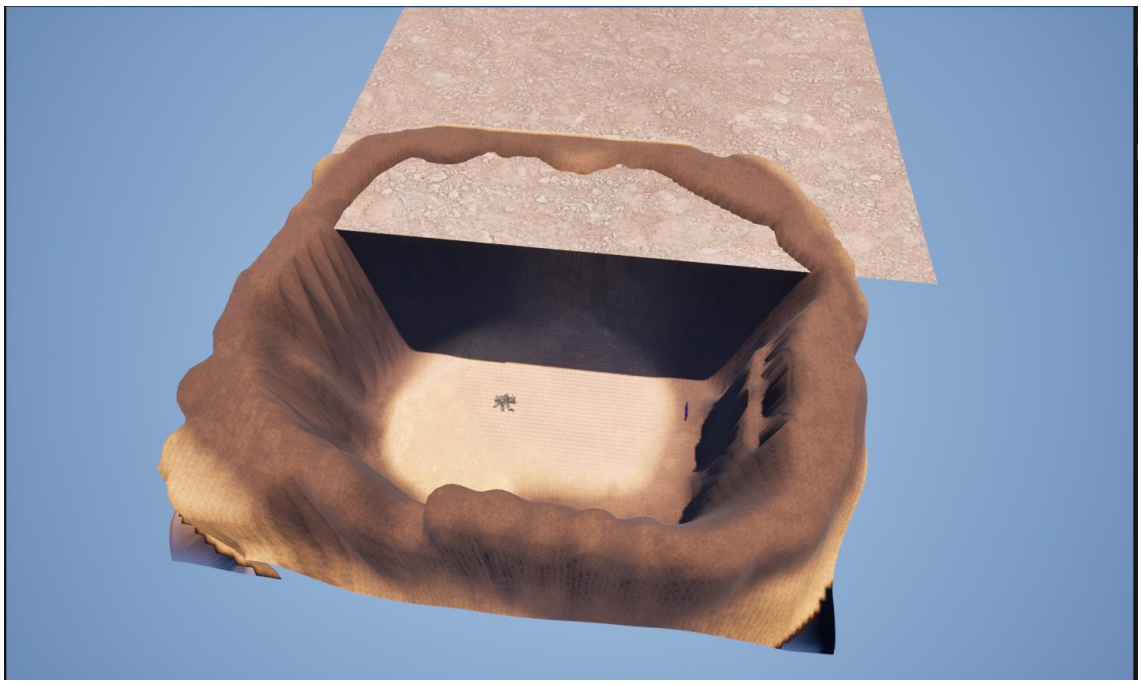


Figura 72**:** Arena del jefe vista desde arriba.

2.6.4. Música

La banda sonora del proyecto se estructura mediante un sistema de audio dinámico que adapta la música al contexto jugable en tiempo real.
El motor detecta variables como:

- Entrada en ciudades o asentamientos.
- Exploración de mundo abierto.
- Proximidad o activación del Boss Final.

Cada uno de estos estados activa pistas musicales específicas que refuerzan la tensión, la épica o la sensación de seguridad del entorno, favoreciendo la inmersión emocional del jugador y potenciando la narrativa del juego a través del sonido.

En cuanto a la producción musical, la mayor parte de la composición ha sido generada inicialmente mediante herramientas de inteligencia artificial, y posteriormente modificada o adaptada manualmente por el desarrollador utilizando *Audacity* para garantizar coherencia estética, ajustabilidad a los eventos dinámicos del juego y calidad sonora final.

Nombre	Autor	Uso	Enlace
AmbianceAndMusic.mp3	Suno AI y temas de naturaleza copyright free, modificada por mí.	Música principal del mundo.	https://drive.google.com/file/d/18I-hW_2opqLq9SMWuGv3FyON1ehybJ5G/view?usp=sharing
Boss-Close to the Edge.mp3	Suno AI, modificada por mí.	Música contra el jefe final.	https://drive.google.com/file/d/1HqNPA-tGbQjuSo_1zxx3zEsl3eJN05Eb/view?usp=sharing
Cave.mp3	julius_galla, modificada por mí.	Música ambiente de la arena del jefe final.	https://freesound.org/people/julius_galla/sounds/232685/
DeathMusic.wav	Suno AI, modificada por mí.	Música de Game Over.	https://drive.google.com/file/d/1L84uwbC90GqGGEh2JSvEUJyURMh9aPtu/view?usp=sharing
Eldhollow-Medieval Skies.mp3	Suno AI, modificada por mí.	Música del pueblo (Eldhollow).	https://drive.google.com/file/d/1QHEgQzqQ_QaLRKs7x4QODo4xu1eQ3DWf/view?usp=sharing
Whispers of the Abyss.WAV	Suno AI, modificada por mí.	Música del menú de inicio.	https://drive.google.com/file/d/1-UJnMNRoVjByLYFJvewmPTjbYmN1HrN-/view?usp=sharing
Dead Valley Echo.WAV	Suno AI, modificada por mí.	Música del valle final.	https://drive.google.com/file/d/1myQFHyfNCEuHLIW5uES7KnIh1ty88M2w/view?usp=sharing
Seraphen-Cobblestone Dreams.mp3	Suno AI, modificada por mí.	Música de la ciudad (Seraphen).	https://drive.google.com/file/d/1MWEa-

			BTTRk1cekaCf_xNrVAeVa5McjN4/view?usp=sharing

Tabla 3: Música Ambiente.

2.6.5. Efectos de sonido

Nombre	Uso	Procedencia
A_Buttonpress	Al presionar el botón de continuar (Resume) o de quitar el juego (Quit) en el menú de pausa.	Propio *asset* de Unreal Engine 5.
A_Hovered	Al colocar el ratón sobre el botón de continuar (Resume) o de quitar el juego (Quit) en el menú de pausa.	Propio *asset* de Unreal Engine 5.
AttackLizard	Cuando ataca el enemigo de infierno al jugador.	Hecho por mí en Audacity.
BiteWolf	Cuando ataca el enemigo de tierra al jugador.	https://pixabay.com/sound-effects/nature-barking-3-47779/ . Modificado.
BossInvulnerable	Cuando intentas dañar al boss en estado invulnerable.	https://pixabay.com/es/sound-effects/pel%c3%adculas-y-efectos-especiales-armor-impact-from-sword-393843/
button_press_799_WAV	Cuando el personaje recibe daño.	Paquete de Fast Travel System.
Chicken	Suena al acercarse a gallinas y pollitos.	https://pixabay.com/es/sound-effects/naturaleza-little-chicken-22922/ . Modificado.
Chicks	Suena al acercarse a gallinas y pollitos.	https://pixabay.com/es/sound-effects/naturaleza-chicken-soundscape-200111/ . Modificado.
DeathMegaSpike	Cuando muere el jefe final.	Hecho por mí en Audacity.
DeathScream	Cuando muere el jugador.	Hecho por mí en Audacity.
Decrease_health	Cuando bajan los corazones del jugador.	Modificado.
DrinkPotion	Cuando el jugador usa una poción.	https://pixabay.com/sound-effects/film-special-effects-magical-whoosh-148459/ . Modificado.
EndDubPotion	Cuando acaba la poción “Obsidian Phial”.	https://pixabay.com/sound-effects/film-special-effects-magic-spell-1-232441/
EndPotion	Cuando acaba el efecto de una poción.	https://pixabay.com/sound-effects/film-special-effects-magical-whoosh-148459/
enemy_death_lizard	Cuando muere el enemigo del infierno.	Hecho por mí en Audacity.
enemy_death_skeleton	Cuando muere el enemigo del cielo.	https://pixabay.com/sound-effects/film-special-effects-falling-bones-87568/
enemy_death_wolf	Cuando muere el enemigo de la tierra.	https://pixabay.com/sound-effects/nature-dog-whine-2-65720/ . Modificado.
EnemyHitSound_Lizard	Cuando el jugador golpea al enemigo del infierno.	Hecho por mí en Audacity.

EnemyHitSound_Skeleton	Cuando el jugador golpea al enemigo del cielo.	https://pixabay.com/sound-effects/film-special-effects-bone-snap-408148/
EnemyHitSound_Wolf	Cuando el jugador golpea al enemigo de la tierra.	https://www.youtube.com/watch?v=8ii2VRUWWzI
fire-sounds	Cuando el jugador está cerca de una hoguera encendida.	https://pixabay.com/sound-effects/film-special-effects-fire-sounds-405444/
FoundAnimal	Cuando el jugador registra un animal nuevo en el bestiario.	Hecho por mí en Audacity.
IncreaseHealth	Cuando el jugador se cura.	Modificado.
LockOffSound	Cuando el jugador desfija a un enemigo.	https://www.youtube.com/watch?v=E4LHX7R-eg8 . Modificado.
LockOnSound	Cuando el jugador fija a un enemigo.	https://www.youtube.com/watch?v=E4LHX7R-eg8
loseMoneySound	Cuando el jugador gasta dinero.	https://www.youtube.com/watch?v=4kVTqUxJYBA
match-lighting	Cuando el jugador descansa por primera vez en una hoguera y la enciende.	https://pixabay.com/sound-effects/film-special-effects-match-96840/
MegaSpikeAttack	Cuando el jefe final ataca.	Hecho por mí en Audacity.
MegaSpikeSummon	Cuando el jefe final hace aparecer súbditos.	Hecho por mí en Audacity.
PARTICLESSWORD	Cuando el jugador se acerca a la espada del tutorial.	Sonido aislado de Shiny del videojuego *Palworld*.
PatrolRandomRoar	Sonido aleatorio cuando un enemigo de infierno no ha detectado al jugador pero está cerca.	https://pixabay.com/sound-effects/film-special-effects-dragon-roar-364478/ . Modificado.
PICKUPITEM	Cuando el jugador recoge un objeto del suelo.	Modificado por mí.
PigGrunt	Suena al acercarse a cerdos.	https://pixabay.com/sound-effects/nature-pig-grunt-100272/
portal	Suena al acercarse al portal.	https://pixabay.com/sound-effects/film-special-effects-warp-portal-63742/ . Modificado.
QuestActivated	Suena al recibir una nueva misión.	Hecho por mí en Audacity.
QuestComplete	Suena al completar una misión.	Hecho por mí en Audacity.
rattling-bones-105394	Sonido aleatorio cuando un enemigo de cielo no ha detectado al jugador pero está cerca.	https://pixabay.com/sound-effects/film-special-effects-rattling-bones-105394/
RoarMegaSpike	Cuando inicia la batalla contra el jefe final y el jugador se acerca lo suficiente.	Hecho por mí en Audacity.

RoarMinion	Cuando el jefe final hace aparecer a sus súbditos y cuando atacan dichos súbditos.	Hecho por mí en Audacity.
Rooster	Suena al acercarse a gallinas y pollitos.	https://pixabay.com/sound-effects/nature-rooster-cry-chicken-rooster-305576/
SkeletonHit	Cuando ataca el enemigo de cielo al jugador.	https://pixabay.com/sound-effects/film-special-effects-bone-breaking-effect-393836/ . Modificado.
wolf-howl-359873	Sonido aleatorio cuando un enemigo de tierra no ha detectado al jugador pero está cerca.	https://pixabay.com/sound-effects/nature-wolf-howl-359873/

Tabla 4: Efectos de Sonido.

2.6.6. Objetos

El juego cuenta con una serie de objetos obtenibles, los cuales se encuentran recogidos en una tabla descriptiva que se presentará a continuación en la Tabla 5.

Nombre	**Categoría**	**Descripción**	**Adición a estadísticas**
Dunbrae Sword	Espadas	A sword forged by Forest Dwarves.	-
Elden Bark	Espadas	A blade of hardened age, split by time, yet bound by the eye of the forest.	-
MoonSword	Espadas	Forged in silver light. It is as sharp and cold as the air on a clear night.	-
Light Wraith	Espadas	A dense, jagged mass of steel, infused with a fractured, chilling light.	-
Rogue Crystal	Espadas	Shattered and reformed—a pulsing, volatile energy barely contained by the grip.	-
Abyzakar Justice	Espadas	A heavy, brutal judge's blade. The crimson rune within glows with an undeniable, scorching heat.	-
Ares Fury	Etiquetas	The power of the war god flows within.	+ 20 ataque
Aegis Protection	Etiquetas	The legendary shield guards your most vital aspect.	+ 1 corazón
Hermes Smoke	Etiquetas	Experience the fleeting swiftness of the winged messenger.	+200 velocidad
Marchosias Curse	Etiquetas	A swifter stride for a duller edge.	+10 ataque y +100 velocidad
Malphas Fortress	Etiquetas	Embrace fragile, piercing power.	-1 corazón y +5 ataque
Buer Oath	Etiquetas	Stubborn life at the cost of flight.	+1 corazón y -300 velocidad
Agate Drops	Pociones	Your strikes burn with borrowed wrath.	-
Jade Brew	Pociones	Life seeps back in, warm and quiet.	-
Lapis Elixir	Pociones	A blessing that hardens the soul’s vessel.	-
Amethyst Brew	Pociones	Gravity loosens its grip for a heartbeat.	-

Sunstone Tincture	Pociones	Bottled sunlight that urges your blood to race.	-
Obsidian Phial	Pociones	A void-black promise that bites back.	-
Daisy	Plantas	A common bloom, but its delicate petals hold a subtle, earthy promise.	-
Lupine	Plantas	Towers above the grass—a spire of color hinting at focused, bottled energy.	-
Brown Mushroom	Plantas	Found in shadowed corners, rumored to bring a foundational strength to the prepared.	-
Red Mushroom	Plantas	A vibrant, alarming cap. Use with care, as its strange power is highly concentrated.	-
Dragon Skull	Plantas	A reminder of a powerful creature.	-
Meat and Hide	Plantas	A deer's meat and hide. Liked by carnivores.	-

Tabla 5: Todos los objetos obtenibles en el juego.

2.6.6.1. Sprites

Dunbrae Sword

Figura 73: Sprite. Dunbrae Sword.

Elden Bark

Figura 74: Sprite. Elden Bark.

MoonSword

Figura 75: Sprite. MoonSword.

Light Wraith

Figura 76: Sprite. Light Wraith.

Rogue Crystal

Figura 77: Sprite. Rogue Crystal.

Abyzakar Justice

Figura 78: Sprite. Abyzakar Justice.

Ares Fury

Figura 79: Sprite. Aegis Protection.

Aegis Protection

Figura 80: Sprite. Aegis Protection.

Hermes Smoke

Figura 81: Sprite. Hermes Smoke.

Marchosias Curse

Figura 82: Sprite. Marchosias Curse.

Malphas Fortress

Figura 83: Sprite. Malphas Fortress.

Buer Oath

Figura 84: Sprite. Buer Oath.

Agate Drops

Figura 85: Sprite. Agate Drops.

Jade Brew

Figura 86: Sprite. Jade Brew.

Lapis Elixir

Figura 87: Sprite. Lapis Elixir.

Amethyst Brew

Figura 88: Sprite. Amethyst Brew.

Sunstone Tincture

Figura 89: Sprite. Sunstone Tincture.

Obsidian Phial

Figura 90: Sprite. Obsidian Phial.

Daisy

Figura 91: Sprite. Daisy.

Lupine

Figura 92: Sprite. Lupine.

Brown Mushroom

Figura 93: Sprite. Brown Mushroom.

Red Mushroom

Figura 94: Sprite. Red Mushroom.

Dragon Skull

Figura 95: Sprite. Dragon Skull.

Meat and Hide

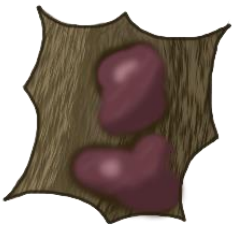


Figura 96: Sprite. Meat and Hide.

3

Análisis, diseño y solución

3.1. Primer concepto

El concepto inicial de *Order Frame* se define como un videojuego de acción en tercera persona centrado en la dualidad moral y en el equilibrio entre tres planos de existencia: Cielo, Tierra e Infierno. La propuesta original presenta al protagonista como una única entidad anímica que adopta tres identidades diferenciadas según el plano en el que se manifiesta: Caelvorn en el Cielo, Elior (o Elandor) en la Tierra y Azrakar en el Infierno, asumiendo en todos los casos el rol sistémico de la denominada "Llama del Juicio".

3.1.1. Estructura narrativa y mecánicas originales

El diseño inicial planteaba una estructura de mundo segmentada en capas interconectadas bajo una configuración conceptual de "marco" (*frame*). El punto de partida narrativo contemplaba diversas posibilidades en relación con la salida del protagonista del plano celestial. Entre ellas se incluían: la ejecución de un mandato divino orientado al exterminio demoníaco, el destierro tras desafiar órdenes superiores (como la negativa a diezmar una aldea) o la comisión de un "acto imperdonable" derivado de la búsqueda de conocimiento prohibido.
En el ámbito de la jugabilidad, se propusieron mecánicas destinadas a reforzar el peso sistémico de las decisiones morales:

- **Mecánica de Hesitación:** Durante la fase tutorial, el sistema monitorizaba la respuesta del jugador ante órdenes de ejecución. La duda o el cuestionamiento del mandato activaban señales visuales en el HUD, acompañadas de un castigo físico al personaje materializado en forma de una "quemadura" de energía divina.

- **Sistema de Alineamiento Oculto:** Se diseñó un medidor no visible para el usuario que clasificaba la progresión moral en tres estados (Neutral, Corrupto y Alineado) en función de diálogos seleccionados y resultados de misiones.

- **Combate Híbrido:** Se planteó un sistema de combate basado en el uso de grandes espadas (*claymores*) imbuidas en poder sagrado o energía infernal, complementado con habilidades de movilidad avanzada como el doble salto de tipo *flutter*.

3.1.2. Elementos modificados o descartados

Debido a limitaciones técnicas, temporales y a decisiones estratégicas de ingeniería orientadas a garantizar la viabilidad del proyecto, diversos elementos del concepto original no fueron implementados en su forma inicial o fueron objeto de simplificación.

- **Alcance del mundo:** La propuesta original contemplaba un mundo completamente abierto que incluía zonas plenamente transitables del Cielo y del Infierno. Este planteamiento fue sustituido por un entorno de grandes dimensiones centrado en una *vertical slice* de alta fidelidad, con el objetivo de asegurar la estabilidad del software. El primer jefe concebido en la planificación inicial fue mantenido, no obstante, como jefe final de la demo jugable.

- **Compatibilidad de periféricos:** Se realizaron intentos de implementación de control mediante mando de Xbox. Sin embargo, se priorizó finalmente el esquema de teclado y ratón para garantizar la robustez y fiabilidad del sistema de entrada.

- **Sistema de adopción y vivienda:** El concepto original incluía la posibilidad de adquirir una vivienda en el plano de la Tierra, junto con un sistema extensivo de adopción compuesto por siete menores específicos (Lina, Nia, Lumi, Faye, Ori, Milo y Noa), cada uno con trasfondo narrativo y misiones de lealtad asociadas que otorgaban recompensas únicas. Este sistema fue descartado por su elevada complejidad de implementación, aunque su trasfondo narrativo permanece documentado para futuras ampliaciones.

- **Bestiario y mascotas:** Se proyectó inicialmente un sistema de domesticación compuesto por ocho mascotas (principalmente lobos y felinos) con habilidades de combate y funciones de apoyo en el cuidado de los menores adoptados (entre ellos Bran, Ash, Whisper, Seraphine o Shade). Este sistema fue reducido, manteniéndose únicamente dos lobos: Skoll y Fenra. Asimismo, el bestiario contemplaba en su concepción inicial una sección dedicada a flora y plantas, finalmente descartada.

- **Gestión de equipo:** Se planteó que las maldiciones aplicadas al equipamiento fueran de carácter aleatorio y que su eliminación dependiera de un NPC especialista mediante pago. Del mismo modo, cada espada estaría vinculada a una alineación específica, infligiendo daño adicional según la tipología del enemigo. Ambas mecánicas fueron simplificadas en el prototipo final.

- **Sistema de logros:** Se diseñó un sistema de logros orientado al completismo, entre los que se incluían:
    - *Full House* (adoptar a todos los niños).
    - *Gotta Pet 'em All* (obtener todas las mascotas).
    - *Is This It?* (derrotar a Dios).
    - *What I Was Meant to Do* (derrotar al Rey de los Demonios).
    - *No Sympathy* (no salvar a ningún niño ni mascota).
    - *Know Thy Kingdom* (completar el bestiario).
    - *One Less* (finalizar el tutorial en su versión original con combate).
    - *Beyond the Map* (acceder a una ciudad oculta alineándose con la Tierra).
    - *Roots at Last* (construir la vivienda propia).

- **Finales predefinidos:** La finalización del juego bajo determinadas condiciones otorgaba títulos permanentes utilizables en multijugador. Entre ellos se encontraban:
    - *The Blind Flame* (alineamiento absoluto con el Cielo y obediencia total, exterminación de todos).
    - *Of the Quiet Dawn* (alineamiento con la Tierra con familia y hogar).
    - *The Bound Blade* (obediencia celestial sin exterminio total).
    - *The Dull Flame* (rechazo a servir al Rey Demonio).
    - *The God Slayer* (alianza infernal y derrota de la deidad).

- **Modo multijugador:** Se contempló la implementación de un modo multijugador independiente de la narrativa principal, centrado en combates, con un enfoque comparable a propuestas como *Kid Icarus: Uprising [28]*. Este sistema no fue desarrollado en la versión final del prototipo.

3.2. Captura de requisitos

3.2.1. Requisitos funcionales

Los requisitos funcionales de Order Frame describen las características técnicas y los comportamientos esenciales implementados en el motor Unreal Engine 5.6 [16] mediante Blueprints [8].

1. Lógica del Personaje y Movimiento
    - **RF1:** El sistema debe permitir el control de movimiento avanzado del jugador, incluyendo carrera, doble salto (tipo *flutter*) y la capacidad de rodar para esquivar ataques.
    - **RF2:** El software debe gestionar dinámicamente el estado de salud y la muerte del personaje, activando una pantalla de fin de partida tras la pérdida total de vida.
    - **RF3:** La aplicación debe aplicar efectos de estado al jugador, como aumento de estadísticas por alineamiento o pociones.

2. Sistemas de Inventario y Combate

- **RF4:** La aplicación proporcionará un inventario organizado por pestañas (armas, magia, diario, misiones y bestiario) donde el usuario podrá gestionar sus recursos y ver su estado actual.
- **RF5:** El sistema debe permitir el equipamiento simultáneo de hasta dos armas de tipo espada y un conjunto limitado de tres bendiciones y tres maldiciones.
- **RF6:** El usuario debe poder interactuar con un sistema de creación para crear pociones combinando ingredientes recolectados en el mundo.
- **RF7:** El usuario podrá utilizar pociones desde su inventario.
- **RF8:** El sistema de combate debe permitir la ejecución de ataques, así como el uso de un sistema de fijación de objetivos (*lock-on*).

3. Inteligencia Artificial y Enemigos
    - **RF9:** La aplicación debe procesar el comportamiento de los enemigos mediante lógica de patrullaje autónomo por puntos de navegación y detección por proximidad.
    - **RF10:** El sistema debe gestionar la aparición y el *respawn* automático de tres tipos distintos de enemigos y objetos recolectables en el mapa.
    - **RF11:** El encuentro con el jefe final (*Gluttony*) debe ser procesado en tres fases de combate diferenciadas, variando sus patrones de ataque y habilidades según la fase activa.

4. Progresión y Mundo de Juego
    - **RF12:** La aplicación debe registrar y actualizar el sistema de alineación dinámica (Cielo, Tierra o Infierno) basándose en las decisiones y acciones del jugador.
    - **RF13:** El sistema debe modificar las estadísticas del personaje (corazones de vida, daño y velocidad) de forma permanente según la rama de alineamiento alcanzada.
    - **RF14:** El usuario debe poder domesticar animales, integrándolos en un "Journal" para que influyan en la jugabilidad y el combate.
    - **RF15:** El sistema debe permitir la interacción de diálogo con NPCs mediante un *blueprint* genérico (*Interface [11]*) que facilite la asignación de misiones y la toma de decisiones.

5. Salida de Datos y Persistencia
    - **RF16:** La aplicación debe generar una representación visual del progreso mediante un mapa con revelación automática de zonas exploradas e iconos de interés.

- **RF17:** El usuario debe poder guardar y cargar su progreso (posición, inventario, misiones y estadísticas) mediante un sistema de persistencia basado en *Save Game Blueprints [15].*
- **RF18:** El software debe gestionar el comercio permitiendo al usuario comprar y vender objetos en tres tiendas distribuidas por el mundo.

3.2.2. Requisitos no funcionales

Los requisitos no funcionales de **Order Frame** definen los atributos de calidad, las restricciones técnicas y los estándares de rendimiento que el software debe cumplir para garantizar una experiencia de usuario óptima y una arquitectura robusta.

1. Rendimiento y Eficiencia
    - **RNF1:** El sistema debe emplear técnicas de carga bajo demanda de recursos para optimizar el uso de memoria y garantizar la fluidez en un entorno de grandes dimensiones.
    - **RNF2:** Se deben simplificar los sistemas de físicas en aquellos elementos que no requieran alta precisión para mantener un rendimiento estable en tiempo real.
    - **RNF3:** El videojuego debe aprovechar las capacidades de Unreal Engine 5.6 [16] para ofrecer gráficos de alta fidelidad y una iluminación avanzada, manteniendo la estabilidad del motor.

2. Usabilidad y Diseño de Interfaz
    - **RNF4:** El sistema de control debe priorizar un esquema de entrada mediante teclado y ratón, asegurando una respuesta técnica robusta y probada frente a otras alternativas de periféricos.
    - Las interfaces gráficas (HUD y menús) deben actualizar la información en tiempo real mediante el uso de *bindings [10]* que vinculen directamente las variables del jugador con los elementos visuales.
    - **RNF6:** El software debe proporcionar retroalimentación visual clara ante eventos críticos, como el daño recibido o la muerte del personaje, para facilitar la comprensión del estado del juego.

3. Arquitectura y Mantenibilidad
    - **RNF7:** El desarrollo debe ser íntegramente modular, utilizando el sistema de *Blueprints [8]* de forma jerárquica (clases padre e hijo) para facilitar la reutilización de código y la escalabilidad.
    - **RNF8:** La comunicación entre diferentes módulos (personaje, enemigos, inventario) debe realizarse mediante *Interfaces [11]* y *Event Dispatchers* [10] para evitar dependencias rígidas y asegurar un sistema mantenible.

- **RNF9:** El proyecto debe estructurarse de manera que permita la depuración visual y el ajuste rápido de variables paramétricas, facilitando el balance de juego durante las fases de prueba.

4. Fiabilidad y Portabilidad
    - **RNF10:** El software debe garantizar la integridad de los datos de la partida mediante un sistema de persistencia fiable basado en Save Game Blueprints [15].
    - **RNF11:** La aplicación debe ser compatible y estar optimizada específicamente para la plataforma de ordenador personal (PC).
    - **RNF12:** El desarrollo debe estar respaldado por un sistema de control de versiones (*GitHub [35]*) para prevenir la pérdida de datos ante posibles corrupciones de archivos y asegurar la trazabilidad del código.

5. Estándares de Diseño y Desarrollo
    - **RNF13:** El diseño debe basarse en una *"vertical slice"* de alta fidelidad, asegurando que todos los sistemas centrales funcionen perfectamente en una zona controlada antes de cualquier expansión.
    - **RNF14:** Los comportamientos de la IA deben ser predecibles y estar vinculados a mecánicas lógicas (detección por distancia o colisión) para evitar comportamientos aleatorios que frustren al jugador.

3.3. Casos de uso

A continuación, se describen los escenarios de interacción que definen el comportamiento funcional de Order Frame. En la Tabla 8 se detalla el caso de uso generalizado de Interacción Moral y Alineamiento Dinámico, el cual representa el núcleo de la propuesta original y la lógica de decisión del jugador. En dicho cuadro se desglosan elementos críticos como la descripción, los actores involucrados (Jugador y Sistemas de Unreal), las precondiciones, postcondiciones y los flujos de eventos.
Posteriormente, en las Tablas 6, 7, 9, 10, 11, 12, 13, 14, 15 y 16, se describen los casos de uso específicos que cubren las áreas de combate, gestión de inventario, inteligencia artificial de enemigos, adiciones al bestiario y persistencia de datos entre otros.

Nombre	**Ejecutar Movimiento y Habilidades Físicas**
Descripción	El jugador desplaza al personaje por el entorno utilizando habilidades de movilidad avanzada.
Actor Principal	Jugador.
Precondiciones	El personaje debe estar en estado activo (no muerto).
Postcondiciones	La ubicación o el estado físico del personaje se actualizan en el mundo.
Flujo Principal	1. El jugador introduce comandos de dirección mediante el teclado (WASD).

	2. El sistema interpreta la entrada y desplaza al personaje en el entorno. 3. El sistema ejecuta la animación correspondiente de caminar o correr. 4. El jugador activa una habilidad de movilidad avanzada, como el doble salto (*flutter*) o la acción de rodar. 5. El sistema actualiza la posición y el estado físico del personaje en el mundo.
Flujo Alternativo	4.a. Si el jugador recibe un ataque durante la acción de rodar. 4.a.1. El sistema evalúa los *frames* de invulnerabilidad. 4.a.2. Si el ataque ocurre dentro de dichos *frames*, el daño es evitado.

Tabla 6: Sistema de Movimiento y Control.

Nombre	**Realizar Ataque Cuerpo a Cuerpo**
Descripción	El jugador utiliza sus espadas para infligir daño a los enemigos mediante ataques.
Actor Principal	Jugador.
Precondiciones	Tener al menos una espada equipada en el inventario en el slot de equipamiento.
Postcondiciones	Reducción de la salud del enemigo o aplicación de efectos de estado.
Flujo Principal	1. El jugador activa el sistema de fijación de objetivo (*lock-on*). 2. El jugador presiona la tecla de equipamiento de arma. 3. El sistema ejecuta la animación de equipamiento. 4. El jugador presiona la tecla de ataque correspondiente. 5. El sistema ejecuta la animación de ataque. 6. El sistema calcula el daño en función del arma equipada y modificadores activos. 7. El sistema aplica el daño y posibles efectos de estado al enemigo.
Flujo Alternativo	2.a. Si el jugador tiene dos espadas equipadas. 2.b. El jugador puede cambiar el arma en mitad del combate sin entrar al inventario.

Tabla 7: Combate y Gestión de Armas.

Nombre	**Actualizar Alineación Dinámica**
Descripción	El sistema modifica las estadísticas y la percepción del mundo del jugador según sus decisiones morales.
Actor Principal	Sistema / Jugador.
Precondiciones	El jugador debe realizar una acción con carga moral (matar, perdonar, hablar).
Postcondiciones	Modificación de corazones de vida, daño o velocidad del personaje.
Flujo Principal	1. El jugador realiza una acción con carga moral (matar, perdonar o hablar). 2. El sistema identifica el tipo de acción realizada.

	3. El sistema asigna puntos invisibles al alineamiento correspondiente (Cielo, Tierra o Infierno). 4. El sistema comprueba si se ha alcanzado un umbral de alineamiento. 5. Al alcanzar el valor establecido, el sistema desbloquea el alineamiento en la interfaz. 6. El jugador puede elegir la alineación que desee. 7. Se modifican las estadísticas del personaje según el alineamiento obtenido.
Flujo Alternativo	-

Tabla 8: Sistema de Alineamiento Moral.

Nombre	**Gestionar Equipamiento y Objetos**
Descripción	El jugador organiza sus armas, bendiciones y maldiciones mediante una interfaz de pestañas.
Actor Principal	Jugador.
Precondiciones	Debe estar dentro del inventario para poder verlos.
Postcondiciones	Los cambios en el equipo de espadas se reflejan en el HUD.
Flujo Principal	1. El jugador abre el inventario desde el menú correspondiente. 2. El sistema muestra la interfaz dividida en pestañas (armas, magia, journal, misiones, bestiario). 3. El jugador navega por las pestañas disponibles. 4. El jugador equipa hasta dos armas y tres bendiciones o maldiciones. 5. El sistema actualiza las armas en el HUD.
Flujo Alternativo	4.a. Si el jugador desequipa el arma o las bendiciones o maldiciones con click derecho. 4.a.1. El arma o bendición o maldición correspondientes dejan de estar equipados y pasan al inventario.

Tabla 9: Gestión de Inventario y Equipamiento.

Nombre	**Crear Pociones de Magia**
Descripción	El jugador combina ingredientes recolectados para generar consumibles con efectos especiales.
Actor Principal	Jugador.
Precondiciones	Tener los ingredientes necesarios recogidos del mundo y tener desbloqueado el menú de creación de pociones.
Postcondiciones	Una nueva poción se añade al inventario y los ingredientes utilizados son consumidos.
Flujo Principal	1. El jugador accede a la pestaña de magia del inventario. 2. Selecciona los ingredientes disponibles (plantas). 3. El sistema valida la combinación seleccionada. 4. El sistema crea la poción correspondiente. 5. La poción se añade al inventario del jugador.
Flujo Alternativo	3.a. Si el tipo del ingredientes es inválido. 3.a.1. El sistema no permite ponerlos en los slots de creación.

Tabla 10: Creación y Magia.

Nombre	**Interactuar con NPC y Misiones**
Descripción	El jugador entabla diálogos con personajes no jugables para avanzar en la historia o recibir recompensas.
Actor Principal	Jugador.
Precondiciones	El NPC debe estar dentro del rango de interacción.
Postcondiciones	Actualización del panel de misiones o adición de objetos o dinero.
Flujo Principal	1. El jugador se aproxima a un NPC dentro del rango de interacción. 2. El jugador ejecuta la acción “Hablar”. 3. El sistema despliega el *Widget* de diálogo. 4. El jugador lee el texto y recibe una misión. 5. El sistema actualiza el estado de misiones.
Flujo Alternativo	4.a. El jugador lee el texto y elige entre dos opciones dadas. 4.a.1. Dependiendo de la opción elegida el personaje gana una alineación específica.

Tabla 11: Interacción Narrativa (NPCs).

Nombre	**Registrar Animal**
Descripción	El jugador interactúa con la fauna del mundo para completar el registro del bestiario.
Actor Principal	Jugador.
Precondiciones	Encontrar un animal en el mundo y acercarse.
Postcondiciones	El animal aparece como "registrado" en el bestiario.
Flujo Principal	1. El jugador localiza un animal específico en el mundo. 2. El jugador se acerca al animal para registrarlo. 3. El sistema desbloquea la entrada correspondiente en el bestiario. 4. El animal queda registrado.
Flujo Alternativo	3.a. El animal ya estaba registrado. 3.a.1 No aparece dos veces en el bestiario.

Tabla 12: Bestiario.

Nombre	**Comprar/Vender en Tienda**
Descripción	Intercambio de objetos por una de las monedas del juego en localizaciones específicas.
Actor Principal	Jugador.
Precondiciones	Acceder a una de las tres tiendas disponibles.
Postcondiciones	Actualización del inventario y del saldo disponible del tipo de moneda correspondiente.
Flujo Principal	1. El jugador interactúa con el actor de la tienda. 2. El sistema muestra el *Widget* de comercio. 3. El jugador selecciona objetos para comprar o vender. 4. El sistema valida la transacción. 5. Se actualizan el inventario y el saldo.

Flujo Alternativo	4.a. Si el jugador no tiene suficiente saldo o espacio. 4.a.1. El sistema cancela la transacción y no ocurre nada.

Tabla 13: Guardado de Progreso.

Nombre	**Guardar Partida**
Descripción	Persistencia de todos los datos del jugador y del estado del mundo.
Actor Principal	Jugador.
Precondiciones	Interactuar con una hoguera.
Postcondiciones	Se genera o actualiza un archivo a través de Save Game Blueprints [15].
Flujo Principal	1. El jugador se acerca e interactúa con una hoguera. 2. Selecciona la opción para descansar en la hoguera. 3. El sistema registra la información mediante *Save Game Blueprints [15]*.
Flujo Alternativo	3.a. El sistema ejecuta un guardado automático al salir del juego.

Tabla 14: Comercio y Economía.

Nombre	**Consultar Mapa Dinámico**
Descripción	Visualización del progreso de exploración y puntos de interés.
Actor Principal	Jugador.
Precondiciones	Ninguna (accesible desde el HUD).
Postcondiciones	El jugador identifica su ubicación y localización previamente visitadas.
Flujo Principal	1. El jugador activa la vista de mapa. 2. El sistema muestra las zonas exploradas y la localización del personaje.
Flujo Alternativo	2.a. Si el jugador entra en una nueva zona. 2.a.1. El sistema registra dicha localización en el mapa.

Tabla 15: Exploración y Mapa.

Nombre	**Combatir contra Boss (Gluttony)**
Descripción	Batalla final dividida en tres etapas de dificultad creciente.
Actor Principal	Jugador.
Precondiciones	Haber avanzado hasta el final de la demo/misión principal.
Postcondiciones	Victoria (final de la demo) o derrota (pantalla de fin de partida).
Flujo Principal	1. Se inicia la Fase 1 con patrones básicos de ataque. 2. Al reducir la vida del jefe, aparece una nueva barra de vida y el sistema activa la Fase 2 mediante *Event Dispatchers [10]*. 3. Al reducir la vida del jefe, aparece una nueva barra de vida y se inicia la Fase 3 con patrones avanzados de huida e invocación de súbditos. 4. El combate finaliza con victoria o derrota.
Flujo Alternativo	4.a. Si el jugador es derrotado. 4.a.1. El sistema reinicia el combate y devuelve al jugador al último punto de guardado.

Tabla 16: Enfrentamiento con Jefe Final.

La Figura 97 muestra el diagrama de casos de uso del sistema, en el que se representa la interacción entre el usuario y las distintas funcionalidades disponibles. Este diagrama permite identificar de forma clara las acciones que el usuario puede realizar, así como su relación con los diferentes componentes del sistema, facilitando la comprensión general del comportamiento y alcance de la aplicación. Fue realizado con Visual Paradigm [34].

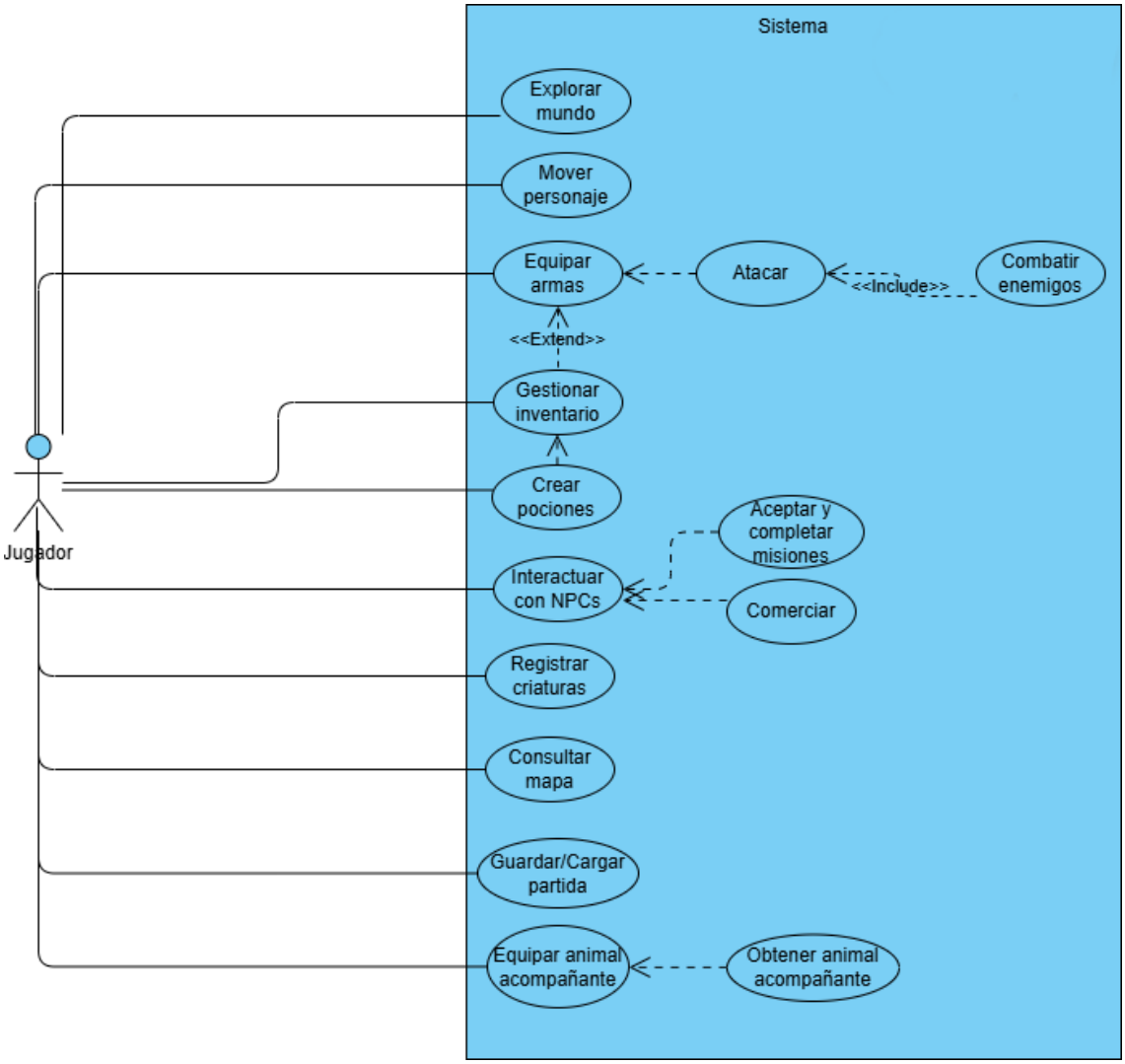


Figura 97: Diagrama Casos de Uso.

3.4. Diagrama de Secuencia

La *Figura 98: Diagrama de Secuencia.* presenta un diagrama de secuencia que ilustra el proceso de asignación de puntos de alineamiento dentro del juego. Este diagrama refleja de manera detallada la interacción entre el jugador y los distintos sistemas implicados en la modificación de la moralidad del personaje, permitiendo comprender cómo las acciones del usuario afectan de forma directa al sistema de alineamiento dinámico implementado.

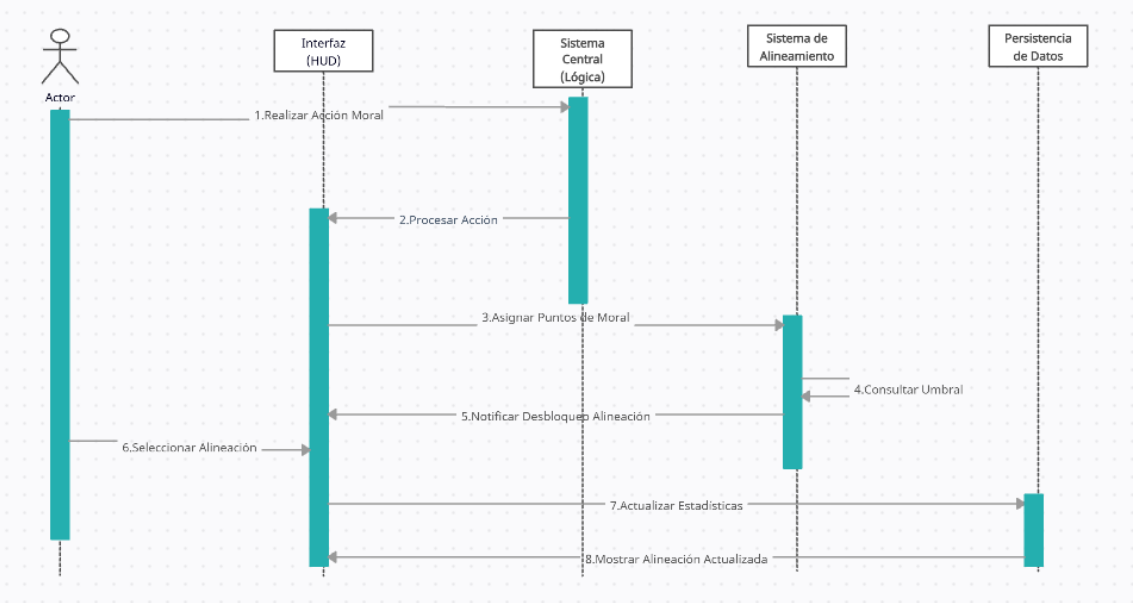


Figura 98: Diagrama de Secuencia.

3.5. Interfaz de usuario

Se debe señalar que la mayor parte de los bocetos de interfaz fueron elaborados durante las fases iniciales del desarrollo del videojuego, por lo que no todos los elementos representados en los diseños conceptuales se corresponden exactamente con la implementación final dentro del prototipo. No obstante, dichos bocetos resultan fundamentales para comprender la planificación estructural de la experiencia de usuario y la organización funcional de los sistemas interactivos.

3.5.1. Boceto de interfaz del menú principal

Se realizaron diversos diseños preliminares del menú principal con el objetivo de establecer la jerarquía visual y la accesibilidad de las opciones básicas del sistema. Entre los elementos dibujados se incluyen:

- Pantalla de **Start Menu** o inicio.

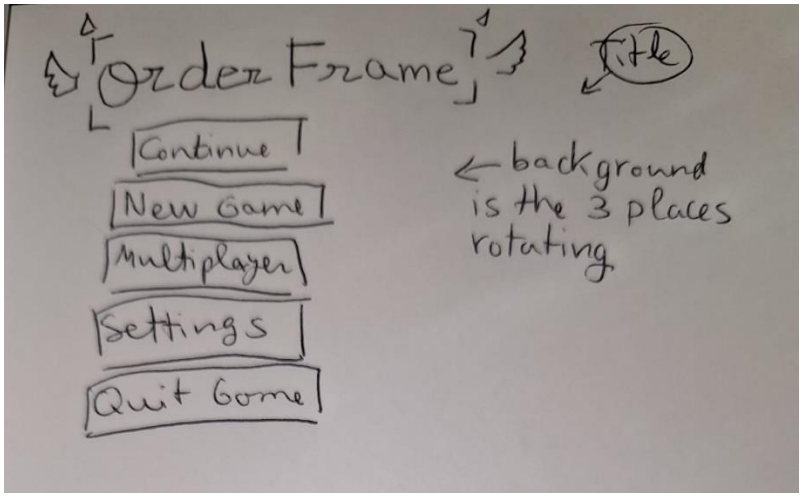


Figura 99: Primer concepto del menú de inicio.

- Interfaz asociada al **botón de pausa (Escape)**.

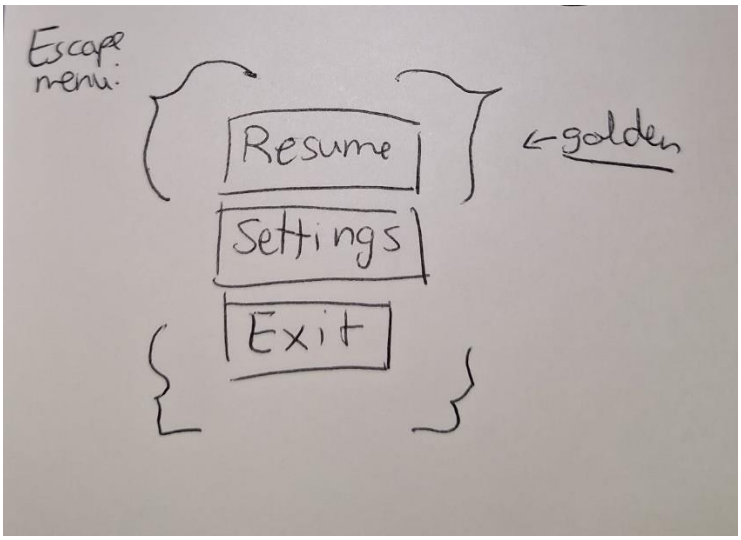


Figura 100: Primer concepto del menú de escape.

- Interfaz de exploración estándar (HUD general).

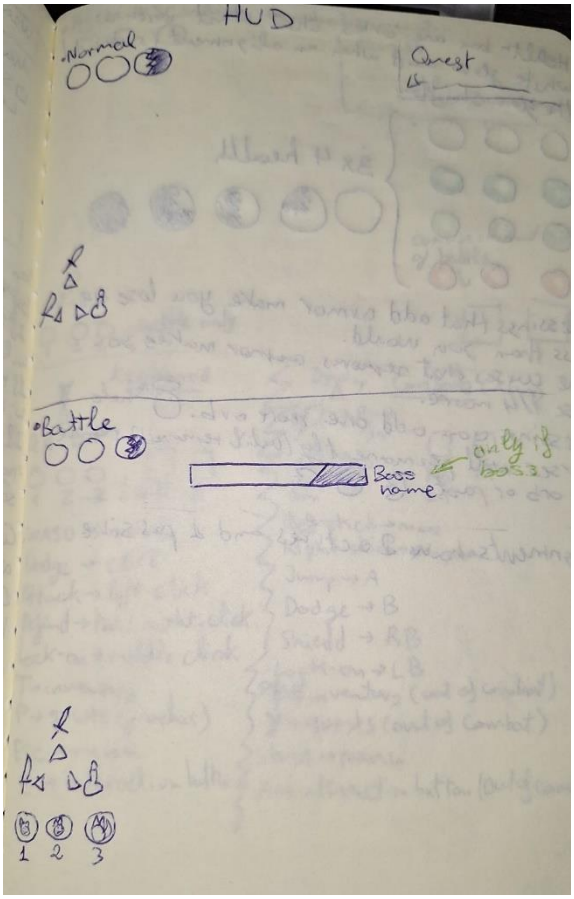


Figura 101: Primer concepto del HUD.

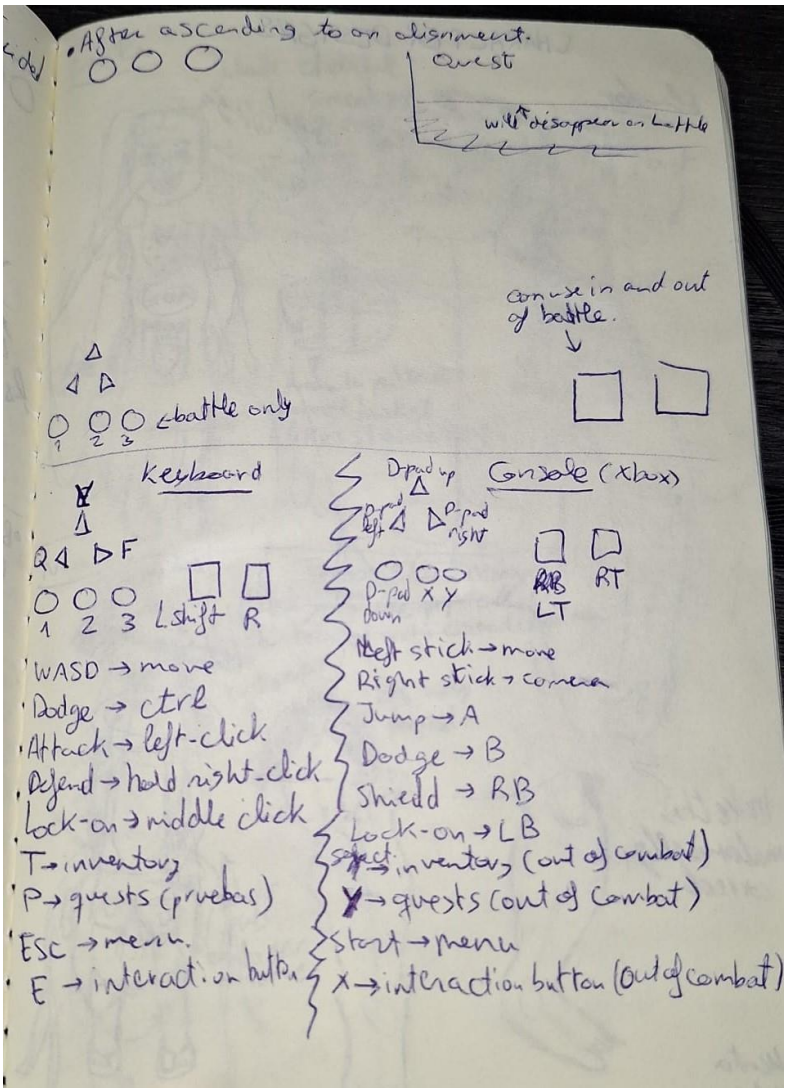


Figura 102: Segundo concepto del HUD y de las teclas.

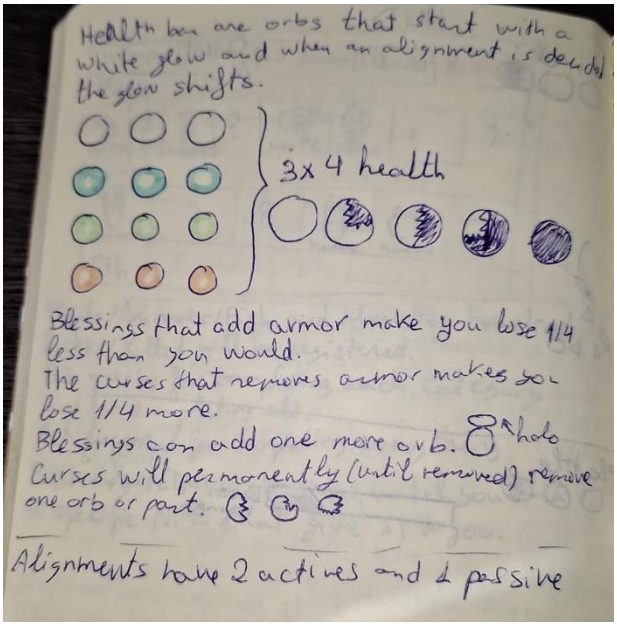


Figura 103: Concepto de la vida y alineamientos.

3.5.2. Boceto de interfaz del inventario

Se desarrolló un conjunto completo de ilustraciones correspondientes al sistema de inventario. En dichos diseños se representaban las distintas pestañas del inventario en detalle.

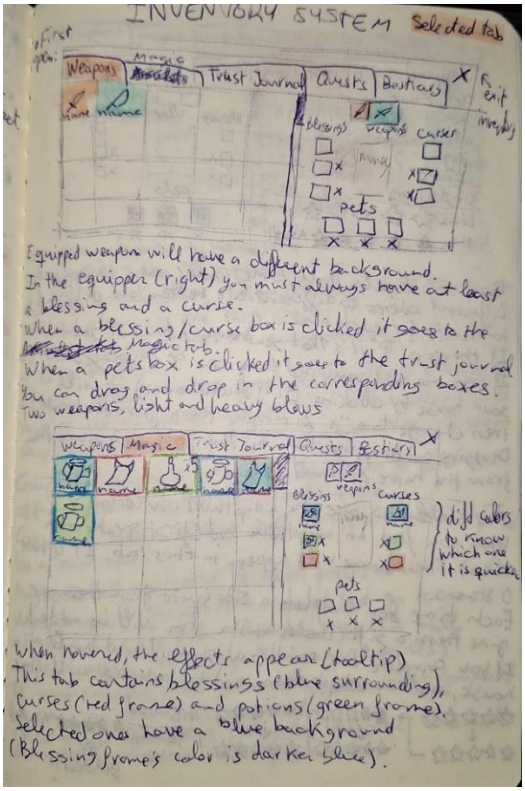


Figura 104: Concepto de las pestañas de armas y magia en el inventario.

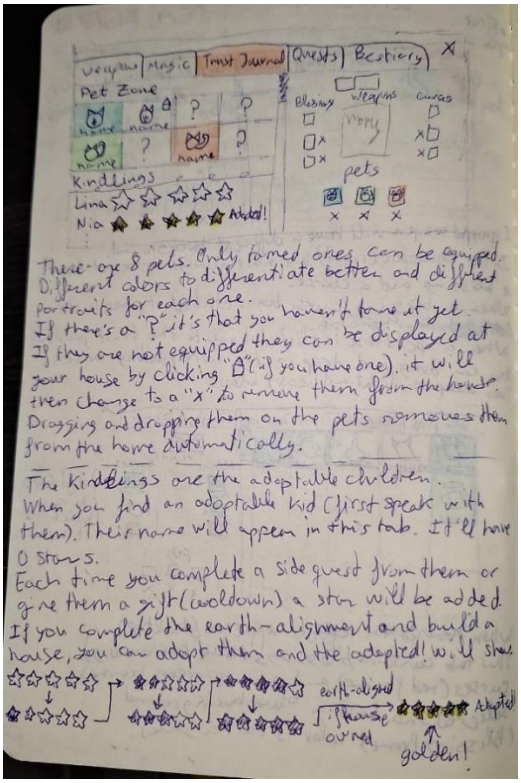


Figura 105: Concepto de la pestaña de acompañantes y adoptables en el inventario.

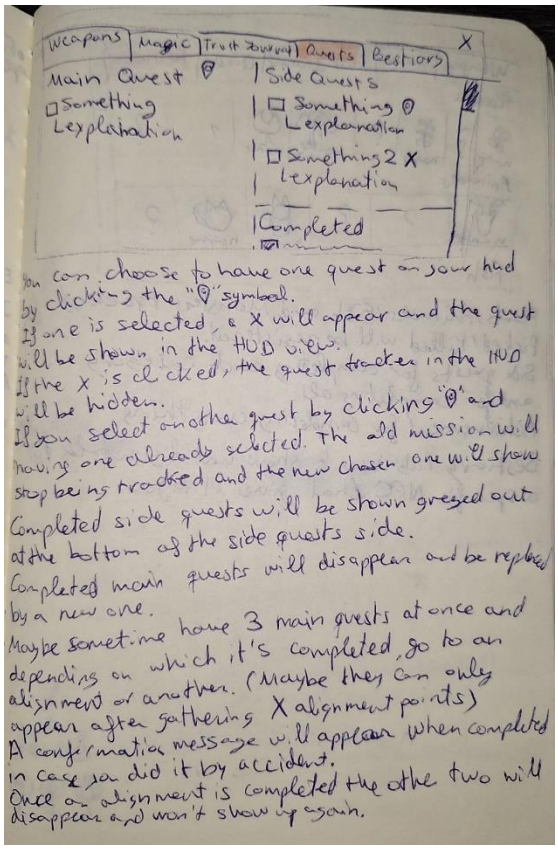


Figura 106: Concepto de la pestaña de misiones en el inventario.

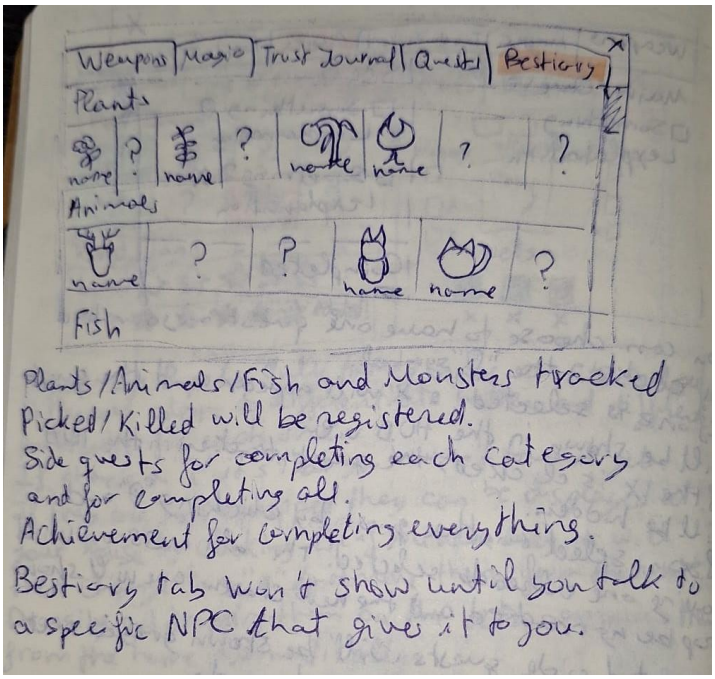


Figura 107: Concepto de la pestaña de bestiario en el inventario.

3.5.3. Boceto de interfaz de la tienda

Asimismo, se elaboró un boceto específico para la interfaz de comercio. Su diseño buscaba facilitar la toma de decisiones del jugador dentro del sistema económico del juego.

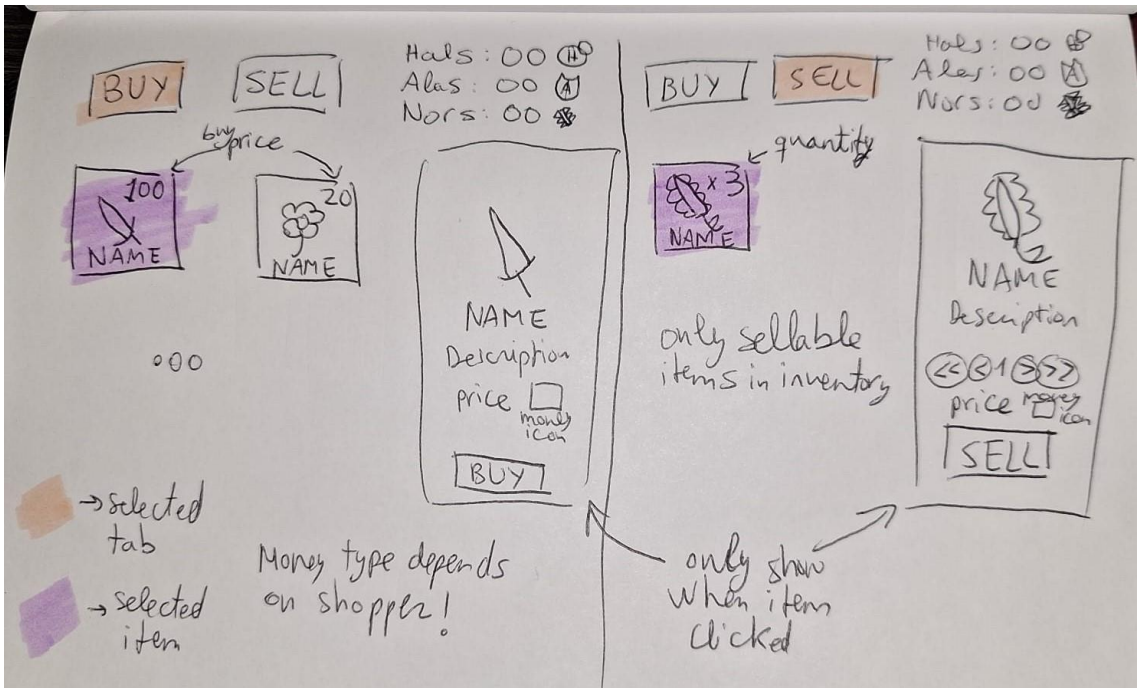


Figura 108: Concepto de la interfaz de comercio.

3.6. Estadísticas y sistema de lucha de los personajes

El sistema de estadísticas fue concebido como una estructura integral destinada a regular el rendimiento del personaje en combate y exploración. Cada estadística incide directamente en variables jugables específicas, afectando tanto a la ofensiva como a la supervivencia y movilidad.

Sin entrar en la implementación técnica o de código, las estadísticas determinan aspectos como:

- Daño infligido.
- Cantidad de vida máxima.
- Velocidad de movimiento.

Este sistema actúa como base matemática sobre la que se articulan las progresiones del personaje.

3.6.1. Mejora de estadísticas mediante pociones

Se diseñó un sistema de mejora basado en consumibles. Las pociones permiten incrementar atributos de manera permanente o temporal, incentivando la exploración y la recolección de recursos. Más adelante se entrará en el aspecto técnico de las pociones y su uso en el juego.

3.6.2. Mejora de estadísticas mediante etiquetas

Las etiquetas funcionan como modificadores adicionales, diferenciándose en bendiciones y maldiciones. Estas aportan bonificaciones concretas que complementan las estadísticas base, permitiendo la configuración de estrategias personalizadas según el estilo de juego.

Asimismo, el diseño sistémico se orientó a garantizar un equilibrio funcional entre ambas categorías. En este sentido, se estableció que las maldiciones no operasen exclusivamente como penalizadores, sino que incorporasen también atributos positivos asociados. De este modo, su uso implica una lógica de riesgo-beneficio, en la que los efectos adversos se compensan con ventajas específicas, fomentando decisiones tácticas más complejas y haciendo viable (y potencialmente ventajosa) la adopción de configuraciones negativas en determinados contextos jugables.

3.6.3. Mejora de estadísticas mediante alineaciones

El alineamiento moral del personaje influye igualmente en sus capacidades. Dependiendo de su afinidad con Cielo, Tierra o Infierno, se activan modificadores que alteran el rendimiento en combate, reforzando la integración entre narrativa y jugabilidad sistémica. Más adelante se entrará en el aspecto técnico de las alineaciones y su impacto en el juego.

3.7. Diseño de mecánicas de juego

El diseño mecánico se estructuró en torno a un conjunto de acciones fundamentales que definen la interacción del jugador con el mundo.

Entre las principales mecánicas implementadas se encuentran:

- **Sistema de Alineamiento Dinámico**: Se establece como la mecánica nuclear del proyecto. El software registra las acciones morales del jugador mediante la acumulación de puntos invisibles asociados a sus decisiones, interacciones y resultados de misiones.

  Al alcanzarse un umbral de 100 puntos, el sistema desbloquea una de las tres alineaciones (**Cielo, Tierra o Infierno**), produciendo una modificación permanente sobre las estadísticas base del *Character Blueprint*, concretamente en los valores de salud, daño y velocidad. Esta transformación afecta tanto al rendimiento en combate como a la interacción sistémica con el mundo.

- **Combate Avanzado y Movilidad Técnica**:

  - **Sistema de Fijación (Lock-on)**: Permite centrar la cámara y los ataques sobre un objetivo específico, garantizando precisión direccional en enfrentamientos en tercera persona.

  - **Evasión con Invulnerabilidad**: Se implementa una acción de rodar (roll) que concede frames de invulnerabilidad temporales, posibilitando la evasión estratégica de ataques enemigos.

  - **Equipamiento Dual de Armas**: El sistema permite portar simultáneamente dos espadas, posibilitando la alternancia dinámica durante el combate sin necesidad de acceder al inventario, manteniendo la fluidez del enfrentamiento.

- **Movimiento del personaje**: desplazamiento libre por el entorno tridimensional, incluyendo acciones de movilidad avanzada.

- **Alquimia y Creación de Objetos (Crafting):** Se implementa un sistema de creación de consumibles integrado en el *Widget Blueprint [17]* de magia. El jugador debe combinar ingredientes recolectados en espacios específicos para generar pociones con efectos temporales (*buffs*), tales como incrementos de velocidad, ampliación de salud máxima o habilidades de movilidad avanzada como el triple salto.

- **Exploración Dinámica y Registro de Criaturas**:

  - **Revelación Progresiva del Mapa**: El entorno emplea un modelo de descubrimiento automático mediante el cual las zonas se revelan cartográficamente conforme son transitadas, incentivando la exploración no lineal.

  - **Mecánica de Avistamiento (Bestiario):** La proximidad a criaturas permite su registro en el Journal, desbloqueando entradas informativas y otorgando recompensas de alineamiento al completarse el compendio.

  - **Domesticación de animales**: captura y vinculación de criaturas específicas.

  - **Recolección de objetos**: obtención de recursos, consumibles y equipo.

  - **Recuperación de vida en fogatas**: puntos de descanso que restauran la salud del personaje.

  - **Guardado de localización**: las fogatas actúan además como nodos de guardado persistente de progreso.

- **Sistema de Bendiciones y Maldiciones:** El sistema permite equipar hasta tres bendiciones y tres maldiciones simultáneamente. Estas funcionan como modificadores estadísticos avanzados. Las maldiciones se diseñan bajo una lógica de **intercambio estratégico (*trade-off*)**, donde una penalización concreta (por ejemplo, reducción de velocidad) se compensa con mejoras sustanciales en otros parámetros (como salud), favoreciendo configuraciones tácticas complejas.

- **Interacción Social y Sistema de NPC**: El sistema permite la comunicación con personajes no jugables mediante diálogos funcionales que habilitan:
  - Activación de misiones
  - Comercio
  - Obtención de información narrativa
  - Modificación de alineamiento

- **Sistema de Inventario y Gestión de Equipo:** El inventario permite la administración de armas, consumibles y etiquetas, posibilitando la personalización del estilo de juego mediante la combinación de equipamiento y modificadores.

En conjunto, estas mecánicas configuran el bucle jugable principal, integrando exploración, progresión y combate dentro de una estructura cohesionada.

3.8. Diagrama de flujo de movimiento de los personajes

Todos los diagramas de flujo serán realizados con Miro [23].

3.8.1. Enemigos básicos

Los enemigos básicos distribuidos a lo largo del mundo de juego comparten una misma lógica de comportamiento y combate. Este sistema común define sus patrones de detección, persecución, ataque y retorno a estado de reposo, garantizando coherencia funcional y eficiencia en la implementación.

Las variaciones entre enemigos se establecen principalmente a nivel de modelo tridimensional, atributos estadísticos (salud, daño, velocidad, resistencia) y, en determinados casos, efectos visuales asociados. Este enfoque modular permite reutilizar la arquitectura de comportamiento base, reduciendo la complejidad de desarrollo sin comprometer la diversidad jugable.

En la Figura 109 se presenta el diagrama de flujo correspondiente a la lógica general de los enemigos básicos.

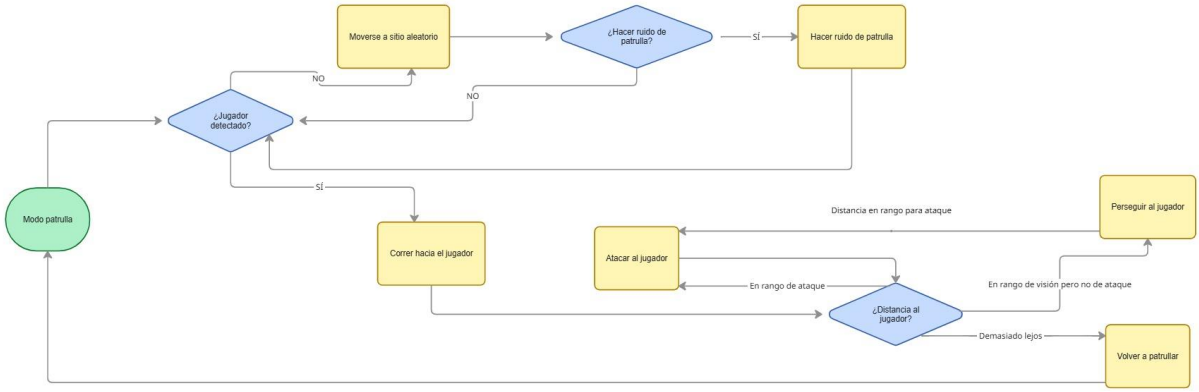


Figura 109: Diagrama de flujo de enemigos básicos.

3.8.2. Jefe final

El Jefe Final constituye la entidad con la lógica de comportamiento más compleja dentro del sistema de combate del juego. Su diseño se estructura en tres fases diferenciadas, cada una con variaciones en patrones de ataque, agresividad, habilidades disponibles y respuesta al jugador.

La transición entre fases se activa mediante condiciones específicas (umbrales de salud), lo que introduce progresión dramática y escalada de dificultad durante el

enfrentamiento. Este planteamiento busca generar una experiencia de combate más dinámica y estratégica en comparación con los enemigos estándar.

En la Figura 110 se muestra el diagrama de flujo que representa la arquitectura completa de comportamiento del Jefe Final y la transición entre sus distintas fases.

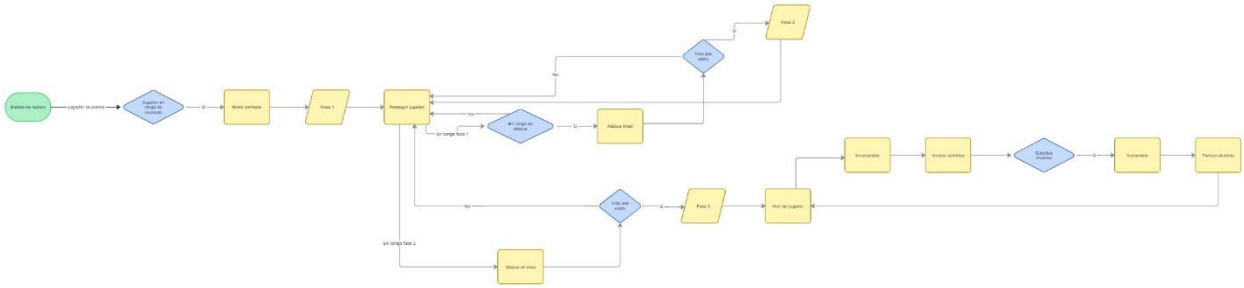


Figura 110: Diagrama de flujo del jefe final.

3.8.3. Súbdito del jefe final

Los súbditos asociados al combate contra el Jefe Final presentan una lógica de comportamiento significativamente más simple, al estar concebidos como unidades de apoyo dentro del enfrentamiento principal.

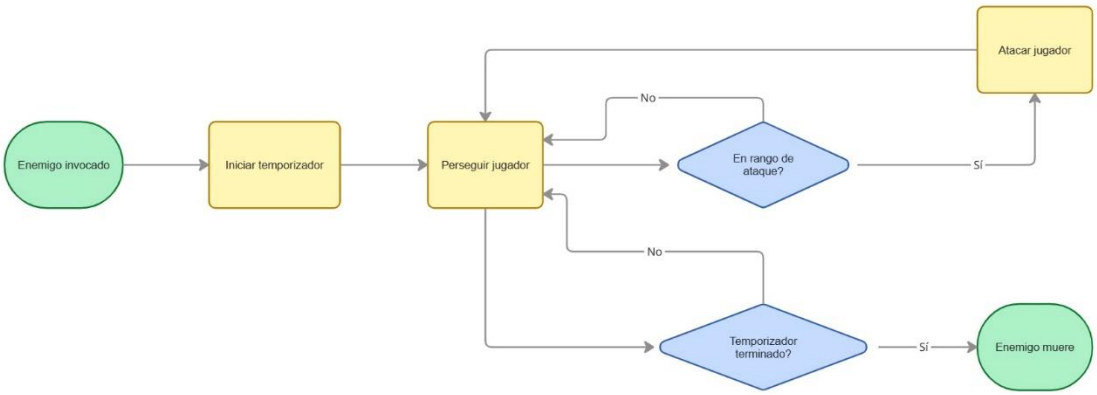


Figura 111: Diagrama de flujo de los súbditos del jefe final.

3.9. Historia del juego

El universo del juego se estructura en una configuración de marco (*frame*) dividida en tres estratos: **Cielo, Tierra e Infierno**. El protagonista, una entidad conocida como la **Llama del Juicio**, asume identidades distintas en cada plano: **Caelvorn** en el Cielo, **Elandor** entre los hombres y **Azrakar** en el Infierno.

La narrativa cuestiona la pureza del plano celestial al presentar un **Cielo corrupto o dogmático**. El conflicto se detona cuando el "Líder Supremo" (Dios) impone mandatos éticamente cuestionables, como la aniquilación de aldeas bajo el pretexto de una supuesta corrupción que no es tal, utilizándolas como pruebas de lealtad.

No obstante, para la construcción de la **demo funcional**, se optó por simplificar este trasfondo narrativo con el objetivo de facilitar la comprensión inicial del jugador y

focalizar los recursos de desarrollo en los sistemas jugables principales. En esta versión reducida, el protagonista ha sido desterrado del plano celestial tras cometer un acto de traición motivado por la codicia: el asesinato del portador legítimo de un arma divina con el fin de arrebatársela.

Como consecuencia de este crimen, la deidad suprema decreta su exilio, permitiéndole conservar el arma sustraída no como recompensa, sino como símbolo de culpa y recordatorio permanente de su necesidad de redención. Este artefacto constituye, además, el arma inicial del jugador, integrando de forma diegética la progresión jugable con el trasfondo narrativo del personaje.

El mundo principal de *Order Frame* cuenta con una geografía diversa que refleja las tensiones entre las facciones:

- Eldhollow: Aldea agrícola y primer contacto del jugador con la Tierra.
- Dunbrae: Aldea oculta en el bosque, hogar de herreros independientes que actúan como refugio neutral.
- Seraphen: Ciudad de clérigos y santos, geográficamente la más próxima al Cielo.
- Froswynd: Asentamiento gélido situado en las regiones del norte.
- Cragwatch: Bastión militar ubicado en las montañas.
- Mirenlake: Ciudad comercial rica con un marcado submundo criminal.
- Bristlebrook: Antigua aldea diezmada tras incursiones demoníacas.
- Umbreth: Zona de influencia directa de los adoradores del Infierno.
- Thaloriel y Abyzakar: Capitales del Cielo y el Infierno, respectivamente.
- Thornsreach Forest: Bosque misterioso que solo deja pasar al jugador si este tiene un objeto específico obtenido al salvar a un NPC. Dentro se encuentra la aldea de Dunbrae.
- Skarvald Peaks: Protege al mundo de los demonios, sirve como barrera natural.
- Mirror Lake: Uno de los pocos sitios donde los aldeanos pueden obtener peces sin tener que someterse a una fe.
- La tribu Veyari: Tribu nómada que adoran a un Dios que se encuentra en el Limbo.
- Limbo: Lo que rodea al mundo entero. El cielo es como una barrera que protege al mundo del poder destructivo del Limbo.

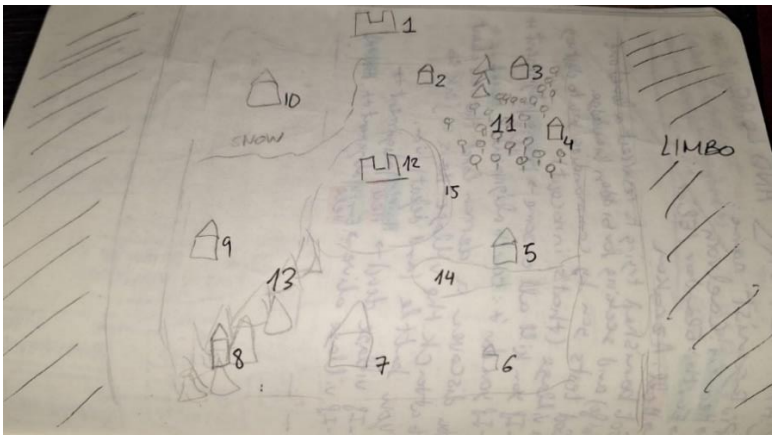


Figura 112: Mapa original de Order Frame.

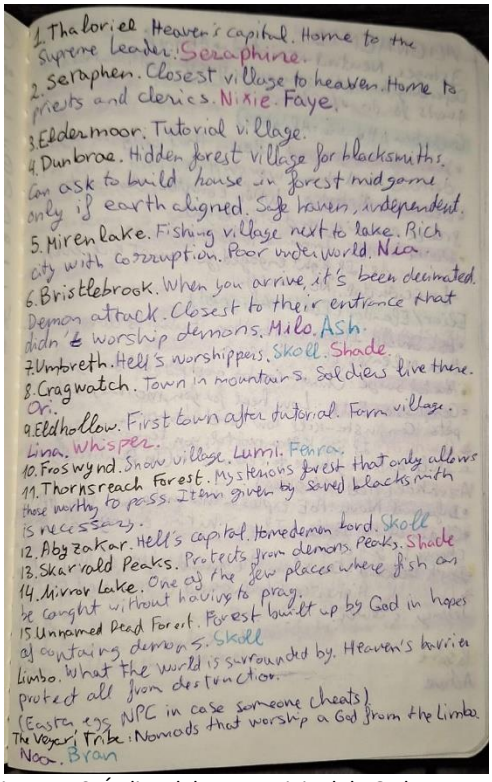


Figura 113: Índice del mapa original de Order Frame.

Un pilar central de la narrativa reactiva es la posibilidad de rescatar y adoptar entidades que influyen en el alineamiento y el final del juego.

Niños Adoptables: El proyecto original contempla siete niños con trasfondos específicos vinculados a las localizaciones:

- **Lina** (Eldhollow).
- **Nia** (MirenLake).
- **Lumi** (Froswynd).
- **Faye** (Seraphen).
- **Ori** (Cragwatch).
- **Milo** (Bristlebrook).
- **Noa** (tribu nómada Veyari).

La adopción de estos personajes es un requisito para finales específicos, como el de "Guardián de la Tierra".

Para poder iniciar el proceso de adopción, el jugador debe ganarse progresivamente la confianza de cada menor mediante interacciones positivas, tales como la entrega de alimentos u otros objetos beneficiosos. Estas acciones incrementan un sistema interno de afecto o afinidad. Una vez alcanzado el nivel máximo de relación, la adopción solo puede formalizarse si se cumplen adicionalmente dos condiciones: que el jugador se encuentre alineado con la Tierra y que disponga de una vivienda previamente adquirida en la ciudad de Dunbrae.

Al completarse el proceso, los personajes adoptados se trasladan automáticamente a la residencia del jugador. Asimismo, si estos se encuentran acompañados por mascotas asignadas a su cuidado, se activa un sistema pasivo de generación de recursos, produciendo objetos o divisas a lo largo del tiempo que pueden ser reclamados mediante interacción directa con ellos.

Mascotas y Acompañantes Animales: El sistema permite la domesticación de criaturas que participan activamente en el combate y la exploración:

- **Caninos/Lobos: Fenra** (rastreadora), **Bran** (protector contra demonios invisibles), **Ash** (potenciador de moral) y **Skoll** (lobo feroz con daño extra contra facciones celestiales).
- **Felinos: Nixie** (detectora de trampas), **Whisper** (exploradora sigilosa), **Seraphine** (guardiana celestial) y **Shade** (detectora de ilusiones demoníacas).

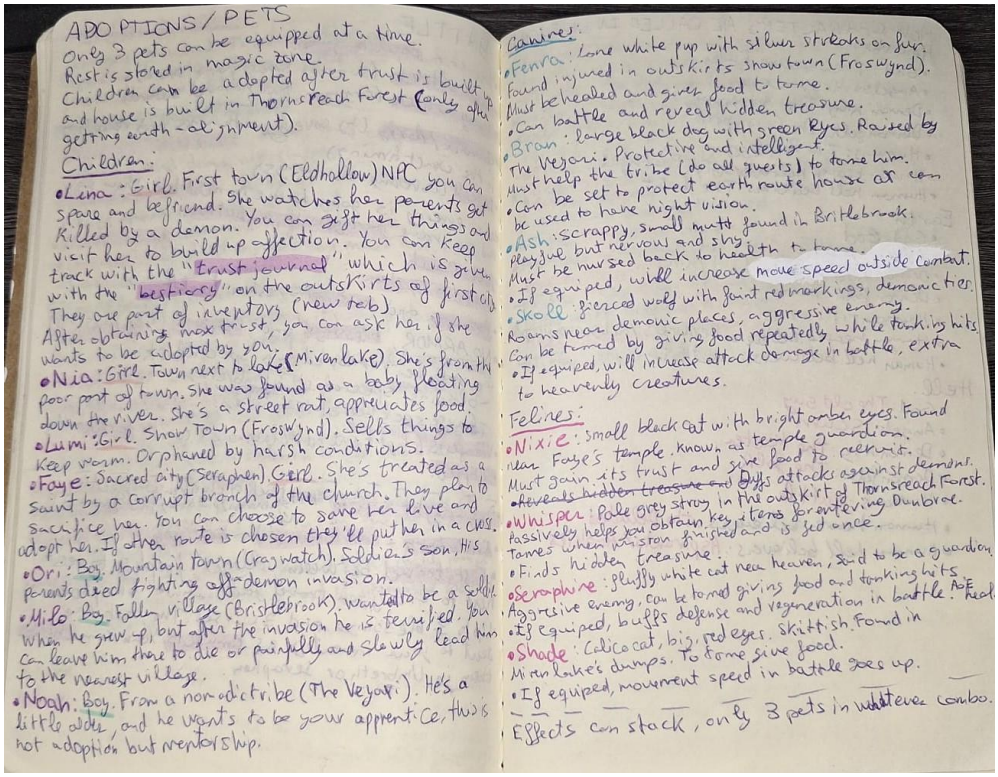


Figura 114: Historia base de todos los niños y mascotas.

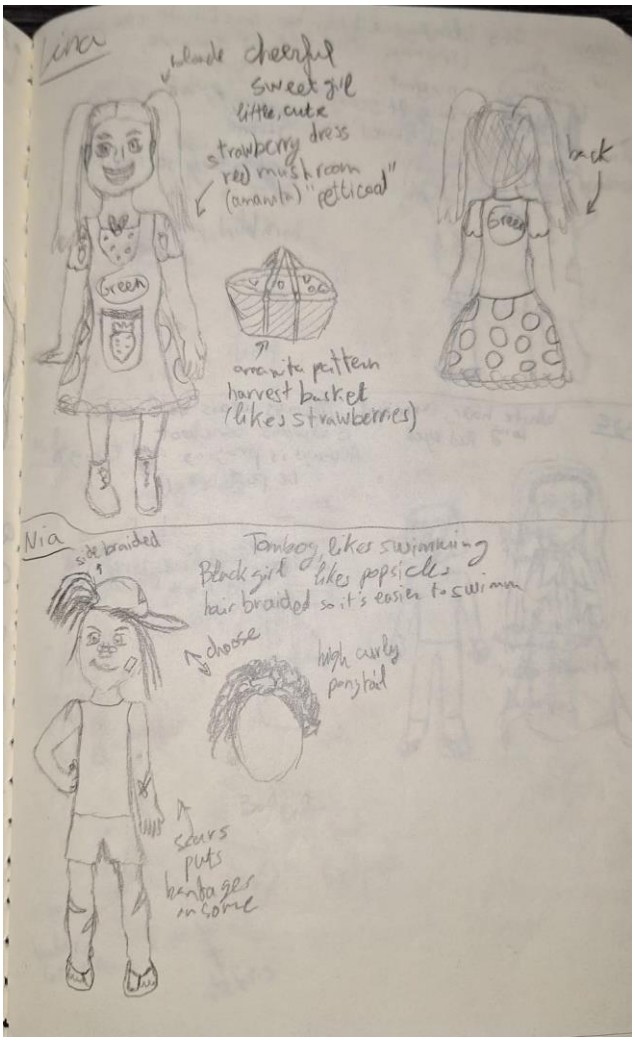


Figura 115: Boceto niños adoptables 1.

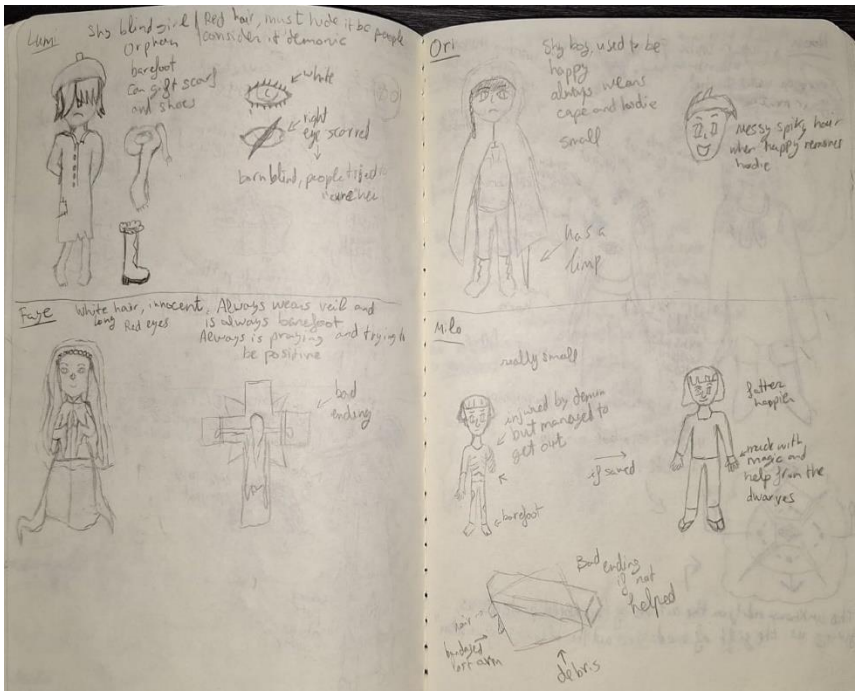


Figura 116: Boceto niños adoptables 2.

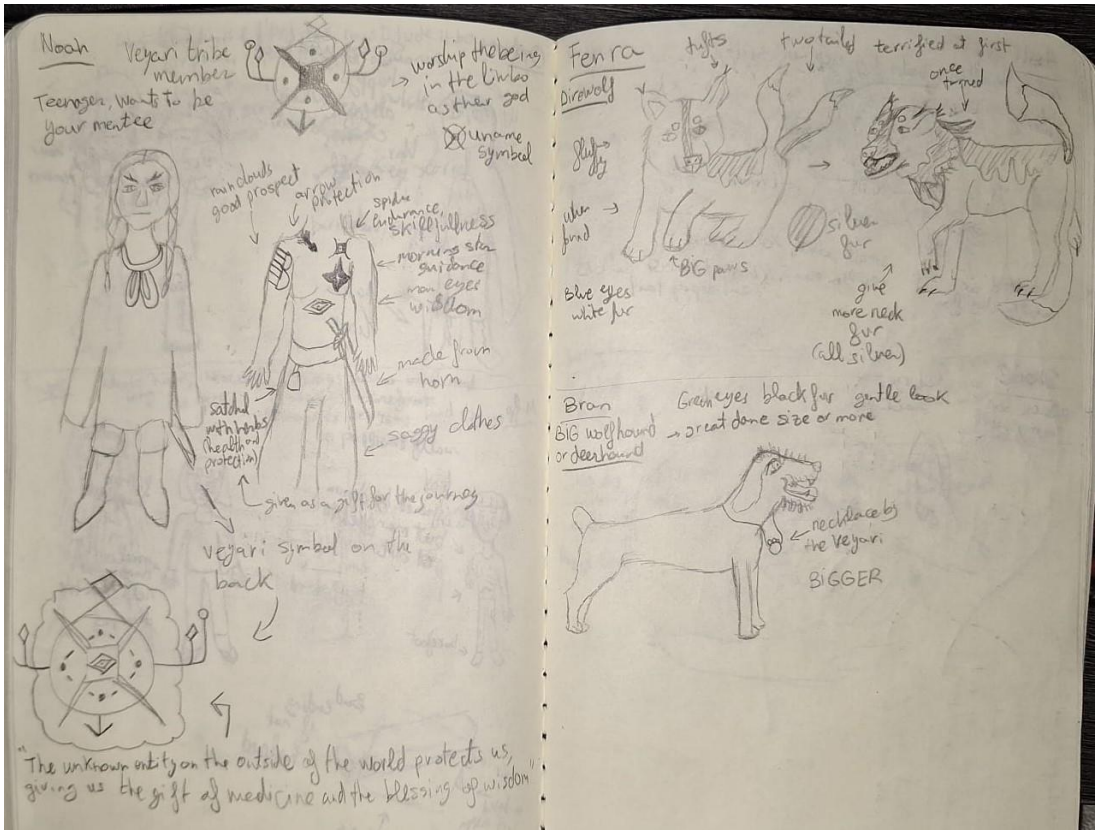


Figura 117: Boceto niños adoptables y macotas.

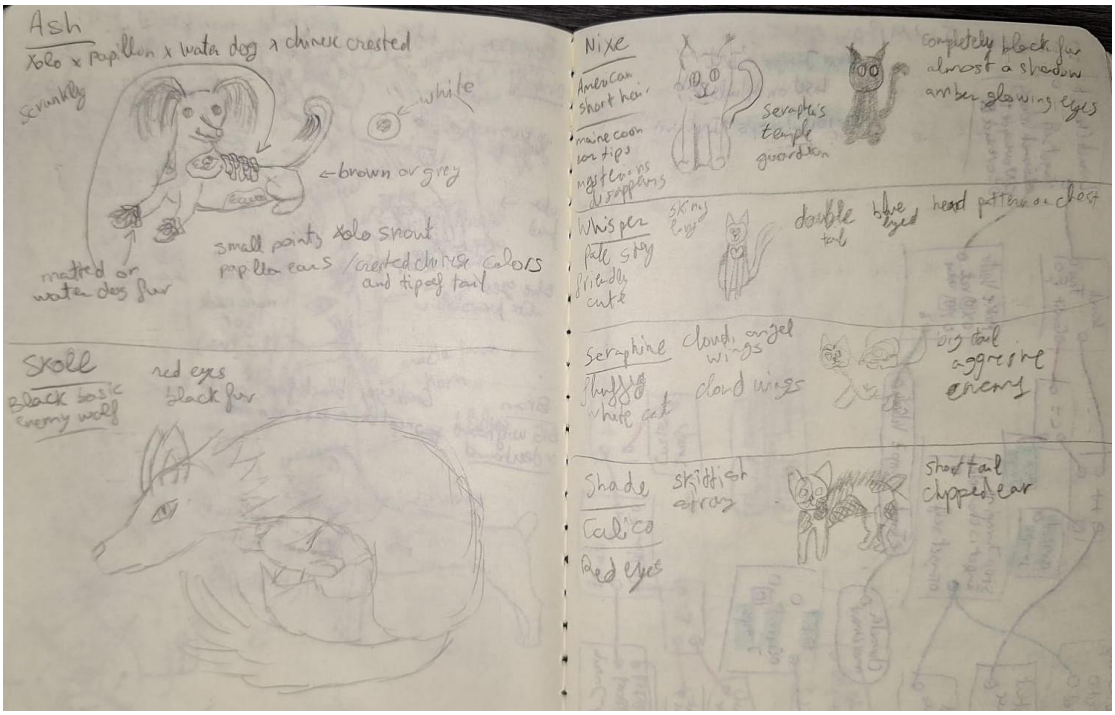


Figura 118: Boceto macotas.

En conjunto, estos elementos configuran una experiencia donde la **agencia del jugador** y su respuesta ante la corrupción del orden establecido definen no solo su alineamiento técnico, sino el destino de los habitantes del mundo.

4

Implementación

4.1. Personaje jugable

El personaje jugable constituye el núcleo central del videojuego, no solo desde el punto de vista de la interacción directa con el jugador, sino también a nivel sistémico. Desde este Blueprint [8] se orquesta una parte sustancial de la lógica global del juego mediante el uso de Event Dispatchers [10] y funciones específicas que comunican estados, eventos y cambios a otros sistemas como el HUD, inventario, combate, IA o misiones.

El Blueprint [8] principal del personaje es BP_MyCharacter, el cual encapsula tanto los componentes físicos como los sistemas lógicos asociados al combate, inventario, estadísticas y control de animaciones.

Esta arquitectura centralizada permite que el personaje actúe como eje de sincronización entre subsistemas, facilitando la escalabilidad del proyecto y la modularidad del desarrollo.

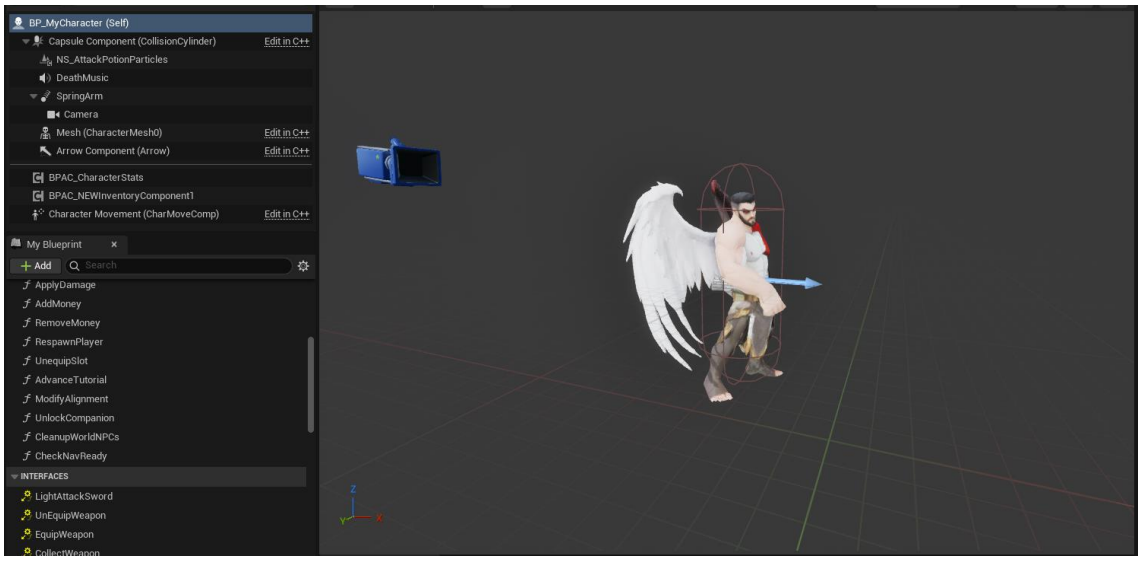


Figura 119: Componentes del Blueprint del personaje jugable.

Componentes principales identificados

1. **Capsule Component (CollisionCylinder)**
    - Define las colisiones físicas del personaje.
    - Gestiona impactos, navegación y detección de suelo.
    - Determina altura y anchura del actor.
2. **Mesh (CharacterMesh0)**
    - Malla esquelética visible del personaje.
    - Vinculada al esqueleto para reproducir animaciones.
    - Permite aplicar montajes de combate.
3. **SpringArm**
    - Brazo articulado que separa la cámara del personaje.
    - Evita clipping en paredes.
    - Permite zoom dinámico.
4. **Camera**
    - Cámara en tercera persona.
    - Sigue la rotación del controlador.
5. **Arrow Component**
    - Referencia direccional para orientación y spawn.
6. **NS_AttackPotionParticles**
    - Sistema de partículas asociado a los efectos de la poción de ataque. [13]
7. **DeathMusic**
    - Componente de audio que reproduce música al morir.
8. **BPAC_CharacterStats**
    - Componente Actor para estadísticas.
9. **BPAC_NEWInventoryComponent1**
    - Gestión del inventario del jugador.

A continuación, se describen de manera individualizada las funciones principales implementadas.

Funciones de control y cámara

Move

La función **Move** permite al jugador controlar el desplazamiento del personaje dentro del entorno tridimensional. Recibe los valores de entrada del teclado o mando y los traduce en vectores de movimiento relativos a la orientación de la cámara.
Incluye:

- Movimiento frontal y lateral.
- Normalización de velocidad.
- Adaptación a pendientes.
- Integración con el Character Movement Component.

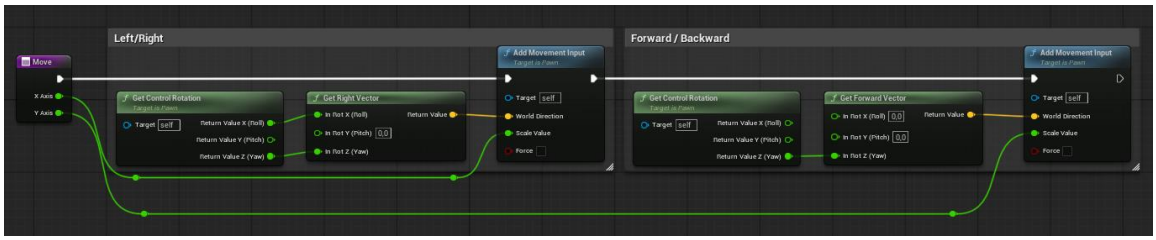


Figura 120: Función Move en BP_MyCharacter.

Aim

La función **Aim** gestiona la rotación de la cámara del jugador, permitiendo orientar la vista en los ejes horizontal y vertical.

Se conecta a:

- Spring Arm.
- Cámara en tercera persona.
- Sensibilidad configurable.

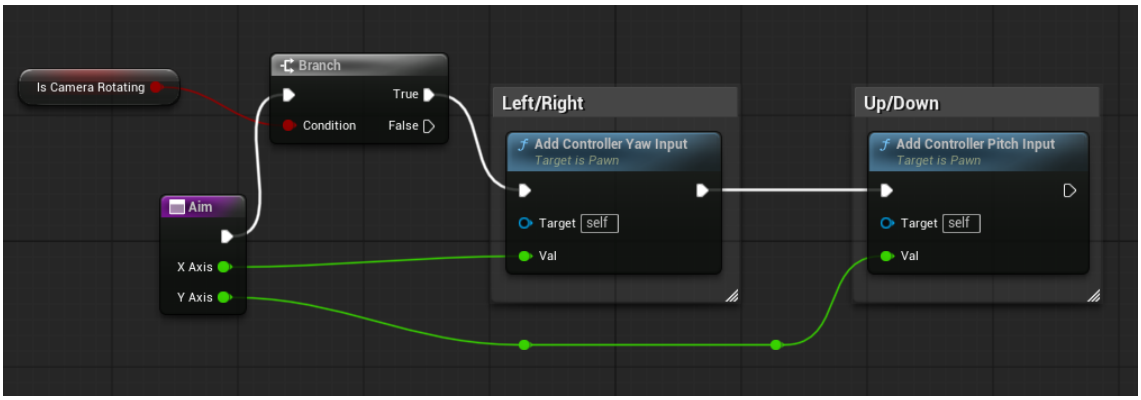


Figura 121: Función Aim en BP_MyCharacter.

Sistema de vida y daño

IncrementHealth

La función **IncrementHealth** permite curar al personaje.

Características:

- Suma una cantidad determinada.
- Limita el valor a **MaxHealth**.
- Evita sobrecuración.

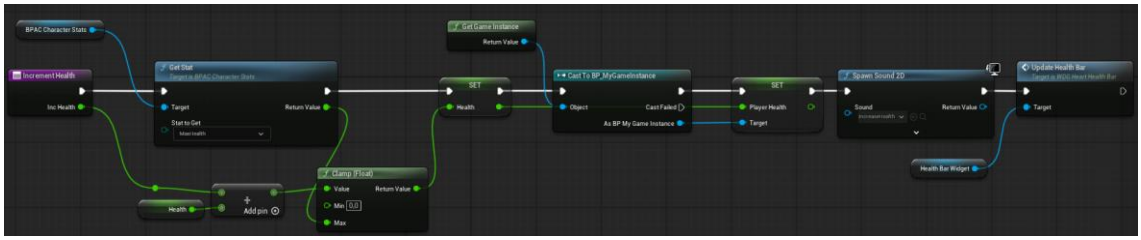


Figura 122: Función IncrementHealth en BP_MyCharacter.

DecrementHealth

La función **DecrementHealth** reduce la vida del personaje cuando recibe daño.
Funciones internas:

- Resta de vida.
- Límite inferior en 0.
- Lanzamiento de Event Dispatcher [10] de daño recibido.
- Comprobación de muerte.

Si la vida llega a 0:

- Se realiza un **Bind** dentro de la propia función.
- Se llama automáticamente a la función de muerte.

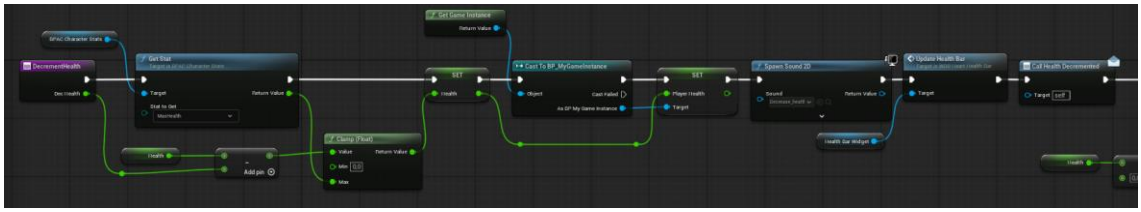


Figura 123: Función DecrementHealth en BP_MyCharacter.

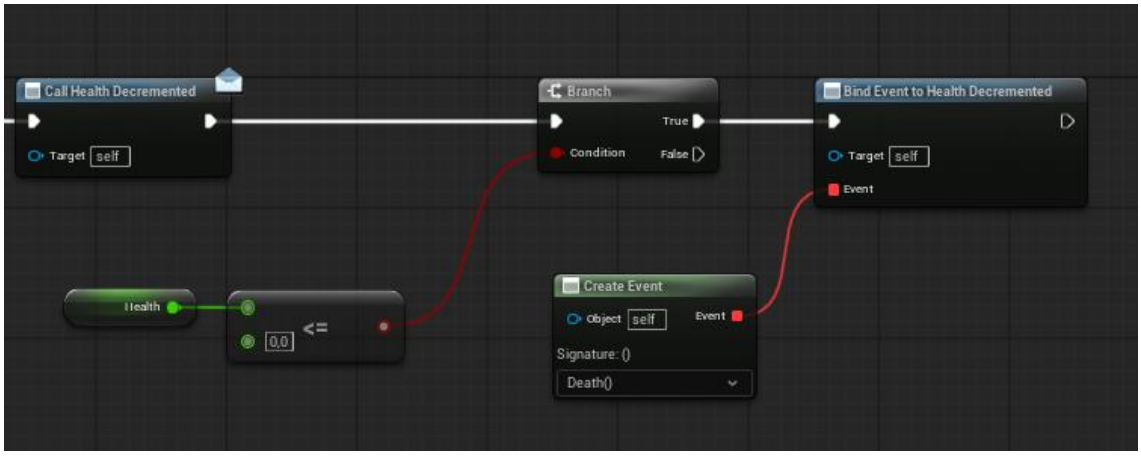


Figura 124: Llamada a evento de muerte en la función DecrementHealth en BP_MyCharacter.

ApplyDamage

La función **ApplyDamage** actúa como capa superior de gestión de daño.
Particularidad clave:

- Si el personaje está rodando (dodge roll), es **invulnerable**.
- Se comprueba el estado antes de aplicar daño real.

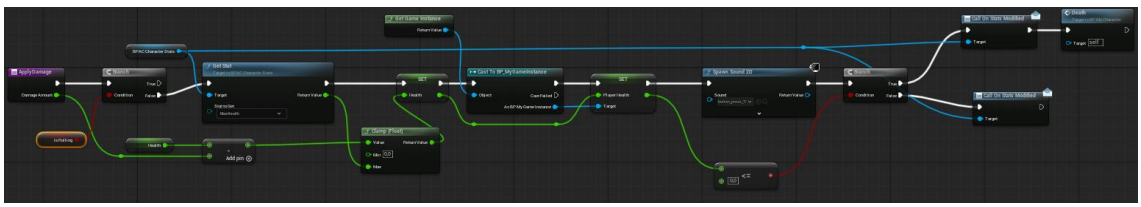


Figura 125: Función ApplyDamage en BP_MyCharacter.

AttachWeaponToHandSocket
Acopla el arma al socket de la mano del personaje cuando se equipa.

AttachWeaponToSheathSocket_R / _L
Ambas funciones realizan la misma operación:

- Colocar la espada en la vaina correspondiente.
- Diferenciación por lateralidad (derecha / izquierda).

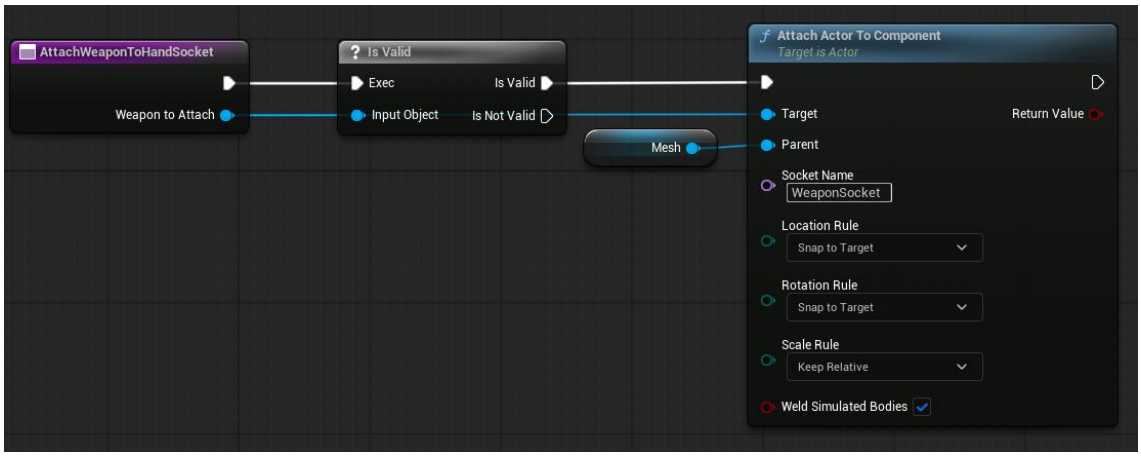


Figura 126: Función AttachWeaponToHandSocket en BP_MyCharacter, sólo cambia el Socket Name en las otras dos.

UpdateEquipmentSlot
Gestiona el cambio de armas dentro del inventario.
Funciones:

- Determina arma equipada.
- Actualiza slot activo.
- Notifica al HUD.
- Controla en qué posición visible está el arma.

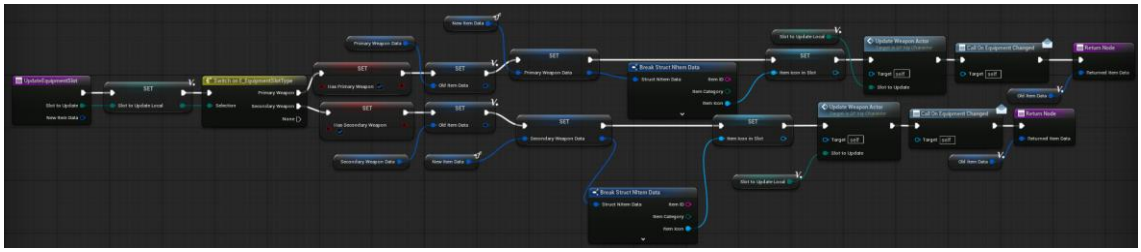


Figura 127: Función UpdateEquipmentSlot en BP_MyCharacter.

UpdateWeaponActor
Se encarga de actualizar el **modelo 3D** del arma equipada.
Permite:

- Cambiar malla de espada.
- Actualizar estadísticas asociadas.
- Sincronizar con animaciones.

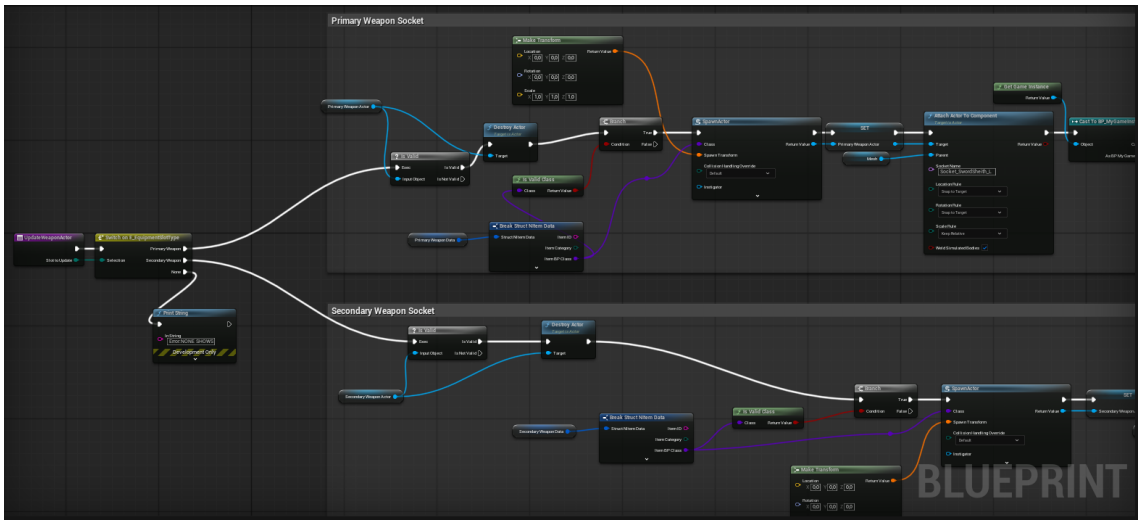


Figura 128: Función UpdateWeaponActor en BP_MyCharacter.

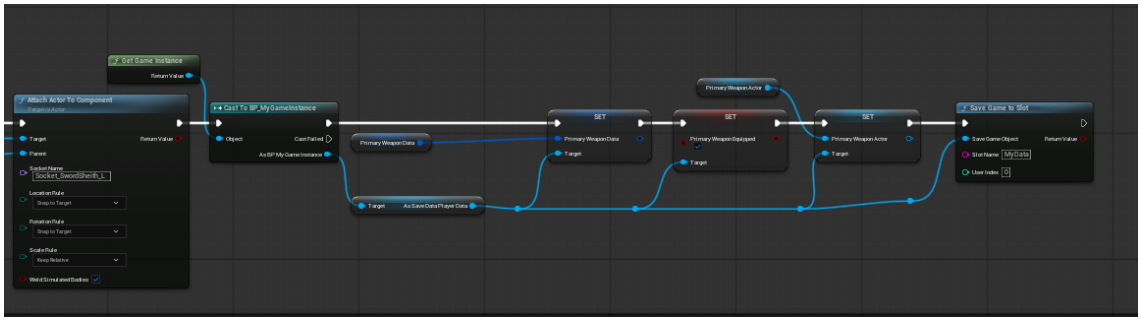


Figura 129: Guardado de la información en la función UpdateWeaponActor en BP_MyCharacter.

Sistema económico

AddMoney
Añade dinero al inventario del jugador y lo guarda.
Parámetros:

- Cantidad.
- Tipo de moneda.

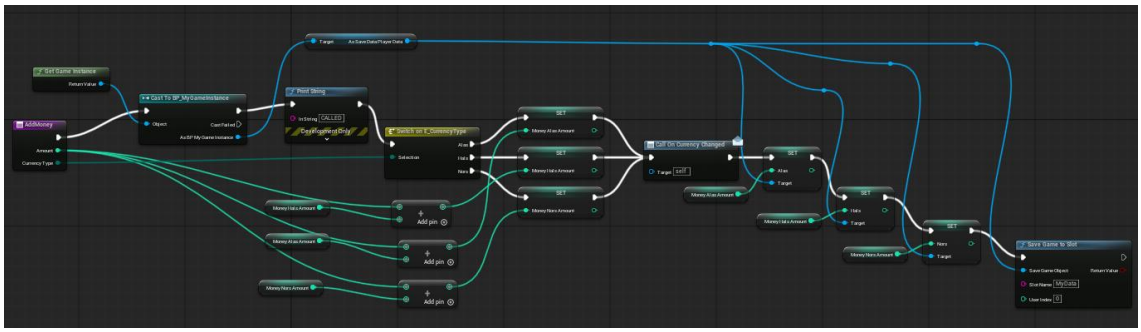


Figura 130: Función AddMoney en BP_MyCharacter.

RemoveMoney
Resta dinero según coste o compra.
Controla:

- Cantidad mínima.
- Tipo de moneda requerido.

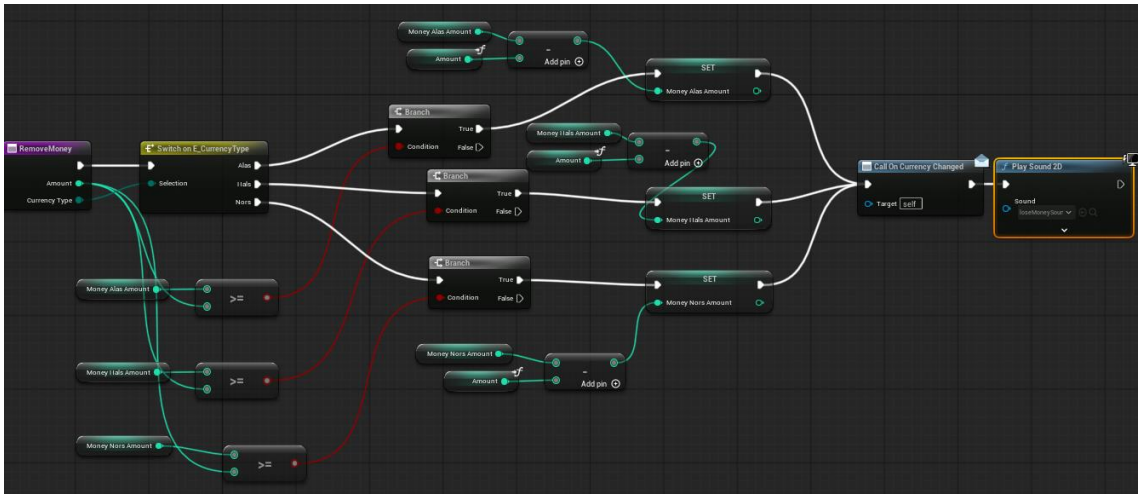


Figura 131: Función RemoveMoney en BP_MyCharacter.

Sistema de muerte y reaparición

RespawnPlayer
Gestiona la reaparición del jugador.
Características:

- Revive en localización correcta del mapa principal.
- Especial control si muere en mapa de jefe final.
- Restaura:
    - Vida máxima.
    - Objetos necesarios.

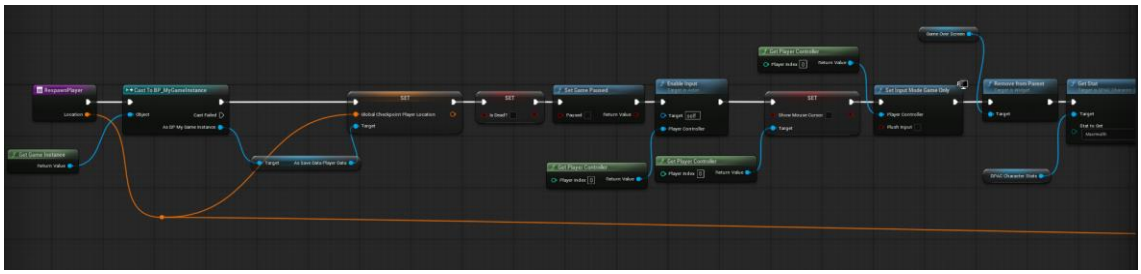


Figura 132: Función RespawnPlayer en BP_MyCharacter parte 1.

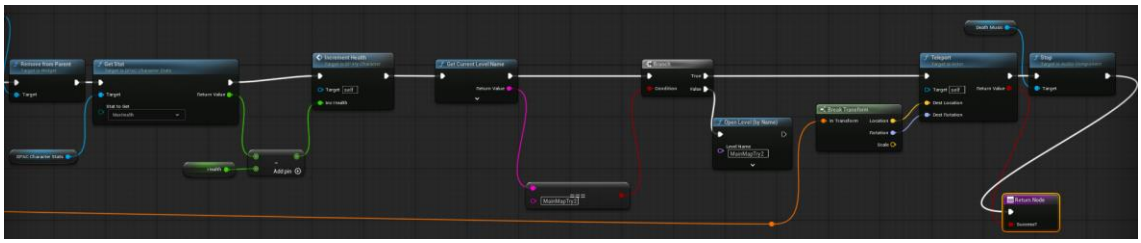


Figura 133**:** Función RespawnPlayer en BP_MyCharacter parte 2.

Figura 134: Evento Death en BP_MyCharacter en el event graph y su relación con RespawnPlayer.

Gestión de equipamiento

UnequipSlot
Permite retirar un arma o equipo.
Funciones:

- Vacía el slot.
- Actualiza estadísticas.
- Evita que el jugador sea considerado armado.

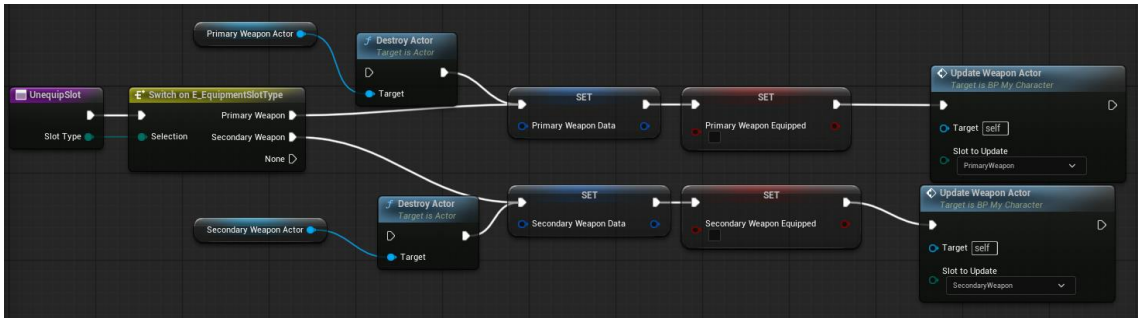


Figura 135: Función UnequipSlot en BP_MyCharacter.

Sistema de misiones y progreso

AdvanceTutorial
Avanza la misión especial de tutorial.
Permite:

- Actualizar objetivos.

- Desbloquear mecánicas.
- Guiar progresión inicial.

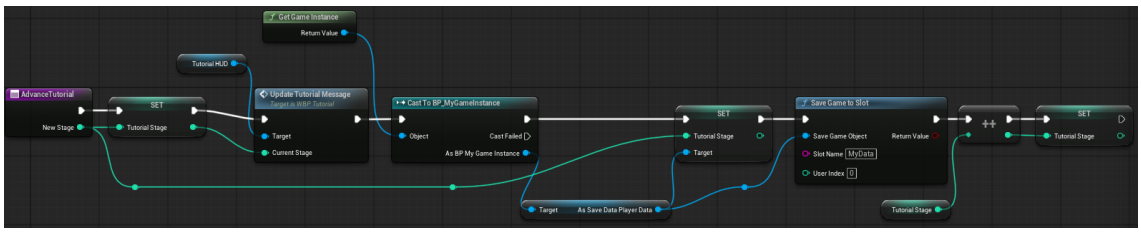


Figura 136: Función AdvanceTutorial en BP_MyCharacter.

Sistema de alineamiento

ModifyAlignment
Gestiona el alineamiento moral del jugador.
Funciones:

- Suma/resta puntos.
- Tipo de alineamiento.
- Comprobación de umbral (100 puntos).

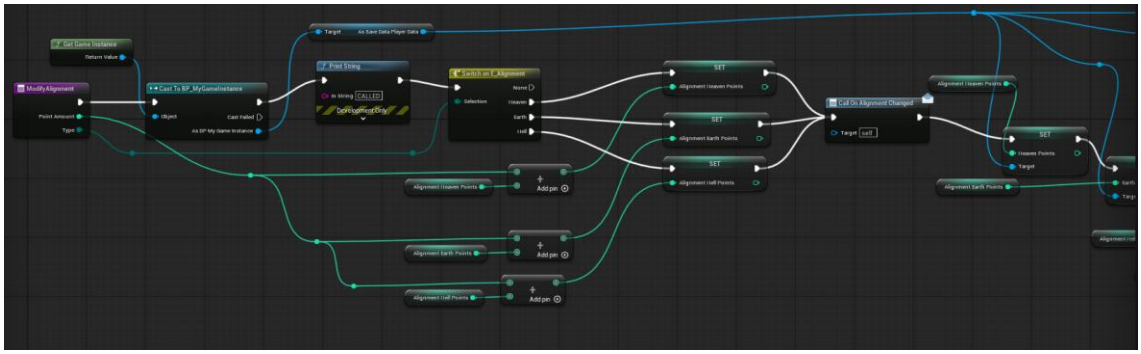


Figura 137: Función ModifyAlignment en BP_MyCharacter.

Al alcanzarlo:

- Se muestra botón de cambio de alineamiento en inventario.

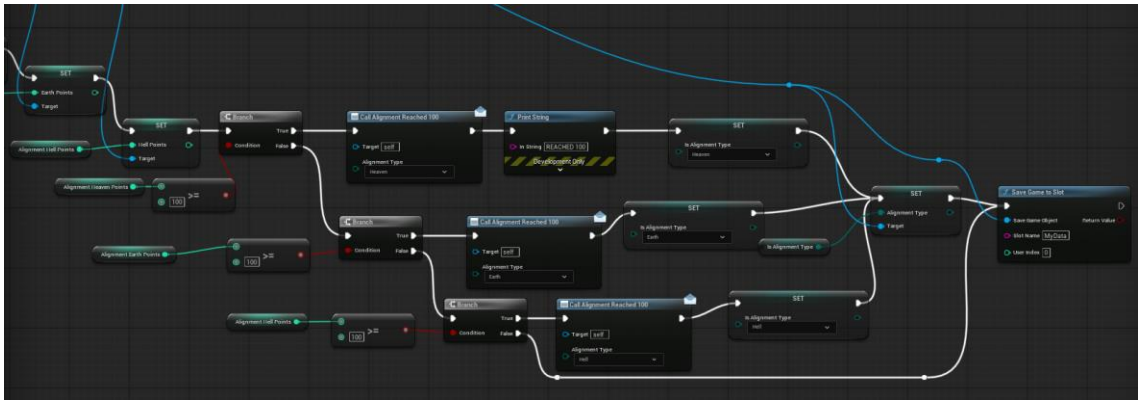


Figura 138: Función ModifyAlignment y la llamada a Alignment Reached en BP_MyCharacter.

Sistema de compañeros

UnlockCompanions
Desbloquea acompañantes tras ser domesticados.
Permite:

- Añadirlos al sistema de seguimiento.
- Activar habilidades asociadas.

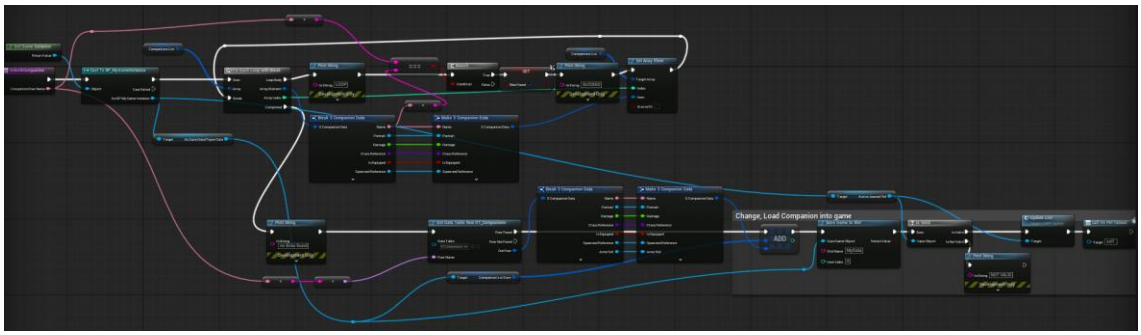


Figura 139: Función UnlockCcompanions en BP_MyCharacter.

CleanUpWorldNPCs
Elimina NPCs redundantes del mundo una vez domesticados.
Objetivo:

- Evitar duplicados.
- Mantener coherencia narrativa.

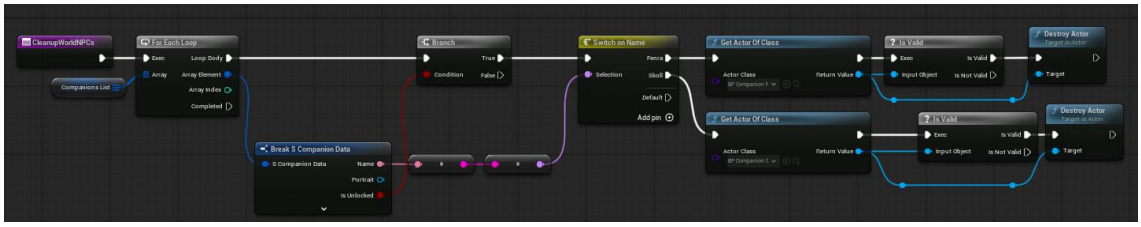


Figura 140: Función CleanUpWorldNPCs en BP_MyCharacter.

En conclusión, el Blueprint [8] del personaje jugable no solo gestiona el control directo del avatar, sino que centraliza sistemas de combate, economía, progresión, alineamiento, inventario, misiones y acompañantes. La utilización de funciones específicas junto con Event Dispatchers [10] convierte a este actor en el eje estructural del videojuego, garantizando la comunicación eficiente entre subsistemas y permitiendo una arquitectura escalable y mantenible.

4.1.2. Animación del personaje

El sistema de animación del personaje jugable se ha implementado mediante Animation Blueprints en Unreal Engine [16], integrando máquinas de estados, transiciones por velocidad y reproducción de montajes de combate. Este enfoque permite gestionar de forma modular las animaciones de desplazamiento, ataque, esquiva y estados especiales, garantizando coherencia visual entre acciones y respuesta jugable.

Inicialmente, se planteó la incorporación de técnicas avanzadas de suavizado de movimiento mediante IK Rigs (Inverse Kinematics Rigs), con el objetivo de mejorar la adaptación de las extremidades al terreno y dotar a los giros de una mayor naturalidad cinemática. Asimismo, se pretendía implementar interpolaciones más complejas en las rotaciones para evitar transiciones bruscas durante los cambios de dirección.

No obstante, debido a limitaciones derivadas de los modelos tridimensionales disponibles en las primeras fases del desarrollo (principalmente relacionadas con la calidad de los esqueletos, jerarquías óseas incompatibles y ausencia de configuraciones preparadas para retargeting avanzado), resultó inviable implementar dichas soluciones en ese momento sin comprometer de forma significativa los tiempos de producción.

Posteriormente, y gracias a la incorporación de modelos gratuitos de mayor calidad técnica, fue posible integrar sistemas de IK de forma parcial, permitiendo una mejora visible en la suavidad de los giros, la estabilidad de las pisadas y la adaptación general del personaje al entorno.

A continuación, se presenta una comparativa visual entre ambas configuraciones:

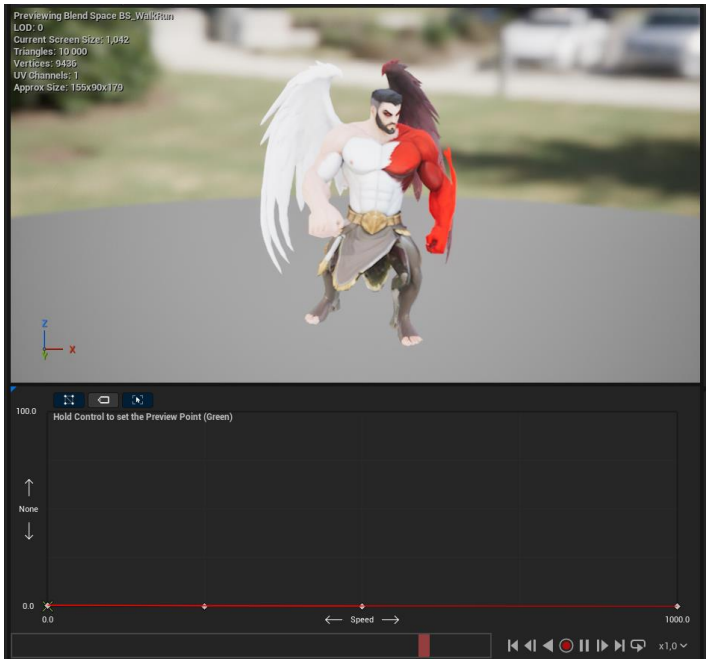


Figura 141: Blend Space de movimiento del personaje principal.

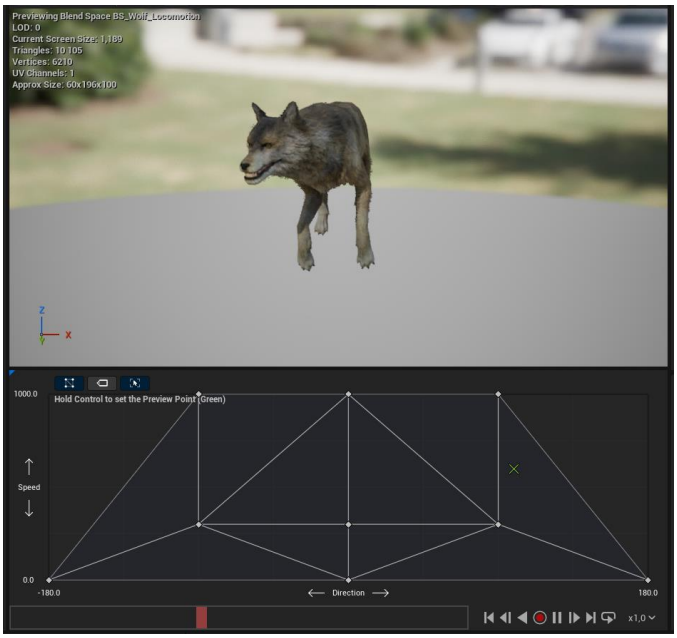


Figura 142: Blend Space de movimiento del lobo en comparación.

Derivado de las limitaciones asociadas a la implementación de los sistemas de IK Rigs en las fases iniciales, se optó por la adopción de una solución alternativa orientada a garantizar la correcta alineación del personaje con la superficie del terreno. En este sentido, se estableció que el modelo del personaje mantuviese en todo momento una orientación perpendicular (90º) respecto al plano del suelo.

Para la consecución de este comportamiento, se empleó un procedimiento basado en la ejecución de un nodo *Line Trace By Channel* proyectado de forma perpendicular al modelo del jugador. A partir de la normal de impacto obtenida mediante dicha traza, se procedió a configurar dinámicamente la rotación del personaje, ajustándola al valor resultante con el fin de asegurar su adecuada adaptación a la inclinación del terreno.

El video utilizado como referencia para realizar esta funcionalidad es de *The Game Dev Cave [31]*.

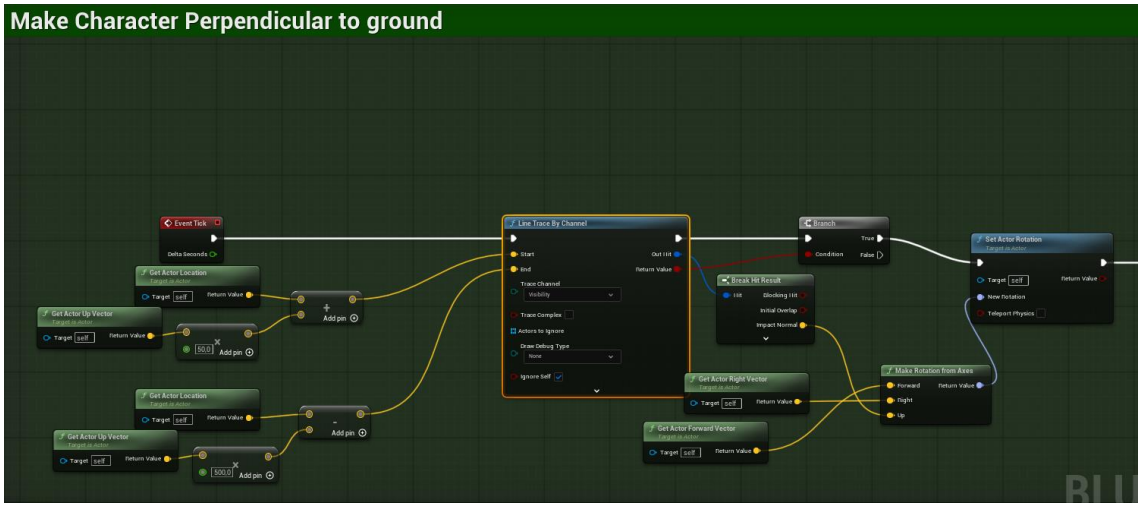


Figura 143: Función para que el jugador se oriente perpendicularmente al suelo.

El sistema actual permite:

- Idle
- Caminata
- Carrera
- Ataques ligeros
- Muerte
- Equipado / desenvainado

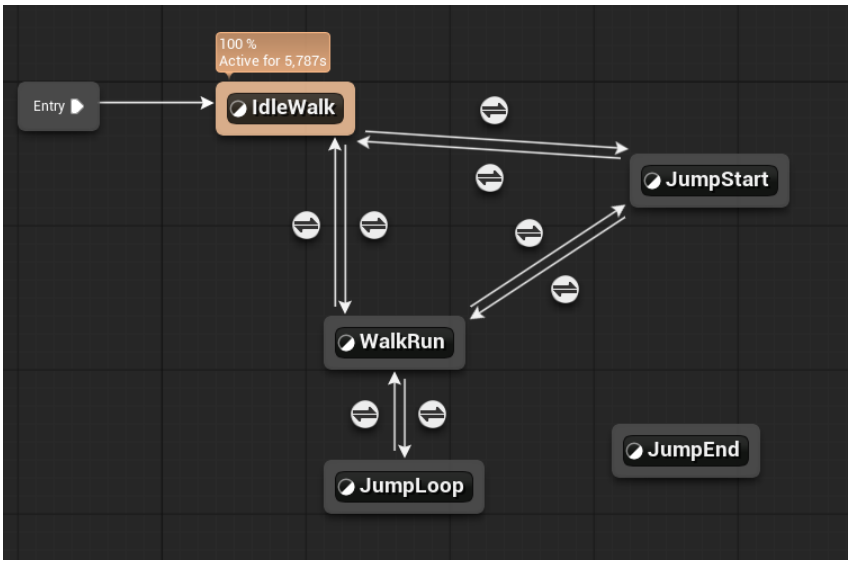


Figura 144: Locomoción ABP_MyCharacter.

Necesitamos añadir un Default Slot para poder utilizar el nodo “Play Anim Montage”

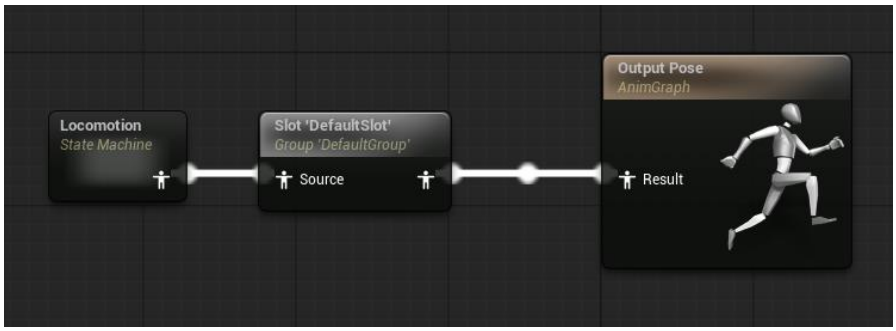


Figura 145: Grafo de animaciones en ABP_MyCharacter.

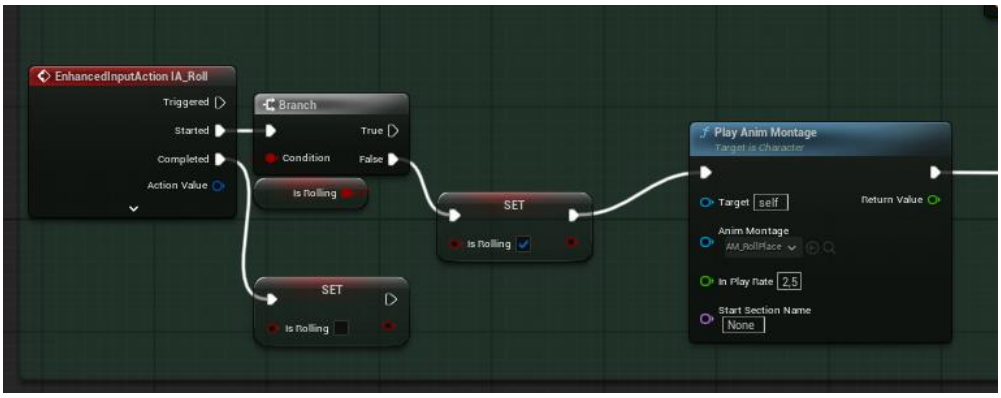


Figura 146**:** **E**jemplo de uso del nodo Play Anim Montage en el rodar.

Elementos clave

- **State Machine locomotora**
  Controla movimiento base.
- **Blend Spaces**
  Mezcla Idle / Walk / Run según velocidad.
- **Animation Montages**
  Usados para ataques con espada.
- **Slots de animación**
  Permiten superponer ataques al movimiento.

Figura 147**:** Animaciones principales del personaje.

4.1.3. Player Controller

El control del jugador se gestiona mediante inputs vinculados al Character Blueprint.

Tipos de entrada implementados:

- Movimiento: WASD
- Cámara: Click izquierdo
- Salto: Espacio
- Ataque: Click
- Equipar: Q o E
- Rodar: Ctrl izquierdo
- Correr: Mayus izquierdo
- Abrir inventario: I
- Abrir mapa: M
- Interactuar con objetos: R
- Interactuar con NPCs: F
- Lock-on: C

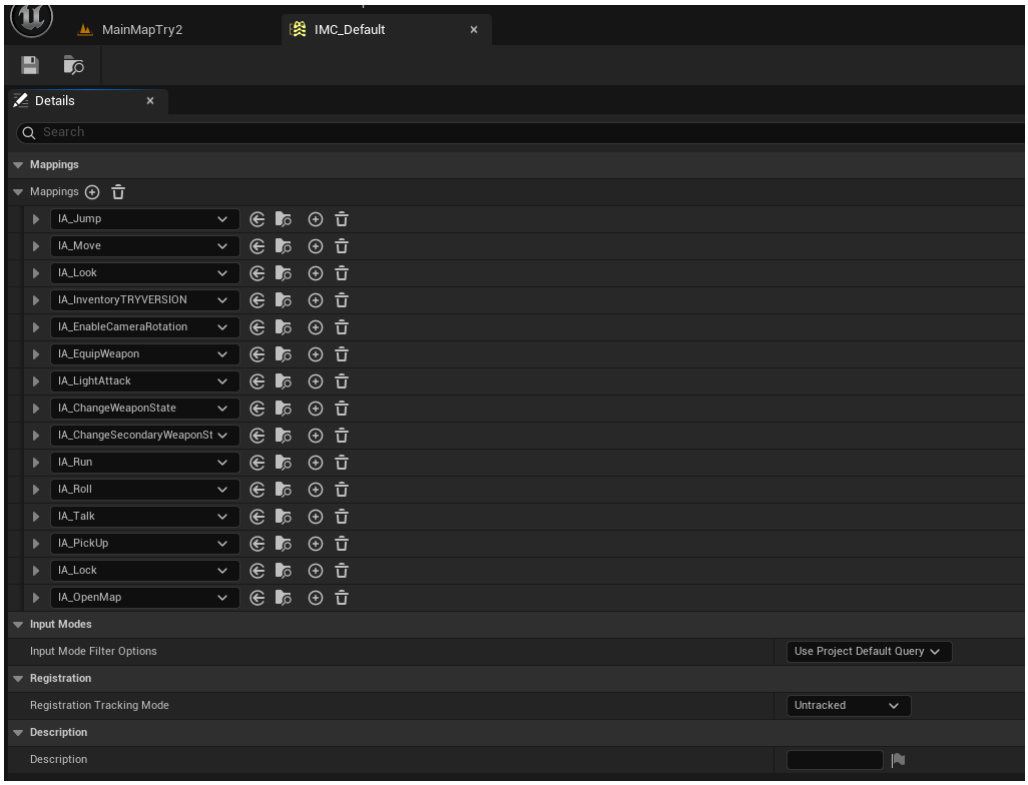


Figura 148**:** Input Mapping Context IMC_Default.

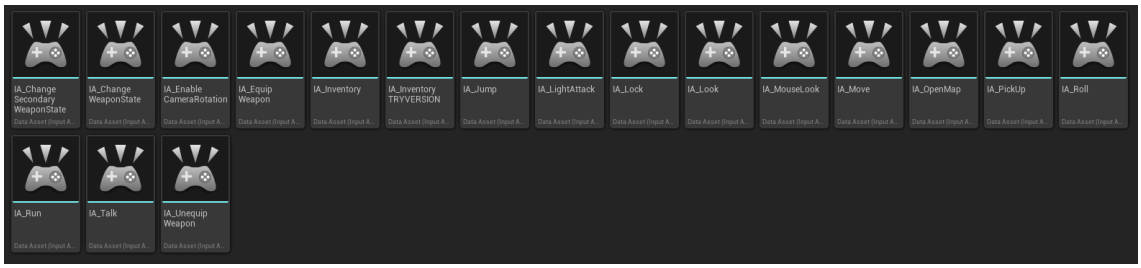


Figura 149: Todos los Input Actions.

En cuanto a la movilidad, el personaje dispone de una acción de rodar (*roll*), durante la cual se vuelve invulnerable a los ataques enemigos. Esta mecánica introduce una capa adicional de habilidad y timing en el combate. Además, se implementó un sistema de fijado de enemigos, que permite centrar la cámara en un objetivo concreto, facilitando el control en enfrentamientos directos.

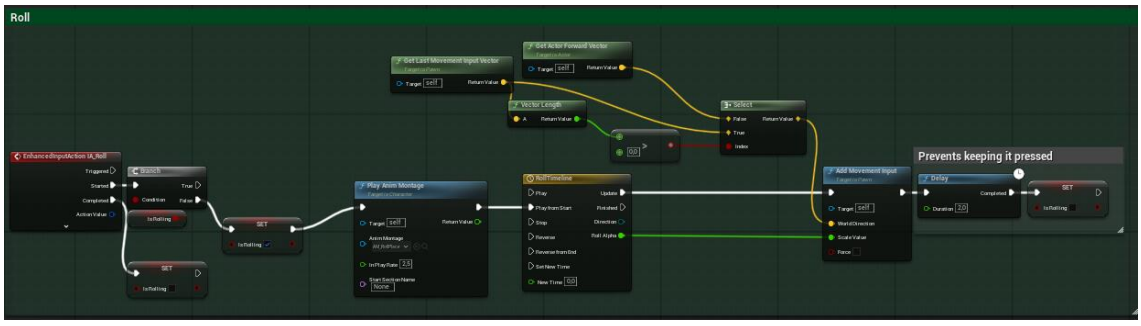


Figura 150: Lógica para rodar.

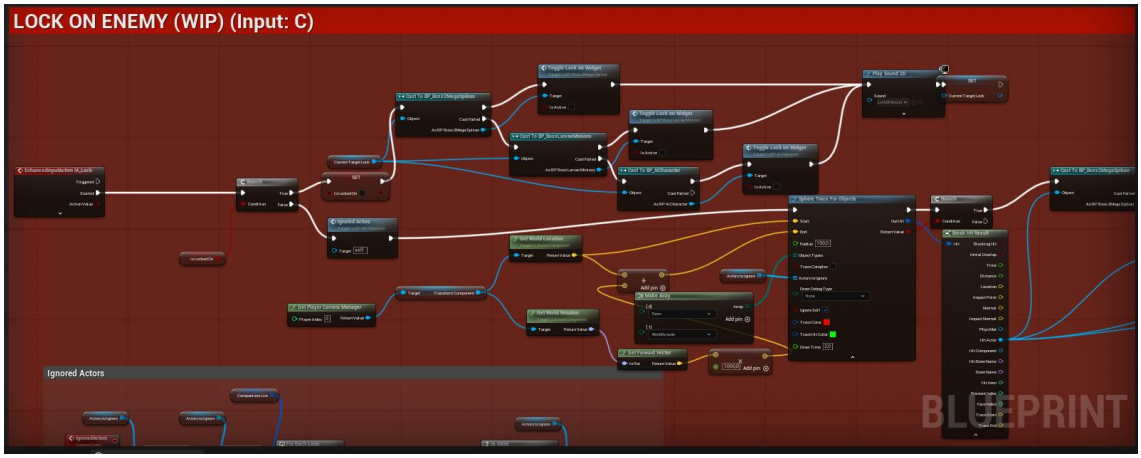


Figura 151: Lógica para fijar enemigo en BP_MyCharacter.

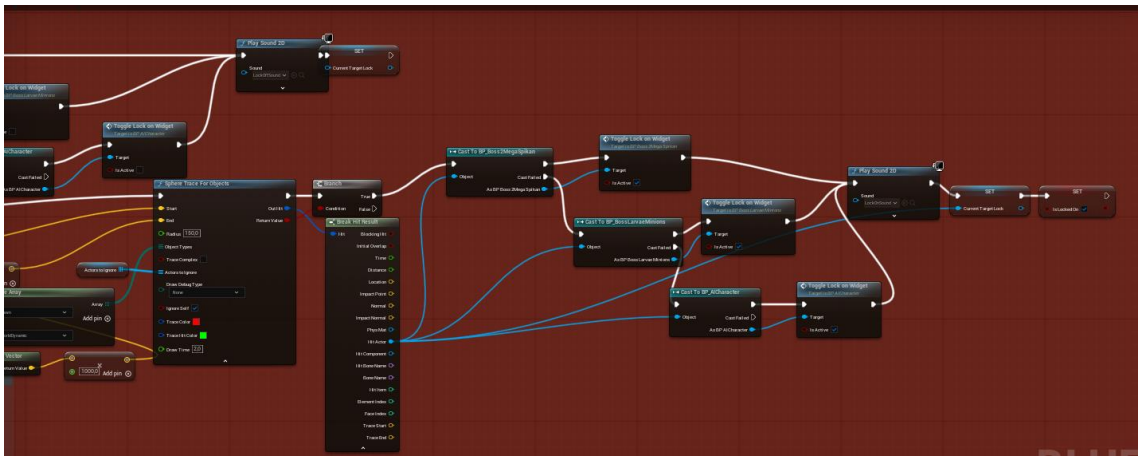


Figura 152: Lógica para fijar enemigo, enemigos específicos en BP_MyCharacter.

4.1.4. Estadísticas de combate

Las estadísticas de combate del personaje jugable se gestionan mediante un componente desacoplado denominado BPAC_CharacterStats (Blueprint Actor Component). Este enfoque modular permite centralizar toda la lógica relativa a atributos sin necesidad de modificar directamente el Blueprint base del personaje, favoreciendo la escalabilidad, reutilización de código y mantenimiento del sistema.

El componente es responsable de administrar la variación dinámica de las estadísticas principales del personaje, concretamente:

- **Vida Máxima**
- **Ataque**
- **Velocidad**

Dichas estadísticas pueden verse alteradas temporal o permanentemente mediante el uso de **pociones**, así como por efectos de **bendiciones** y **maldiciones** asociados al sistema de alineamiento y progresión narrativa.

Las estadísticas se almacenan mediante estructuras de tipo diccionario (Map), donde:

- **Clave**: Enumerado que define el tipo de estadística (E_StatType puede ser Max Health, Attack o Speed).
- **Valor**: Magnitud numérica asociada a la estadística.

Este sistema permite:
- Acceso eficiente a cada atributo.
- Extensibilidad para futuras estadísticas.
- Aplicación simultánea de múltiples modificadores.

Contiene 3 funciones:

Get Stats
Esta función permite recuperar el valor actual de una estadística específica a partir de su clave enumerada. Se utiliza de forma recurrente en sistemas de combate, interfaz y cálculos de daño para garantizar que siempre se empleen los valores actualizados tras la aplicación de modificadores.

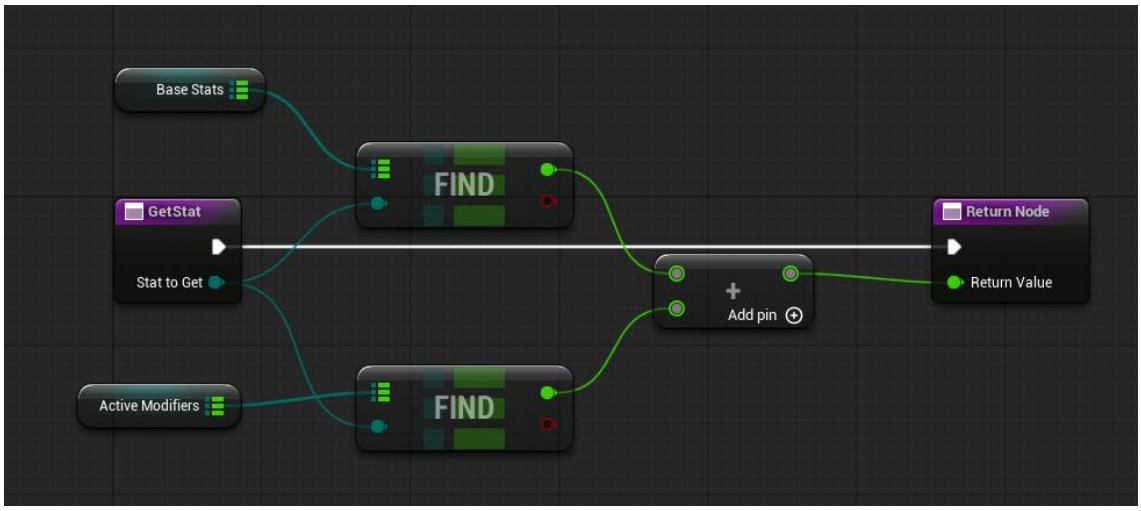


Figura 153: Función GetStat de BPAC_CharacterStats.

Apply Modifier
La función **ApplyModifier** se encarga de aplicar alteraciones sobre las estadísticas base del personaje. Dichos modificadores pueden proceder de consumibles, equipamiento o efectos narrativos.

El sistema suma el valor del modificador al atributo correspondiente, actualizando automáticamente el diccionario de estadísticas activas.
Este enfoque permite la coexistencia de múltiples efectos simultáneos, facilitando la creación de builds, estados alterados y sinergias entre sistemas.

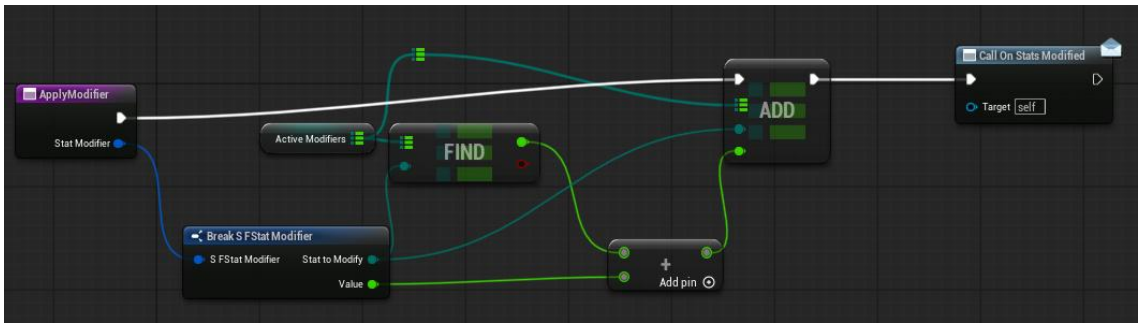


Figura 154: Función ApplyModifier de BPAC_CharacterStats.

Remove Modifier

La función **RemoveModifier** realiza el proceso inverso, eliminando los efectos previamente aplicados cuando:

- Expira la duración de una poción.
- Se disipa una bendición o maldición.
- Se desequipa un objeto.

El sistema recalcula el valor final de la estadística tras retirar el modificador, garantizando la consistencia de los atributos en todo momento.

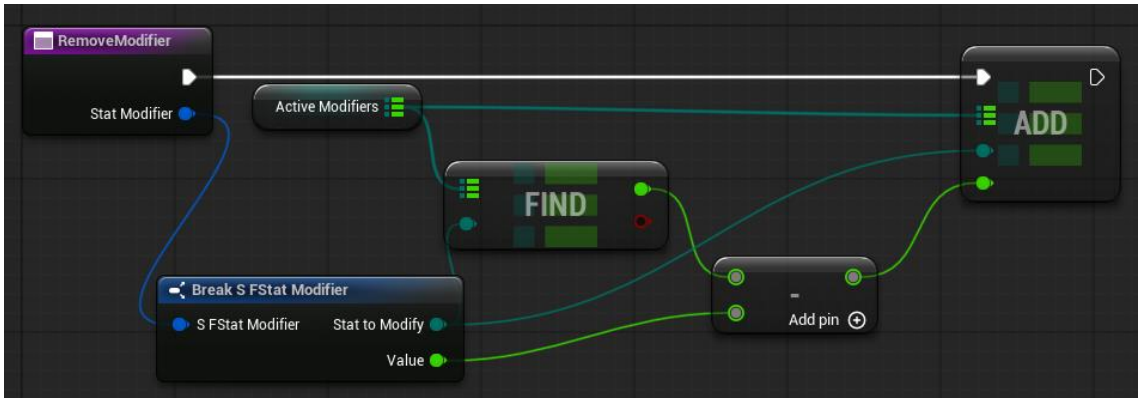


Figura 155: Función RemoveModifier de BPAC_CharacterStats.

4.1.5. Vida y salud

El sistema de vida del personaje jugable se estructura mediante un modelo híbrido que combina lógica interna de atributos con representación visual modular en la interfaz.

1. **Flujo de daño**
   El procesamiento del daño sigue un flujo secuencial claramente definido dentro de la arquitectura del personaje:
2. **Recepción de impacto**
   El sistema detecta la colisión o evento de ataque procedente de enemigos, proyectiles o áreas de efecto.
3. **Llamada a la función ApplyDamage**
   Se invoca la función encargada de gestionar el daño entrante, centralizando el cálculo y evitando modificaciones directas sobre la variable de salud.

4. **Reducción de Health**
   La función decrementa el valor actual de vida teniendo en cuenta posibles estados especiales, como la invulnerabilidad temporal durante acciones evasivas (por ejemplo, la esquiva o rodamiento).
5. **Comprobación de umbral ≤ 0**
   Tras aplicar el daño, el sistema verifica si la salud del personaje ha alcanzado o descendido por debajo de cero.
6. **Activación del estado de muerte**
   En caso afirmativo, se ejecuta la lógica de muerte del personaje, que incluye animación, bloqueo de control, reproducción de audio y posterior llamada al sistema de respawn.

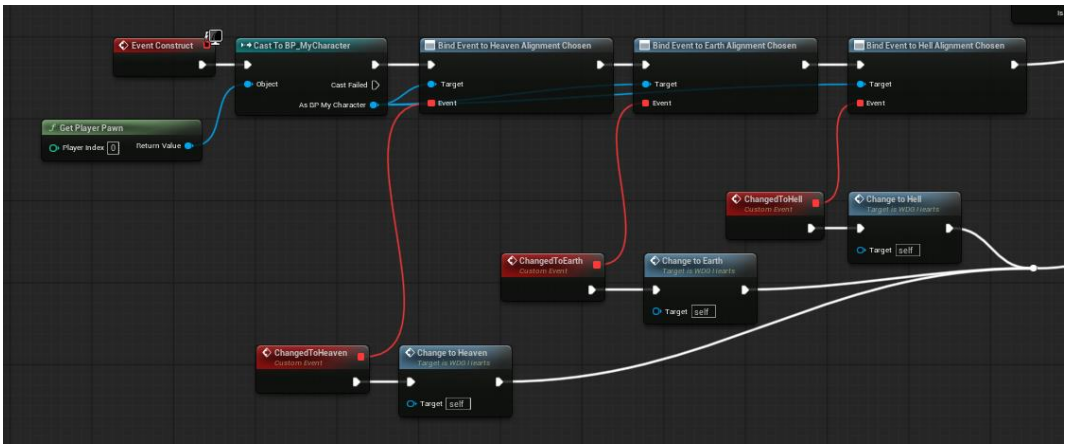


Figura 156: Vemos en los corazones individuales si deben ser cambiados de alineación.

Figura 157: Cada función pone sus texturas correspondientes.

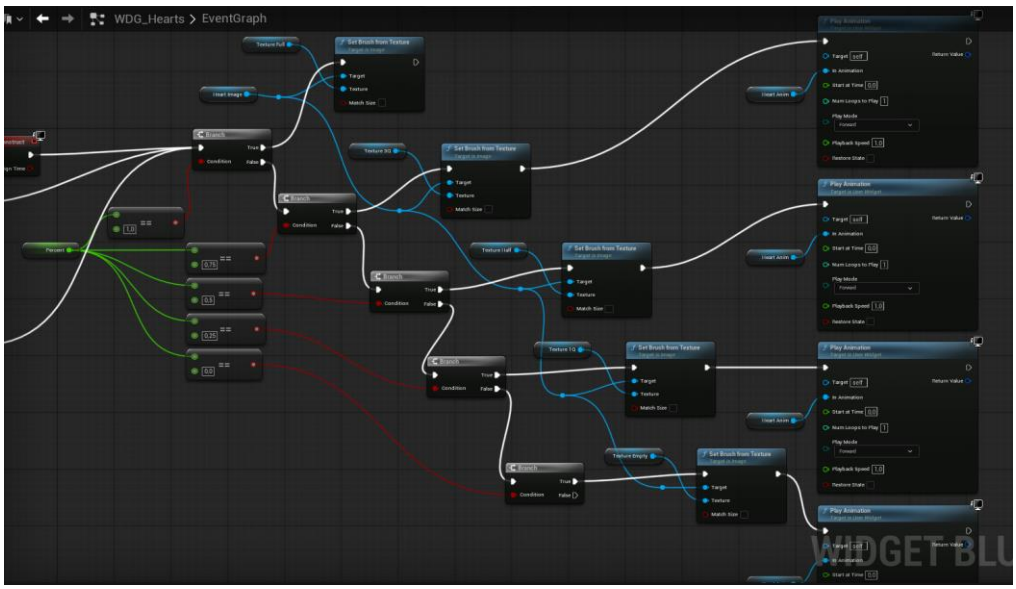


Figura 158: Comprueba la textura que debe tener cada corazón y en caso de que cambie hace una animación de parpadeo.

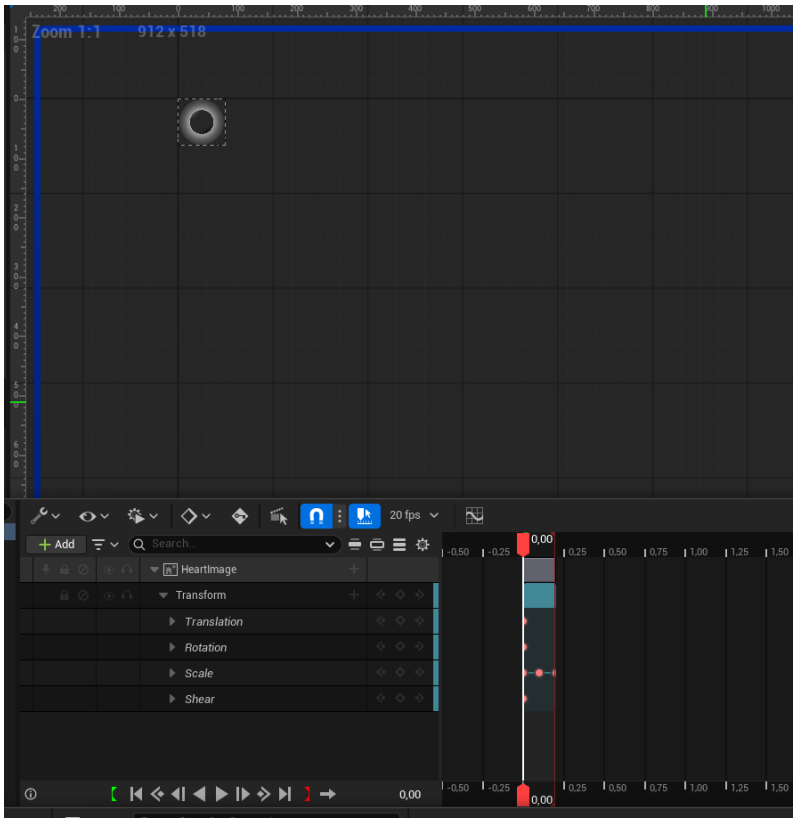


Figura 159: Animación del corazón en el diseñador.

La vida máxima base del personaje se ha establecido en un total de **seis corazones**.

Cada corazón representa una unidad de vida segmentada compuesta por **cuatro impactos** individuales, estableciendo así la siguiente equivalencia:
1 corazón = 4 golpes
6 corazones = 24 puntos de vida totales

Este sistema segmentado permite ofrecer un feedback visual más claro al jugador, facilitando la lectura del daño recibido y la planificación de combate. Esto puede cambiar con el uso de pociones, bendiciones, maldiciones o alineamientos.

Con el objetivo de garantizar la escalabilidad del sistema de salud, cada corazón ha sido implementado como un Widget Blueprint [17] independiente, en lugar de formar parte de una única barra estática.

Este enfoque permite:

- Añadir o eliminar corazones dinámicamente.
- Reflejar aumentos de vida máxima por bendiciones, objetos o progresión.
- Modificar visualmente corazones individuales (vacíos, medios, completos).

Los corazones individuales se agrupan dentro de un contenedor principal denominado HealthBar, también desarrollado mediante Widget Blueprint [17].

Este contenedor no posee un número fijo de elementos, sino que:

- Genera automáticamente la cantidad de corazones necesaria en función de la vida máxima actual del personaje.
- Se actualiza cuando se aplican modificadores de estadísticas que alteran la salud máxima.
- Reorganiza visualmente los corazones para mantener la coherencia de la interfaz.

Este sistema dinámico permite integrar de forma directa las mecánicas de progresión y alineamiento con la representación visual de la vida, reforzando la claridad informativa sin comprometer la inmersión. Fue realizado basándose en un video de *NirnaethGameDev* [27].

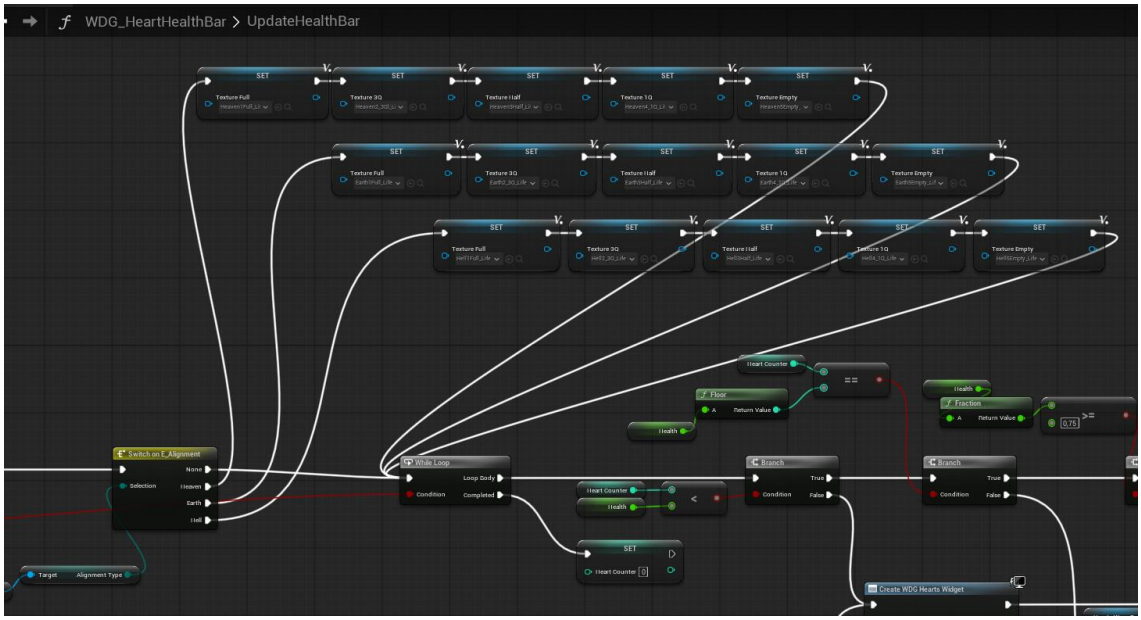


Figura 160: Asegura tener la textura correcta dependiendo de la alineación.

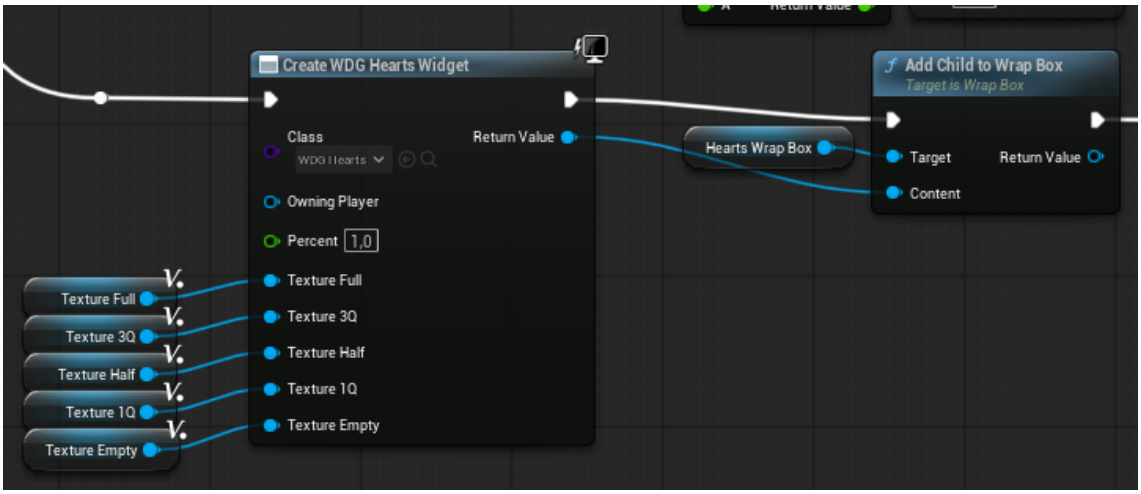


Figura 161: Para cada combinación (1, 0.75, 0.5, 0.25 o 0) de vida crea el corazón que necesita.

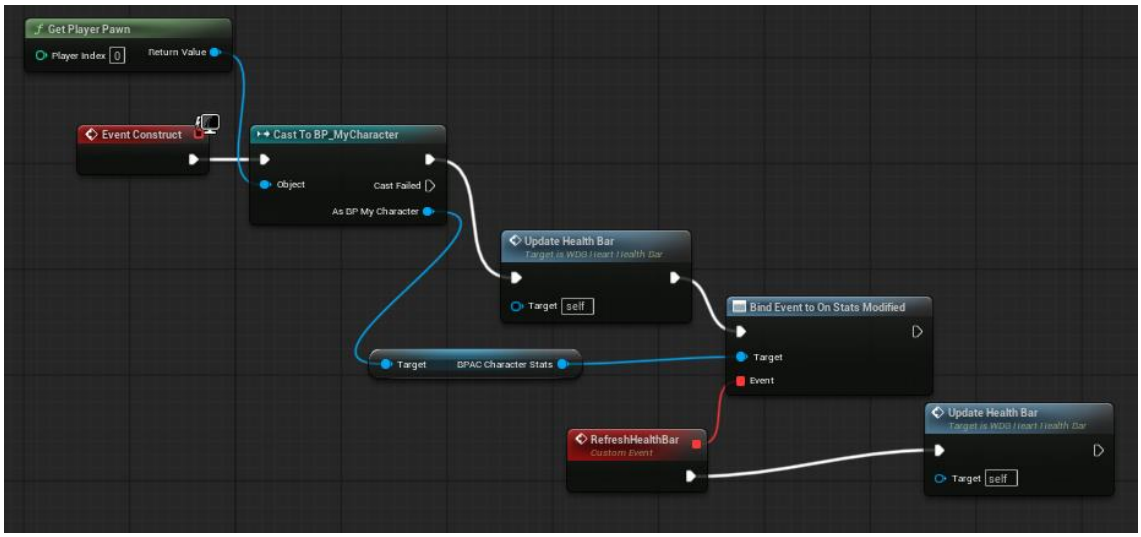


Figura 162: Evento inicial, cuando cambian las estadísticas aseguramos que se actualice la vida.

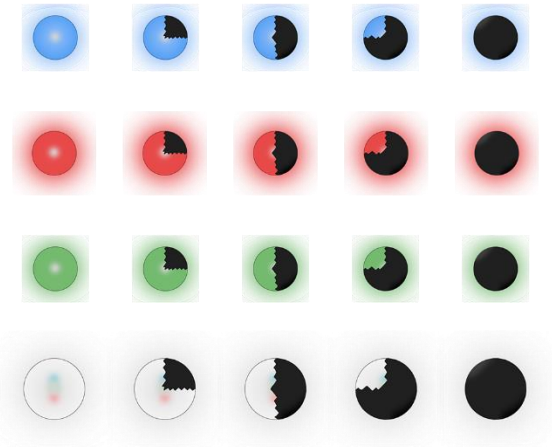


Figura 163: Todas las vidas.

4.2. Objetos

Como se ha presentado previamente en la **Tabla 5**, el sistema de objetos del proyecto se compone de diversas categorías y tipologías. A continuación, se procede a su análisis en mayor profundidad, abordando tanto sus características funcionales como su implementación técnica dentro del sistema de juego.

4.2.1. Espadas

El combate cuerpo a cuerpo se basa en el uso de espadas, las cuales aplican un multiplicador sobre el daño base del personaje. Las espadas disponibles en el juego son:

- **Moonsword:** espada inicial, con un multiplicador de daño ×1.
- **Dunbrae Sword:** multiplicador de daño ×1.2.
- **Light Wraith:** multiplicador de daño ×1.5.
- **Elden Bark:** multiplicador de daño ×1.8.

- **Rogue Crystal:** multiplicador de daño ×2.2.
- **Abyzakar Justice:** multiplicador de daño ×2.6.

Asimismo, todas ellas derivan de una clase base de tipo espada que concentra la lógica común del sistema, permitiendo la reutilización de funcionalidades y una mayor coherencia en el diseño.

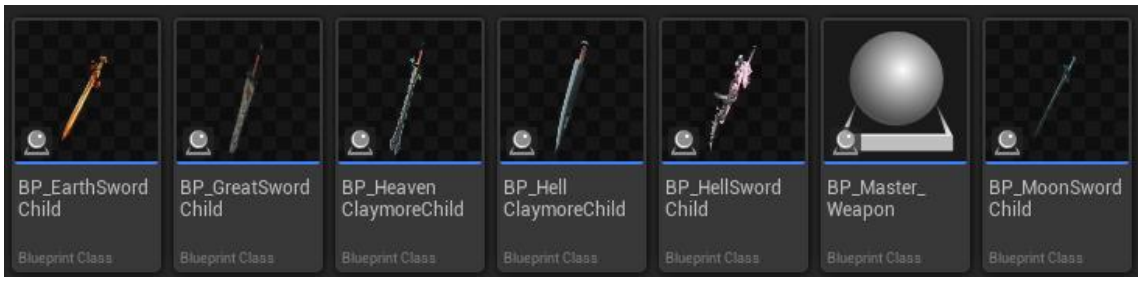


Figura 164: Tipos de armas.

Se planteó inicialmente la incorporación de efectos especiales para todas las espadas pesadas, aunque por limitaciones de tiempo solo se implementó en *Abyzakar Justice*. Esta espada aplica un efecto de quemadura que reduce la vida del enemigo en un punto durante diez *ticks* consecutivos.

Los efectos propuestos para otras armas incluían la ralentización de enemigos en un 30 % durante tres segundos en el caso de *Light Wraith* (reducido al 10 % frente a jefes) y un aumento del 40 % del daño cada cuatro golpes consecutivos para *Elden Bark*.

El sistema permite al jugador llevar equipadas dos espadas simultáneamente, aunque solo puede utilizar una a la vez. El cambio entre armas se realiza de forma dinámica durante el combate, sin necesidad de acceder al inventario, evitando interrupciones en la experiencia de juego.

Desde el punto de vista técnico, la lógica de aplicación de daño se basa en una caja de colisión asociada al arma, la cual detecta la colisión con el enemigo y emite un mensaje notificando el impacto. A partir de esta comunicación, el sistema de combate del enemigo procesa la reducción de vida correspondiente y, en el caso de que el arma equipada disponga de efectos adicionales, como la quemadura de *Abyzakar Justice*, se aplican de forma complementaria.

Adicionalmente, se contempló un sistema en el que cada espada estuviese asociada a un tipo de alineación, infligiendo mayor o menor daño en función del tipo de enemigo. No obstante, esta funcionalidad no llegó a implementarse en el prototipo final.

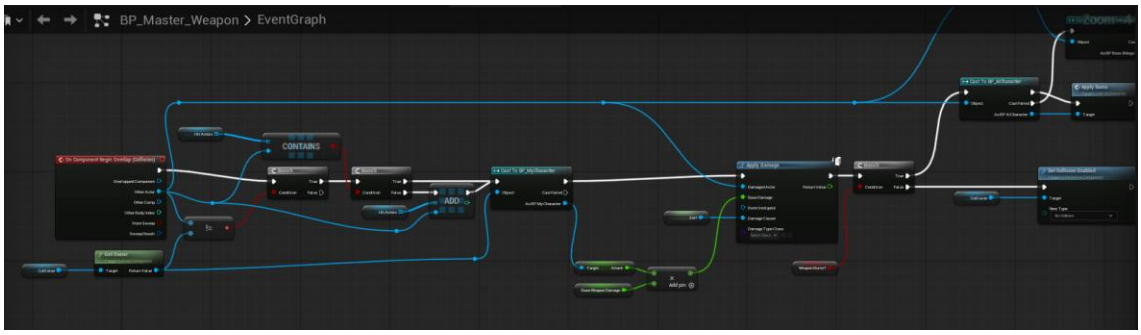


Figura 165: Manejo de colisión en BP_Master_Weapon.

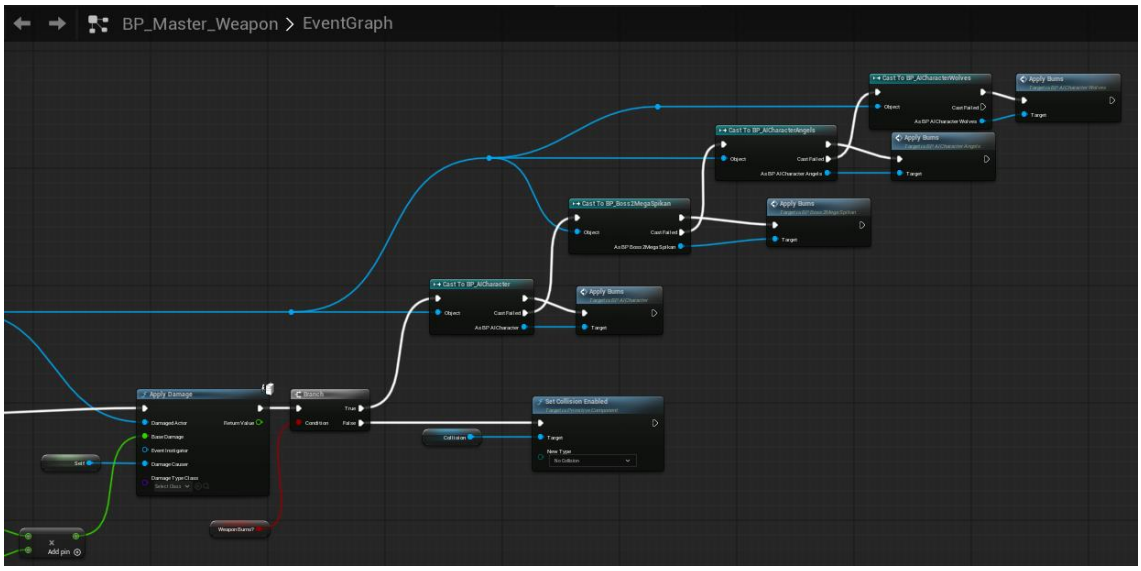


Figura 166: Manejo de colisión cotra enemigos en BP_Master_Weapon.

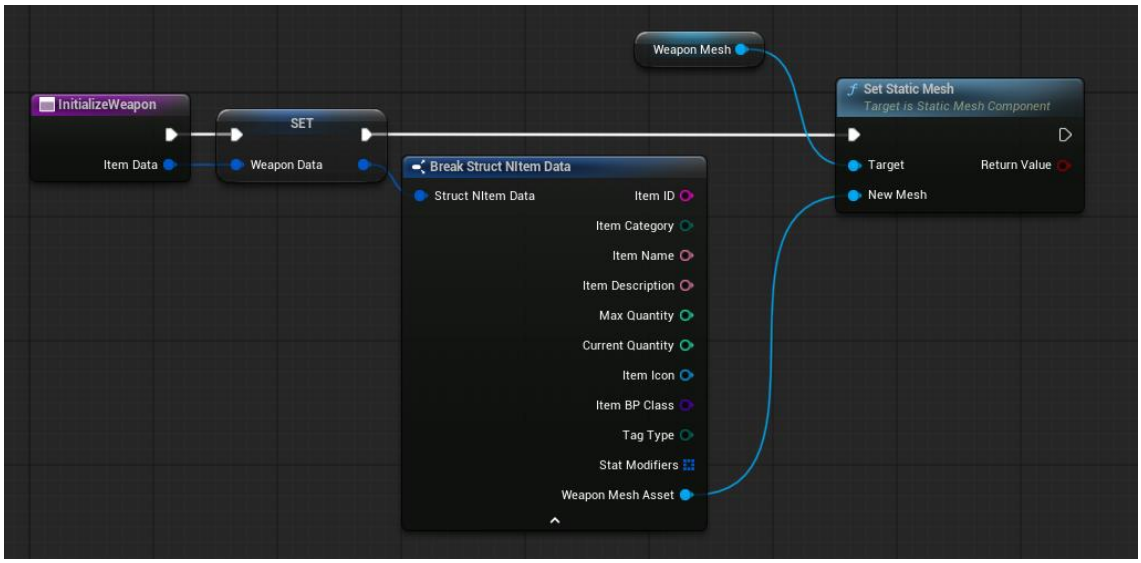


Figura 167: Inicialización de arma en BP_Master_Weapon.

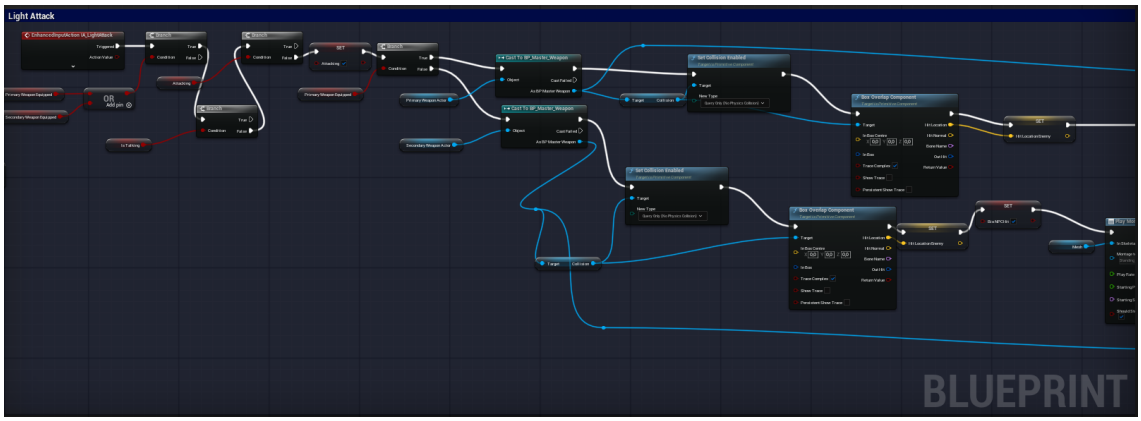


Figura 168: Lógica de ataque en BP_MyCharacter.

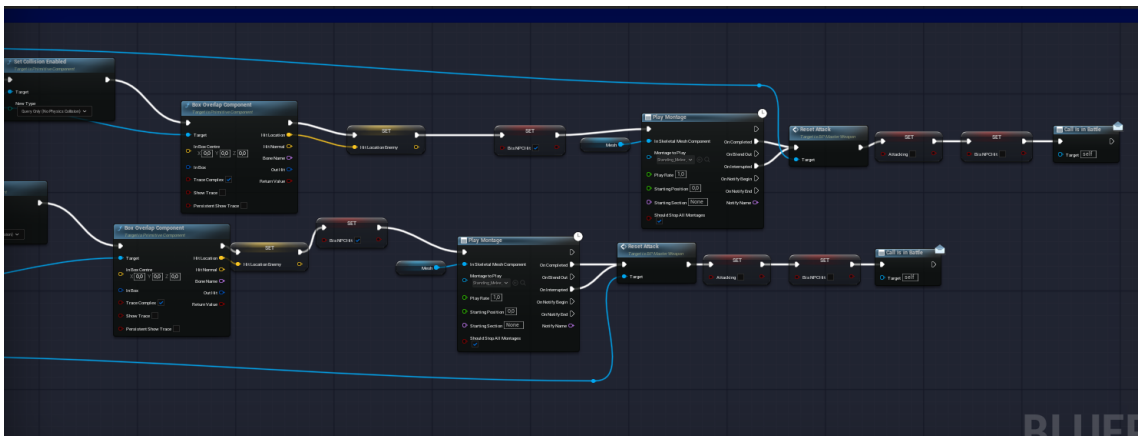


Figura 169: Lógica de animación de ataque en BP_MyCharacter.

4.2.2. Pociones

Las pociones constituyen un sistema de consumibles que permiten alterar temporal o permanentemente las estadísticas del personaje, además de introducir efectos adicionales en algunos casos. Las pociones disponibles en el juego son las siguientes:

- **Poción de recuperación (Jade Brew):** restaura hasta un máximo de dos corazones de vida.
- **Poción de vida (Lapis Elixir):** añade un corazón adicional a la vida máxima durante un periodo de cinco minutos.
- **Poción de salto (Amethyst Brew):** permite al personaje realizar un salto adicional, habilitando un triple salto en lugar del doble salto estándar.
- **Poción de daño (Agate Drops):** incrementa el ataque base en 20 puntos durante cinco minutos.
- **Poción de velocidad (Sunstone Tincture):** aumenta la velocidad base del personaje en 500 unidades durante cinco minutos.
- **Poción dudosa (Obsidian Phial):** reduce la vida actual en dos corazones y, simultáneamente, incrementa el ataque base en 5 puntos durante tres minutos, sin que el jugador conozca inicialmente sus efectos negativos.

Este último tipo de poción introduce un elemento de riesgo e incertidumbre, reforzando la toma de decisiones del jugador.

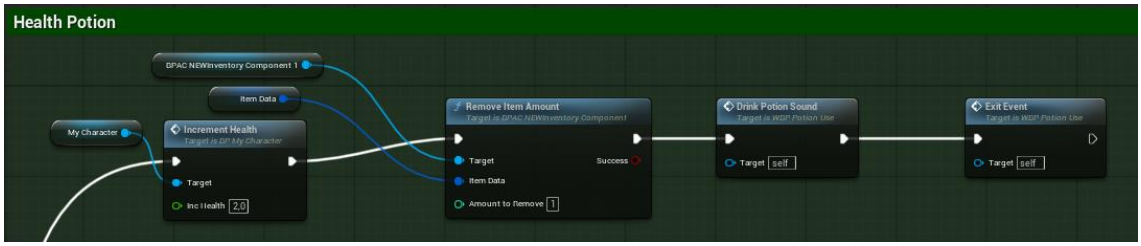


Figura 170**:** Implementación de **Jade Brew**.

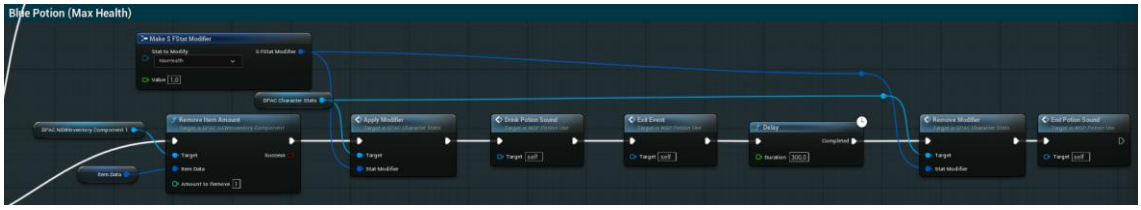


Figura 171**:** Implementación de **Lapis Elixir**.

Figura 172**:** Implementación de **Amethyst Brew**.

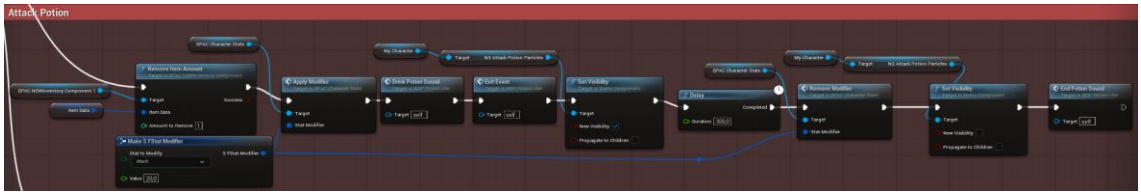


Figura 173**:** Implementación de **Agate Drops**.

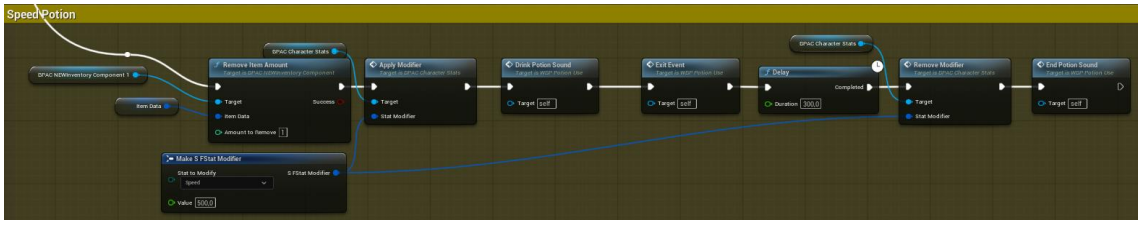


Figura 174**:** Implementación de **Sunstone Tincture**.

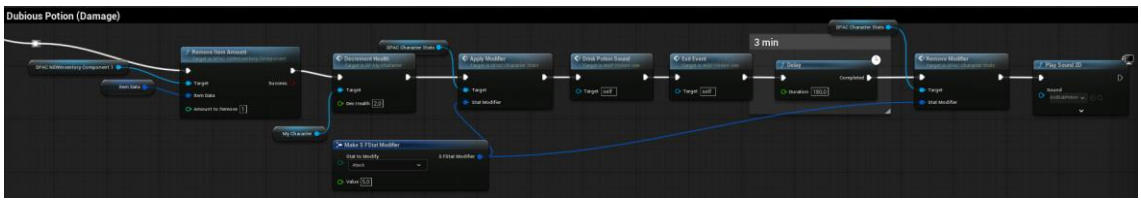


Figura 175**:** Implementación de **Obsidian Phial**.

4.2.3. Maldiciones y bendiciones

El sistema de etiquetas, que incluye bendiciones y maldiciones, es un mecanismo de modificación de estadística. Estas alteraciones suelen tener efectos más duraderos y están diseñadas para fomentar intercambios entre estadísticas, incentivando al jugador a asumir consecuencias positivas y negativas. Estas son objenidas a través de NPCs repartidos por el mundo o comprándolas en las tiendas como se verá más delante.

Las bendiciones disponibles son:

- **Bendición de ataque (Ares Fury):** incrementa el ataque base en 20 puntos.
- **Bendición de vida (Aegis Protection):** añade un corazón adicional a la vida máxima.
- **Bendición de velocidad (Hermes Smoke):** incrementa la velocidad base en 200 unidades.

Las maldiciones implementadas son:

- **Maldición de ataque (Marchosias Curse):** reduce el ataque base en 10 puntos, pero incrementa la velocidad de movimiento en 100 unidades.
- **Maldición de vida (Malphas Fortress):** elimina un corazón de vida máxima, a cambio de aumentar el ataque base en 5 puntos.
- **Maldición de velocidad (Buer Oath):** reduce la velocidad base en 300 unidades, pero añade un corazón adicional a la vida máxima.

Las maldiciones han sido diseñadas para no ser completamente negativas, incentivando al jugador a aceptar intercambios estratégicos. La idea original del sistema contemplaba que estas maldiciones no pudieran eliminarse de forma inmediata, obligando al jugador a localizar un NPC específico y pagar una cantidad determinada para su eliminación.

Asimismo, se planteó que la maldición recibida fuese aleatoria, con el objetivo de aumentar la rejugabilidad.

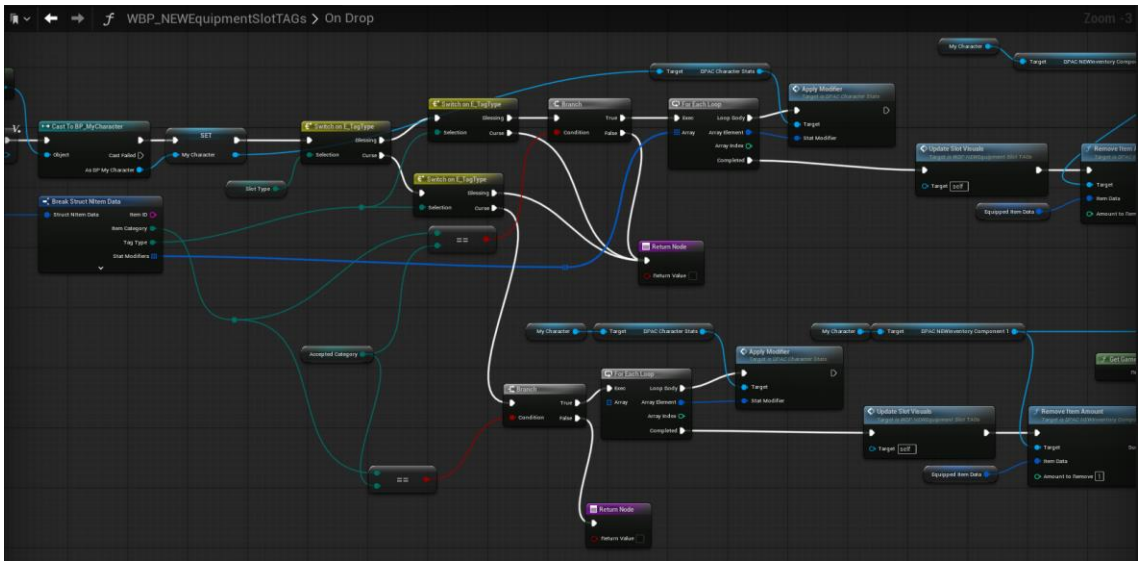


Figura 176: Sistema de equipamiento de bendiciones y maldiciones.

Figura 177: Personajes que dan las maldiciones (derecha) y bendiciones (izquierda).

Las funcionalidades de cada una de las etiquetas proviene de una Tabla de Datos.

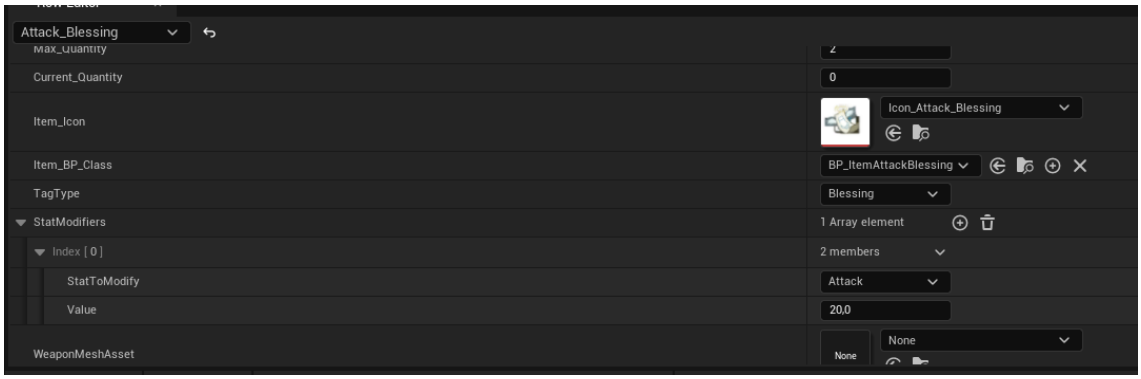


Figura 178: Ejemplo de etiqueta en Tabla de Datos.

4.2.4. Miscelánea

Dentro de la arquitectura interna del sistema de inventario y recolección, se ha establecido una categorización funcional unificada en la que la mayoría de objetos recolectables del mundo son denominados genéricamente como “plantas” a nivel de implementación. Esta nomenclatura no responde necesariamente a su naturaleza visual o narrativa, sino a criterios de diseño sistémico orientados a simplificar la gestión de ítems.

De este modo, bajo esta categoría se agrupan elementos heterogéneos que comparten una misma finalidad mecánica: su utilización en sistemas de creación de pociones, los cuales serán desarrollados en apartados posteriores o su importancia a lo largo de la historia.

Todas las entidades pertenecientes a esta categoría derivan de una **clase maestra común**, encargada de centralizar la lógica base de comportamiento de los objetos en el mundo. Este enfoque basado en herencia permite:

- Reutilización de funcionalidades.
- Reducción de redundancias en Blueprints hijos.
- Escalabilidad para la incorporación de nuevos ítems.

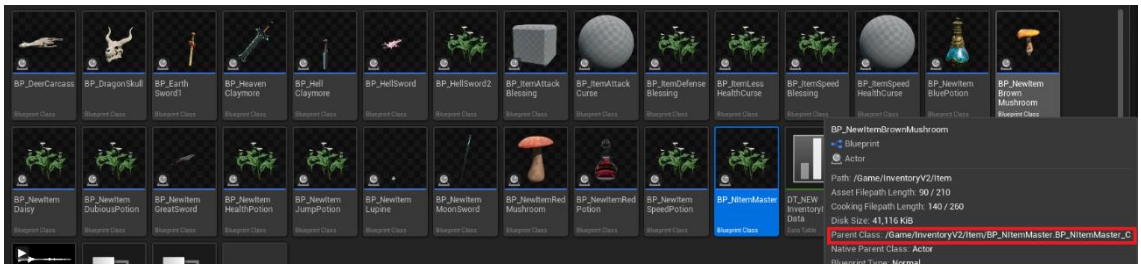


Figura 179: Clase maestra y sus hijos.

Cada objeto específico actúa como hijo de esta clase, heredando sus funciones principales y personalizando únicamente variables como modelo, icono, nombre o efectos.

El caso de las espadas constituye la principal divergencia dentro de esta jerarquía de clases. Su implementación responde a una doble naturaleza sistémica:

1. **Espadas como objeto recolectable**
   Funcionan como cualquier otro ítem del mundo, pudiendo ser recogidas, almacenadas o intercambiadas.
2. **Espadas como objeto equipado**
   Al ser equipadas, pasan a depender de una clase distinta orientada a la lógica de combate, sockets de animación, daño y efectos asociados.

Por ello, las espadas poseen **dos clases padre diferenciadas** según su estado:

- Clase de objeto (inventario / mundo).
- Clase de arma equipada (combate / animación).

Esta separación permite desacoplar la lógica de interacción del suelo de la lógica de combate, optimizando la organización del sistema.

La clase base de objetos recolectables integra diversas funciones esenciales para su comportamiento en el entorno:

Función de recogida (Pickup Function)

Gestiona la interacción entre el jugador y el objeto cuando este se encuentra en el mundo. Al activarse:

- Añade el ítem al inventario.
- Reproduce efectos visuales o sonoros.
- Elimina temporalmente el objeto del escenario.

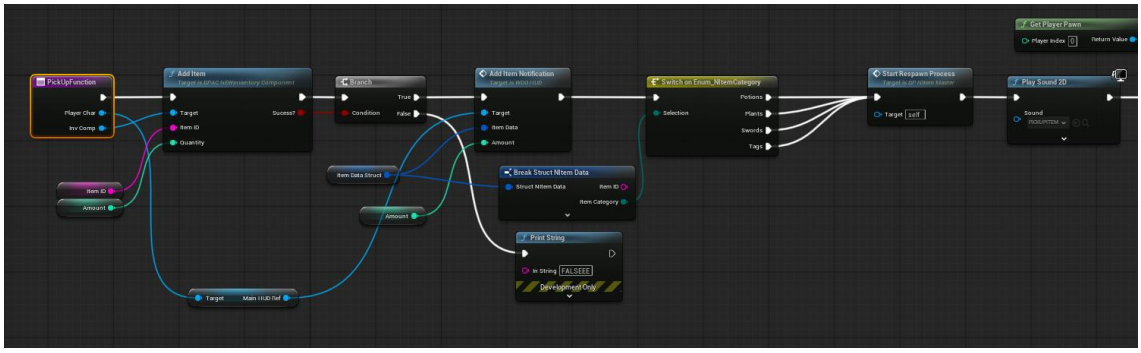


Figura 180: PickUpFunction en la clase maestra de objetos.

Orientación hacia el jugador

Para mejorar la legibilidad visual e interacción, los objetos recargan su orientación de forma constante para mirar hacia el personaje jugable. Este comportamiento tipo *billboard* facilita su identificación dentro del entorno, especialmente en zonas de alta densidad ambiental.

Desde el punto de vista técnico, esta actualización de rotación no se ejecuta mediante un evento Tick continuo. En su lugar, se ha implementado un temporizador con una frecuencia de 0,5 segundos.

La adopción de este enfoque responde a criterios de optimización del rendimiento, ya que evita cálculos innecesarios en cada frame, reduciendo la carga de procesamiento cuando existen múltiples objetos activos simultáneamente en el mundo.

Asimismo, se ha considerado el factor perceptivo: la diferencia entre una actualización por frame y una actualización semestral mediante temporizador resulta prácticamente imperceptible para el ojo humano en este contexto de uso, manteniendo la eficacia visual del sistema sin comprometer la eficiencia técnica.

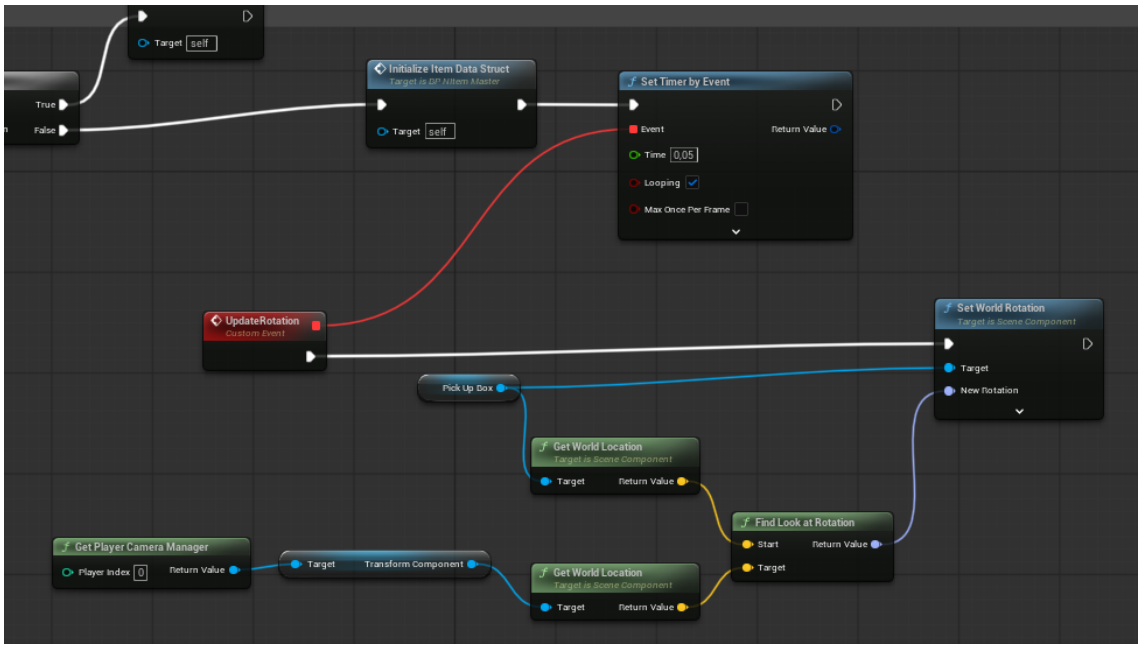


Figura 181: Rotación hacia el jugador cada 0.5 segundos.

Sistema de respawn

Con el fin de mantener la persistencia de recursos en el mundo abierto, el objeto maestro incorpora un sistema de reaparición automática.

Tras ser recogido:

- Se inicia un temporizador interno.
- El objeto vuelve a instanciarse en su localización original.
- El tiempo establecido de respawn es de 20 minutos.

Este sistema garantiza la renovabilidad de materiales necesarios para crafteo sin comprometer la exploración.

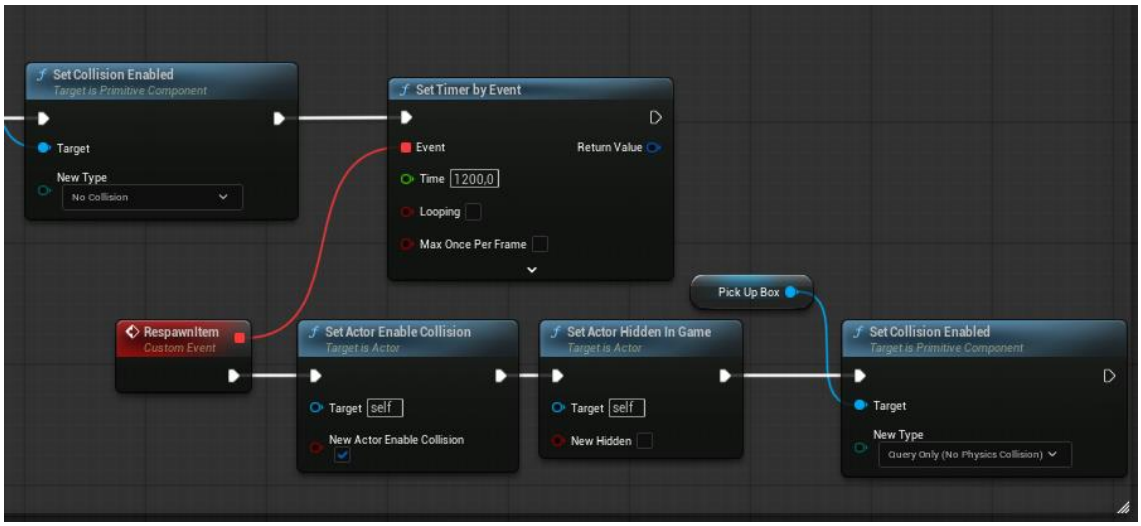


Figura 182: El objeto reaparece tras 1200 segundos (20 minutos).

4.3. Enemigos

La arquitectura del sistema de combate implementado en Order Frame se fundamenta en una estricta jerarquía de Actor Blueprints, cuya finalidad primordial es garantizar la modularidad del código y la consecuente optimización de su reutilización.

Dicha aproximación metodológica se refuerza mediante la integración de las entidades de generación de enemigos (Enemy Spawners). Estos componentes operativos exhiben la capacidad de instanciar de manera dinámica cualquier Blueprint [8] de enemigo disponible, lo que contribuye a la flexibilidad estructural del sistema y a la simplificación de la gestión del poblamiento del nivel.

A nivel funcional, la clasificación de las entidades enemigas se articula en dos categorías diferenciadas. Por un lado, se encuentran las unidades básicas o entidades estándar, caracterizadas por un comportamiento predefinido de patrulla. Por otro lado, se contempla un encuentro de elevada complejidad algorítmica y técnica, concretado en la implementación del antagonista final (jefe final), el cual representa el desafío más complejo dentro del ecosistema de combate.

4.3.1. Enemigos Normales

La implementación de las unidades enemigas de baja complejidad se sustenta en el principio de reutilización de la lógica operativa, permitiendo así una rápida expansión del catálogo de adversarios con una mínima sobrecarga de desarrollo. Dentro de esta categoría se identifican tres entidades primarias:

Lizard
Esta entidad constituye el arquetipo de unidad enemiga fundamental o base dentro del ecosistema de combate. Se encuentra conceptualmente asociada al bioma del Infierno, y por tanto es el encargado de que el jugador gane la moneda *Nors*. La lógica de movimiento y agresión de esta entidad sirve como plantilla funcional para las unidades subsiguientes.

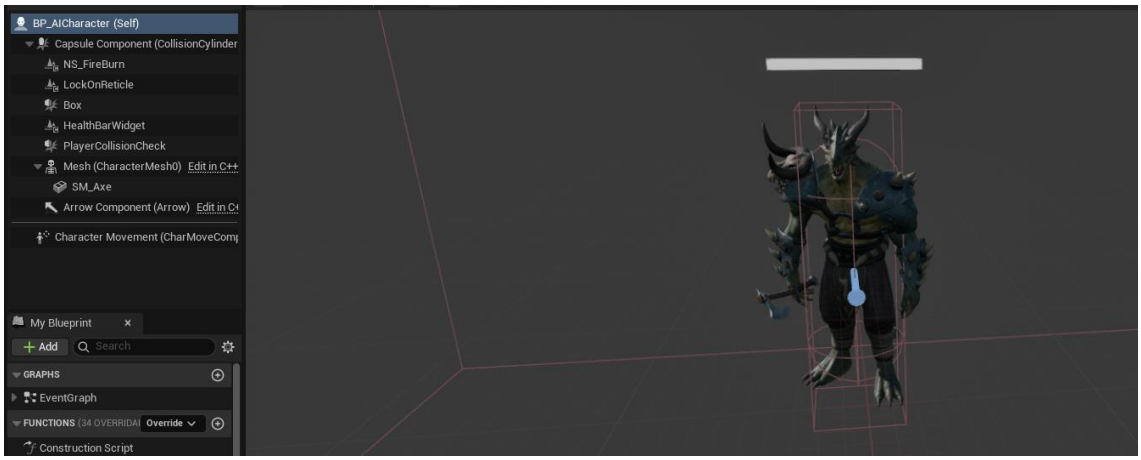


Figura 183: Enemigo Lizard Modelo.

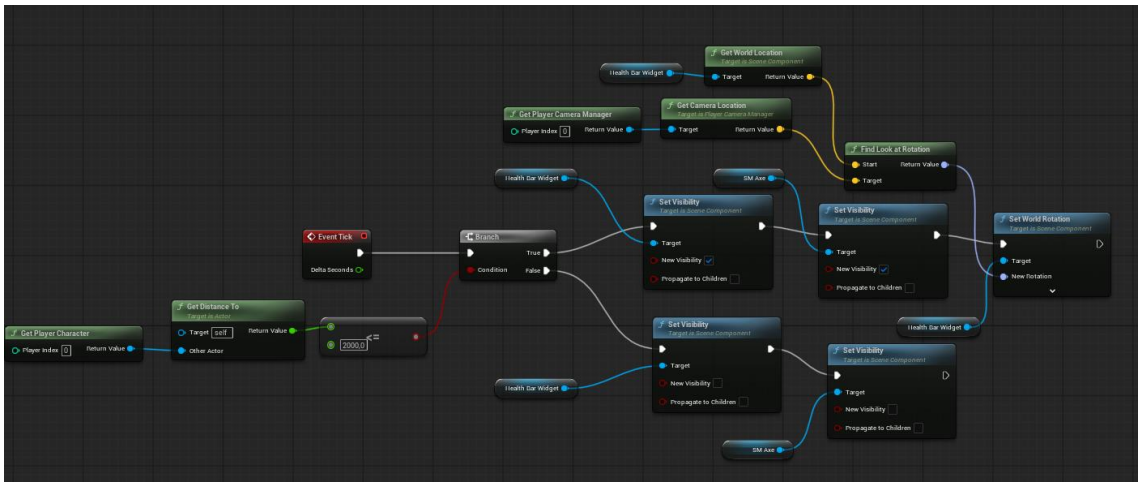


Figura 184: Evento tick para que la barra de vida siempre mire al jugador.

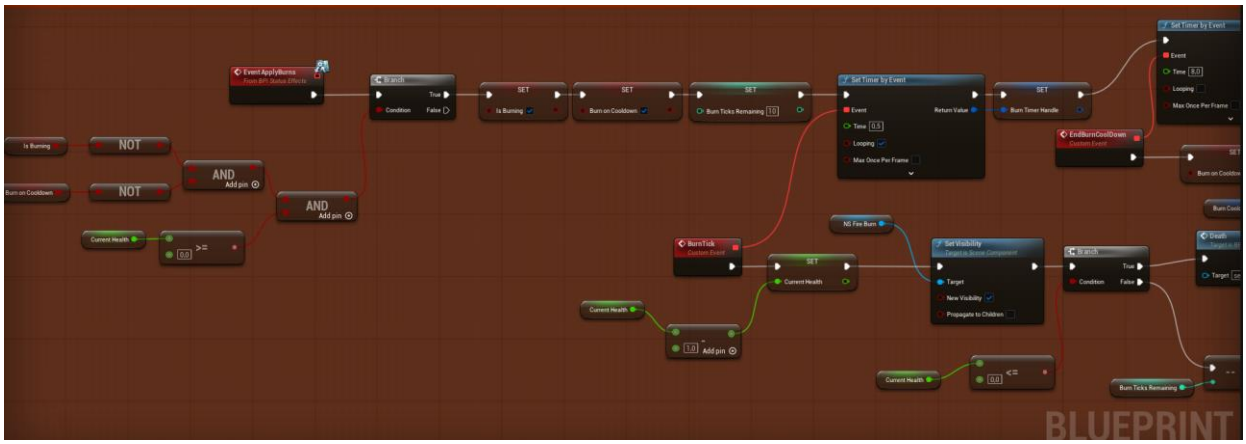


Figura 185: Evento para aplicar quemaduras al enemigo.

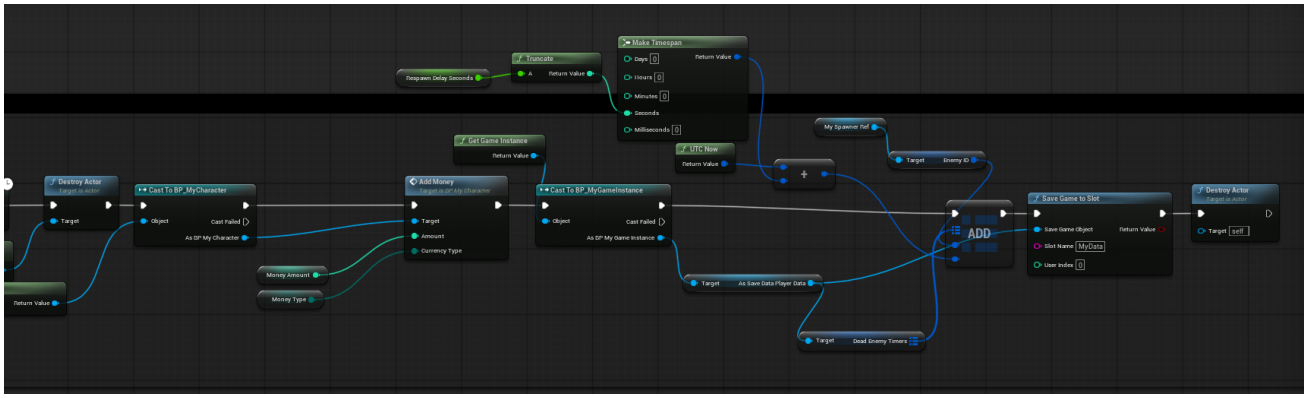


Figura 186: Evento de muerte para el enemigo que se encarga de hacerlo reaparecer pasado el tiempo.

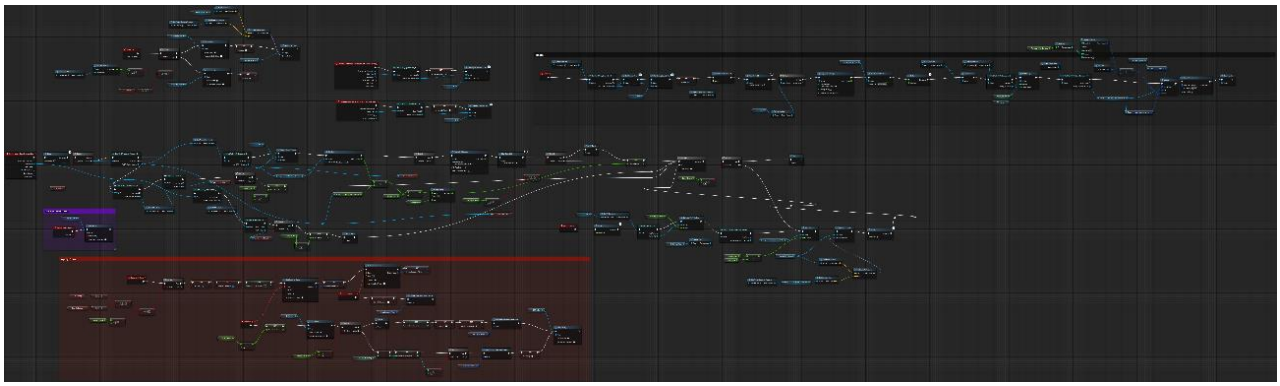


Figura 187**:** Blueprint [8] completo del enemigo Lizard.

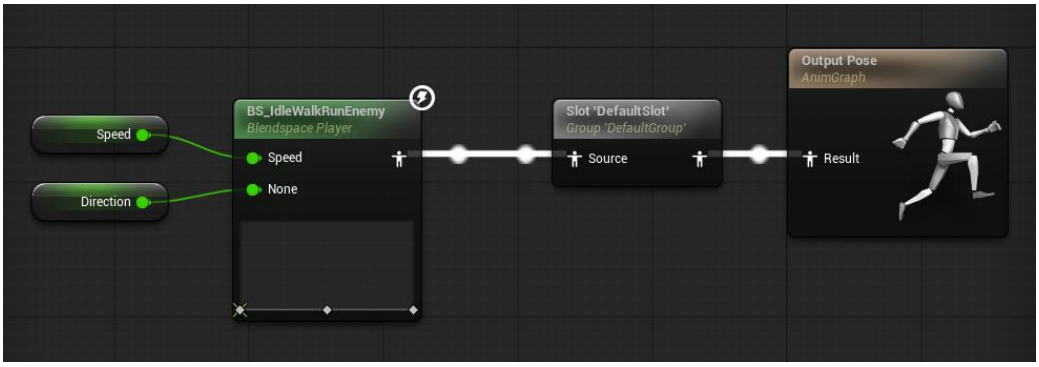


Figura 188**:** Enemigo Lizard Blueprint de animación.

Wolf

Vinculada temáticamente al bioma de la Tierra, esta entidad se localiza predominantemente en entornos boscosos, donde opera como un obstáculo de posición durante las secuencias de exploración. Es el encargado de que el jugador gane la moneda *Alas.* Es relevante destacar que la entidad 'Wolf' hereda y aplica la misma lógica operacional que la entidad 'Lizard'. Tanto el *Animation Blueprint* como el *Blueprint [8]* de lógica del lobo son idénticos a los del enemigo previo, sólo diferenciándose en las variables utilizadas, el modelo, las animaciones y los sonidos que hacen.

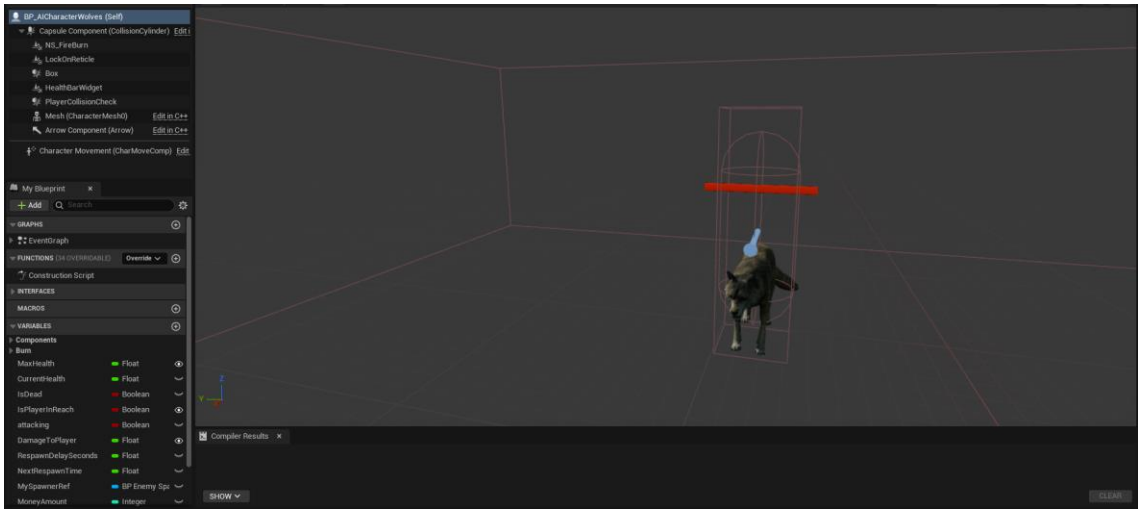


Figura 189**:** Enemigo Wolf Modelo.

Esqueleto

Esta unidad está asociada al entorno temático del Cielo y su introducción se reserva para la segunda mitad del desarrollo del juego. Es el encargado de que el jugador gane la moneda *Hals.* Al igual que la entidad 'Wolf', el 'Esqueleto' se adhiere rigurosamente a la lógica de funcionamiento preestablecida por el enemigo base 'Lizard'.

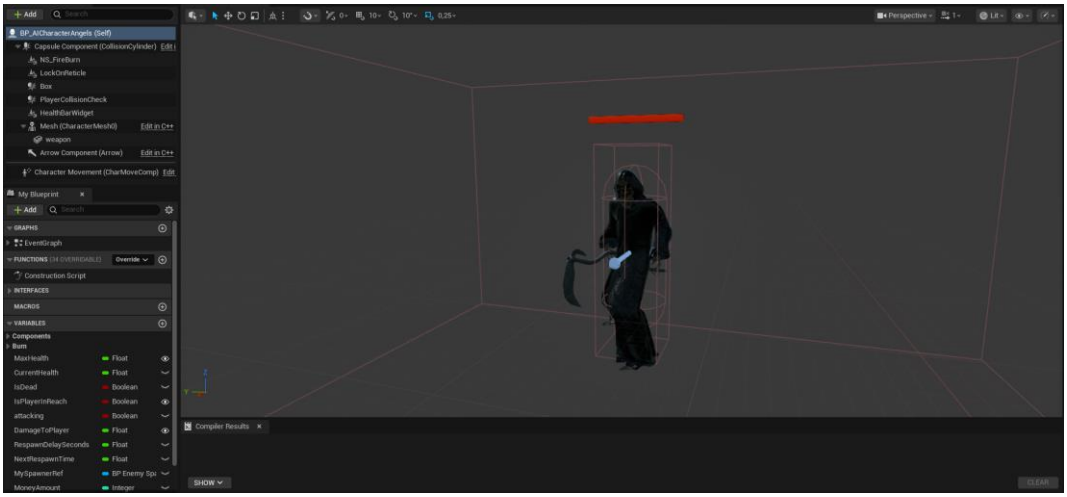


Figura 190: Enemigo Esqueleto Modelo.

La diferenciación entre estas tres entidades (Lizard, Wolf y Esqueleto) se limita estrictamente a aspectos superficiales y cosméticos. A nivel de la arquitectura de software, la homogeneidad funcional de estas unidades se garantiza mediante el uso de un mismo componente de control de inteligencia artificial (AI Controller) compartido por todas ellas.

Las únicas variaciones implementadas residen en:

1. Los activos gráficos (modelo 3D y conjunto de animaciones).
2. La configuración del *loot* (el tipo específico de moneda otorgada al jugador al ser derrotado el enemigo).

Esta estrategia de diseño maximiza la eficiencia del sistema al permitir la variación estética y contextual sin incurrir en la necesidad de desarrollar lógicas de comportamiento adicionales para las unidades de patrulla estándar.

Adicionalmente a las entidades estándar implementadas, el proceso de diseño consideró la inclusión de un arquetipo de enemigo estático, conceptualizado como una "estatua". La lógica funcional de este prototipo estaba definida por un mecanismo de defensa basado en la proximidad, donde la entidad solo infligiría daño al jugador al este vulnerar un umbral de cercanía preestablecido.

Técnicamente, el elemento diferenciador de esta unidad radicaba en su mecánica de neutralización. Se planificó que la entidad se desintegraría en múltiples fragmentos tras recibir un número específico de impactos (tres), empleando para ello el sistema **Chaos Destruction** [9] de Unreal Engine [16], específicamente a través de su **modo Fracture**. No obstante, debido a las dificultades inherentes a la manipulación y optimización de dicho sistema de destrucción en el contexto del *pipeline* de desarrollo, se tomó la

decisión de desestimar la implementación de este tipo de adversario, priorizando la estabilidad y la finalización de las funcionalidades esenciales del ecosistema de combate.

4.3.2. Jefe final

Enemigo único que aparece al atravesar el portal final. Este enemigo recibe daño reducido por parte del jugador, incrementando la dificultad del enfrentamiento. Su vida máxima es 1000,0 que se dividen en 3 barras de vida. Cuenta con 3 fases:

- 1ª fase: Gluttony se acerca al jugador y lanza un ataque de vómito en línea recta.

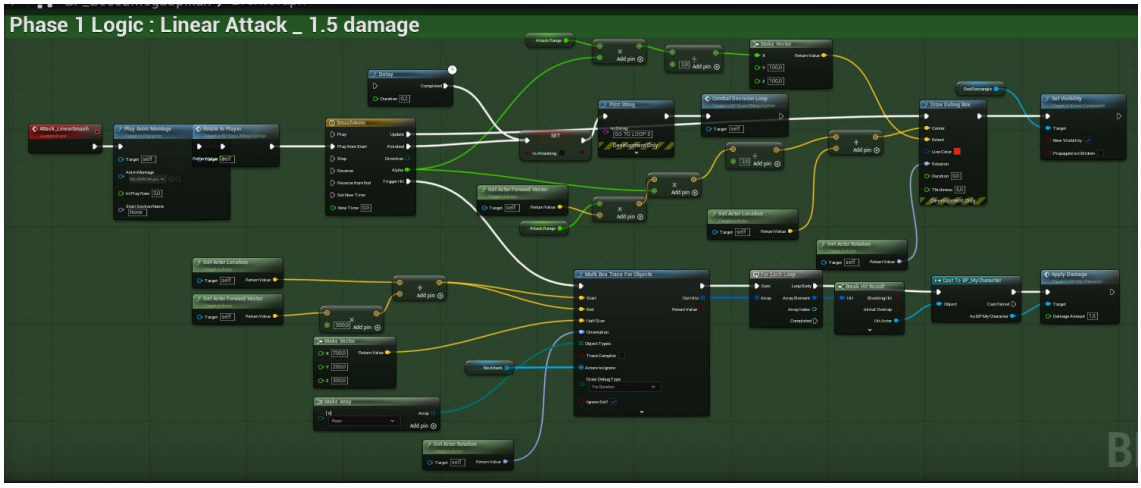


Figura 191: Lógica para fase 1 del jefe.

- 2ª fase: Gluttony se acerca al jugador y hace un círculo a su alrededor que hace daño.

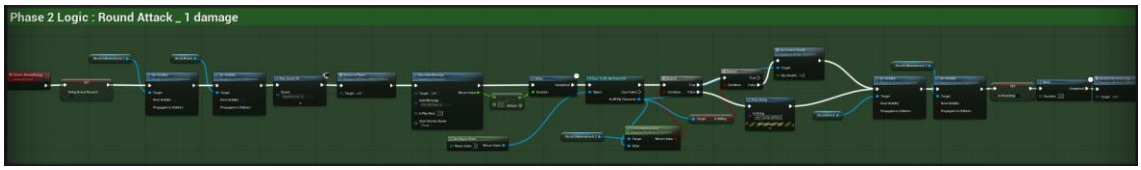


Figura 192: Lógica para fase 2 del jefe.

- 3ª fase: Gluttony huye del jugador y *spawnea* a 3 súbditos que persiguen al jugador y le muerden. Estos morirán en 13 segundos, son invencibles y mientras estén vivos el jefe es invulnerable a los ataques.

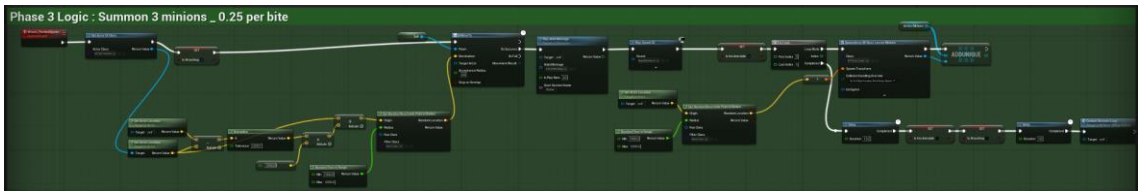


Figura 193: Lógica para fase 3 del jefe.

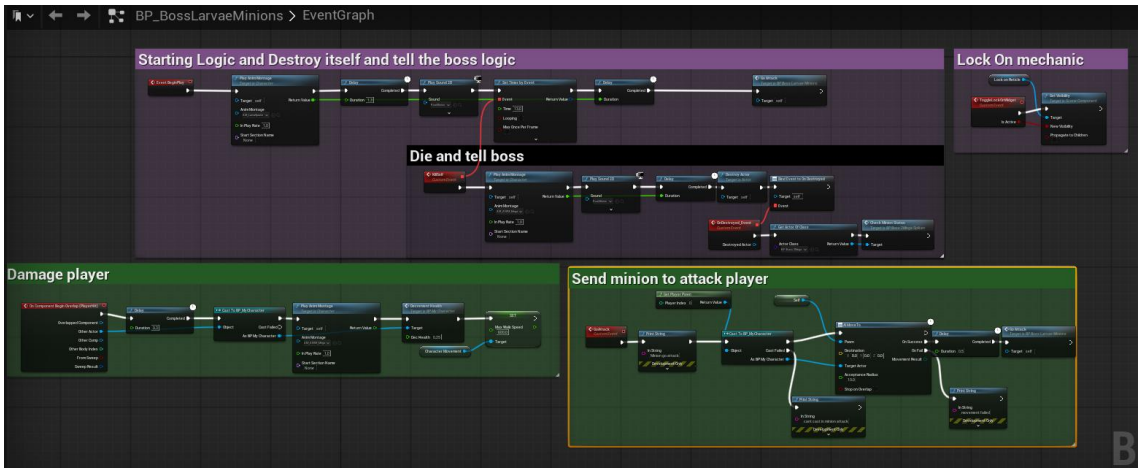


Figura 194: Lógica súbditos generados del jefe.

4.4. Animales acompañantes

Como complemento al sistema de combate, se implementó un sistema de acompañantes domesticados que permite al jugador contar con apoyo adicional durante los enfrentamientos. El jugador puede llevar equipados hasta un máximo de tres acompañantes de manera simultánea, aunque en la versión actual del proyecto únicamente se encuentran disponibles dos de ellos. Una vez equipados, los acompañantes siguen de forma autónoma al personaje principal durante la exploración.

Cuando un enemigo entra dentro del rango de detección de los acompañantes, estos activan su comportamiento ofensivo y se desplazan hacia el objetivo para atacarlo de manera automática. Este sistema introduce una capa adicional de estrategia en el combate, al permitir al jugador apoyarse en aliados controlados por la inteligencia artificial sin necesidad de microgestionar constantemente a cada uno.

Los acompañantes implementados comparten una lógica de inteligencia artificial común, encargada de la detección de enemigos, el desplazamiento hacia el objetivo y la ejecución de ataques. No obstante, se diferencian en los efectos y el daño infligido, lo que aporta variedad y progresión al sistema. Los acompañantes disponibles son los siguientes:

- **Fenra:** se encuentra disponible en una zona cercana al primer pueblo del juego. Este acompañante inflige 15 puntos de daño por ataque al enemigo.

- **Skoll:** se desbloquea aproximadamente en la mitad del juego, tras superar la segunda ciudad. Inflige 20 puntos de daño por ataque y aplica adicionalmente un efecto de quemadura sobre el enemigo.

Este sistema de acompañantes refuerza la sensación de progresión del jugador y contribuye a diversificar los enfrentamientos, integrándose de forma natural con el resto de mecánicas de combate y con el diseño general del juego.

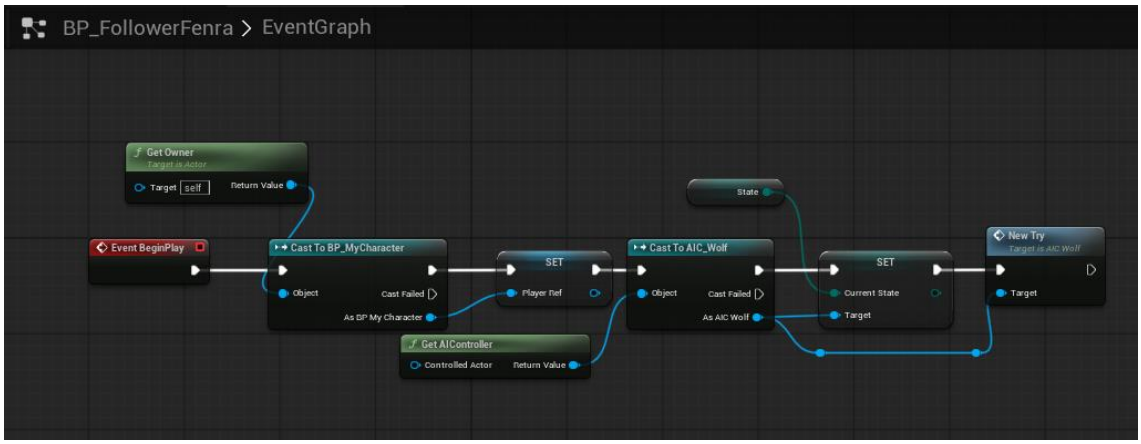


Figura 195: Implementación con el controlador común a uno de los acompañantes.

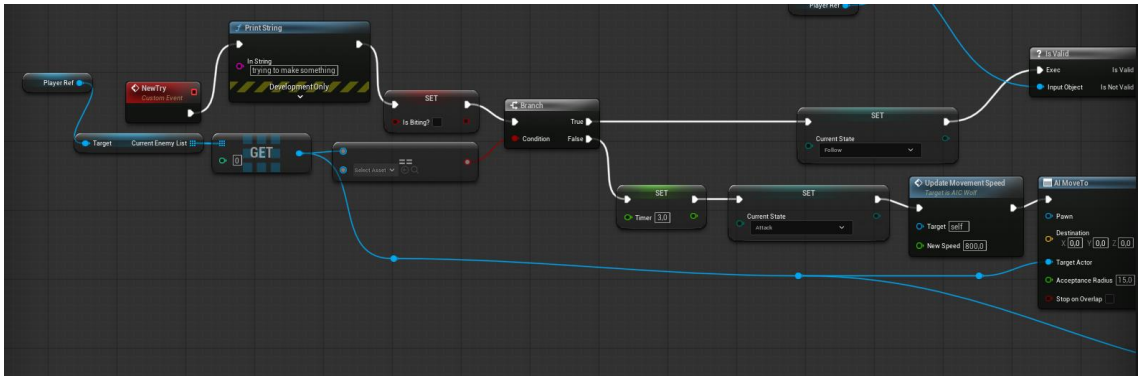


Figura 196: Controlador del ataque y seguimiento del acompañante.

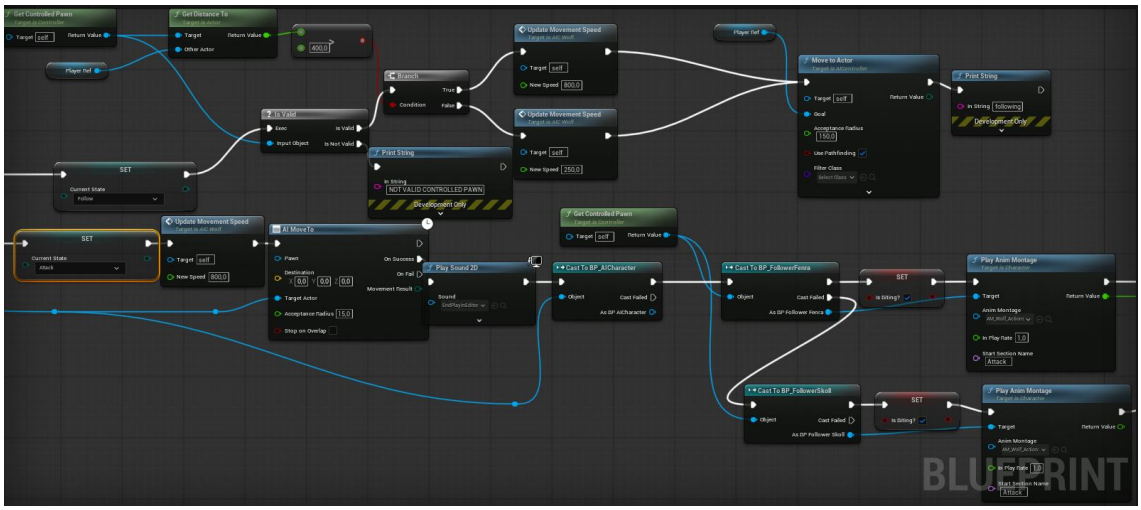


Figura 197: Lógica del ataque y seguimiento del acompañante.

4.5. Diálogo NPC

El sistema de diálogo implementado en la interacción con los Personajes No Jugables (NPC) se fundamenta en el uso de una **Interfaz [11]** denominada **Talk**.

En el entorno de desarrollo de Unreal Engine [16], una Interfaz [11] es un contrato de funcionalidad que define un conjunto de funciones sin implementación propia. Permite que diferentes Actor Blueprints, sin necesidad de pertenecer a la misma jerarquía de herencia, puedan compartir una lógica funcional común. Un Actor Blueprint simplemente implementa la interfaz, obligándose a definir el comportamiento de cada función declarada. Esta técnica promueve la modularidad, la decoupling (desacoplamiento) entre clases y facilita la comunicación entre actores diversos.

Para optimizar la creación y el mantenimiento de las entidades NPC, se ha adoptado una estrategia de diseño basada en la herencia de Actor Blueprints. Se ha desarrollado un conjunto de Blueprints padres (Parent Blueprints) [8] genéricos, los cuales contienen la lógica base para la interacción y el diálogo. Estos Blueprints padres [8] incorporan variables expuestas (visibles) que pueden ser modificadas directamente en el editor sin alterar la lógica de programación. Variables como el conjunto de líneas de diálogo o el modelo 3D (mesh) son editables, lo que permite la creación de múltiples instancias de NPC con distintas apariencias y diálogos (instanciación polimórfica) de manera eficiente, evitando la necesidad de crear un Blueprint [8] único para cada personaje.

La interfaz [11] Talk es implementada por las clases NPC (para responder al llamado de diálogo) y es invocada por el Blueprint [8] del personaje controlable (BP_MyCharacter).

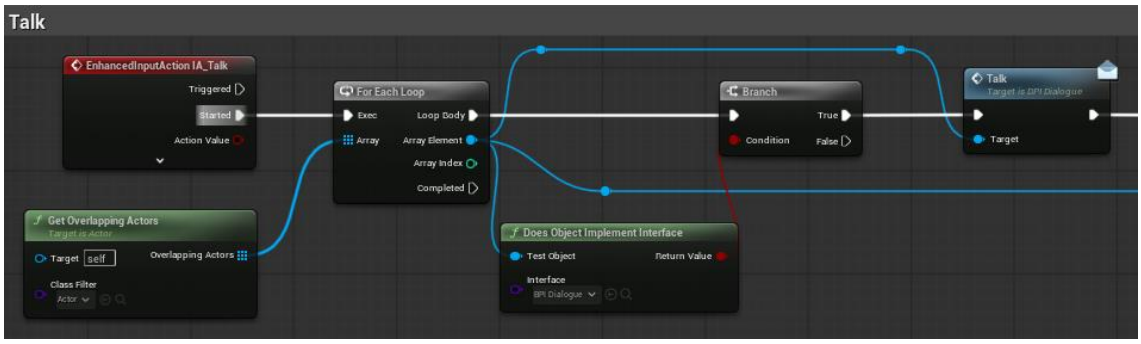


Figura 198: Acción de hablar en BP_MyCharacter.

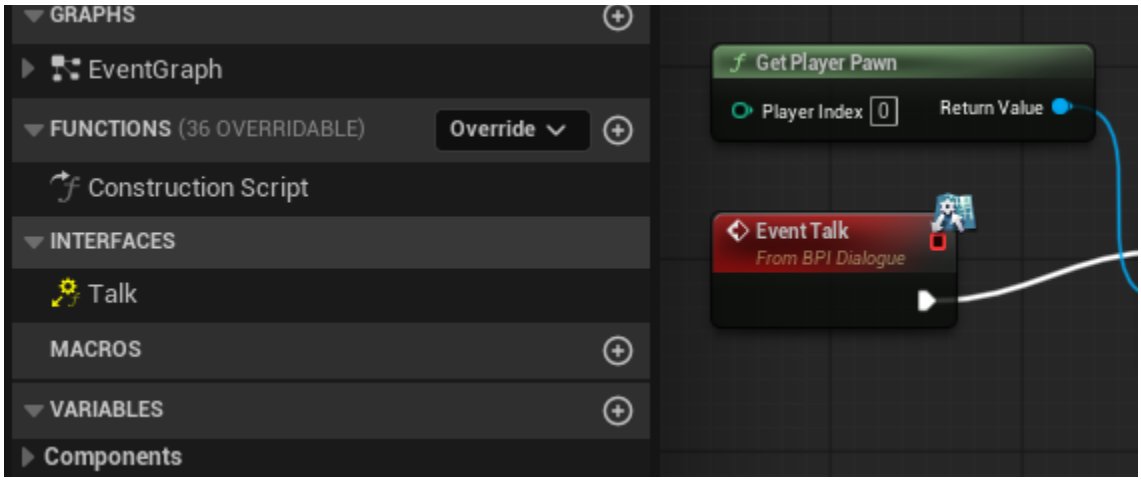


Figura 199: Interfaz de hablar implementada en un NPC.

Figura 200: Interfaz de interacción.

4.5.1. Diálogos simples

Los diálogos simples representan la modalidad de interacción más básica y directa. El NPC inicia la conversación con una secuencia de diálogo predefinida, cuyo avance está supeditado a la acción explícita del jugador (al presionar la tecla de interacción, F).

Para aumentar el realismo y la inmersión narrativa, el sistema integra una lógica de persistencia conversacional. En las interacciones posteriores a la primera, el NPC no reproduce la totalidad del diálogo inicial. En su lugar, el sistema conmuta la respuesta a un diálogo final o de clausura (repeat dialogue), indicando que el contenido informativo principal ya ha sido transmitido.

Figura 201: Diálogo básico dentro del juego. NPC Guía.

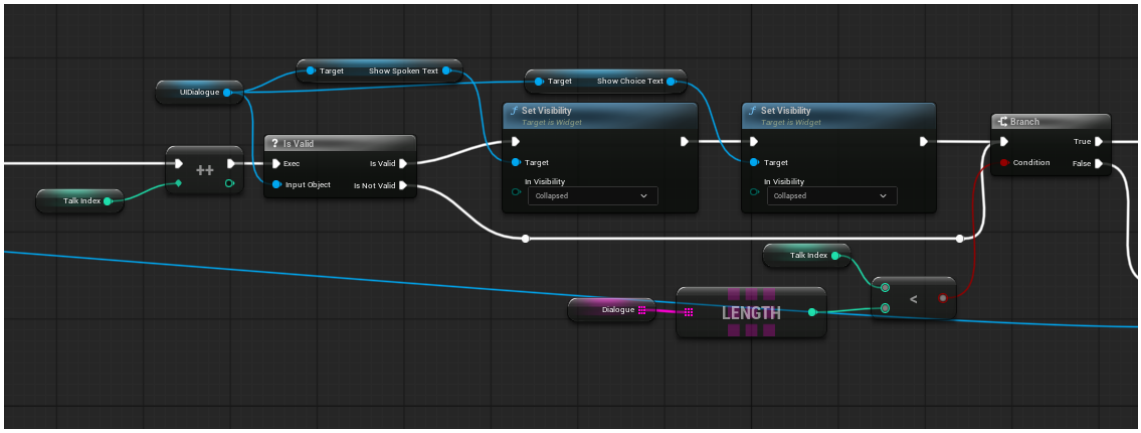


Figura 202: Lógica del diálogo básico, comprobar que no haya acabado el diálogo.

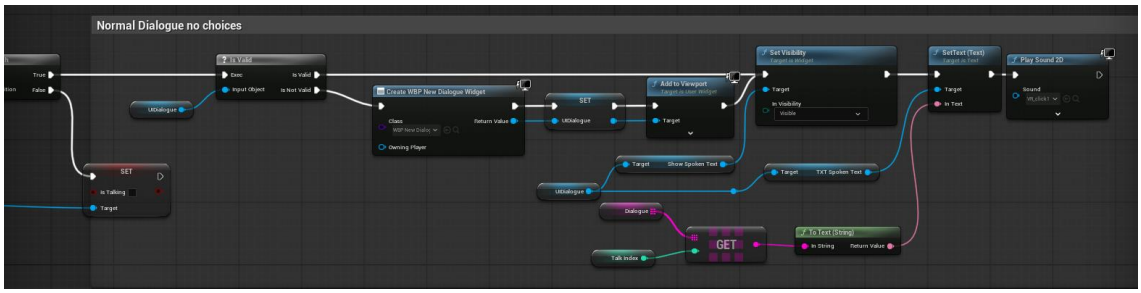


Figura 203: Lógica del diálogo básico, dice el índice correspondiendo del array de diálogo.

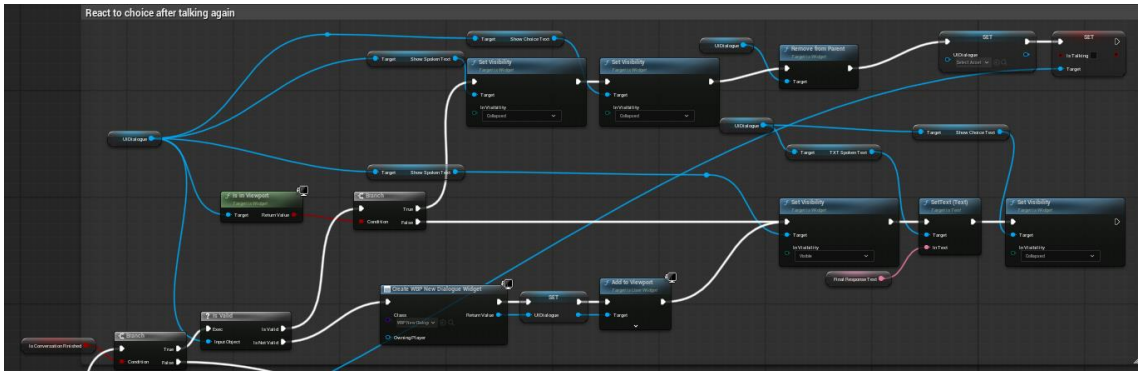


Figura 204: Lógica del repetición de diálogo.

4.5.2. Diálogos con elecciones

Esta modalidad de diálogo introduce una mayor complejidad interactiva y se encuentra intrínsecamente ligada al sistema de alineamiento del jugador. Tras la activación de la conversación con este tipo de NPC, se presenta al jugador un punto de ramificación en la narrativa, ofreciendo dos opciones de respuesta mutuamente excluyentes.

Durante la fase de toma de decisión, se implementa una interrupción en el flujo de juego (gameplay) mediante la inhabilitación de las capacidades de movimiento y ataque del personaje controlable. Simultáneamente, se habilita la visualización y el control del cursor del ratón, permitiendo al jugador la interacción con los elementos de la interfaz de usuario (UI) para la selección de la opción deseada.

La selección de una opción específica no solo determina el contenido del diálogo de repetición futuro (diálogo final distinto) sino que también desencadena una modificación en la variable que gestiona el alineamiento ético o moral del personaje del jugador. De esta manera, las decisiones tomadas durante estos diálogos tienen un impacto directo en el alineamiento del personaje.

Figura 205: Diálogo con opciones dentro del juego. NPC Guía.

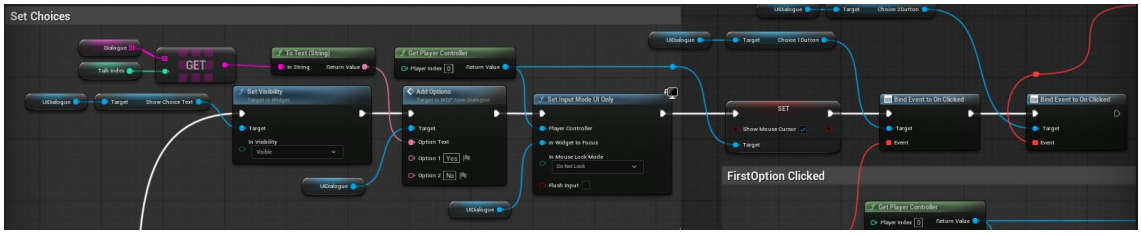


Figura 206: Lógica de creación para poder crear manualmente los NPC sin necesidad de crear más Blueprints.

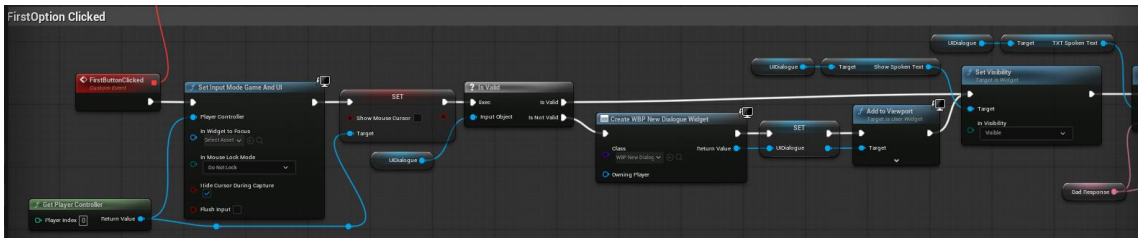


Figura 207: Lógica de creación de uno de los diálogos al elegir una de las opciones.

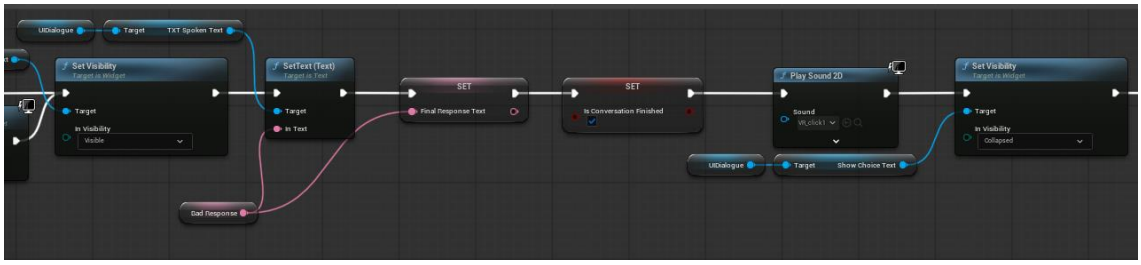


Figura 208: Guardar la repetición de diálogo pertinente a la elección.

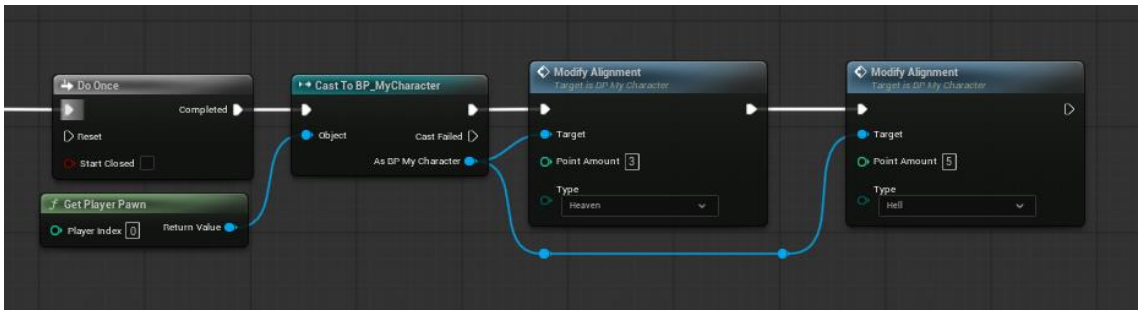


Figura 209: Modificar alineamiento dependiendo de la respuesta una vez.

4.6. Misiones

El sistema de misiones constituye uno de los principales mecanismos de guía y progresión del jugador a lo largo de la experiencia de juego. Desde el inicio de la partida, el jugador es orientado mediante un conjunto de **misiones principales**, encargadas de estructurar la narrativa central y marcar los objetivos fundamentales del desarrollo del juego. De manera complementaria, el mundo del juego incluye diversas **misiones secundarias**, las cuales pueden ser descubiertas de forma opcional durante la exploración y contribuyen a enriquecer tanto la narrativa como la inmersión del jugador.

Con el objetivo de facilitar la escalabilidad y el mantenimiento del sistema, todas las misiones se encuentran definidas en una **Tabla de Datos (Data Table)**. Esta estructura permite añadir, modificar o eliminar misiones de forma sencilla sin necesidad de alterar directamente la lógica principal del sistema, lo que resulta especialmente adecuado para proyectos con previsión de ampliación futura.

El estado de las misiones se gestiona de manera dinámica mediante un **mapa de datos**, cuya clave corresponde a una cadena de texto (*string*) que identifica de forma única cada misión, coincidiendo con su identificador (ID) en la tabla de datos. El valor asociado a cada clave es una **enumeración** que representa el estado actual de la misión. Los estados contemplados en el sistema son los siguientes:

- Misión secundaria no otorgada.
- Misión principal no otorgada.
- Misión activa.
- Misión completada.

Esta estructura permite un control preciso del progreso del jugador, así como la activación condicional de eventos narrativos, diálogos y recompensas en función del estado de cada misión, sentando las bases para una **narrativa reactiva** que responde a las decisiones y acciones realizadas durante la partida.

La lógica asociada a los NPCs y a la asignación y finalización de misiones es común tanto para las misiones principales como para las secundarias. La única diferencia entre ambos tipos radica en la categoría de la misión, utilizada para su correcta clasificación y visualización en el apartado correspondiente del menú de misiones del inventario.

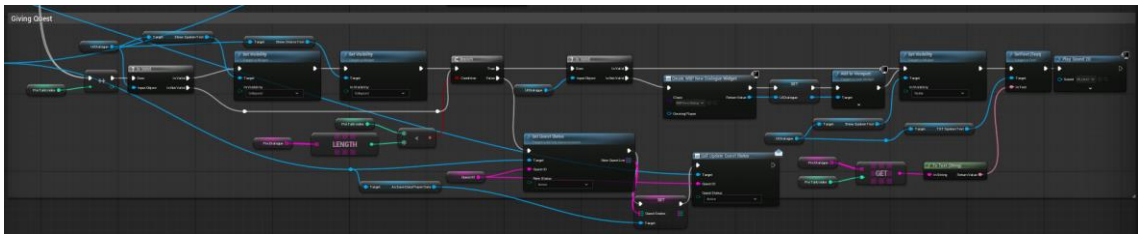


Figura 210: Lógica para activar misión con NPC.

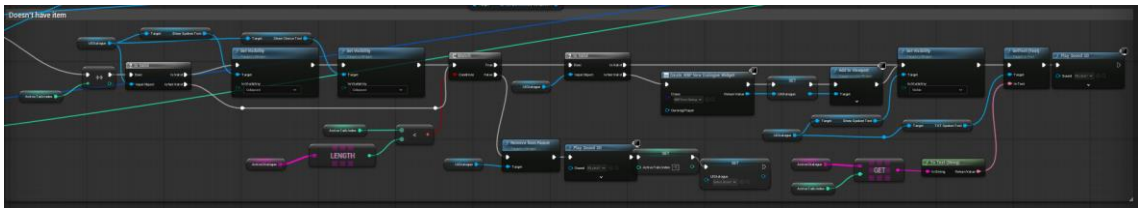


Figura 211: Lógica de diálogo de NPC de misión si ya está dada y aún no completada.

Figura 212: Lógica de diálogo de NPC de misión si no está completada y el jugador tiene los objetivos.

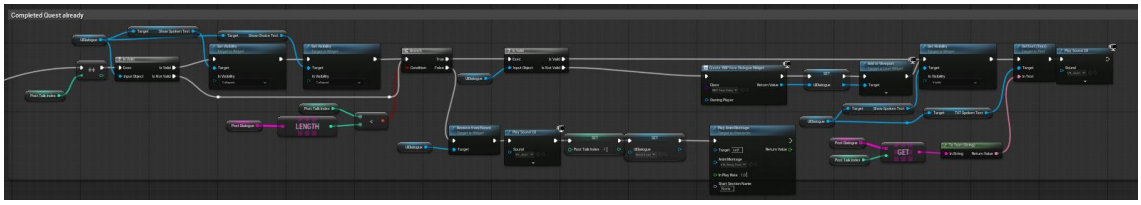


Figura 213: Lógica de diálogo de NPC de misión si ya está completada.

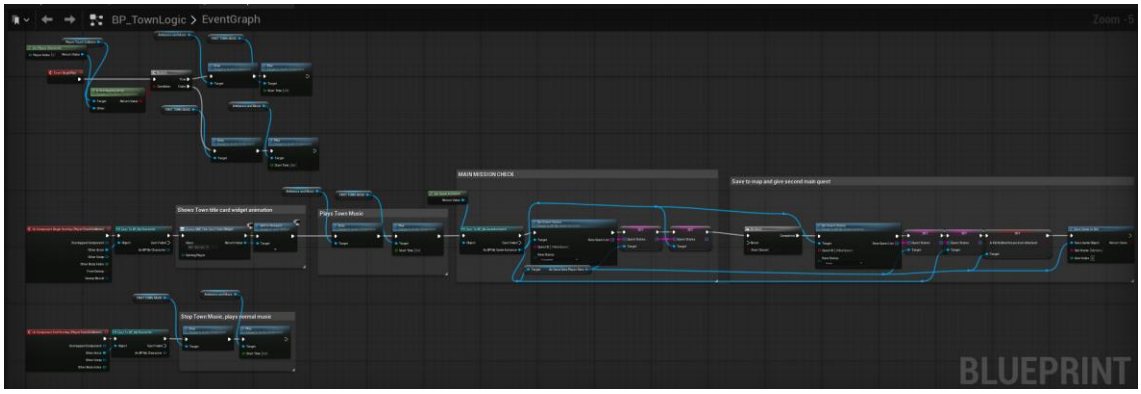


Figura 214: Lógica entrar a primera ciudad que incluye avanzar en una misión principal sin necesidad de hablar con un NPC.

4.6.1. Tutorial

El módulo tutorial se presenta como una funcionalidad que opera de forma heterogénea a la estructura normalizada del sistema de misiones, ya que no se define como una

entidad de misión propiamente dicha. Su activación se produce al iniciar una nueva partida, manifestándose en la esquina superior derecha de la interfaz.

Figura 215: Tutorial activo arriba a la derecha.

El avance del tutorial se basa en un sistema de progresión condicional. Su estado se actualiza dinámicamente en función de la ejecución exitosa de las acciones instruidas por el jugador. Una vez completada la secuencia de instrucciones, el módulo tutorial se desactiva permanentemente.

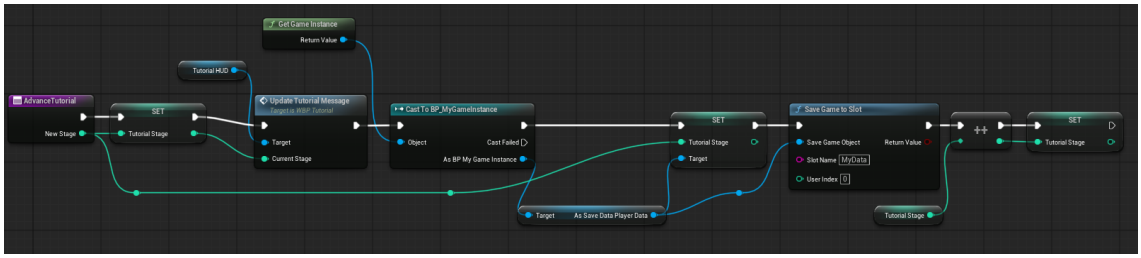


Figura 216: Función para avanzar tutorial en BP_MyCharacter.

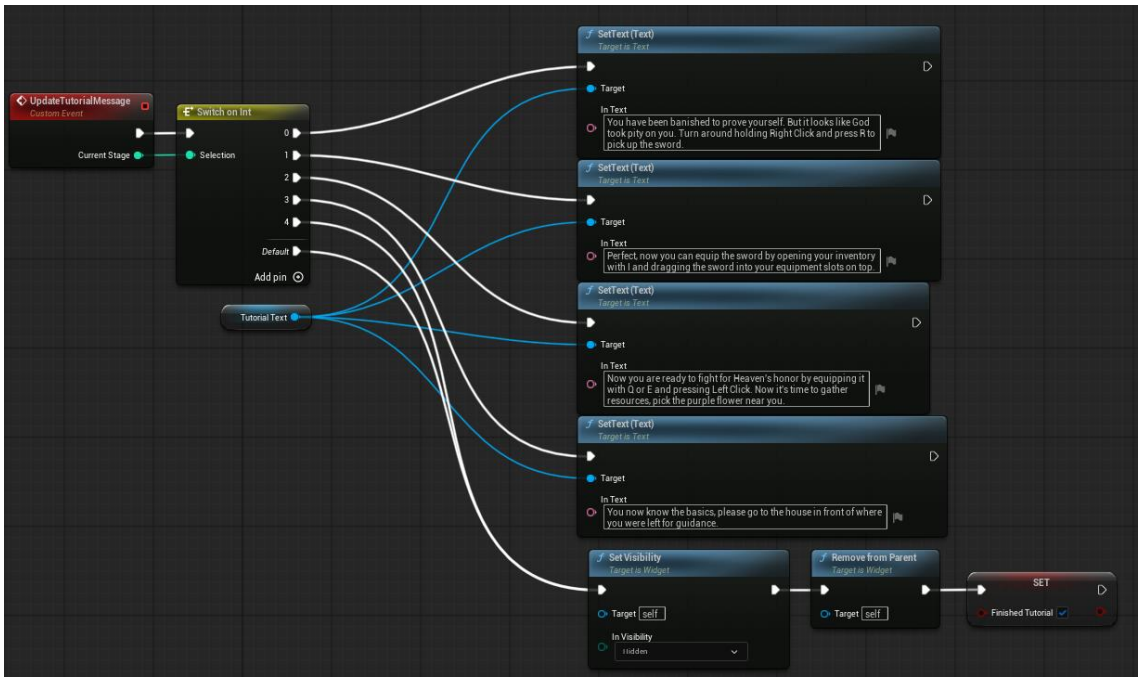


Figura 217: Función para poner el texto del tutorial en BP_MyCharacter.

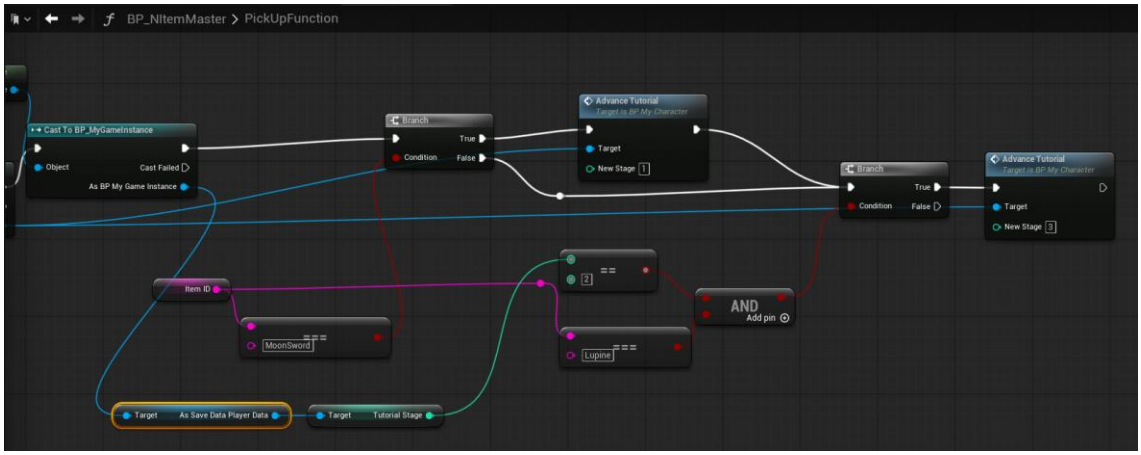


Figura 218: Ejemplo de avance de tutorial al recoger objetos.

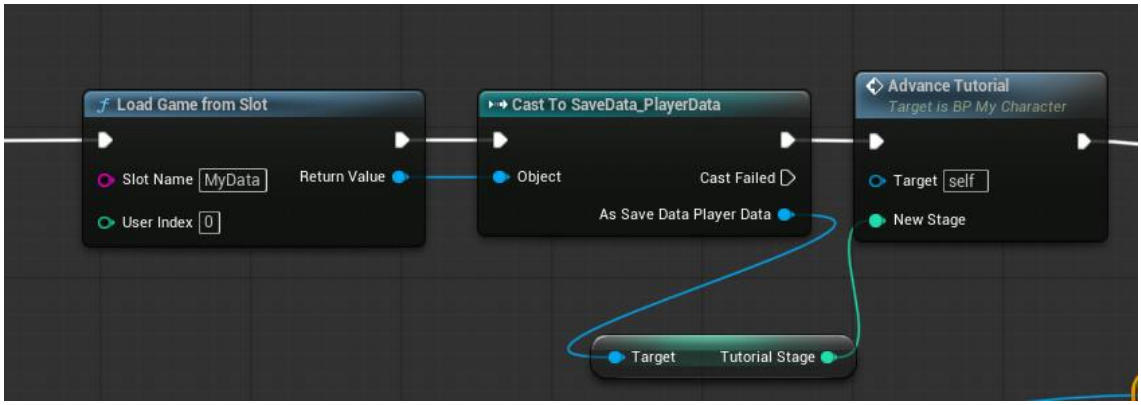


Figura 219: Cargado del estado del tutorial en Event BeginPlay de BP_MyCharacter.

Para garantizar la persistencia de su estado a través de las diferentes sesiones de juego, la información relativa a la finalización del tutorial se almacena utilizando el sistema de

guardado nativo del motor (Save Game Blueprint [15]), impidiendo su reaparición en partidas subsiguientes.

4.6.2. Misiones principals

- **Go To The First Town**: Esta misión constituye el primer objetivo del juego. Se obtiene automáticamente al finalizar el tutorial inicial y tras interactuar con el personaje guía. La misión se considera completada cuando el jugador accede por primera vez a la ciudad de **Eldhollow**, que actúa como el primer núcleo urbano del mundo del juego. No se asocia a una recompensa material directa, ya que su función principal es introducir al jugador en la estructura del mundo y en el flujo narrativo principal.

- **Help is Needed**: La segunda misión principal se activa al entrar en Eldhollow. El objetivo de esta misión es ayudar a un personaje clave del pueblo, el presidente situado en la zona central. La misión se completa al entregar **10 unidades de piel** y **5 setas marrones**, introduciendo al jugador en las mecánicas básicas de recolección y entrega de objetos. La recompensa está orientada al progreso narrativo y a la continuidad de la historia principal.

- **Road to Seraphen**: Esta misión se desbloquea automáticamente tras completar *Help is Needed*. Su objetivo es guiar al jugador hacia la segunda ciudad del juego, **Seraphen**, reforzando la exploración del mundo y la transición entre zonas principales. La misión se completa al entrar en dicha ciudad, sin requerir acciones adicionales, y sirve como nexo entre los primeros actos narrativos.

- **Monetary Trouble**: La cuarta misión principal se obtiene al acceder a Seraphen. En este caso, el jugador debe ayudar a un personaje relevante, la princesa situada junto a la iglesia de la ciudad. La misión se completa entregando **5 setas rojas** y **20 unidades de piel**, reforzando nuevamente el uso de recursos y la interacción con NPCs clave. Su finalización permite continuar con la progresión de la historia principal.

- **Death Valley Venture**: Esta misión se desbloquea tras completar *Monetary Trouble*. El objetivo consiste en llegar al **Death Valley**, una zona avanzada del juego que representa un incremento significativo de dificultad y relevancia narrativa. La misión se considera completada al entrar en dicha región, preparando al jugador para el desenlace de la historia principal.

- **The Boss Needs to Die**: Se trata de la sexta y última misión principal del juego. Se obtiene automáticamente al completar *Death Valley Venture*. El objetivo final es derrotar al jefe principal del juego. La misión se completa al eliminar al jefe, acción que provoca la aparición del portal situado al final del Death Valley, marcando el cierre narrativo y jugable de la experiencia principal.

4.6.3. Misiones secundarias

- **Strengthen Up**: Esta misión se obtiene al interactuar con un NPC situado en el primer pueblo. El personaje solicita una poción de fuerza (**Agate Drops**), introduciendo al jugador en el sistema de creación y uso de pociones. La misión se completa al entregar el objeto solicitado, y la recompensa consiste en 40 Alas, incentivando la exploración de mecánicas opcionales.

- **Rich Kid**: Esta misión se activa al interactuar con un NPC ubicado en la zona central de la segunda ciudad. El objetivo es entregar una **calavera de dragón**, lo que implica la exploración de zonas más peligrosas o avanzadas. La misión se completa al entregar dicho objeto y la recompensa es **200 Alas, 100 Hals y 100 Nors**, constituyendo una de las recompensas económicas más relevantes entre las misiones secundarias. Esto es debido a la gran dificultad que conlleva encontrar dicha calavera, ya que hay una al inicio del juego solo accesible al tomar una poción que permita realizar 3 saltos y hay otra en la zona de dificultad media pasado Seraphen pero que hay que salir del camino principal para hallarla.

4.7. Sistema de alineamiento moral

Se trata de un mecanismo de modificación de estadísticas. Es el mecanismo central del juego. Se pueden obtener diferentes puntuaciones a lo largo del juego realizando diversas actividades que veremos más adelante. Al alcanzar el umbral correspondiente y seleccionar una alineación, el personaje recibe mejoras permanentes en sus estadísticas base:

- **Alineación con el Cielo:** incremento de 400 unidades de velocidad y un corazón adicional de vida máxima.
- **Alineación con la Tierra:** incremento de 3 corazones de vida máxima, 10 puntos de ataque base y 50 unidades de velocidad.
- **Alineación con el Infierno:** incremento de 1 corazón de vida máxima y 20 puntos de ataque base.

Estas mejoras refuerzan estilos de juego diferenciados y consolidan el impacto de las decisiones del jugador a largo plazo.

Figura 220: Sistema de elección de alineamiento.

4.7.1. Alineamiento mediante diálogo

La modificación del valor de alineamiento se produce directamente a través de los diálogos que presentan elecciones al jugador, tal como se detalla en la **sección 4.5.2**.

Cada opción de respuesta en estos diálogos está programada para otorgar un incremento específico en una de las categorías de alineamiento, reforzando la naturaleza del sistema como un reflejo de las decisiones éticas o morales tomadas por el personaje en la narrativa.

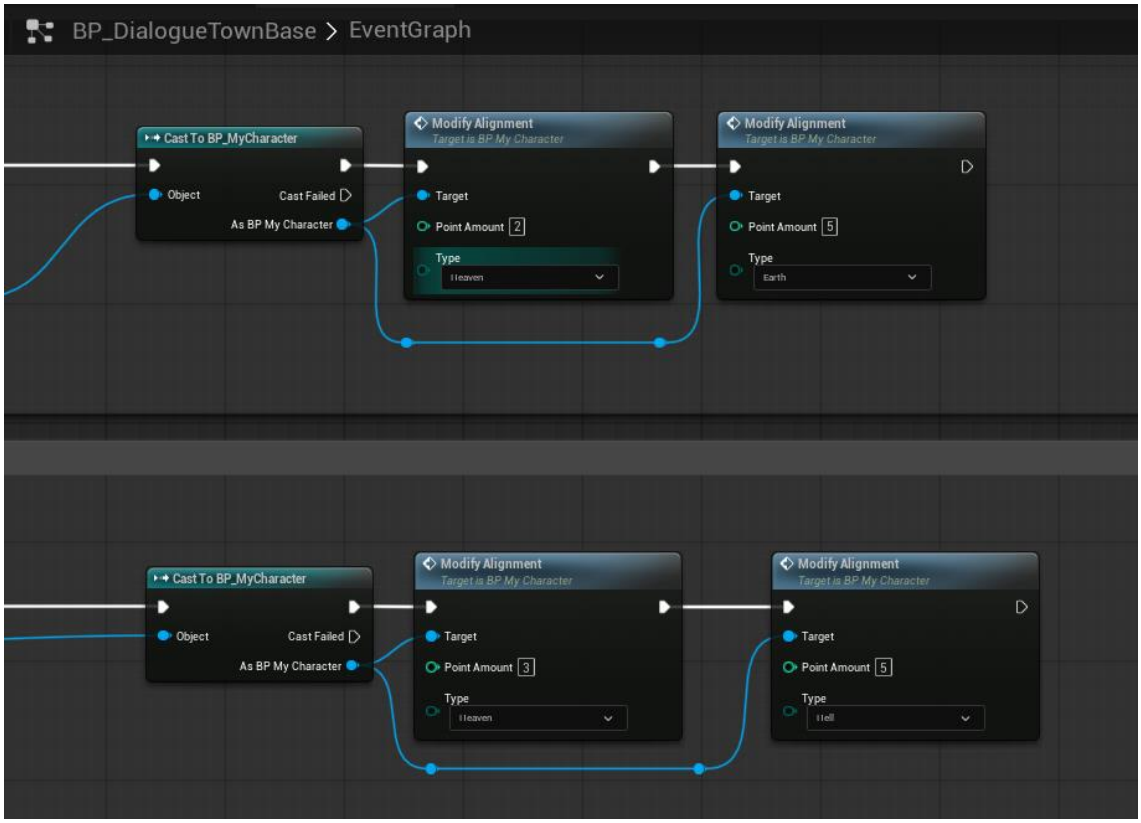


Figura 221: Comparativa de alineamiento al elegir diferentes opciones de diálogo.

4.7.2. Alineamiento mediante Batalla

La finalización exitosa de encuentros de combate contribuye a la progresión del alineamiento, aunque con una distribución de puntos que varía significativamente en función de la entidad derrotada:

- **Encuentros de Alta Complejidad (Gluttony):** La neutralización del jefe final otorga una cantidad sustancial de puntos de alineamiento, reconociendo el logro de un objetivo de alto impacto dentro de la narrativa.
- **Entidades Estándar (Enemigos Normales):** La derrota de enemigos de patrulla reporta una contribución mínima al alineamiento (establecida en 1 punto por unidad). No obstante, la dirección de esta ganancia está ligada al origen temático del enemigo:

- Los enemigos asociados al **Infierno** otorgan alineamiento en la categoría del **Cielo**.
- Los enemigos asociados al **Cielo** otorgan alineamiento en la categoría del **Infierno**.
- Los enemigos asociados a la **Tierra** otorgan alineamiento en la categoría de la **Tierra**.

Esta estructura gamifica el principio de conflicto temático al incentivar la lucha contra entidades de un alineamiento específico. Además al volver a generarse los enemigos, esto permite que siempre sea posible llegar a una alineación específica.

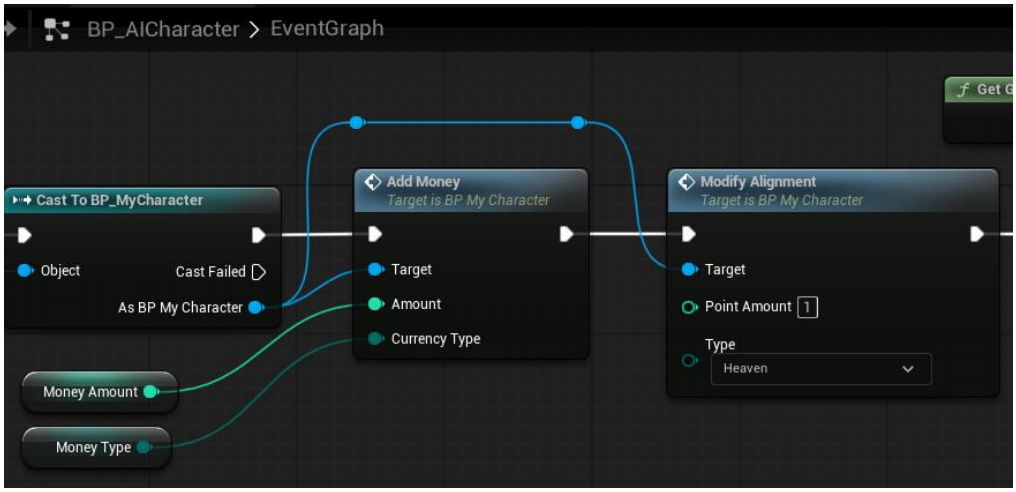


Figura 222: Enemigo normal añadiendo los puntos correspondientes.

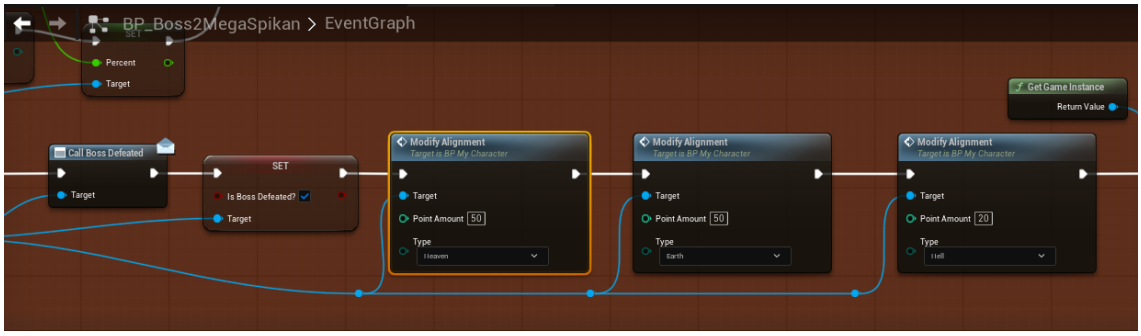


Figura 223: Jefe añadiendo una gran cantidad de puntos.

4.7.3. Alineamiento mediante eventos

Además del diálogo y el combate, el alineamiento se puede modificar a través de eventos puntuales y estructurales que marcan hitos en la progresión o la exploración del mundo:

- **Eventos de Interacción Específicos:** Ciertos sucesos, como la intervención (o no intervención) en un evento de rescate (ejemplo: un niño siendo atacado en las afueras de la ciudad de Seraphen), que se gestiona de forma más próxima a una cinemática que a un diálogo interactivo, pueden afectar el alineamiento.
- **Completado de Tareas Sistemáticas:** La consecución de objetivos secundarios, tales como la finalización del Bestiario, la conclusión de Misiones Secundarias, o la interacción clave con ciertos *NPCs* (ejemplo: al hablar con la bruja para

desbloquear la funcionalidad de pociones), también reporta puntos de alineamiento.

- **Interacciones de Impacto Directo:** La recepción de bendiciones o maldiciones específicas por parte de ciertos *NPCs* constituye otra vía directa para la alteración del valor del alineamiento del personaje.

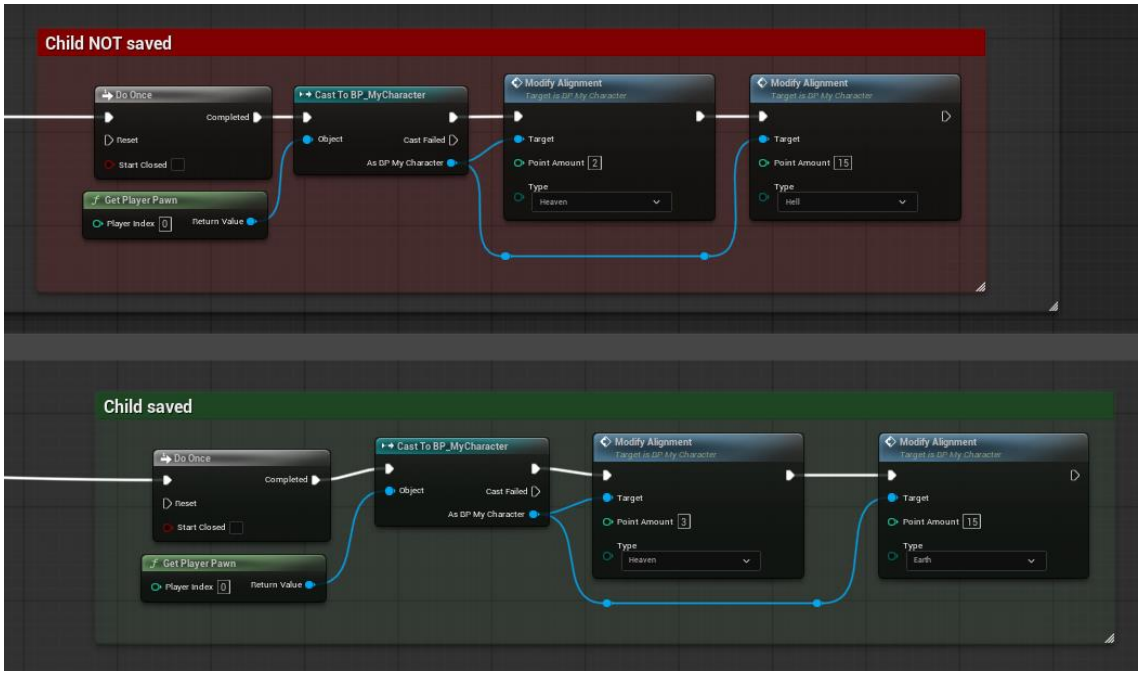


Figura 224: Añadir alineamiento debido a la cinemática de rescate.

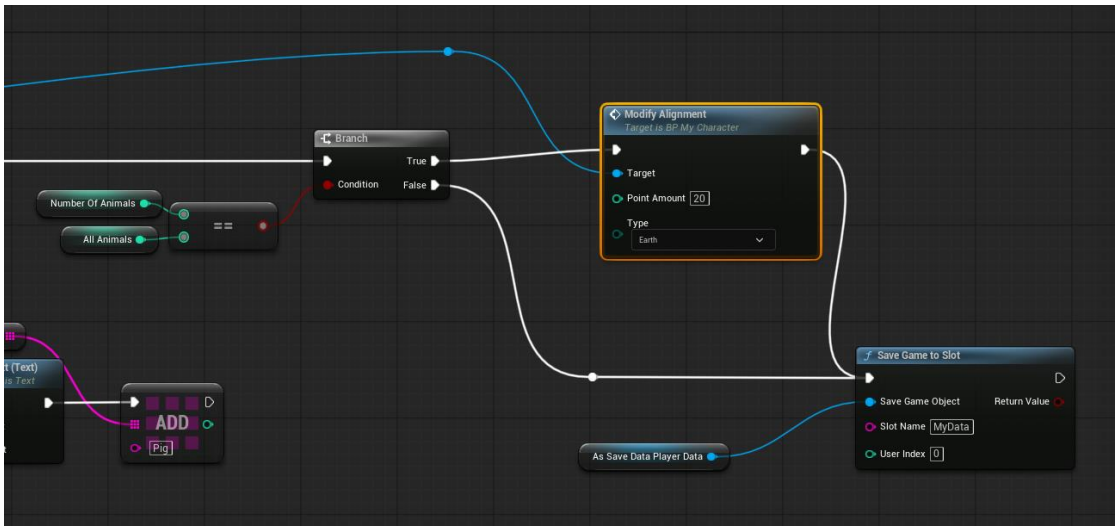


Figura 225: Añadir alineamiento debido a compleción del bestiario.

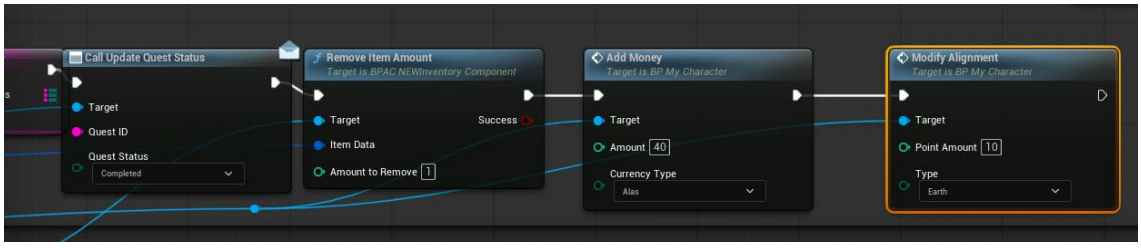


Figura 226: Añadir alineamiento debido a compleción de una misión.

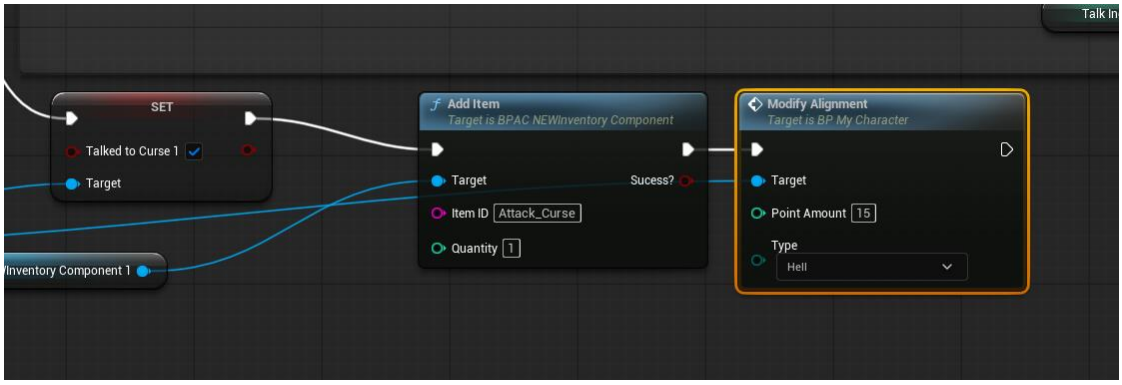


Figura 227: Añadir alineamiento debido l recibimiento de una maldición.

4.8. Menús e interfaces

4.8.1. Menú principal e interfaz del jugador

El Menú Principal del juego ha sido implementado como un Nivel independiente dentro de la arquitectura del proyecto. Esta segregación garantiza que los recursos del entorno de juego principal no se carguen innecesariamente.

El diseño escénico de este nivel es minimalista, consistiendo únicamente en la disposición de una Cámara fija y un Actor base, los cuales definen el punto de vista y el contexto visual de la escena. La inmersión inicial se complementa con la reproducción de una pieza musical de ambientación.

A nivel funcional, el menú presenta una interfaz de usuario (UI) dinámica, cuya composición se ajusta en función del progreso de juego. El número de opciones de navegación disponibles oscila entre tres (3) y cuatro (4) botones, siendo la variación determinada por la existencia o ausencia de un archivo de partida guardada (Save Game [15]). La presencia de un archivo guardado desbloquea la opción de "Continuar Partida", completando así el conjunto de opciones.

Figura 228: Pantalla de inicio sin partida ya empezada.

Figura 229: Pantalla de inicio con partida ya empezada.

Figura 230: Pantalla de inicio con la configuración seleccionada.

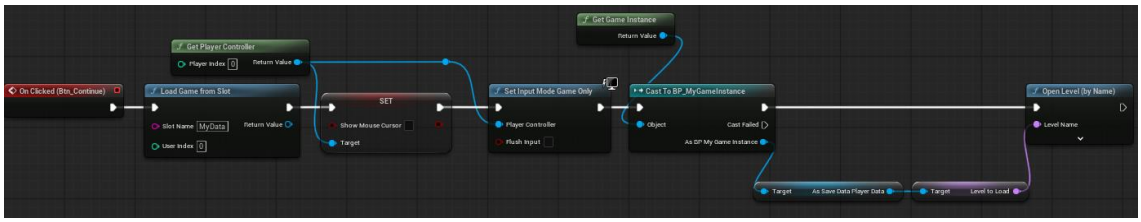


Figura 231: Lógica botón de continuar.

Figura 232: Lógica botón de nueva partida.

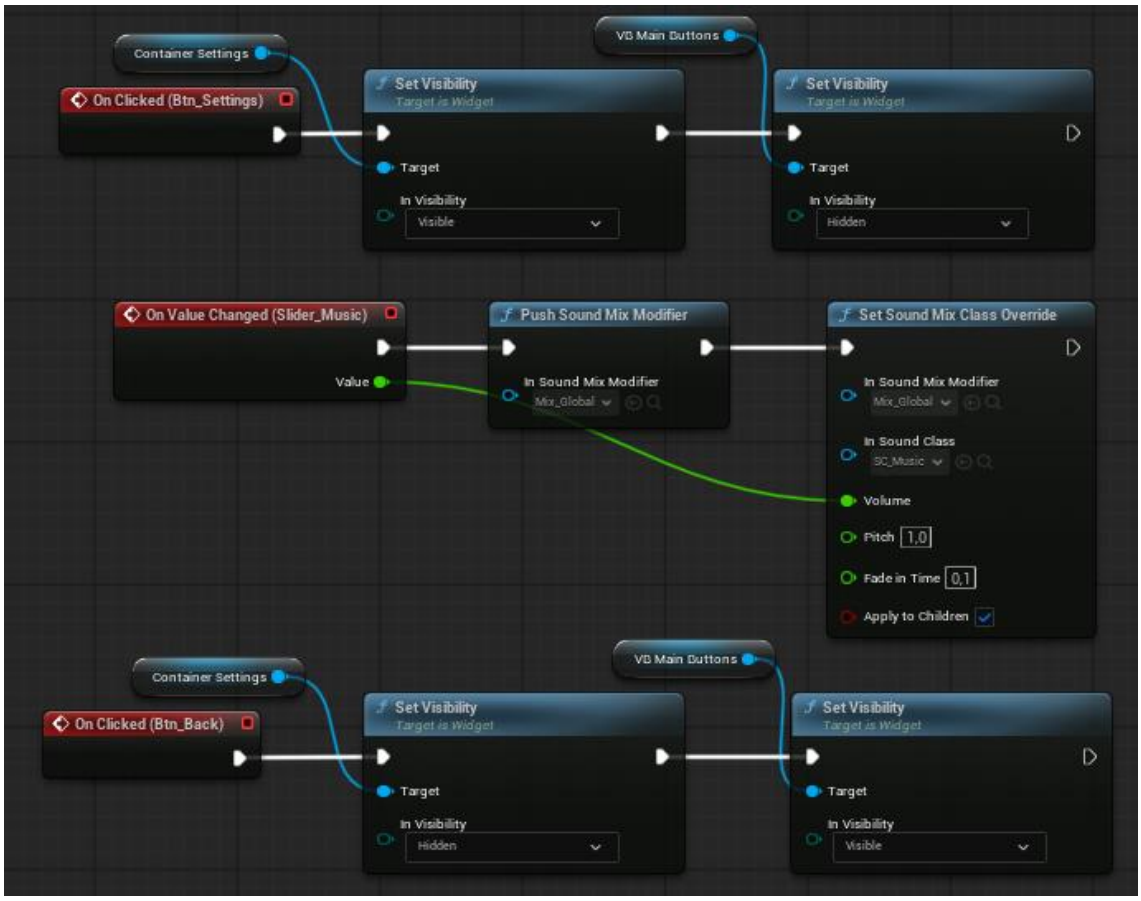


Figura 233: Lógica botón de configuración.

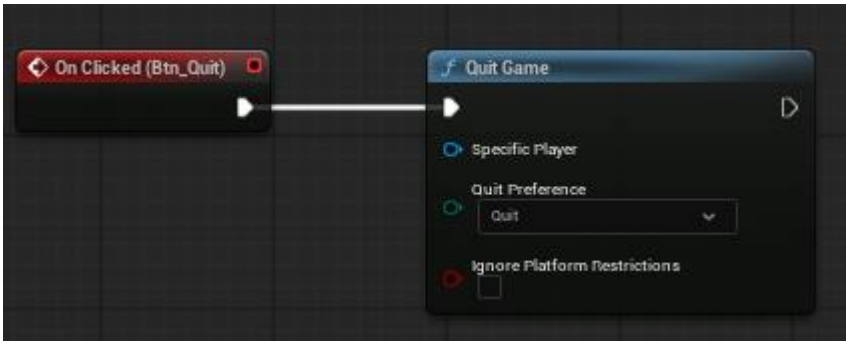


Figura 234: Lógica botón de salir del juego.

El diseño de la interfaz de usuario se ha planteado con el objetivo de minimizar la sobrecarga de información y favorecer una lectura clara e inmediata del estado del jugador durante la partida. Para ello, se ha optado por una disposición sencilla y coherente, en la que únicamente se muestran en pantalla los elementos estrictamente necesarios para la experiencia de juego.

El indicador principal de estado es el sistema de vida, representado mediante corazones situados en la esquina superior izquierda de la pantalla. La pérdida de salud se muestra de forma progresiva, reduciéndose en cuartos de corazón, lo que permite una representación visual precisa del daño recibido.

En la parte inferior izquierda de la pantalla se muestran dos espacios destinados a las armas equipadas. Estos slots reflejan visualmente las espadas disponibles y se acompañan de un indicador que señala la tecla correspondiente para alternar entre ellas, permitiendo un cambio rápido de arma sin necesidad de acceder al inventario.

Figura 235: Interfaz de usuario básica del jugador.

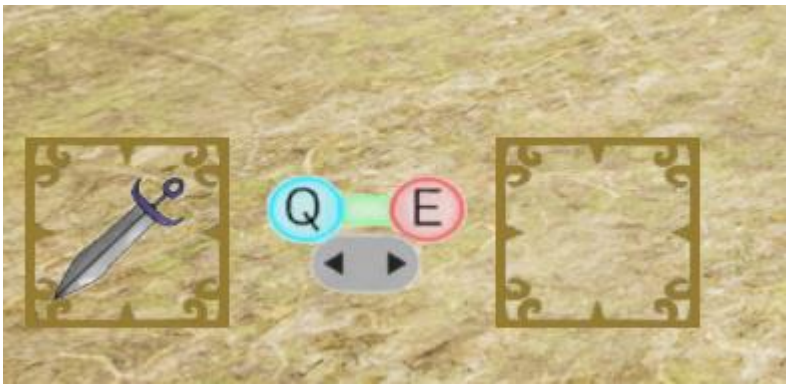


Figura 236: Interfaz de usuario básica del jugador con una espada equipada.

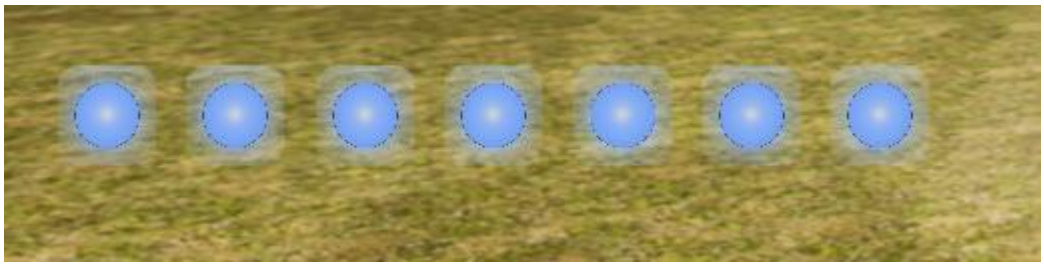


Figura 237: Vida de personaje con alineación al cielo.

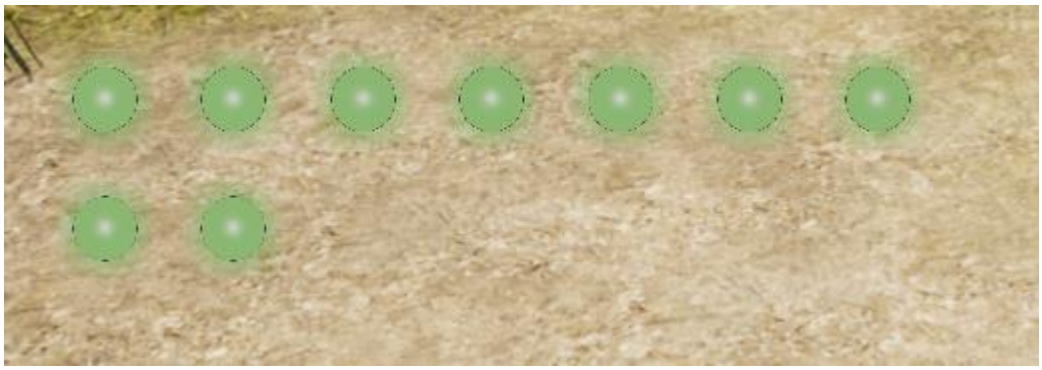


Figura 238: Vida de personaje con alineación a la tierra.

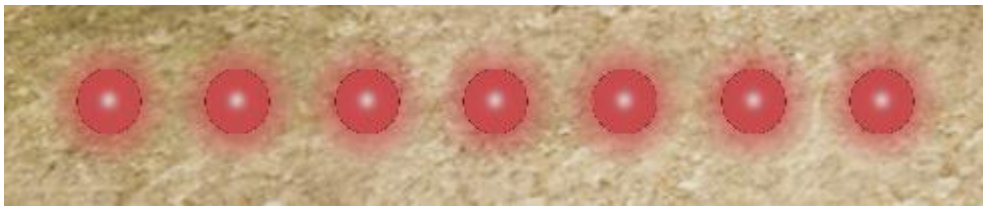


Figura 239: Vida de personaje con alineación a la tierra.

4.8.2. Menú de pausa

El sistema de pausa se implementa mediante un menú accesible durante la partida, el cual ofrece tres opciones principales: reanudar la partida, acceder al menú de opciones y salir del juego. Dentro del apartado de opciones, el jugador puede ajustar el volumen de la música según sus preferencias, proporcionando un control básico pero funcional sobre la experiencia audiovisual.

El botón de configuración sigue exactamente la misma lógica que el que tienen el mismo nombre en el menú principal. El botón para salir del juego, además de hacer lo que su nombre indica, guarda el estado actual del jugador sin incluir la localización actual.

Figura 240: Pantalla de pausa.

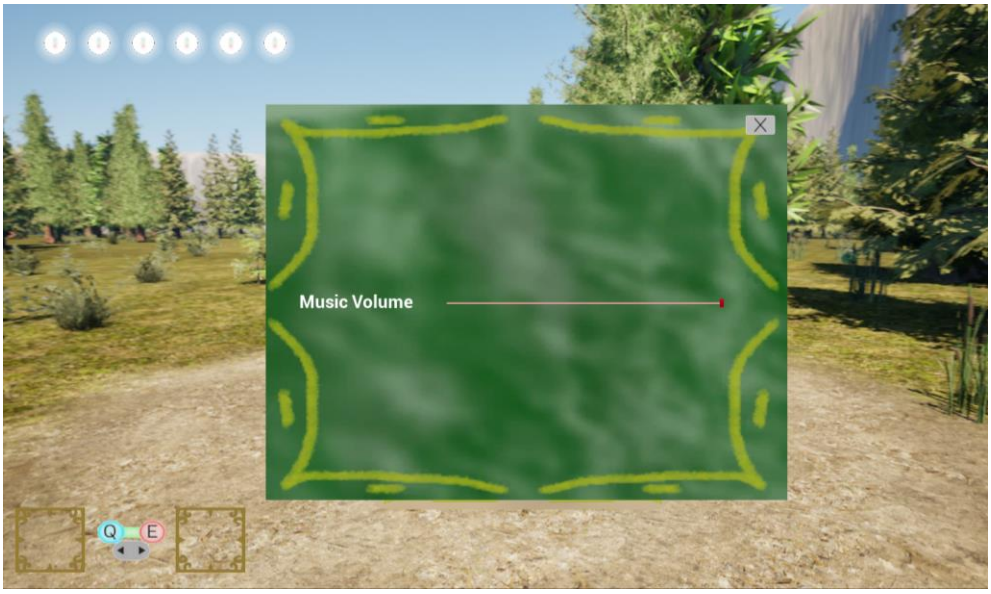


Figura 241: Pantalla de pausa con la configuración abierta.

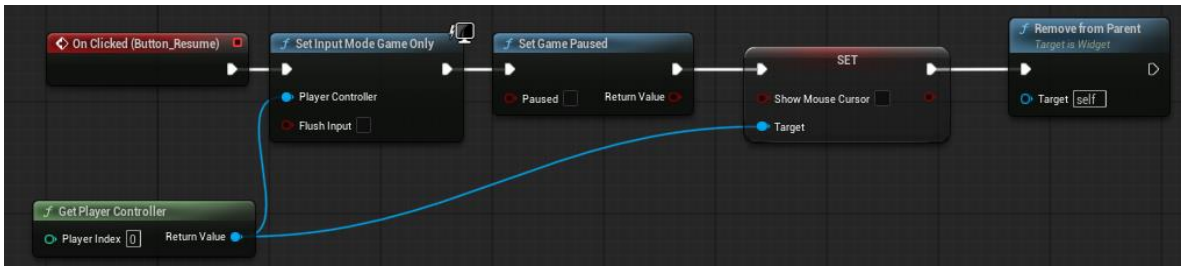


Figura 242: Lógica botón continuar.

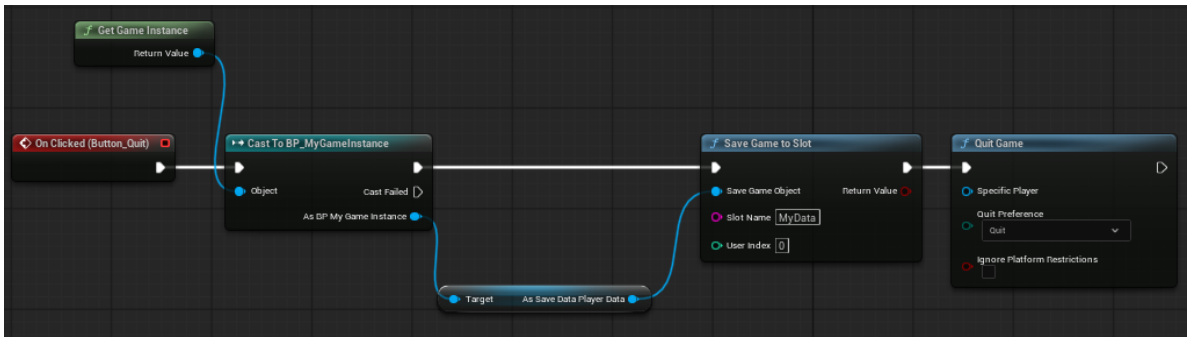


Figura 243: Lógica botón salir del juego.

4.8.3. Menú de Game Over

El Menú de Fin de Partida es una interfaz transitoria que se activa de forma exclusiva cuando la variable de salud del personaje del jugador alcanza el valor cero (0). Su diseño se ha concebido bajo un principio de simplicidad funcional.

Este menú presenta al usuario una única opción operativa: la de Reaparecer (Respawn). Al accionar esta opción, el sistema ejecuta la función de resurrección previamente definida en el gameplay (referida en secciones anteriores), lo cual restaura el estado del personaje y del juego. Tras la invocación exitosa de dicha

función, la interfaz del Menú de Fin de Partida se desactiva y oculta, devolviendo el control al jugador y reanudando la sesión de juego.

Figura 244: Pantalla de Game Over en el juego.

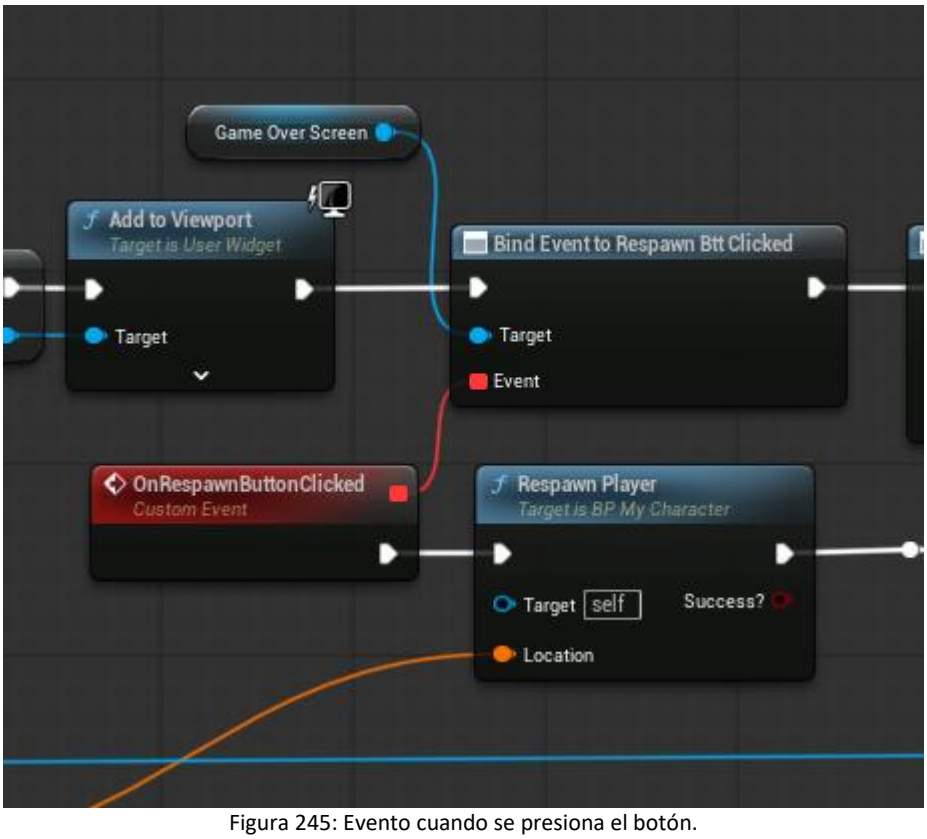


Figura 245: Evento cuando se presiona el botón.

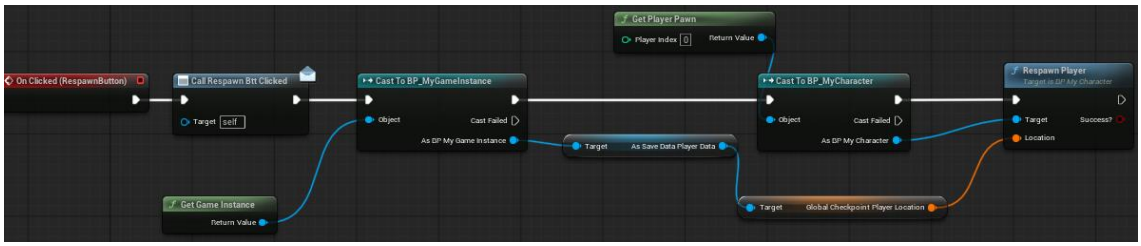


Figura 246: Lógica cuando se presiona el botón de reaparecer.

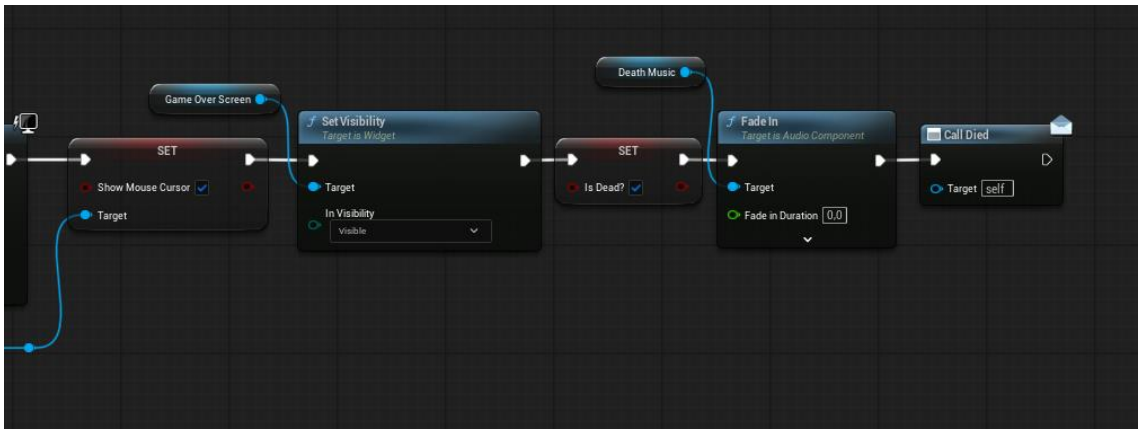


Figura 247: Lógica del Game Over Screen cuando muere el jugador.

4.8.4. Interfaz de la tienda y economía

El videojuego incorpora un sistema de economía y comercio diseñado para reforzar la identidad de las distintas zonas del mundo de juego y aportar profundidad a la gestión de recursos por parte del jugador. Este sistema se basa en la coexistencia de tres monedas diferenciadas, cada una asociada a una región específica del entorno narrativo y jugable.

Las monedas implementadas son Hals, Alas y Nors, correspondientes respectivamente a las zonas del Cielo, la Tierra y el Infierno. Esta separación monetaria tiene como objetivo no solo diversificar la economía del juego, sino también incentivar la exploración y la interacción con los distintos territorios, evitando una economía global única que pudiera trivializar la progresión del jugador.

A lo largo de la partida, el jugador puede interactuar con tres personajes comerciantes, cada uno ubicado en una de las zonas principales del juego. En concreto, existe un comerciante del Cielo, uno de la Tierra y uno del Infierno. Cada comerciante opera exclusivamente con la moneda propia de su región, reforzando la coherencia entre narrativa, ambientación y mecánicas de juego.

Figura 248: Tienda tierra.

Figura 249: Tienda cielo.

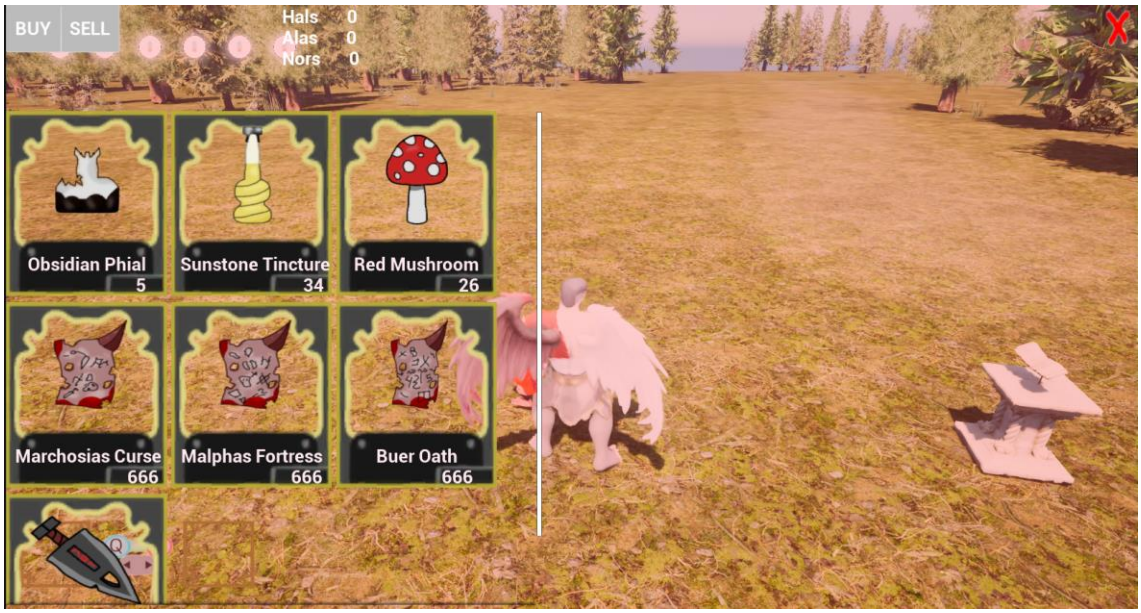


Figura 250: Tienda infierno.

El sistema de comercio permite al jugador vender objetos almacenados en su inventario, facilitando así la obtención de moneda y la gestión estratégica de los recursos acumulados durante la exploración y el combate. Durante las operaciones de compra y venta, el valor de la moneda disponible se actualiza en tiempo real, reflejándose de forma inmediata en la interfaz de usuario para proporcionar al jugador una respuesta clara y directa a cada acción económica realizada.

La interfaz de venta muestra de forma clara los objetos disponibles, su valor económico y la moneda correspondiente, garantizando una interacción intuitiva y accesible. Esta actualización dinámica contribuye a mejorar la sensación de control y feedback del jugador durante las interacciones con los comerciantes.

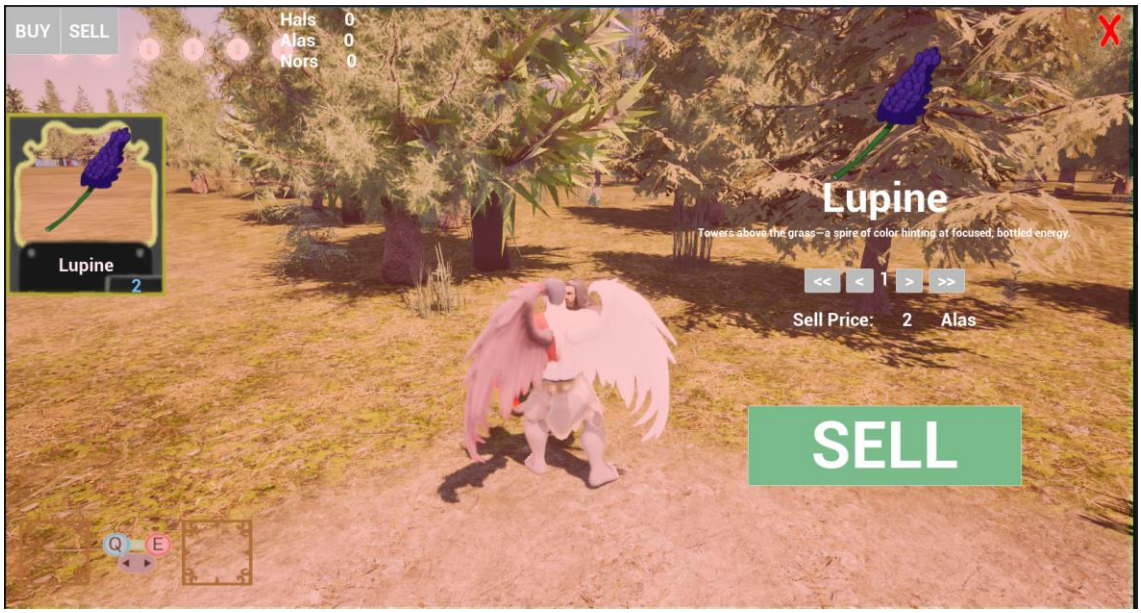


Figura 251: Venta de objetos.

Los objetos que cada comerciante ofrece, así como sus precios, están definidos mediante tablas de datos, lo que permite una separación clara entre lógica y contenido. Estas tablas se recogen en la **Tabla 16**, donde se especifican los artículos disponibles en cada tienda junto con su coste en la moneda correspondiente. También se puede observar en la **Tabla 17** los precios por los que los comerciantes obtienen los bienes del jugador. Este enfoque facilita tanto el equilibrio económico del juego como futuras modificaciones o ampliaciones del contenido sin necesidad de alterar la lógica principal del sistema. Esto se implementa en dos matrices de estructuras dentro de cada uno de los NPC tienda.

Nombre Objeto	Precio	Moneda	Comerciante
Lupine	10	Alas	Tierra
Daisy	10	Alas	Tierra
Agate Drops	40	Alas	Tierra
Jade Brew	40	Alas	Tierra
Dunbrae Sword	400	Alas	Tierra
Daisy	17	Hals	Cielo
Brown Mushroom	25	Hals	Cielo
Lapis Elixir	25	Hals	Cielo
Amethyst Brew	36	Hals	Cielo
Ares Fury	405	Hals	Cielo
Aegis Protection	333	Hals	Cielo
Hermes Smoke	609	Hals	Cielo
Light Wraith	777	Hals	Cielo
Obsidian Phial	5	Nors	Infierno
Sunstone Tincture	34	Nors	Infierno
Red Mushroom	26	Nors	Infierno
Marchosias Curse	666	Nors	Infierno
Malphas Fortress	666	Nors	Infierno
Buer Oath	666	Nors	Infierno
Abyzakar Justice	999	Nors	Infierno

Tabla 17: Todos los objetos en venta de cada comerciante.

Nombre Objeto	Precio comerciante cielo (Hals)	Precio comerciante tierra (Alas)	Precio comerciante infierno (Nors)
Agate Drops	7	5	4
Jade Brew	8	5	4
Lapis Elixir	10	7	10
Amethyst Brew	19	32	28
Sunstone Tincture	7	7	7
Obsidian Phial	0	1	4
Lupine	4	2	2
Daisy	4	2	2

Brown Mushroom	16	10	6
Red Mushroom	2	8	18
Dragon Skull	25	30	30
Meat and Hide	15	15	20

Tabla 18: Todos los objetos vendibles a los comerciantes.

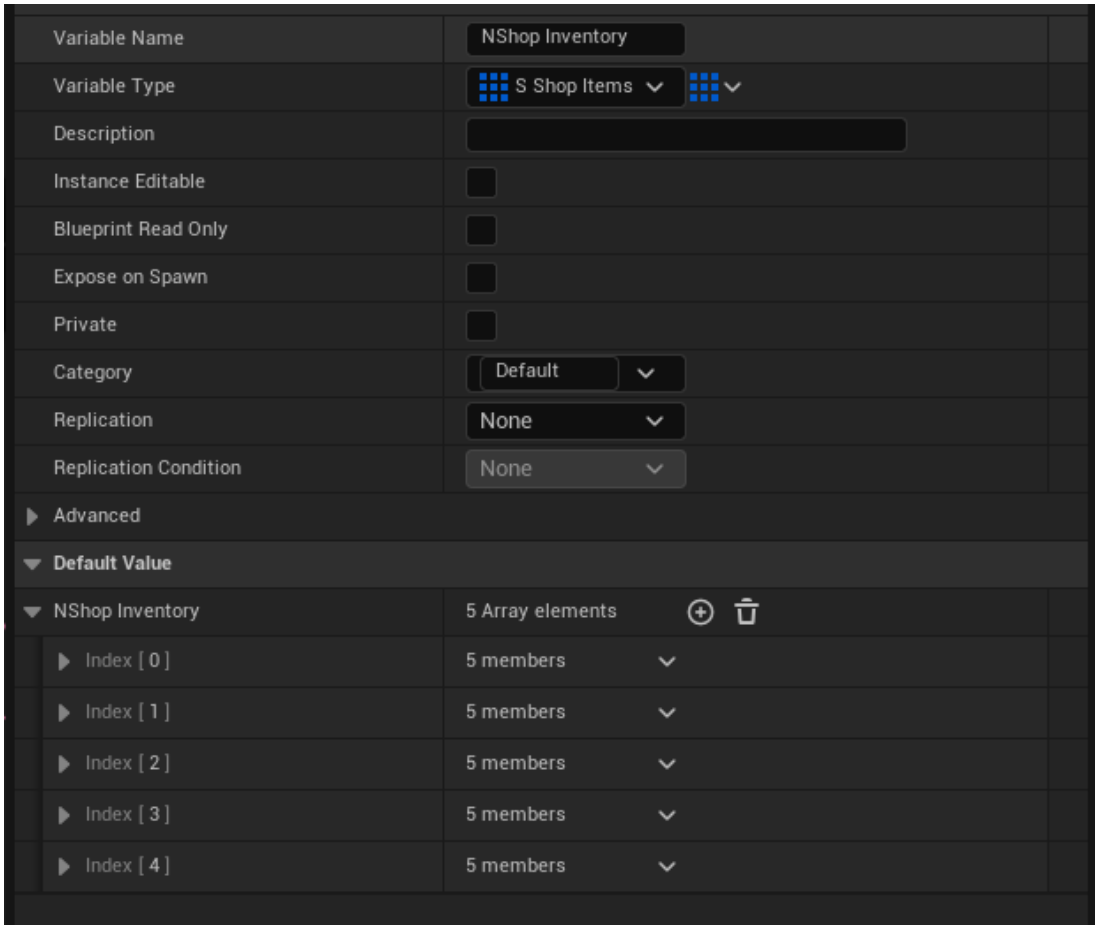


Figura 252: Matriz de objetos que vende el NPC.

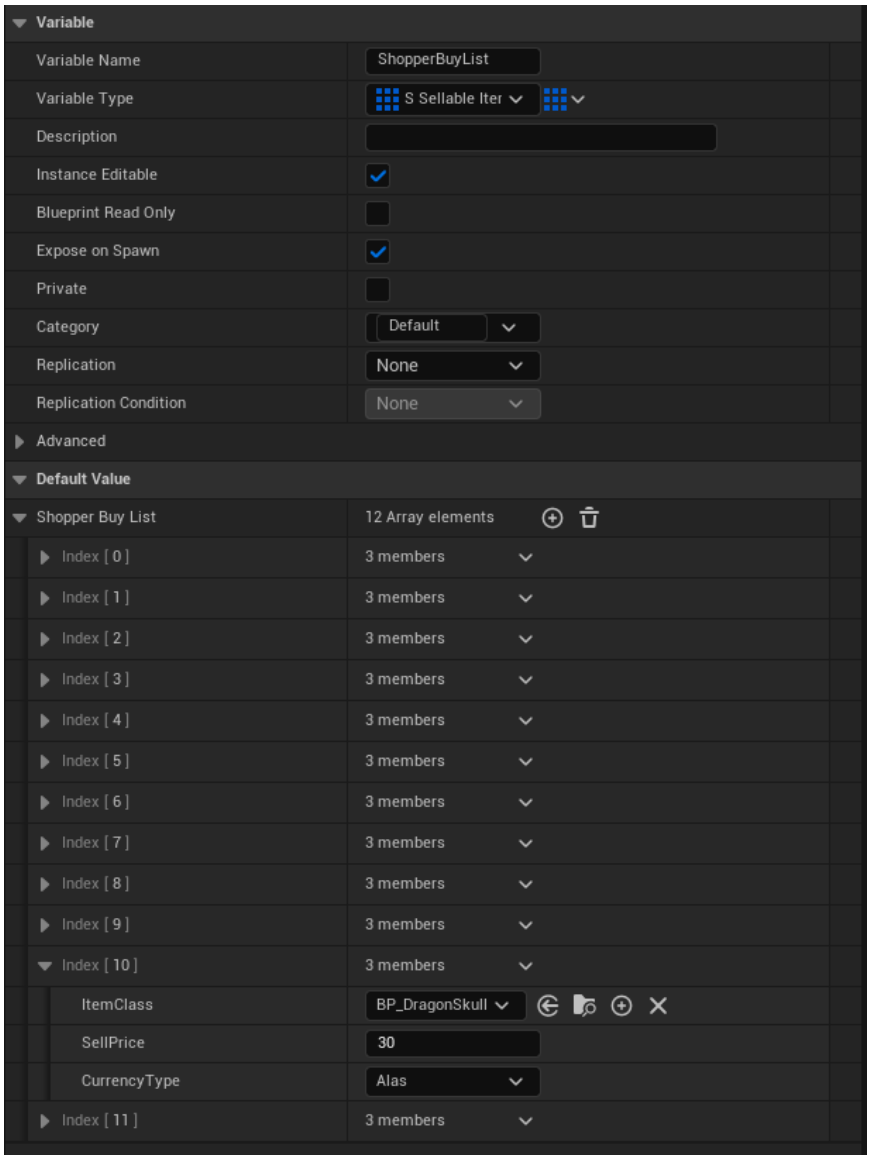


Figura 253: Matriz de objetos que compra el NPC.

En conjunto, la implementación del sistema de economía y comercio contribuye de forma significativa a la profundidad del juego, proporcionando al jugador decisiones estratégicas relacionadas con la gestión del inventario, la obtención de recursos y la planificación de compras. Además, el uso de monedas regionales refuerza la diferenciación entre zonas y apoya la construcción del mundo de juego desde una perspectiva tanto narrativa como mecánica.

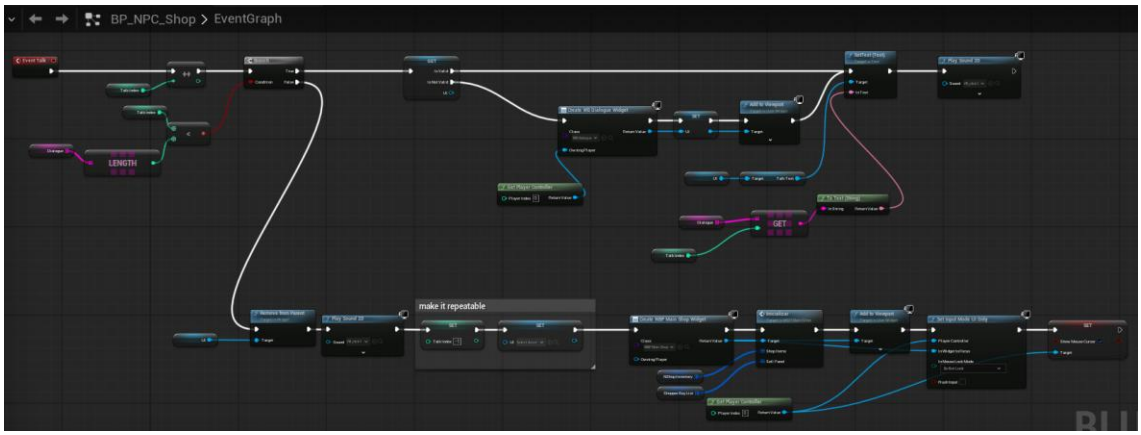


Figura 254: Creación de la tienda.

4.8.5. Interfaz de inventario

El sistema de inventario se ha implementado mediante un Widget Blueprint [17] que utiliza un Widget Switcher para gestionar las distintas pestañas disponibles, que a su vez también son Widget Blueprints [17]. El acceso al inventario se realiza mediante la pulsación de la tecla I, permitiendo al jugador consultar y gestionar los distintos elementos obtenidos durante el desarrollo del juego.

La lógica de apertura del inventario se lleva a cabo en BP_MyCharacter con la utilización de una función Toggle:

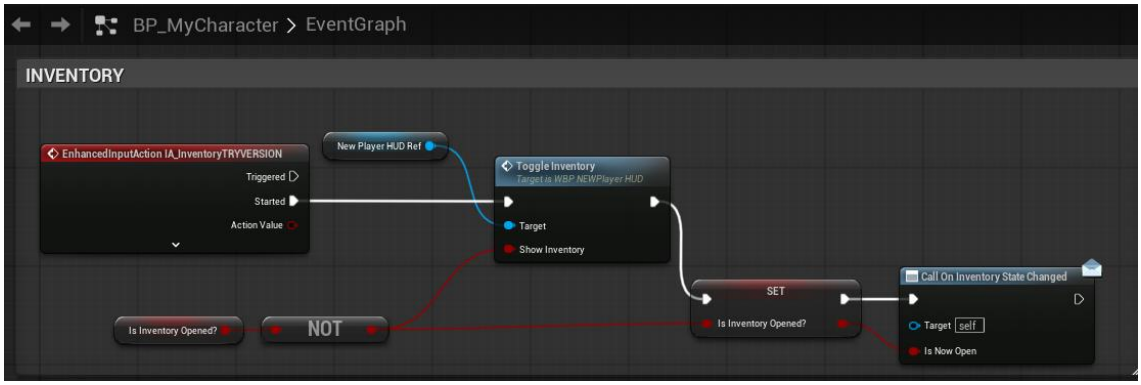


Figura 255: Lógica de apertura de inventario en BP_MyCharacter.

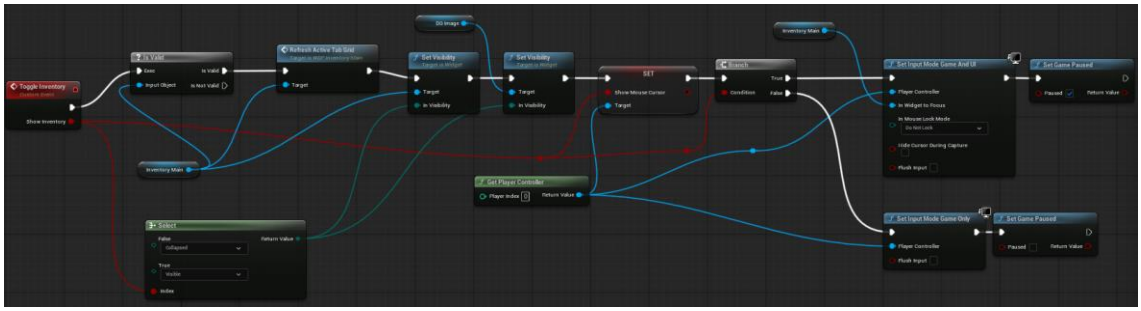


Figura 256: Lógica interna de apertura de inventario.

Este sistema se estructura en un total de **cinco pestañas**, cada una orientada a un tipo concreto de información o gestión de recursos:

- **Armas**: destinada al almacenamiento y equipamiento de las armas obtenidas por el jugador.

- **Magia**: dedicada a la gestión de plantas, pociones y etiquetas, así como a la creación y uso de pociones.

- **Diario de acompañantes**: muestra los acompañantes domesticables y permite su equipamiento para apoyar al jugador en combate.

- **Misiones**: recoge tanto las misiones principales como las secundarias, activas y completadas, junto con su información relevante.

- **Bestiario**: registra las criaturas avistadas a lo largo del juego y gestiona las recompensas asociadas a su completado.

Además en el panel de equipamiento se integra la interfaz de visualización de las divisas del sistema.
Dicho sistema monetario se compone de tres tipos de divisa: **Hals**, **Alas** y **Nors**, que se corresponden, respectivamente, con las representaciones conceptuales de Cielo, Tierra e Infierno.
La funcionalidad de estas divisas abarca su utilización en los puntos de venta (tiendas del entorno de juego). Asimismo, su mecanismo de adquisición se articula mediante la obtención como recompensa tras la finalización exitosa de misiones o a través de la derrota de enemigos.

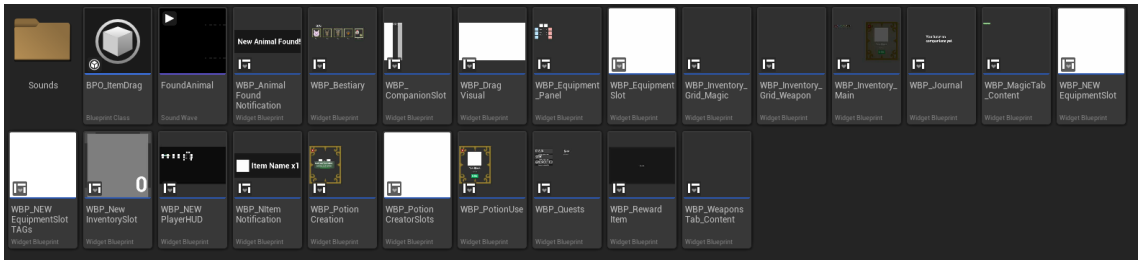


Figura 257: Todos los Widget Blueprint [17] utilizados para el inventario.

Asimismo, la totalidad de la lógica funcional del sistema de inventario se encuentra encapsulada en el componente **BPAC_NEWInventoryComponent**, responsable de la inicialización estructural del inventario y de la carga de los objetos correspondientes en sus respectivos espacios. Este componente actúa como núcleo de gestión de datos, centralizando las operaciones de almacenamiento, consulta, modificación y persistencia de la información relativa a los ítems.

Entre las principales funciones implementadas en dicho componente, destacan las siguientes:

- **Add Item**: gestiona la adición de nuevos objetos al inventario, verificando previamente la existencia del ítem. En caso de encontrarse ya registrado, incrementa su cantidad acumulada hasta alcanzar el límite máximo permitido.

- **GetInventoryItems**: función de consulta (*getter*) destinada a la obtención del conjunto actual de objetos almacenados.

- **InitializeInventory**: inicializa la totalidad de los espacios (*slots*) del inventario, estableciendo su estado base.

- **Check For Empty Slots**: identifica y devuelve el primer espacio vacío disponible para el almacenamiento de un nuevo objeto, en caso de existir.

- **Check For Existing Item Slot**: dado un objeto específico, localiza el espacio en el que se encuentra almacenado si este ya existe dentro del inventario.

- **RemoveItemByID**: elimina del inventario la totalidad de las instancias asociadas a un objeto concreto a partir de su identificador.

- **GetItemQuantity**: devuelve la cantidad actual almacenada de un objeto determinado.

- **RemoveItemAmount**: sustrae una cantidad específica de un objeto dentro del inventario, ajustando su valor acumulado.

- **SaveInventory**: gestiona la persistencia de los datos del inventario mediante su almacenamiento en un objeto de tipo *Save Game*.

- **LoadInventory**: permite la recuperación y carga del estado del inventario previamente guardado desde el objeto *Save Game*.

- **FindItemInventory**: dada la identificación de un objeto, devuelve la instancia completa del mismo junto con un valor booleano que indica si ha sido localizado satisfactoriamente.

Adicionalmente, el componente incorpora un sistema de comunicación basado en **Event Dispatchers [10]**, orientado a la notificación de cambios relevantes dentro del sistema. Se implementan un total de cuatro:

- **OnItemAdded**: notifica la incorporación de un nuevo objeto al inventario.

- **OnInventoryInitialized**: se ejecuta tras la inicialización completa de la estructura del inventario.

- **OnInventoryUpdated**: informa de cualquier modificación producida en el contenido del inventario.

- **OnBestiaryUnlocked**: comunica el desbloqueo de la pestaña correspondiente al bestiario dentro de la interfaz.

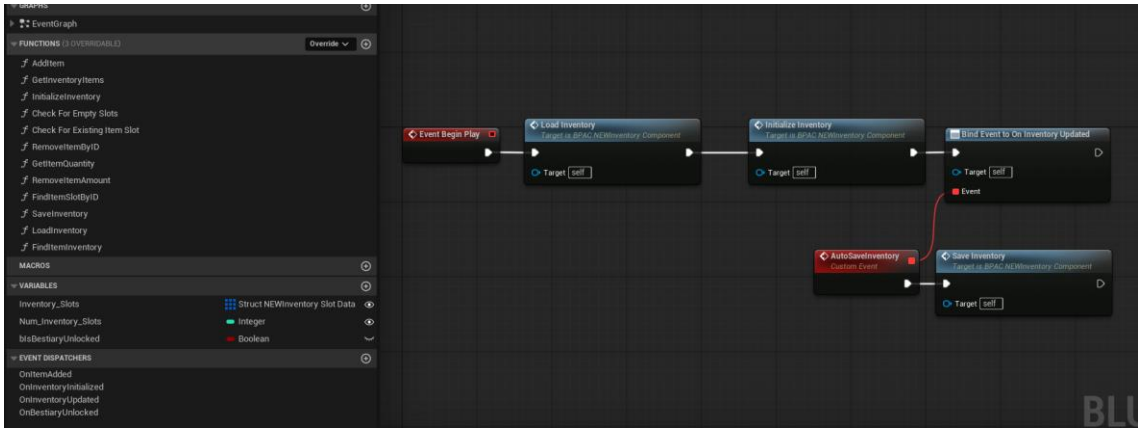


Figura 258: Event Graph de BPAC_NEWInventoryComponent.

4.8.5.1. Armas

La primera pestaña corresponde a la gestión de armas, identificada mediante la pestaña representada con una **W** (*Weapon*). En esta sección se almacenan todas las armas obtenidas por el jugador a lo largo del juego.

Las principales características de este sistema son las siguientes:

- Las armas son objetos únicos y, por tanto, no pueden acumularse en pilas superiores a una unidad.
- Cada arma ocupa un único espacio dentro del inventario.
- En el lado derecho de la interfaz se muestra el menú de equipamiento correspondiente a las armas.

El proceso de equipamiento se realiza de la siguiente manera:

- El jugador puede arrastrar una arma desde el inventario hasta cualquiera de los dos *slots* superiores del menú de equipamiento.
- Al soltar el arma en uno de estos espacios, esta se equipa automáticamente.
- Para desequipar un arma, el jugador debe realizar un clic derecho sobre la misma dentro del inventario.

Este diseño permite una gestión rápida del equipamiento sin necesidad de menús adicionales.

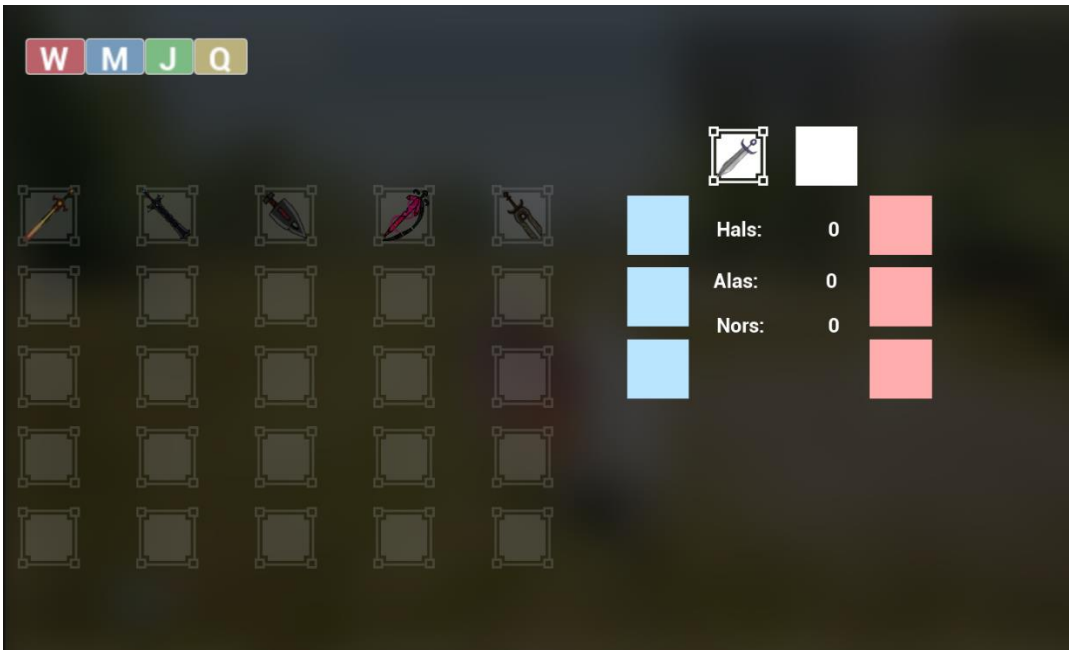


Figura 259: Inventario en pestaña de armas.

En lo relativo al subsistema de armas, la lógica de equipamiento no se gestiona directamente desde el componente general de inventario, sino que se encuentra delegada en los **Widget Blueprints [17]** específicos asociados a los espacios de equipamiento. Esta decisión de diseño responde a la necesidad de vincular de forma directa la interacción de la interfaz con las operaciones de equipamiento, favoreciendo una respuesta inmediata a las acciones del usuario.

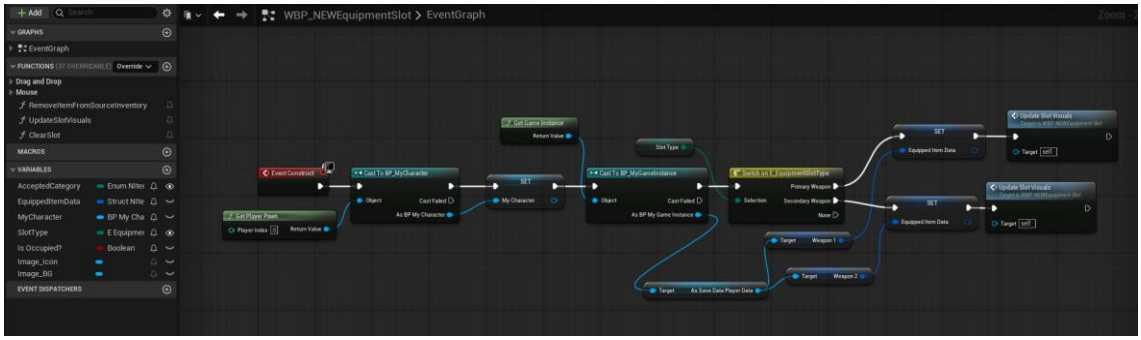


Figura 260: Lógica slots de equipamiento de armas.

Las principales funciones implementadas en dichos *widgets [17]* son las siguientes:

- **On Drop**: gestiona la lógica asociada al evento de arrastre y liberación de un arma sobre un espacio de equipamiento. En primera instancia, el sistema verifica si el espacio de destino se encuentra ya ocupado; en caso afirmativo, la operación no se ejecuta. Si el espacio está disponible, se procede a comprobar que el objeto depositado corresponde tipológicamente a un arma de tipo espada, restringiendo así el equipamiento a los ítems válidos.

  Una vez validado el tipo de arma, la función determina el espacio concreto de equipamiento en el que se ha realizado la acción (mano izquierda o mano

derecha). En función de ello, se actualiza la representación visual del espacio, se almacena la referencia del arma en la variable correspondiente (**Weapon 1** o **Weapon 2**) y se ejecuta el proceso de guardado de la partida.

Adicionalmente, en este mismo flujo lógico se integra una comprobación específica vinculada al sistema de tutorial, destinada a verificar si el arma equipada corresponde a la requerida para el avance de dicha fase formativa.

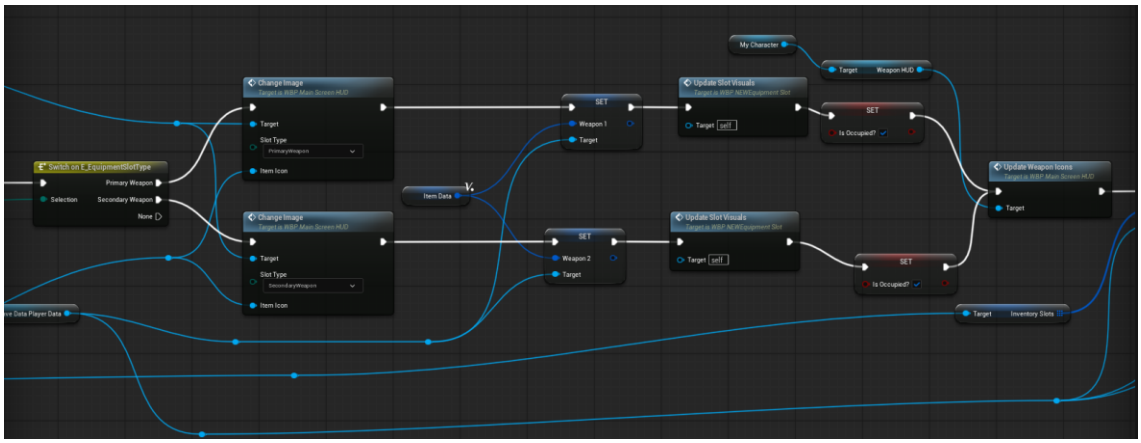


Figura 261: Lógica al soltar un arma en el slot de equipamiento de armas (OnDrop).

- **On Mouse Button Down**: administra la interacción mediante clic del usuario sobre los espacios de equipamiento. Concretamente, al detectarse una pulsación del botón derecho del ratón, el sistema evalúa si existe un arma equipada en el espacio seleccionado. En caso afirmativo, se procede a su desequipamiento, ejecutando la totalidad de la lógica asociada a la retirada del arma y a la actualización de su estado dentro del inventario.

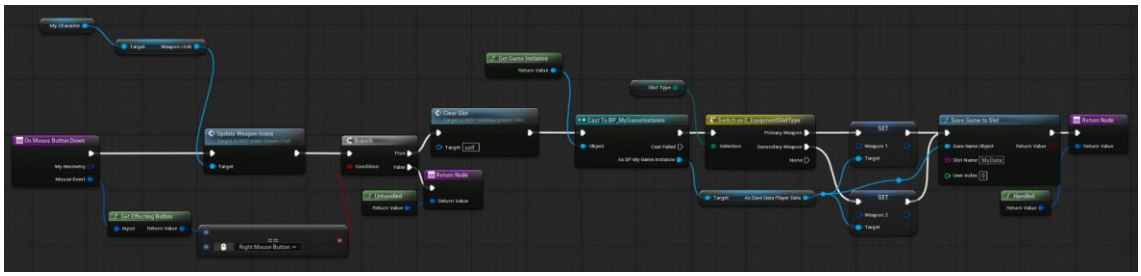


Figura 262: Lógica para quitar un arma del slot de equipamiento de armas.

- **UpdateVisuals**: función auxiliar encargada de actualizar los elementos gráficos asociados al espacio de equipamiento, particularmente los iconos representativos de las armas equipadas, garantizando la coherencia visual de la interfaz.

- **ClearSlot**: función auxiliar destinada a la limpieza completa del espacio de equipamiento cuando se produce el desequipamiento de un arma, eliminando tanto su representación visual como sus referencias lógicas asociadas.

Complementariamente, el subsistema de armas hace uso de *Widget Blueprints [17]* especializados que estructuran y gestionan la visualización de los objetos dentro de la interfaz de inventario.

En primer lugar, el **WBP_Inventory_Grid_Weapon** actúa como elemento responsable de la inicialización de la totalidad de los espacios (*slots*) destinados al almacenamiento de armas. Este *widget* se encarga, asimismo, de ejecutar los procesos de actualización visual del contenido, refrescando dinámicamente la información mostrada a partir de los datos procedentes del componente de inventario. De este modo, garantiza que los objetos representados en la cuadrícula se correspondan en todo momento con el estado real del inventario del jugador.

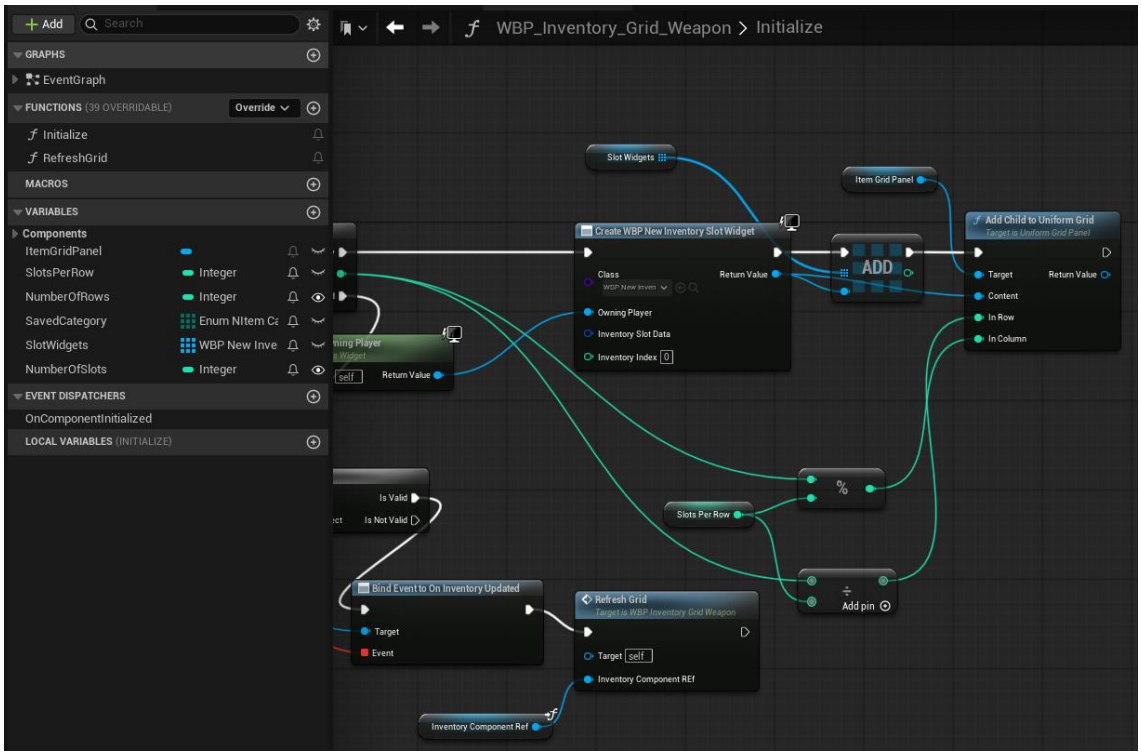


Figura 263: Lógica para la creación de la matriz de slots de armas.

Por otra parte, el **WBP_WeaponsTab_Content** cumple una función estructural dentro de la pestaña de armas, encargándose de la inicialización global del inventario asociado a dicha categoría. Para ello, se apoya en las funciones previamente implementadas en **WBP_Inventory_Grid_Weapon**, reutilizando su lógica de creación de espacios y carga de información. Esta relación jerárquica permite modularizar el sistema, favoreciendo la reutilización de funcionalidades y optimizando la organización interna de la interfaz.

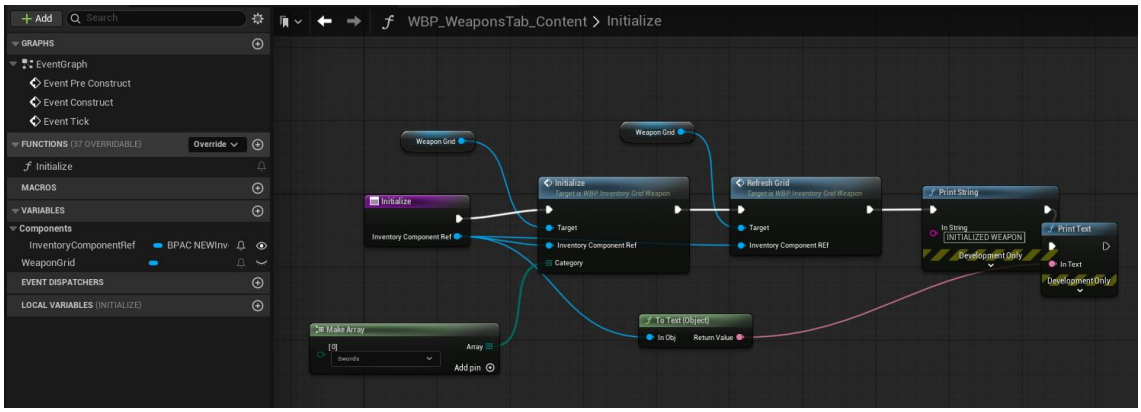


Figura 264: Lógica para la inicialización del inventario de armas.

4.8.5.2. Magia

La segunda pestaña corresponde al sistema de magia, accesible mediante la pestaña representada con una **M** (*Magic*). En esta sección se almacenan los objetos relacionados con la modificación de estadísticas y efectos especiales.

Dentro de esta pestaña se distinguen los siguientes tipos de objetos:

- **Plantas:**
  - Se pueden acumular en pilas de hasta 200 unidades.
  - Esta cantidad se ha establecido para evitar la saturación visual del inventario.

- **Pociones:**
  - Se pueden acumular en pilas de hasta 16 unidades.
  - Este límite permite al jugador disponer de una cantidad razonable sin eliminar la necesidad de gestión de recursos.

- **Etiquetas (bendiciones y maldiciones):**
  - Son objetos únicos.
  - Solo pueden apilarse de una en una.

En esta pestaña también se incluye, de forma permanente, el menú de equipamiento de etiquetas, situado en el lado derecho de la interfaz. Su funcionamiento es el siguiente:

- Las bendiciones pueden arrastrarse al *slot* izquierdo.
- Las maldiciones pueden arrastrarse al *slot* derecho.
- Para retirar una etiqueta equipada, se utiliza el clic derecho sobre el objeto correspondiente.

Figura 265: Pestaña de magia.

Desde el punto de vista de implementación, esta pestaña presenta una estructura funcional análoga a la descrita previamente para el subsistema de armas, si bien incorpora un grado adicional de complejidad derivado de la coexistencia de tres tipologías diferenciadas de objetos (plantas, pociones y etiquetas), cada una con reglas de almacenamiento, uso y equipamiento específicas.

En primer lugar, se aborda el tratamiento de las **etiquetas** (bendiciones y maldiciones). Estas disponen de espacios de equipamiento dedicados dentro del panel lateral derecho de la interfaz. La lógica de gestión de dichos espacios se encuentra implementada en el *Widget Blueprint* [17] **WBP_NEWEquipmentSlotTAGs**.

El funcionamiento de este *widget* sigue los mismos principios estructurales y las mismas funciones base descritas para los espacios de equipamiento de armas (gestión de eventos de arrastre, validación de objeto, equipamiento, desequipamiento y actualización visual). No obstante, introduce una capa adicional de validación tipológica: en lugar de verificar si el objeto corresponde a un arma y determinar su asignación a mano izquierda o derecha, el sistema comprueba que el ítem sea una etiqueta y que su naturaleza coincida con el espacio de destino. De este modo, las **bendiciones** únicamente pueden equiparse en el espacio reservado para bendiciones, mientras que las **maldiciones** quedan restringidas a su espacio correspondiente, evitando configuraciones inválidas.

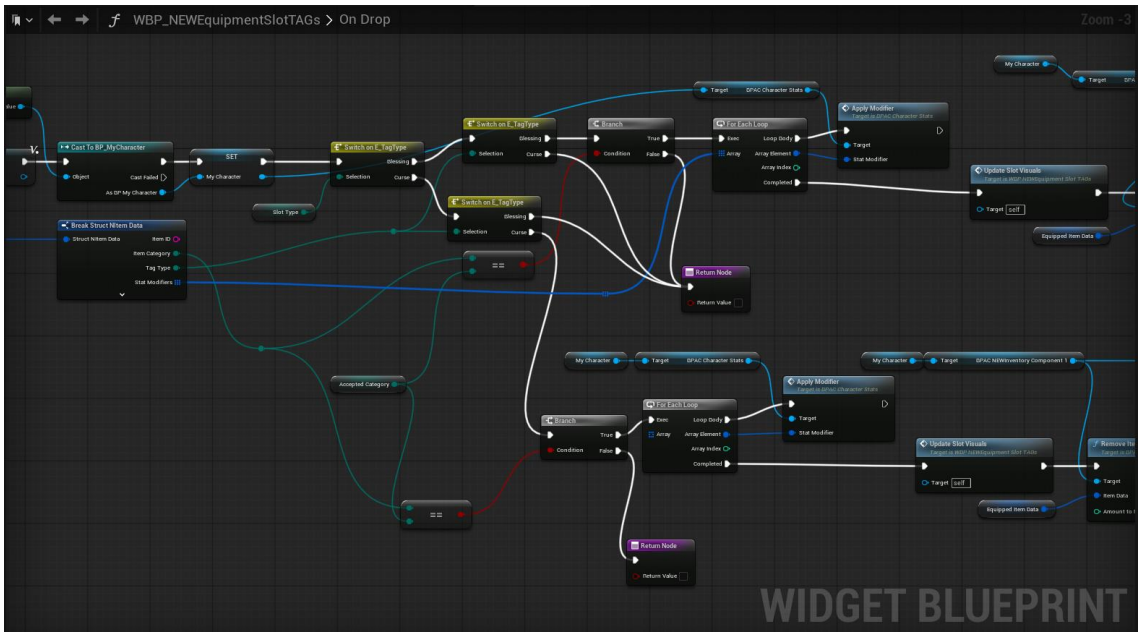


Figura 266: Parte de la lógica de etiquetas que se diferencia de la de armas.

En lo relativo a la estructura de visualización del inventario mágico, los *Widget Blueprints [17]* **WBP_Inventory_Grid_Magic** y **WBP_MagicTab_Content** replican funcionalmente la arquitectura establecida en sus equivalentes del sistema de armas. El primero se encarga de la inicialización y actualización de los espacios destinados al almacenamiento de objetos mágicos, mientras que el segundo actúa como contenedor estructural de la pestaña, coordinando la carga e inicialización de los datos mediante las funciones implementadas en la cuadrícula.

Como elemento diferencial, **WBP_MagicTab_Content** incorpora un botón adicional cuya funcionalidad específica se desarrollará en apartados posteriores, al encontrarse vinculada a los sistemas de uso y elaboración de objetos alquímicos.

Finalmente, cabe señalar que en esta pestaña se centralizan también las mecánicas relativas al **consumo y creación de pociones**, procesos que, debido a su complejidad sistémica y a su integración con otros subsistemas (recolección, recetas, efectos y modificación de estadísticas), serán analizados con mayor profundidad en secciones subsiguientes.

Creación de pociones

En relación con la funcionalidad de creación de pociones, esta se implementa mediante la interacción con el botón **Create Potion**, ubicado dentro de la pestaña de magia en el *Widget Blueprint [17]* **WBP_MagicTab_Content**. Dicho elemento de interfaz se encuentra inicialmente bloqueado y su desbloqueo se produce tras la interacción del jugador con el personaje de la bruja, en el contexto de una misión secundaria específica.

Una vez habilitado, la activación del botón provoca que un nuevo *Widget Blueprint [17]*, denominado **WBP_PotionCreation**, pase a estado visible, superponiéndose sobre el panel de equipamiento dentro de la misma pestaña de magia.

Este menú de creación incorpora una interfaz funcional compuesta por dos espacios (*WBP_PotionCreatorSlots*) destinados a la inserción de ingredientes, configurados para albergar las plantas necesarias para la elaboración de pociones. Estos siguen una lógica parecida a los otros dos slots vistos anteriormente.

El proceso de creación se articula conforme a la siguiente secuencia operativa:

- El jugador selecciona las plantas requeridas desde el inventario y las arrastra hacia cualquiera de los dos espacios disponibles.

- El orden de colocación de los ingredientes no resulta determinante, siempre que la combinación introducida se corresponda con una receta válida registrada en el sistema.

- Los ingredientes depositados pueden retirarse en cualquier momento mediante la interacción de clic derecho sobre el espacio correspondiente.

- Una vez introducidos ambos ingredientes, la activación del botón de creación ejecuta la validación de la receta. En caso de coincidencia, la poción resultante se genera y se añade automáticamente al inventario del jugador.

- Si el botón de creación es accionado sin haber introducido ingredientes, o habiendo introducido únicamente uno, el sistema devuelve una notificación de error, impidiendo la ejecución del proceso hasta que se cumplan los requisitos mínimos de combinación.

	Lupine	**Daisy**	**Brown Mushroom**	**Red Mushroom**	**Dragon Skull**	**Meat and Hide**
Lupine	Obsidian Phial	Jade Brew	Lapis Elixir	Obsidian Phial	Obsidian Phial	Obsidian Phial
Daisy	Jade Brew	Jade Brew	Sunstone Tincture	Amethyst Brew	Obsidian Phial	Obsidian Phial
Brown Mushroom	Lapis Elixir	Sunstone Tincture	Obsidian Phial	Agate Drops	Obsidian Phial	Obsidian Phial
Red Mushroom	Obsidian Phial	Amethyst Brew	Agate Drops	Agate Drops	Obsidian Phial	Obsidian Phial
Dragon Skull	Obsidian Phial	Obsidian Phial	Obsidian Phial	Obsidian Phial	Obsidian Phial	Obsidian Phial
Meat and Hide	Obsidian Phial	Obsidian Phial	Obsidian Phial	Obsidian Phial	Obsidian Phial	Obsidian Phial

Tabla 19: Todas las combinaciones de pociones posibles.

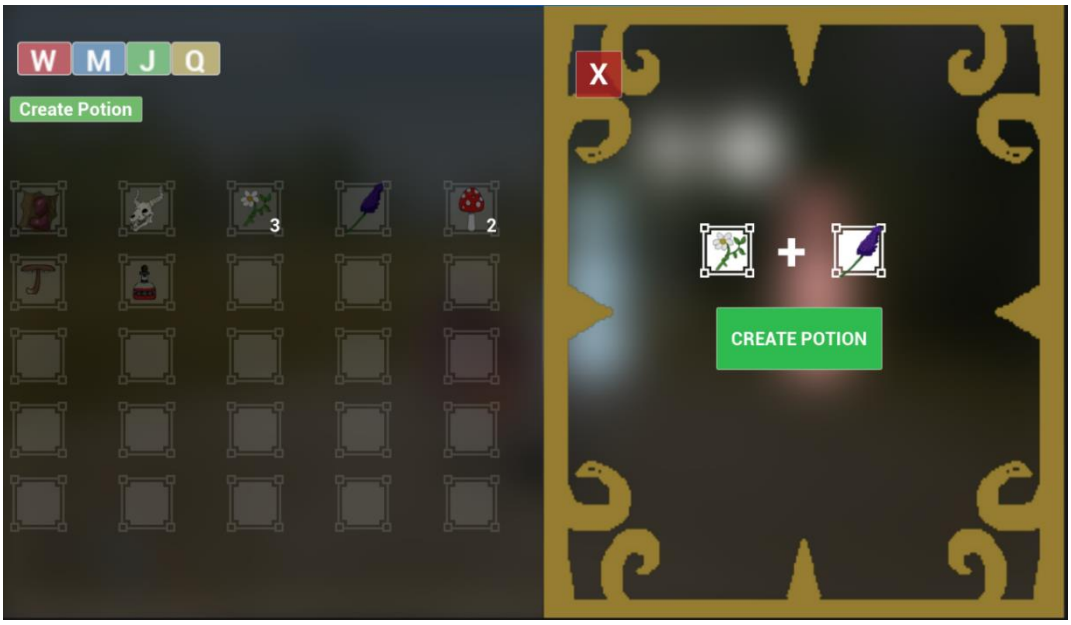


Figura 267: Menú de creación de pociones.

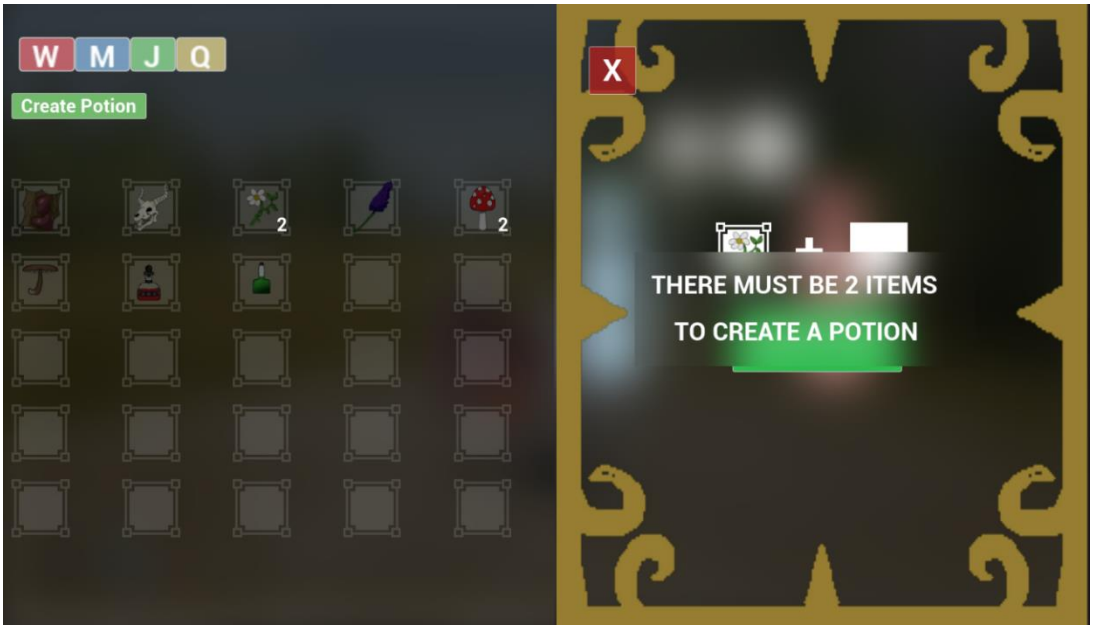


Figura 268: Error de creación de pociones.

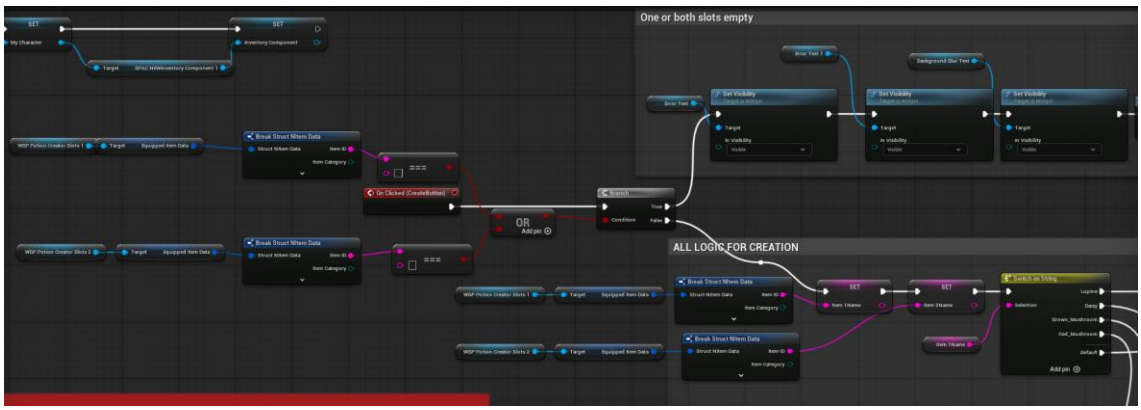


Figura 269: Lógica al presionar el botón de creación de pociones.

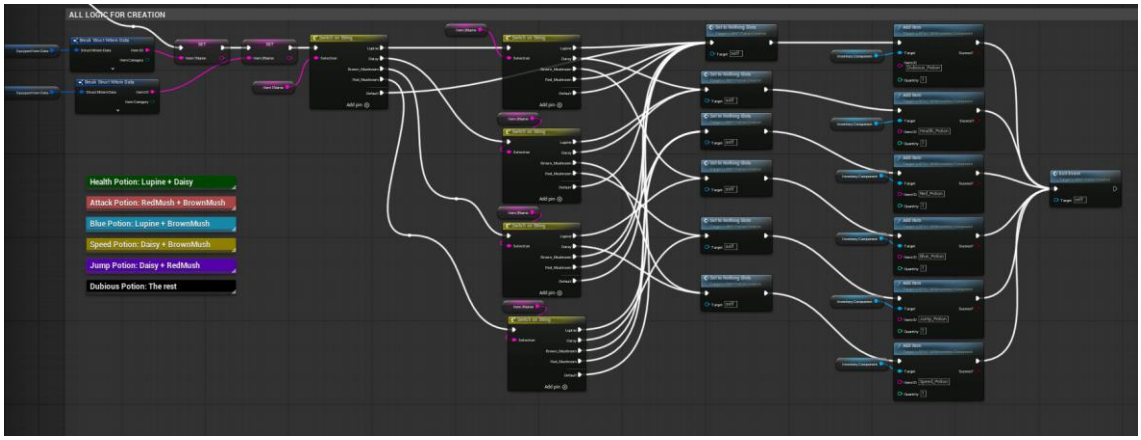


Figura 270: Lógica de creación de pociones.

Uso de pociones

Adicionalmente, las pociones pueden ser utilizadas de forma directa desde el propio inventario, sin necesidad de ser equipadas previamente en un espacio específico.
El procedimiento de uso se articula mediante la interacción contextual del usuario:

- Al realizar clic derecho sobre una poción almacenada en el inventario, se despliega un menú informativo lateral situado en el lado derecho de la interfaz. Dicho panel se encuentra implementado mediante el *Widget Blueprint* [17] **WBP_PotionUse**.

- En este menú se presenta información detallada del objeto seleccionado, incluyendo su descripción funcional y los efectos asociados a su consumo.

Una vez abierto el panel informativo, el jugador dispone de la opción de activar el uso de la poción mediante el botón correspondiente:

- Al seleccionar la opción de uso, el sistema ejecuta la lógica de consumo del objeto y procede a la activación del efecto temporal asociado a la poción, conforme a los parámetros y comportamientos previamente descritos en el apartado 4.2.2.

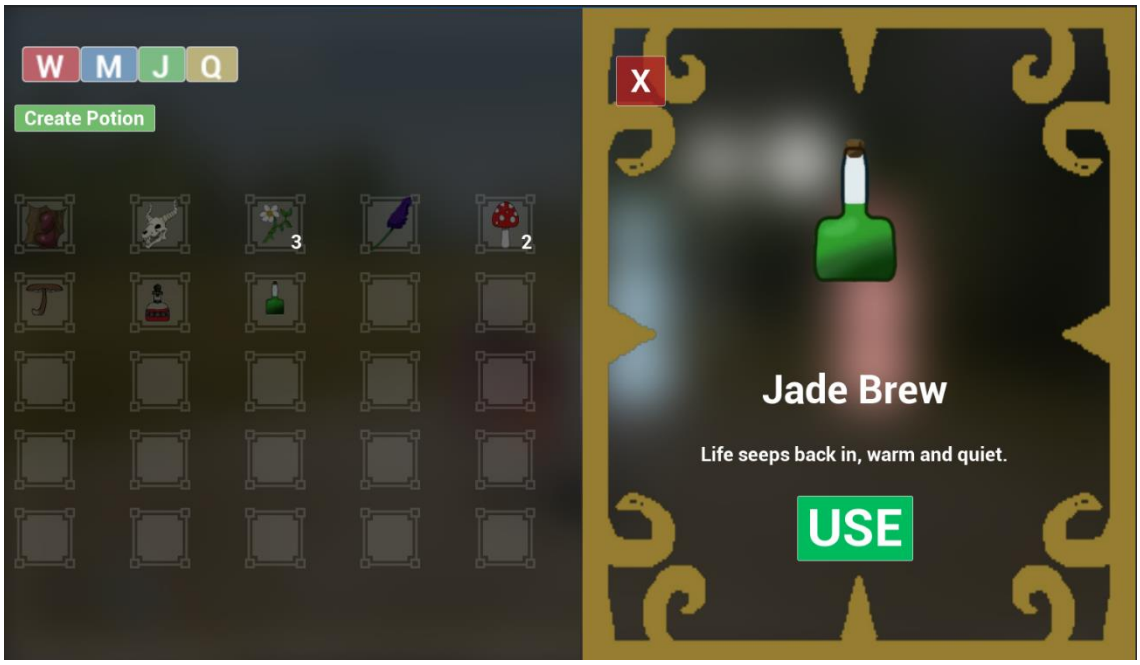


Figura 271: Menú de uso de pociones.

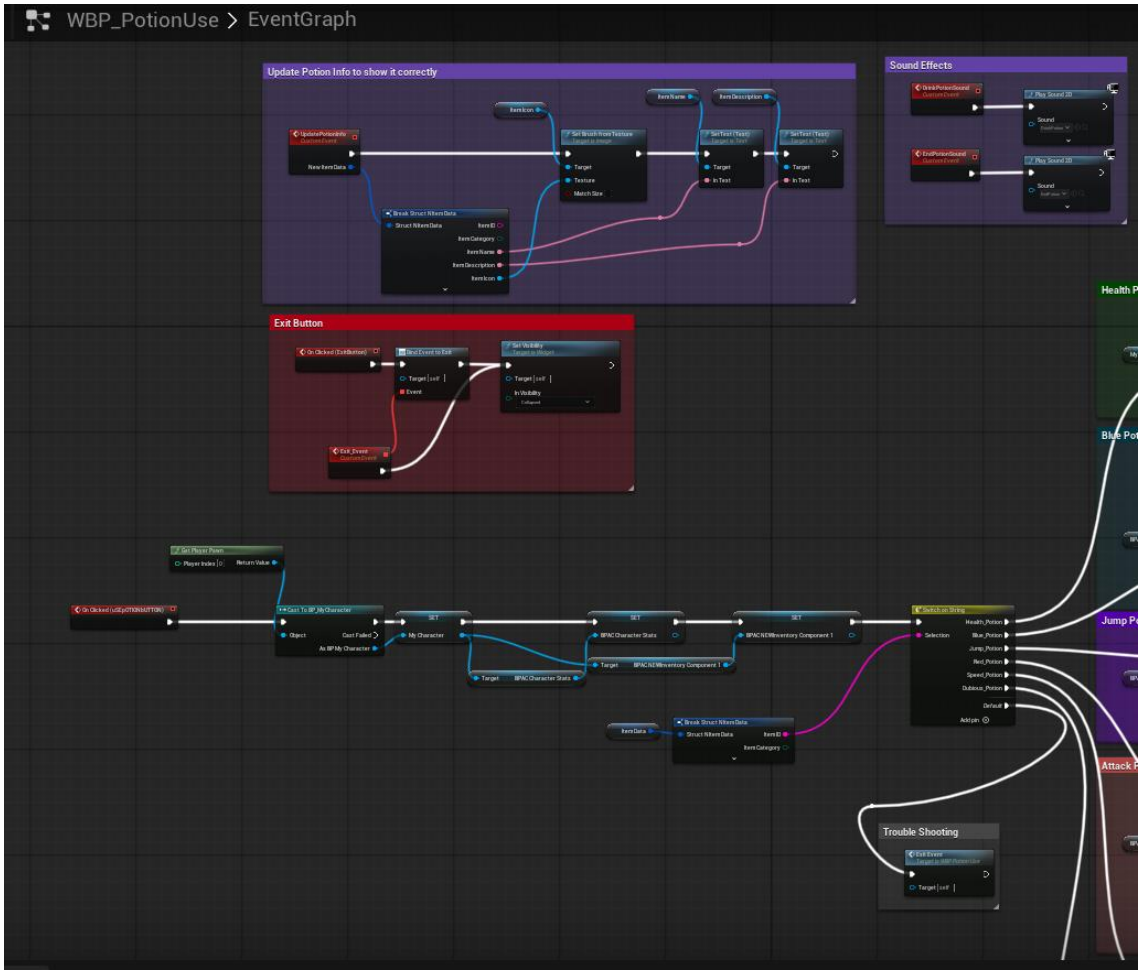


Figura 272: Lógica de menú de uso de pociones.

4.8.5.3. Diario de acompañantes

Desde el punto de vista de implementación, el sistema asociado al **Diario de acompañantes** se ha desarrollado mediante el uso de diversos *Widget Blueprints [17]* especializados, orientados tanto a la representación visual de la información como a la gestión de su lógica funcional.

En primer lugar, se dispone del *Widget Blueprint [17]* **WBP_CompanionSlot**, el cual constituye la unidad básica de representación dentro de la interfaz. Este *widget* es instanciado dinámicamente cada vez que un nuevo acompañante es domesticado. Su función principal consiste en mostrar la información específica asociada al acompañante correspondiente (en la versión actual, lobos), incluyendo sus datos identificativos y su estado dentro del sistema.

Por otra parte, el *Widget Blueprint [17]* **WBP_Journal** actúa como contenedor estructural del diario. Este elemento es responsable de recibir la información relativa a los acompañantes desbloqueados, actualizar la lista visible en la interfaz y gestionar la lógica asociada a su estado de equipamiento.

En este sentido, **WBP_Journal** administra los procesos de activación y desactivación de acompañantes, determinando cuáles se encuentran equipados en cada momento y garantizando el cumplimiento de las restricciones establecidas por el sistema (tanto el límite técnico máximo como la limitación aplicada en la versión demo).

Esta arquitectura modular, basada en la separación entre unidades de representación (*slots*) y el contenedor lógico principal, permite una gestión escalable del sistema, facilitando la futura incorporación de nuevos acompañantes sin necesidad de modificar la estructura base de la interfaz.

Figura 273: Menú de equipamiento de acompañantes.

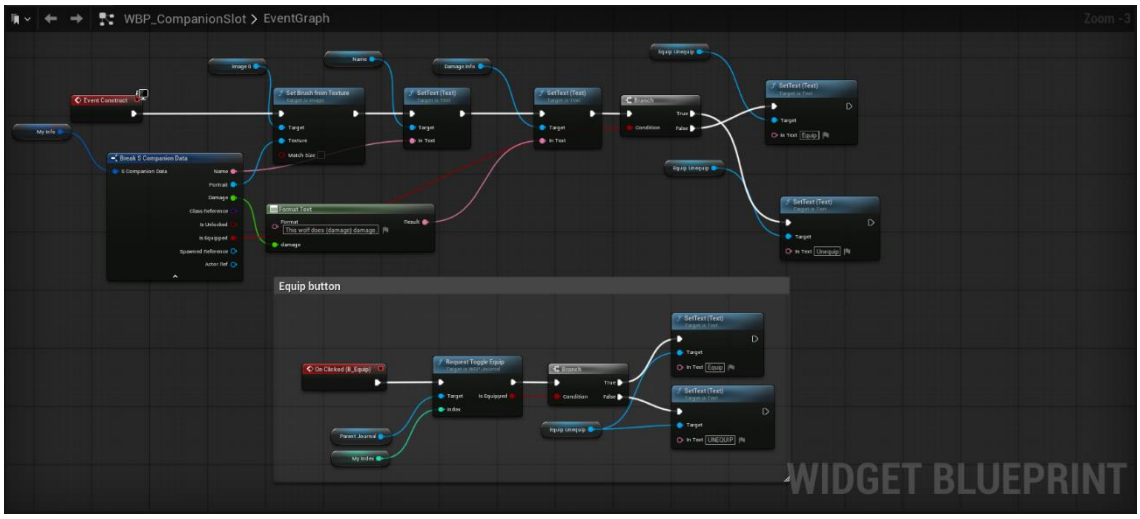


Figura 274: Lógica menú de equipamiento de acompañantes.

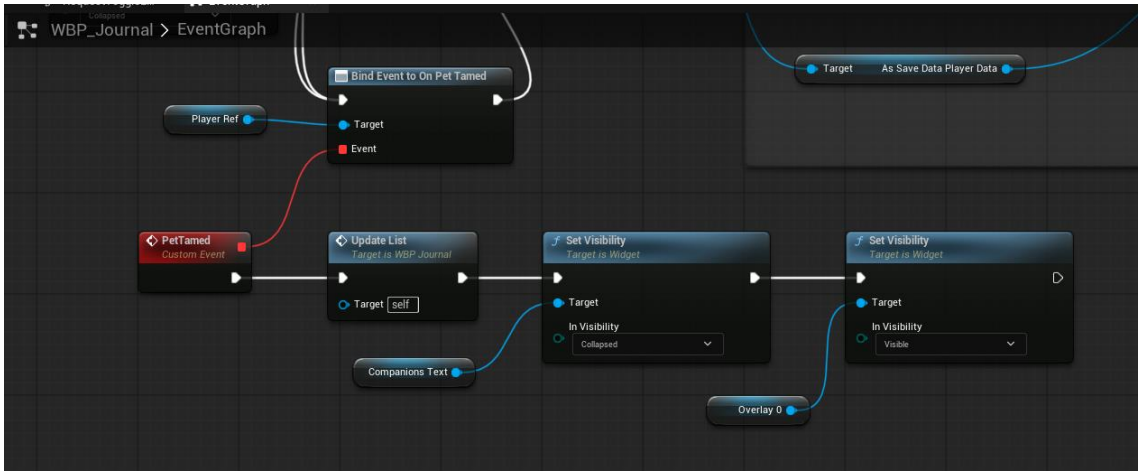


Figura 275: Lógica de añadir un acompañante.

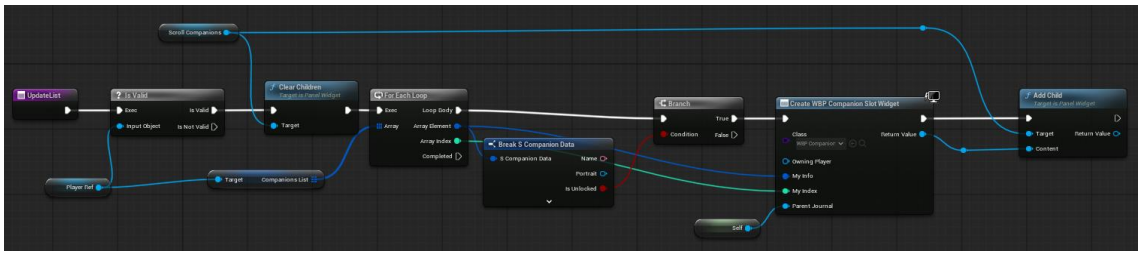


Figura 276: Lógica para actualizar los slots de WBP_Journal.

4.8.5.4. Misiones

Desde el punto de vista de implementación, el sistema de misiones se articula principalmente a través del *Widget Blueprint [17]* **WBP_Quests**, el cual centraliza tanto la representación visual como la lógica de actualización de la información mostrada en la pestaña.

Dentro de este *widget* se han implementado diversas funciones destinadas a garantizar la correcta organización, clasificación y actualización dinámica de las misiones.

El **Event Graph** actúa como núcleo de ejecución lógica, encargándose de la actualización de las listas de misiones mediante la invocación de la función auxiliar **RefreshQuestLists**. Esta función gestiona la distribución automática de cada misión dentro del componente **Expandable Area** que le corresponde, atendiendo tanto a su tipología como a su estado actual. En consecuencia, las misiones se clasifican en las siguientes categorías visuales:

- Misiones principales activas.
- Misiones secundarias activas.
- Misiones completadas, con independencia de su tipología original.

Adicionalmente, el *Event Graph* se encuentra suscrito al **Event Dispatcher [10] UpdateQuestStatus**, a través del cual recibe notificaciones relativas a cambios en el estado de las misiones. Al activarse dicho evento, el sistema procede a actualizar la misión correspondiente, reflejando su nueva condición dentro de la estructura de listas y reasignándola, si procede, al área expandible adecuada.

Por otra parte, se ha implementado un evento específico asociado a la interacción del usuario con la pestaña de misiones. Cada vez que el jugador accede a esta sección mediante el botón correspondiente del inventario, se ejecuta un proceso de refresco de la información detallada de la misión seleccionada. Esta operación se lleva a cabo mediante la llamada a la función auxiliar **UpdateSelectedQuestDetails**, responsable de actualizar los datos descriptivos y las recompensas asociadas que se muestran en el panel informativo.

Esta lógica de actualización bajo demanda permite asegurar la coherencia entre el estado interno del sistema de misiones y su representación en la interfaz, optimizando simultáneamente el rendimiento al evitar procesos de refresco innecesarios.

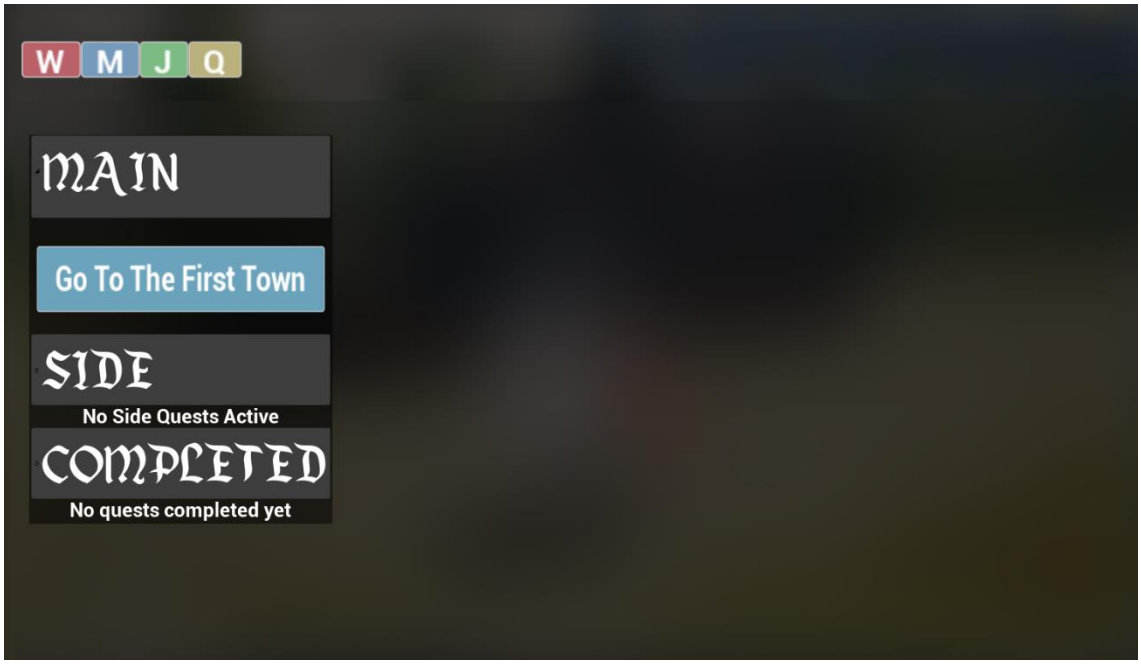


Figura 277: Menú de misiones.

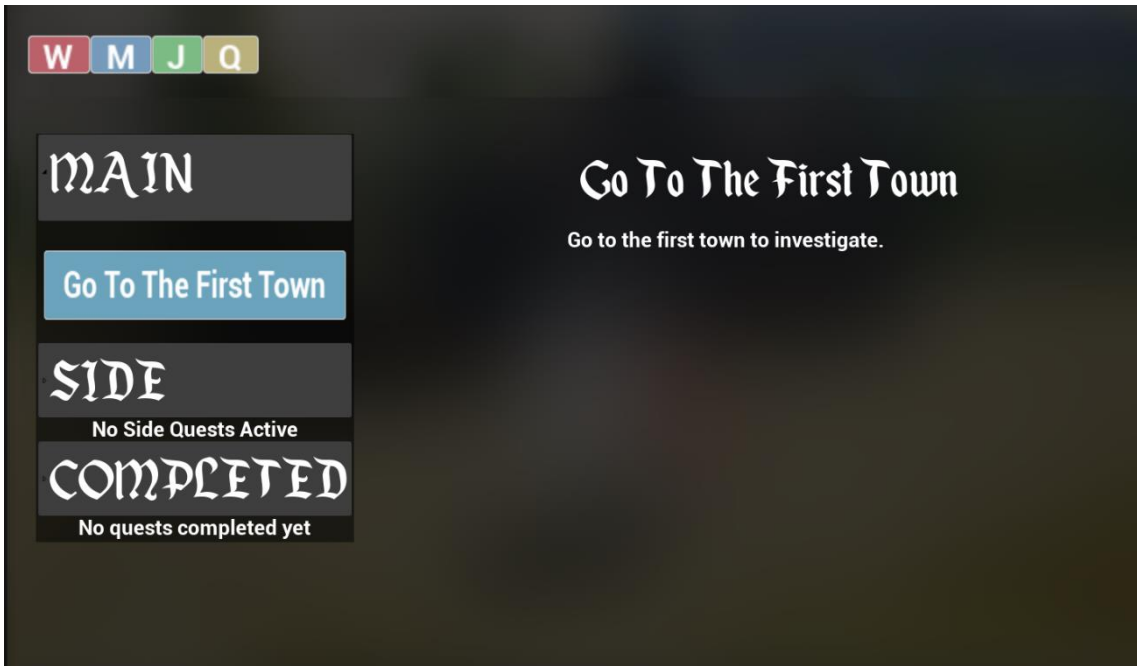


Figura 278: Menú de misiones con misión principal seleccionada.

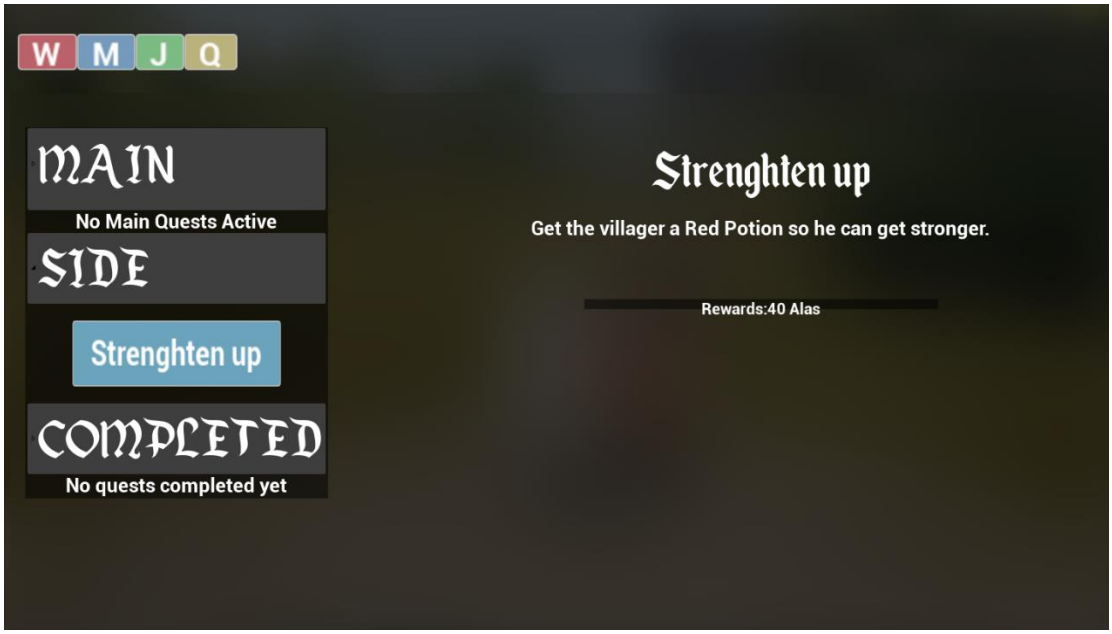


Figura 279**:** Menú de misiones con misión secundaria con recompensa seleccionada.

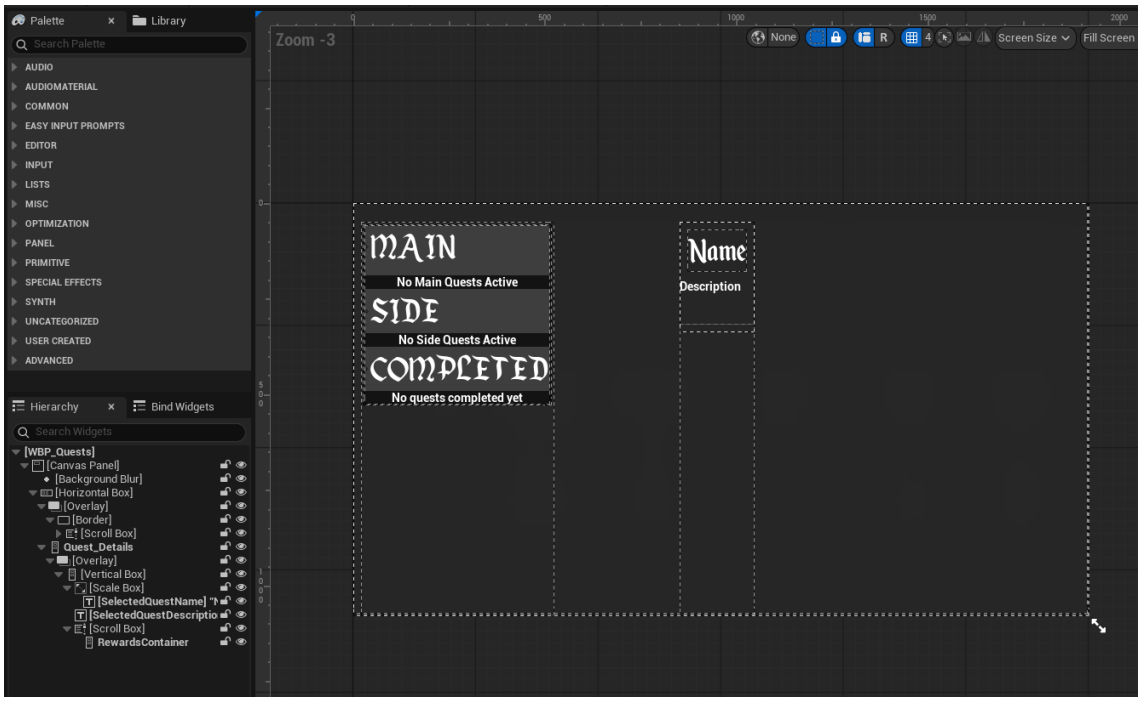


Figura 280: Diseñador del Widget Blueprint [17] de misiones.

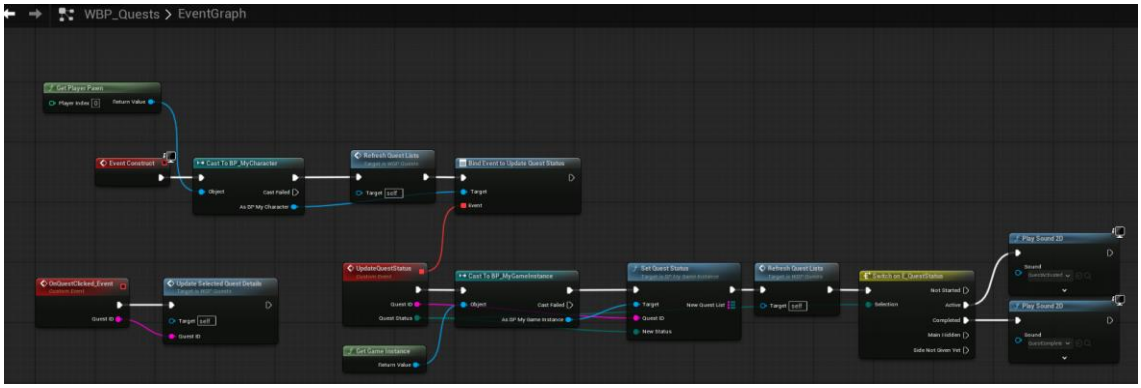


Figura 281: Event Graph del Widget Blueprint [17] de misiones.

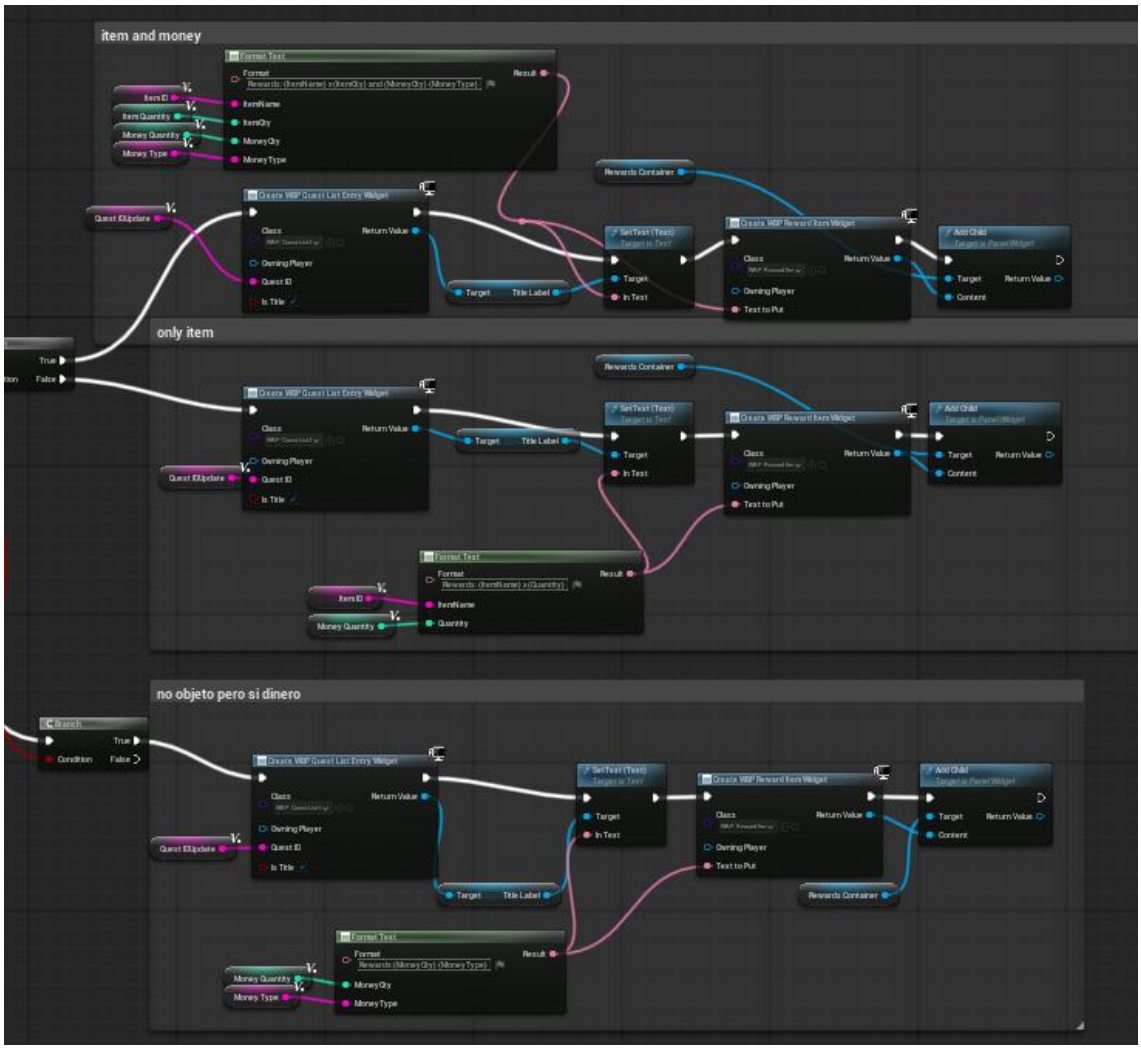


Figura 282: Actualizaciones de las recompensas de misiones en la función UpdateSelectedQuestDetails.

4.8.5.5. Bestiario

Desde el punto de vista de implementación, el sistema de **Bestiario** se ha desarrollado mediante una arquitectura basada en eventos, orientada a la detección automática de avistamientos y a la actualización dinámica del progreso del jugador dentro de esta colección.

El desbloqueo de la pestaña se gestiona mediante lógica de progresión narrativa, activándose en una fase avanzada del juego, concretamente en el tramo previo al enfrentamiento final al hablar con un NPC específico. Una vez habilitada, el sistema comienza a registrar los avistamientos que el jugador haya realizado o realice a partir de ese momento.

El mecanismo de registro se fundamenta en la detección de proximidad entre el jugador y los animales catalogables. Cuando se cumple el umbral de distancia establecido:

- Se muestra una notificación en pantalla informando del avistamiento.
- El animal correspondiente se añade automáticamente al registro del bestiario.

A nivel técnico, esta funcionalidad se implementa mediante el uso de **Event Dispatchers [10]**.
Cada entidad animal susceptible de ser registrada dispone de la lógica necesaria para lanzar un evento en el momento en que se satisface la condición de proximidad.

Dicho evento se encuentra enlazado con el sistema central del bestiario, el cual, al recibir la señal:

- Procede al registro del animal correspondiente dentro de la colección.
- Ejecuta una comprobación del estado global de completitud del bestiario.

En caso de verificarse que la totalidad de las entradas disponibles han sido registradas, el sistema activa la recompensa asociada, otorgando al jugador **20 puntos de alineación con la Tierra**, conforme a la lógica de progresión moral y sistémica del proyecto.

Cabe señalar que, desde el punto de vista de diseño jugable, los animales incluidos en el bestiario presentan un comportamiento pasivo: no reaccionan ante la presencia del jugador y no pueden ser dañados. Esta decisión responde a la intención de orientar el sistema hacia la exploración y la observación, en lugar de vincularlo a dinámicas de combate o captura.

Durante las fases de planificación se contempló la incorporación de una sexta entrada correspondiente a un zorro. No obstante, su implementación fue descartada con el objetivo de priorizar otros sistemas considerados críticos para la entrega de la versión actual del proyecto.

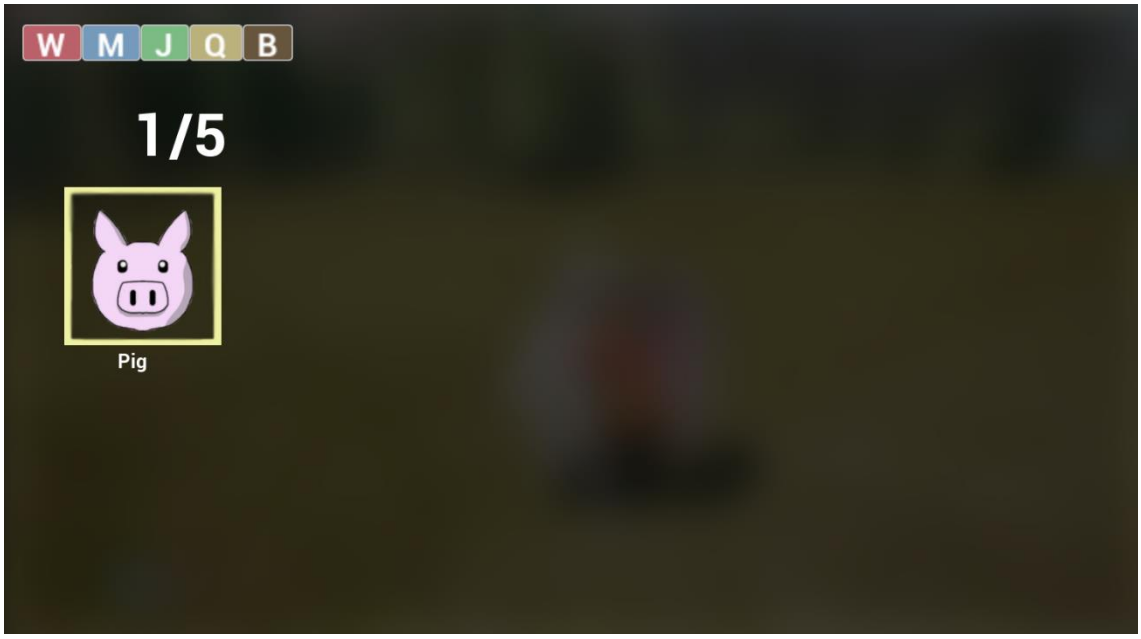


Figura 283: Menú de bestiario con 1 solo animal descubierto.

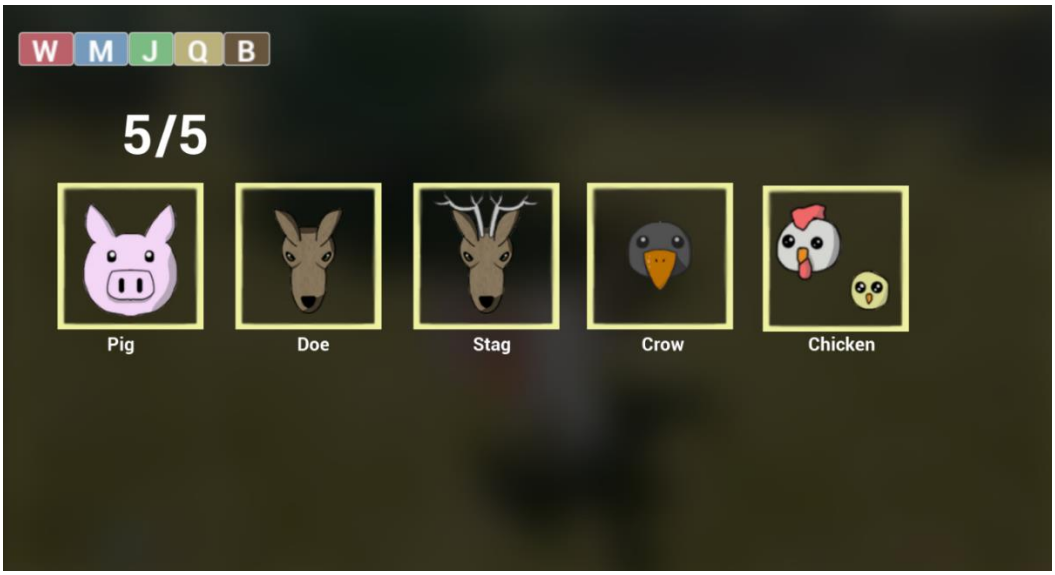


Figura 284: Menú de bestiario con todos los animales descubiertos.

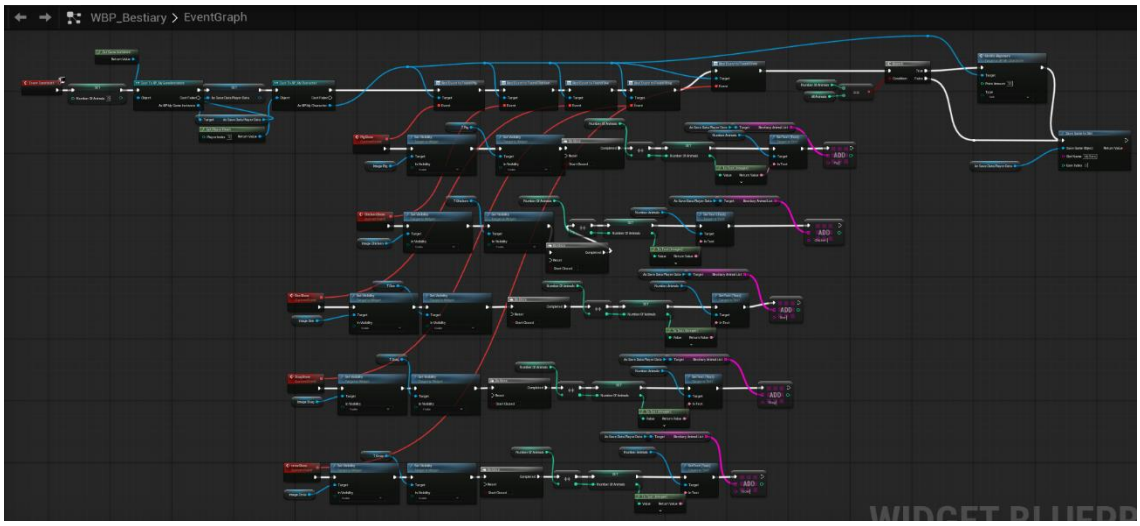


Figura 285: Lógica del bestiario en WBP_Bestiary.

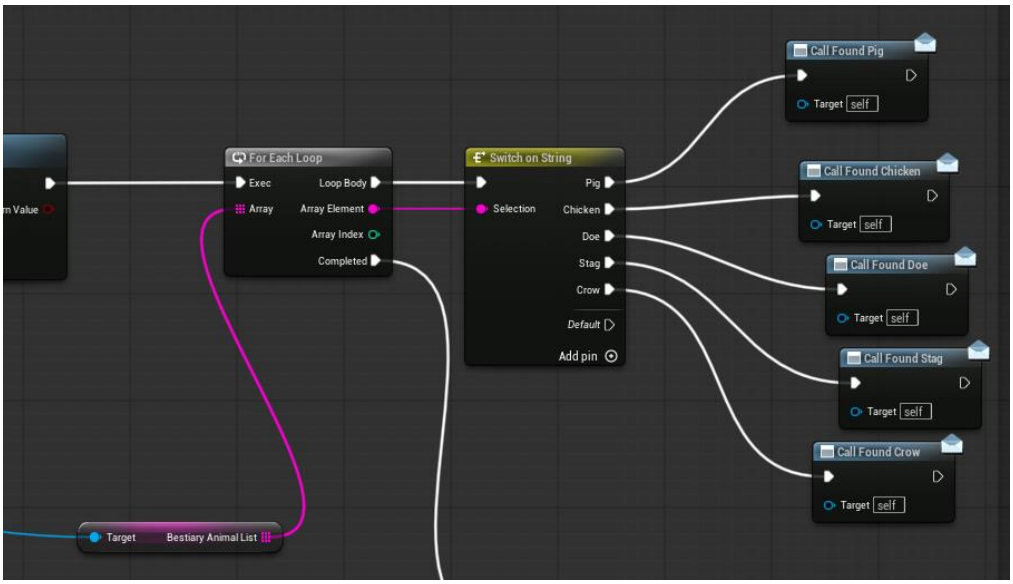


Figura 286: Lógica del carga de bestiario en BP_MyCharacter.

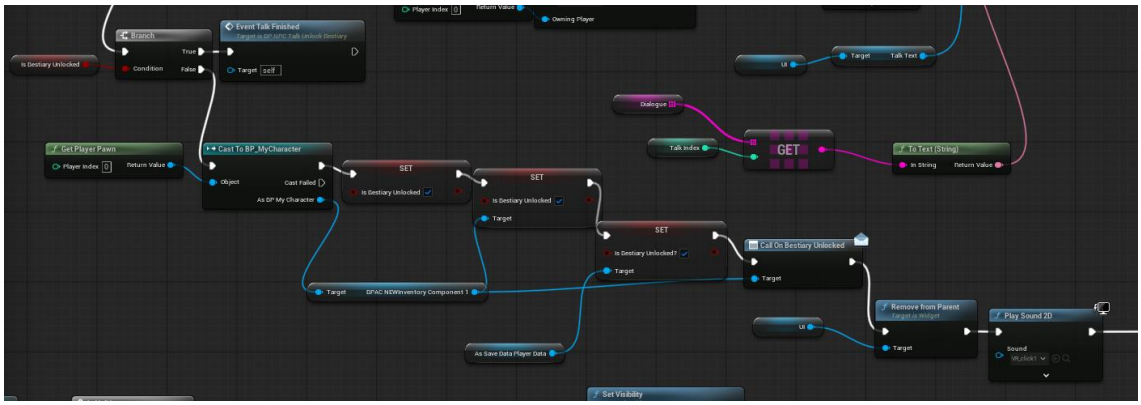


Figura 287: NPC que da el bestiario.

4.8.6. Notificaciones

El sistema de notificaciones del juego se ha implementado mediante tres animaciones diferenciadas desarrolladas a través de *Widget Blueprints [17]*. Todas ellas han sido diseñadas y configuradas utilizando la ventana de animaciones integrada en los propios *widgets*, permitiendo definir transiciones, desplazamientos y temporizaciones de forma visual.

Las notificaciones cumplen una función informativa inmediata, orientada a comunicar al jugador eventos relevantes sin interrumpir el flujo de juego.

Se distinguen las siguientes tipologías:

Animación de entrada a ciudades
Al acceder a una ciudad, se activa una notificación contextual que presenta el nombre de la localización. Visualmente, consiste en un panel que desciende desde la parte superior de la pantalla, permanece visible durante un breve intervalo de tiempo y, posteriormente, vuelve a ascender hasta desaparecer.

Esta animación se reutiliza para ambas ciudades disponibles en la versión actual del proyecto, garantizando uniformidad estética y funcional en la presentación de descubrimientos de localizaciones.

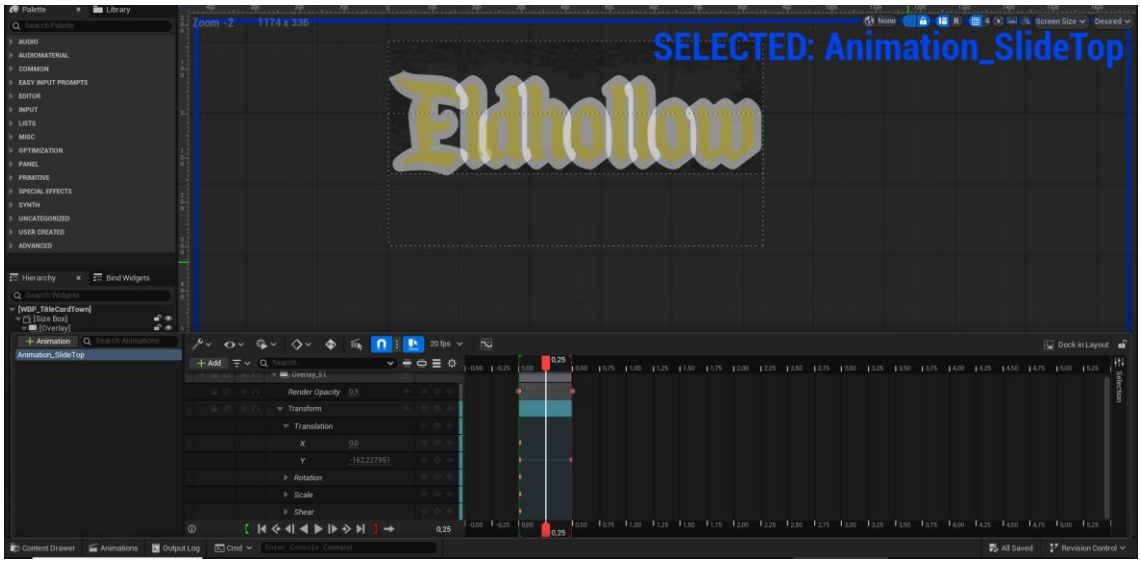


Figura 288: Animación WBP_TitleCardTown.

Animación de registro en el bestiario

Cuando el jugador avista un animal por primera vez y este queda registrado en el bestiario, se despliega una notificación textual en el centro de la pantalla con el mensaje **“New Animal Found”**.

La animación sigue una lógica de movimiento vertical similar a la anterior: el texto desciende desde la zona superior hasta su posición de visualización y, tras un intervalo determinado, asciende nuevamente hasta desaparecer.

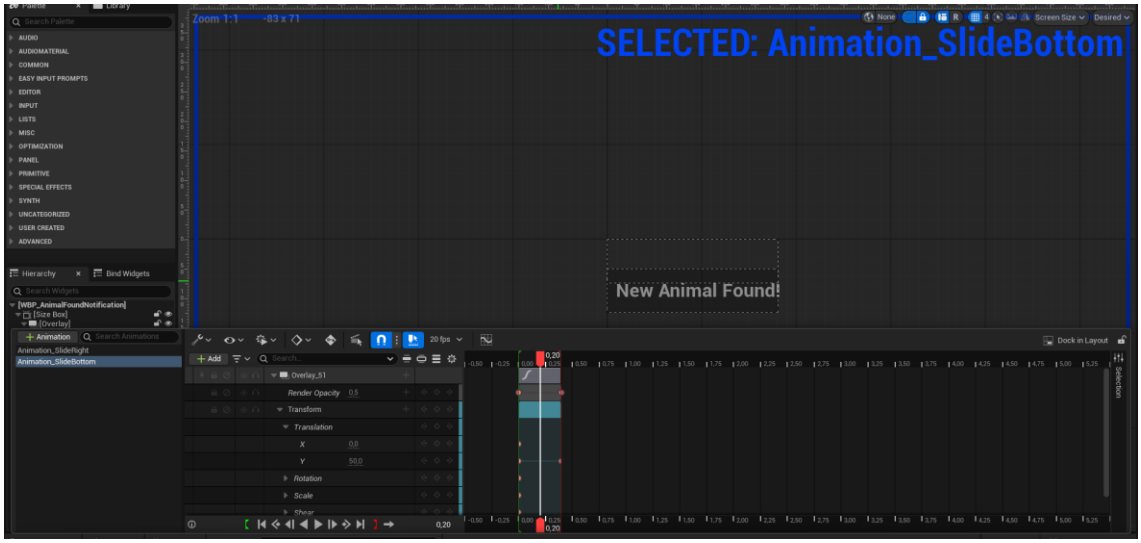


Figura 289: Animación WBP_AnimalFoundNotification.

Animación de recogida de objetos

Al recoger cualquier objeto del entorno, se genera una notificación informativa situada en el lateral izquierdo de la interfaz, concretamente bajo el indicador de salud representado por los corazones.

Esta notificación muestra el nombre del objeto obtenido junto con la cantidad recogida y el icono, permitiendo al jugador confirmar de forma inmediata la adquisición del recurso. La animación aparece de manera dinámica y desaparece tras un breve periodo, evitando la saturación visual de la interfaz.

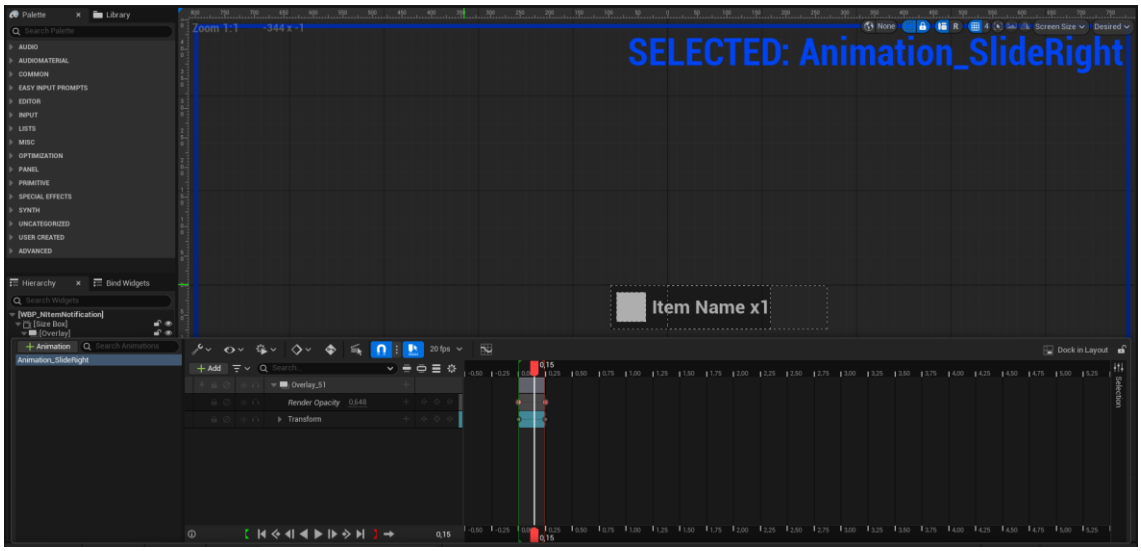


Figura 290: Animación WBP_NItemNotification.

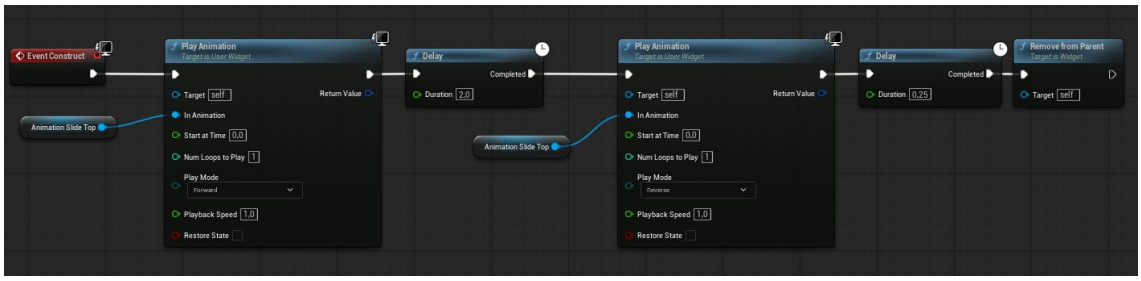


Figura 291: Ejemplo de lógica de animación.

4.9. Sistema de guardado

El sistema de guardado del juego se ha implementado mediante una instancia personalizada de Game Instance, denominada BP_MyGameInstance. Este elemento actúa como núcleo de persistencia global, permitiendo conservar información entre sesiones de juego mediante el almacenamiento estructurado de datos.

Dicha instancia incorpora múltiples variables y funciones auxiliares destinadas a la serialización y recuperación del estado del juego. Entre ellas, destaca una de especial relevancia: una referencia a un objeto de tipo Save Game denominado SaveData_PlayerData [15], encargado de contener y registrar el estado de las distintas variables persistentes del sistema.

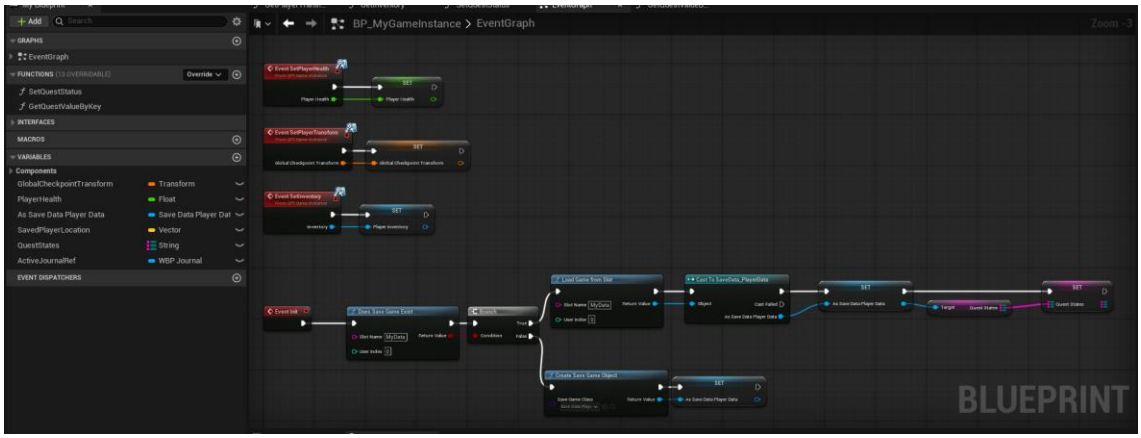


Figura 292: BP_MyGameInstance.

A continuación, se detallan los principales eventos y elementos cuyo estado es almacenado:

Progresión narrativa y eventos clave

- Al derrotar al jefe final, se registra un valor booleano que indica su derrota.
- Se almacena el estado de desbloqueo del portal cuando este es activado.
- La primera vez que se completan las misiones de acceso a ambas ciudades, se guarda el desbloqueo de sus iconos correspondientes en el mapa.
- Tras interactuar con la bruja, se registra tanto la aparición del icono de su localización en el mapa como el desbloqueo del botón de creación de pociones.

Sistema de objetos y reaparición

- Cada objeto recogido registra su identificador dentro de un *Set* de IDs de objetos recolectados, permitiendo gestionar sus temporizadores de reaparición.
- Cuando un enemigo muere, se añade a la lista de entidades con temporizador de reaparición activo.
- Una vez que un enemigo reaparece, se elimina de dicha lista para actualizar correctamente su estado.

Sistema de misiones

- Cada vez que se completa una misión, se guarda el estado actualizado de la lista de misiones.
- Asimismo, cualquier refresco del estado de una misión desencadena su correspondiente proceso de guardado.

Hogueras y puntos de control

- Al interactuar con una hoguera, se almacena la localización actual del jugador.
- Se registra también un valor booleano que indica si dicha hoguera ha sido encendida previamente.

Datos del personaje (BP_MyCharacter)

El personaje jugable constituye uno de los principales focos de persistencia de información. Entre los datos almacenados se incluyen:

- Coordenadas de posición en el mundo.

- Datos y referencia de actor de ambas armas equipadas.
- Estado de equipamiento de cada arma mediante valores booleanos.
- Cantidad acumulada de cada tipo de divisa.
- Nivel de progreso del tutorial.
- Nivel de alineamiento moral del jugador.
- Acompañantes desbloqueados.

Inventario y equipamiento

- El inventario se guarda cada vez que se añade un objeto.
- Asimismo, se almacenan los espacios (*slots*) del inventario en cada proceso de guardado.
- El sistema registra también los eventos de equipamiento y desequipamiento tanto de etiquetas como de armas.

Bestiario

- Cada nuevo animal registrado en el bestiario desencadena su correspondiente proceso de guardado.

Eventos de guardado manual y control de sesión

- Se ejecuta un guardado al presionar el botón de salida desde el menú de pausa, garantizando la persistencia del progreso antes de cerrar la sesión.
- Al seleccionar la opción **Nuevo Juego** en el menú principal, el estado de guardado previo es eliminado. Posteriormente, se genera un nuevo archivo de guardado base, evitando inconsistencias derivadas de diferencias nominales o estructurales entre partidas.

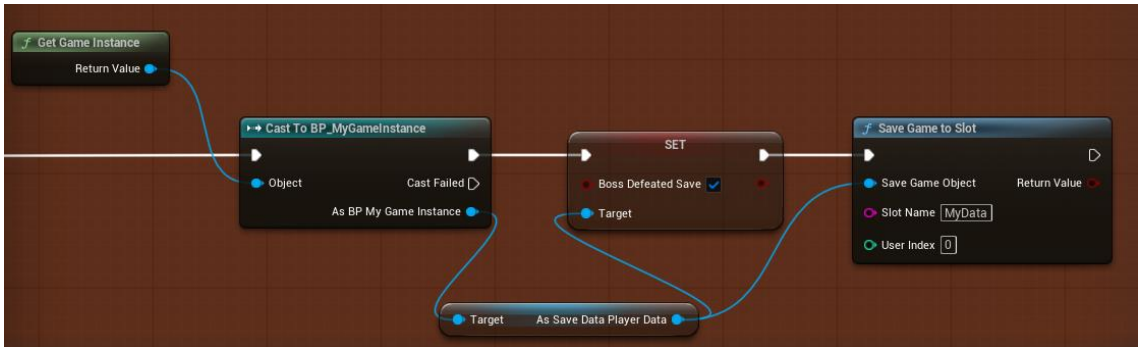


Figura 293: Ejemplo de guardado de una variable.

Este enfoque centralizado, sustentado en **BP_MyGameInstance** y el objeto **SaveData_PlayerData [15]**, permite una gestión integral de la persistencia, asegurando la continuidad del progreso del jugador y la coherencia sistémica entre sesiones de juego.

4.10. Mapa

El corazón de este sistema se basa en una técnica de enmascaramiento dinámico mediante un Texture Render Target de Unreal Engine [16], al que se ha denominado

Map Mask. Esta textura funciona como una "memoria" bidimensional que registra las áreas exploradas. En lugar de modificar el asset original del mapa, la lógica del juego dibuja sobre esta máscara dinámica, permitiendo que el estado de revelación sea persistente y eficiente.

La visualización final del mapa en la interfaz de usuario combina la imagen base del mapa con esta máscara, mostrando solo las áreas que han sido descubiertas por el jugador.

La lógica para actualizar la Niebla de Guerra reside en el Blueprint del personaje principal (BP_MyCharacter).

- **Control de Frecuencia**: Para optimizar el rendimiento y evitar cálculos por cada frame, el proceso de revelación se activa mediante un Timer configurado para ejecutarse en bucle cada 0.1 segundos, a través del evento UpdateMapMask (Imagen 1).

- **Obtención y Transformación de Coordenadas**: Es fundamental traducir la posición 3D del jugador (Get Actor Location) en el extenso mundo a una coordenada bidimensional, normalizada (rango 0.0 a 1.0), que pueda ser utilizada en la Map Mask. Este proceso se realiza en dos etapas:

- **Traslación (Offset)**: Se aplica una adición de vectores (+100800.0, +100800.0) para establecer el origen de coordenadas (0,0) del mapa en una esquina predefinida del mundo.

- **Normalización (Escalado)**: El vector resultante se escala (división implícita por 116400.0). Esto convierte la posición de un valor de mundo a un valor normalizado, perfecto para coordenadas UV.

- **Comunicación con Materiales**: La posición normalizada del jugador se envía al sistema visual mediante el Material Parameter Collection (MPC_MapParameters), estableciendo el valor del parámetro PlayerLocation.

La acción de pintar la máscara ocurre dentro del contexto del nodo “Begin Draw Canvas to Render Target”.

1. **Lienzo:** El nodo selecciona la Map Mask como lienzo para el dibujo.
2. **Brocha:** Se utiliza el nodo Draw Material empleando un material específico (MI_MapBrush, inferido como una brocha circular o con *fade*).
3. **Aplicación:** El material de brocha se pinta en la Map Mask utilizando la posición normalizada del jugador (Screen Position), expandiendo dinámicamente el área de exploración.

El sistema incluye lógica para la gestión de la interfaz y la persistencia del progreso:

- **Persistencia:** La presencia del nodo Save Game to Slot [15] garantiza que el estado actual de la Map Mask (es decir, el progreso de la exploración) se

almacena de forma persistente, permitiendo que el jugador continúe la partida manteniendo las áreas descubiertas.

- **Activación y Modo de Juego:** El mapa se abre al activarse un *Enhanced Input Action*. Al crearse el *widget* del mapa (WBP_MapUI), el juego se pausa (Set Game Paused: True) y el modo de entrada cambia a **UI Only**, enfocando el control del cursor en la interfaz del mapa.
- **Salida:** Al pulsar el botón de salida del *widget*, se revierten los estados: se reanuda el juego, y el modo de entrada vuelve a ser Game Only, devolviendo el control al personaje.

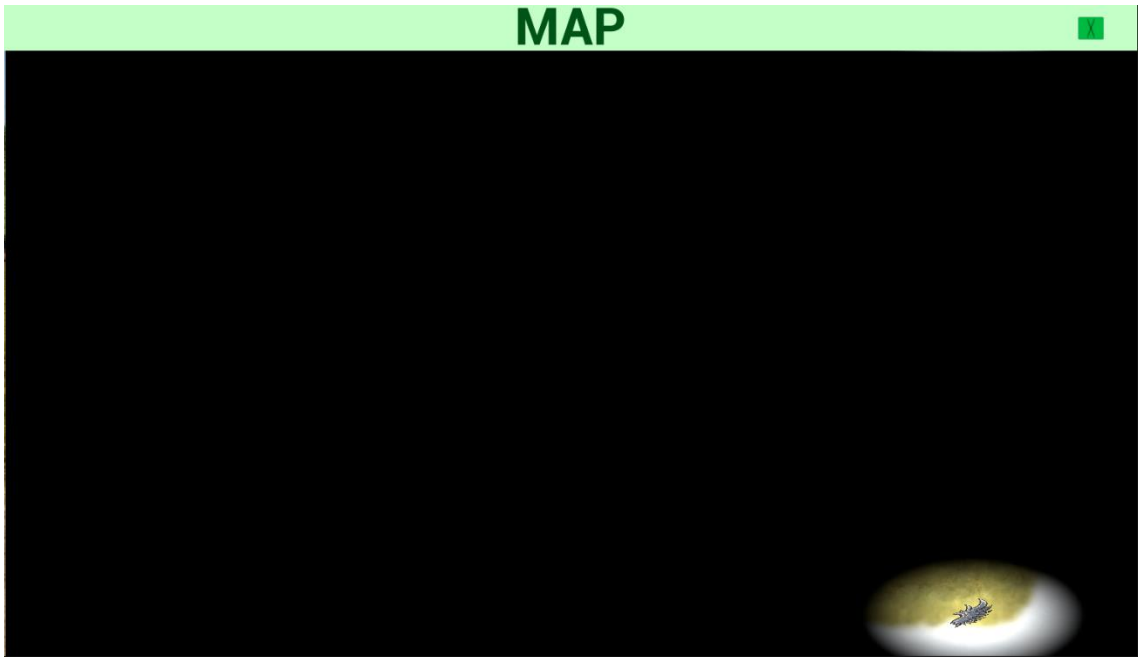


Figura 294: Mapa de revelamiento progresivo.

Figura 295: Mapa completo del juego.

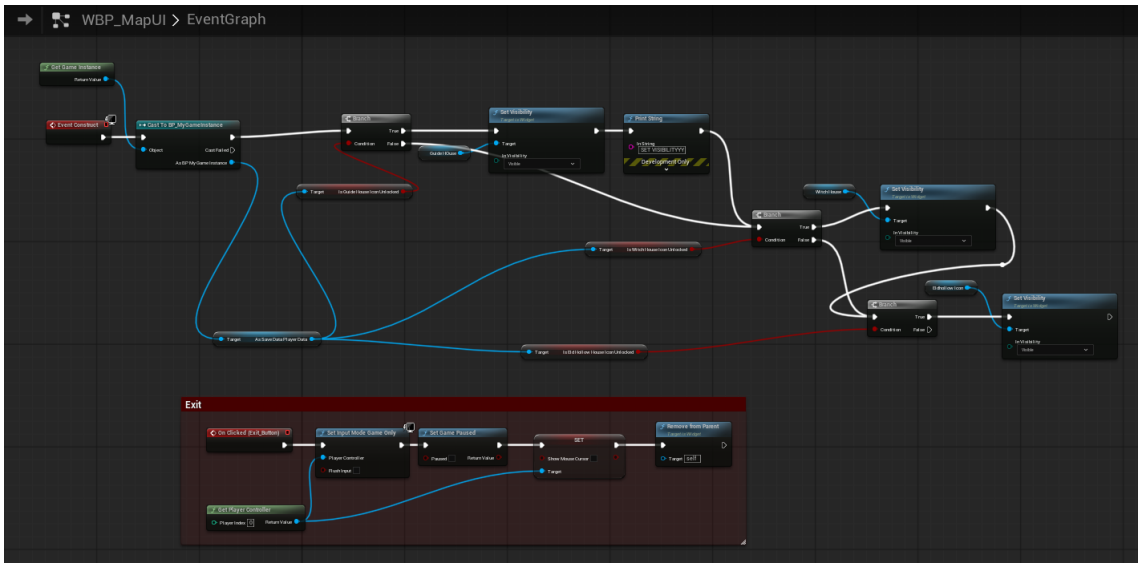


Figura 296: Lógica interna funcionalidad mapa.

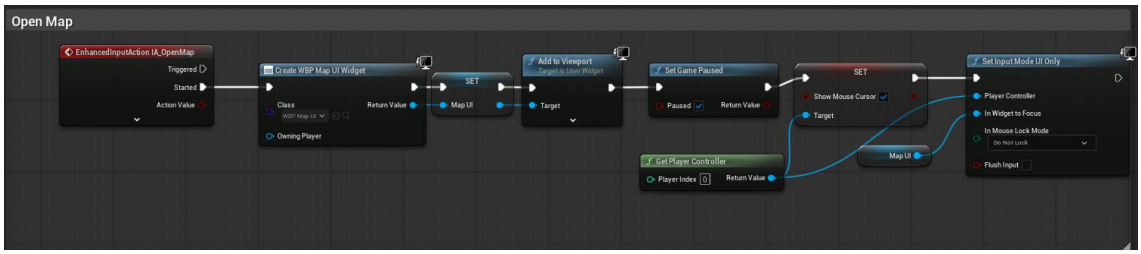


Figura 297: Lógica de abrir mapa en el jugador.

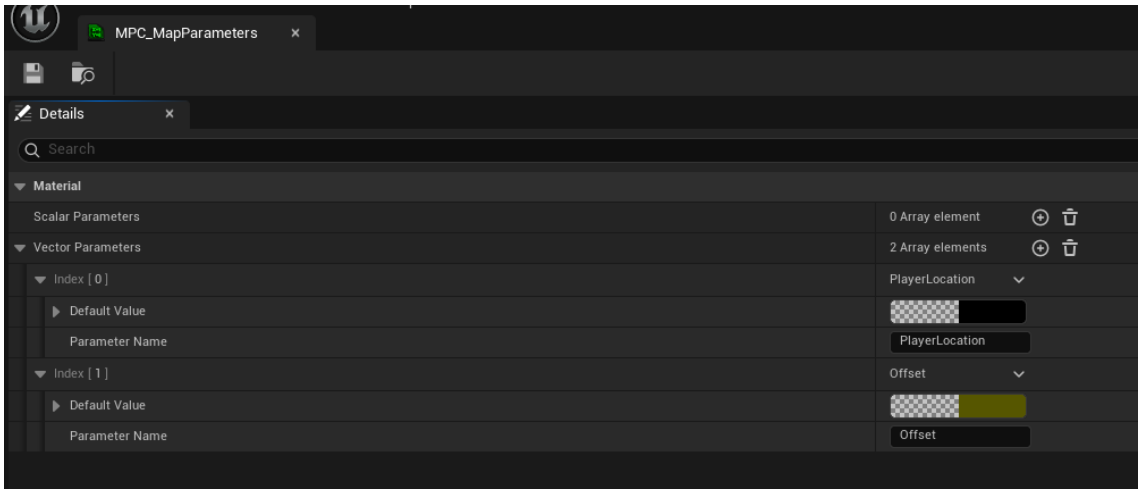


Figura 298: Parámetros del mapa.

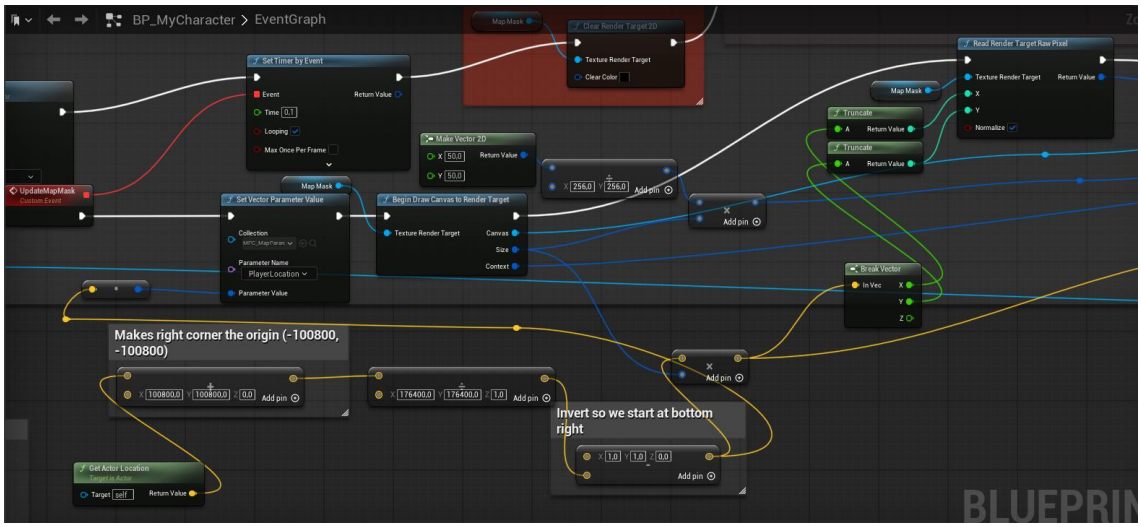


Figura 299: Lógica mapa en BP_MyCharacter.

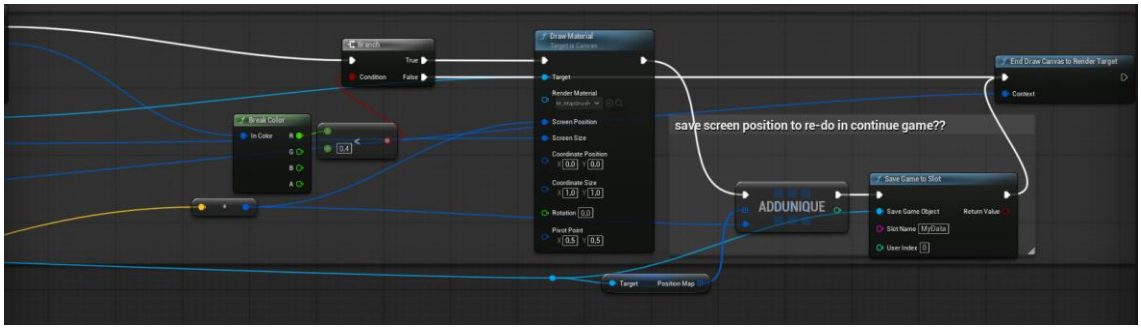


Figura 300: Lógica mapa en BP_MyCharacter que se encarga de dibujarlo.

5

Pruebas

5.1. Pruebas por el desarrollador

Las tareas de pruebas y depuración del proyecto se han realizado principalmente de forma manual y continua a lo largo del desarrollo, siguiendo un enfoque iterativo orientado a la detección temprana de errores y a la validación del comportamiento de los sistemas implementados.

La técnica principal empleada para la depuración ha sido el uso de mensajes de salida en tiempo de ejecución mediante la función *Print String*, lo que ha permitido verificar el flujo de ejecución de los Blueprints [8] y comprobar el estado de variables clave en distintos momentos del juego. Este método ha resultado especialmente útil para validar sistemas dependientes de eventos, como el inventario, el sistema de alineamiento o el bestiario.

Para la comprobación del sistema de guardado y carga, se diseñaron mapas de prueba específicos que contenían todos los elementos relevantes del sistema a evaluar. Por ejemplo, en el caso del bestiario, se utilizó un nivel de pruebas en el que se encontraban disponibles todos los animales registrables. El procedimiento consistía en interactuar con dichos elementos, cerrar el juego y volver a iniciarlo para comprobar si el estado se conservaba correctamente.

En los casos en los que el comportamiento no era el esperado, se procedía a eliminar manualmente el archivo de guardado desde la carpeta correspondiente, ajustar la lógica del sistema afectado y repetir el proceso hasta asegurar su correcto funcionamiento.

Durante el proceso de depuración también se recurrió de forma habitual a recursos externos, como foros especializados de Unreal Engine [16] y herramientas de asistencia basadas en inteligencia artificial, en particular Google AI Studio. Esta herramienta se utilizó para analizar mensajes de error generados por la consola del motor, facilitando la identificación del origen de determinados fallos y agilizando el proceso de corrección. Este enfoque manual, aunque laborioso, permitió obtener un alto grado de control sobre el comportamiento del sistema y asegurar la estabilidad de las funcionalidades principales dentro de las limitaciones temporales propias del proyecto.

Además de las técnicas básicas de depuración, Unreal Engine [16] ofrece herramientas avanzadas que pueden emplearse para analizar el comportamiento interno del sistema de forma más detallada. Entre ellas destaca el Blueprint Debugger, que permite ejecutar el juego en modo depuración y observar en tiempo real el flujo de ejecución de los nodos, los valores de las variables y las llamadas a funciones y eventos. Esta herramienta resulta especialmente útil para detectar errores lógicos complejos o condiciones que no se cumplen según lo esperado.

Otra herramienta relevante es el uso de breakpoints en Blueprints [8], que permite pausar la ejecución del juego en puntos concretos del código visual para inspeccionar el estado del sistema en ese instante. Asimismo, el Output Log del motor proporciona información detallada sobre advertencias, errores y mensajes personalizados, lo que facilita la identificación de problemas relacionados con referencias nulas, fallos de carga o conflictos entre sistemas.

Aunque estas herramientas han sido empleadas de manera puntual durante el desarrollo del proyecto, especialmente para la inspección de flujos de ejecución y la identificación de errores lógicos concretos, el proceso de verificación se ha apoyado principalmente en pruebas manuales iterativas. No obstante, el conocimiento y la correcta aplicación de estas herramientas resultan fundamentales en entornos profesionales de desarrollo, especialmente en proyectos de mayor complejidad y en equipos de trabajo de mayor tamaño.

5.2. Pruebas por usuarios

A continuación, se presentan los resultados de la Primera Prueba (Alpha), realizada con 10 usuarios, enfocada en evaluar el perfil del jugador, las mecánicas principales y la usabilidad inicial del videojuego.

5.2.1. Perfil del jugador

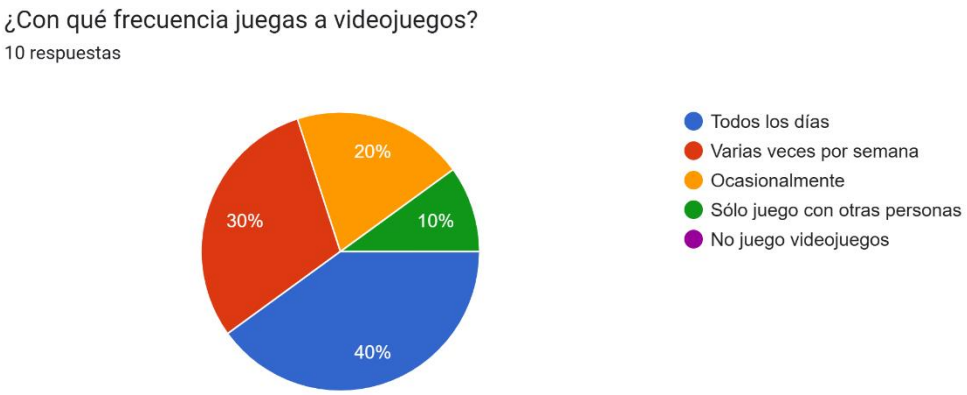


Figura 301: Gráfico relacionado con la relación de los usuarios de prueba con los videojuegos.

El perfil de los usuarios que participaron en esta prueba demuestra que la muestra está compuesta principalmente por jugadores asiduos.

Se puede observar en la Figura 301 que el 70% de los participantes juega a videojuegos con alta frecuencia (40% todos los días y 30% varias veces por semana), lo que sugiere que la muestra tiene experiencia en el género y puede ofrecer feedback valioso.

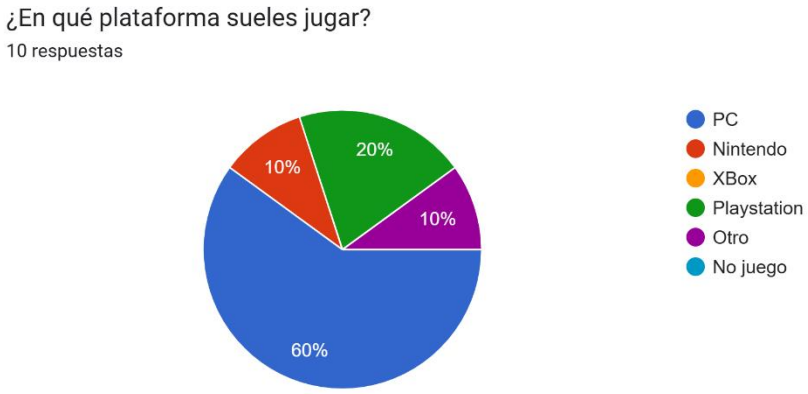


Figura 302: Gráfico de la plataforma de juegos habitual de los usuarios de prueba.

En cuanto a la plataforma de juego habitual, la Figura 302 indica que el 60% de los usuarios prefiere jugar en PC, siendo esta la plataforma dominante. El resto se distribuye entre Xbox (20%), Nintendo (10%) y otras plataformas (10%).

5.2.2. Experiencia de juego

Figura 303: Gráfico de la dificultad de control percibida de los usuarios de prueba.

Figura 304: Gráfico de la dificultad de la primera misión de los usuarios de prueba.

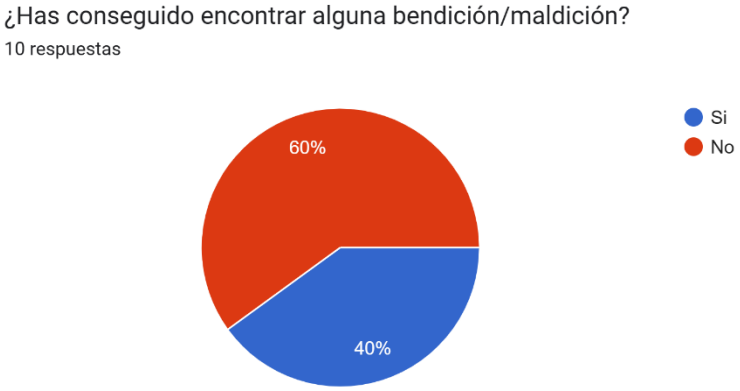


Figura 305: Gráfico de la cantidad de los usuarios de prueba que encuentran etiquetas.

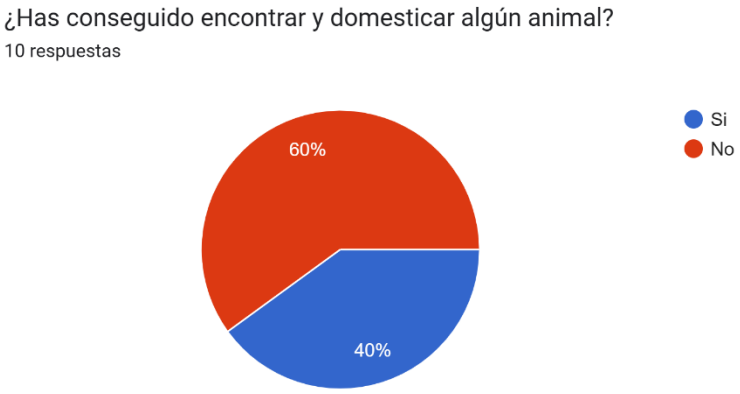


Figura 306: Gráfico de la cantidad de los usuarios de prueba que encuentran acompañantes.

Los resultados sobre la experiencia de juego en las mecánicas iniciales son positivos, aunque se identifican áreas de mejora en la visibilidad de contenidos secundarios.

Dificultad inicial:

- **Control del personaje**: La Figura 303 muestra que el 80% de los usuarios reportó "Ninguna" (50%) o "Poca" (30%) dificultad para controlar al personaje, lo que indica que el gameplay principal es intuitivo y la curva de aprendizaje es adecuada.

- **Misión inicial**: De manera similar, la Figura 304 refleja que el 90% de los usuarios sintió "Ninguna" (20%) o "Poca" (70%) dificultad para encontrar el primer pueblo, confirmando que la guía o el diseño de nivel para la misión introductoria es eficaz.

Mecánicas secundarias y progresión:

- Las Figuras 305 y 306 exponen un punto a revisar: el 60% de los usuarios no logró encontrar alguna bendición/maldición ni tampoco encontró/domesticó algún animal. Esto sugiere que estas mecánicas secundarias clave pueden carecer de la suficiente visibilidad o pistas dentro del mundo de juego y deben ser objeto de ajustes en futuras iteraciones.

- El progreso de misiones, detallado en la Figura 308, muestra un alto nivel de avance en la historia principal inicial (100% de finalización hasta la Misión Principal 3). El porcentaje de finalización cae progresivamente, llegando al 40% para la Misión Principal 6 y la Misión Secundaria 2. Este descenso es esperado en una prueba de este tipo, pero indica que los puntos de la historia o la dificultad después de la Misión Principal 4 (completada por el 90%) podrían necesitar ser revisados para asegurar un flow de juego constante.

Satisfacción general:

La Figura 311 muestra una puntuación general muy alta. El 80% de los usuarios calificó el juego con una puntuación de 8 o superior (30% con 8, 30% con 9, y 20% con 10), lo que refleja una excelente recepción y satisfacción general.

5.2.3. Experiencia de uso

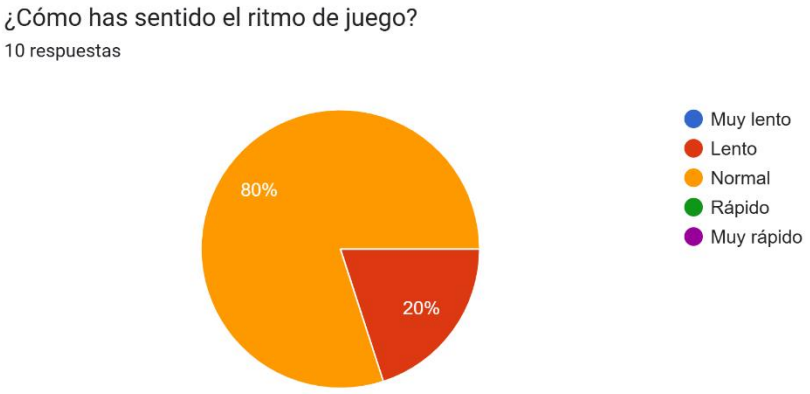


Figura 307: Gráfico de percepción del ritmo de juego de los usuarios de prueba.

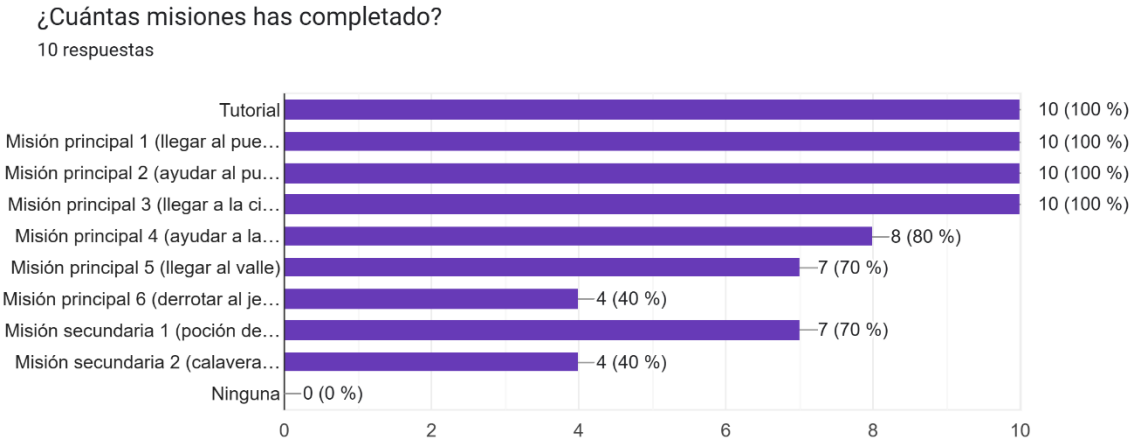


Figura 308: Gráfico de misiones completadas por el usuario de pruebas.

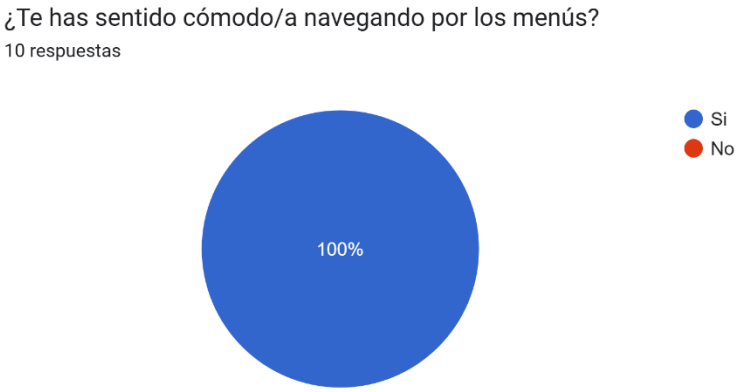


Figura 309: Gráfico de comodidad de uso de menús de los usuarios de prueba.

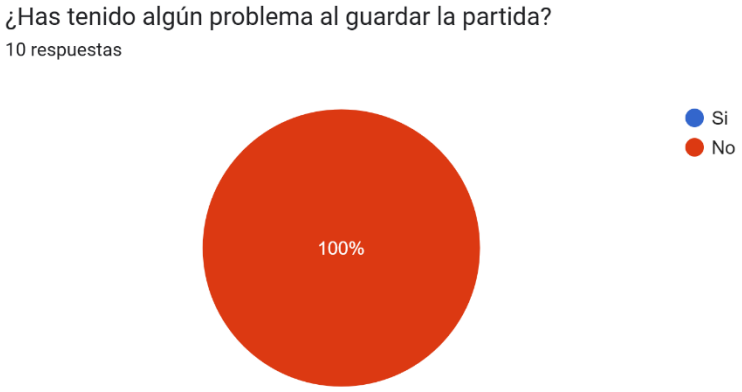


Figura 310: Gráfico de problemas de guardado de partida de los usuarios de prueba.

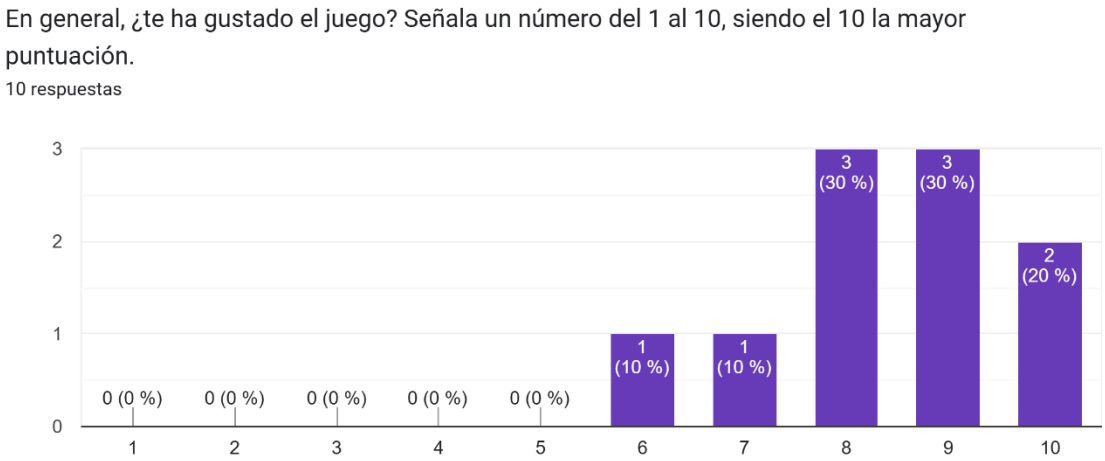


Figura 311: Puntuación general dada por los usuarios de prueba.

La experiencia de uso y la usabilidad del sistema han demostrado ser extremadamente sólidas en esta primera fase.

- **Ritmo de juego**: El ritmo de juego fue percibido como Normal por el 60% de los probadores, como se ve en la Figura 307. Esto confirma que el equilibrio entre la exploración, el combate y la narrativa es adecuado.

- **Navegación y guardado**: La Figura 309 y la Figura 310 demuestran una usabilidad perfecta para estas funciones: el 100% de los usuarios se sintió cómodo navegando por los menús, y el 100% no reportó ningún problema al guardar la partida.

En conclusión, la primera prueba masiva ha validado el core gameplay, la interfaz de usuario y la estabilidad de funciones críticas (guardado y menús). Las áreas a mejorar se centran en aumentar la visibilidad y accesibilidad de las mecánicas secundarias

(bendiciones/maldiciones y compañeros) para asegurar que un mayor número de jugadores las descubran y las utilicen.

Además los usuarios han añadido algunas sugerencias que se tendrán en cuenta a la hora de mejorar el juego:

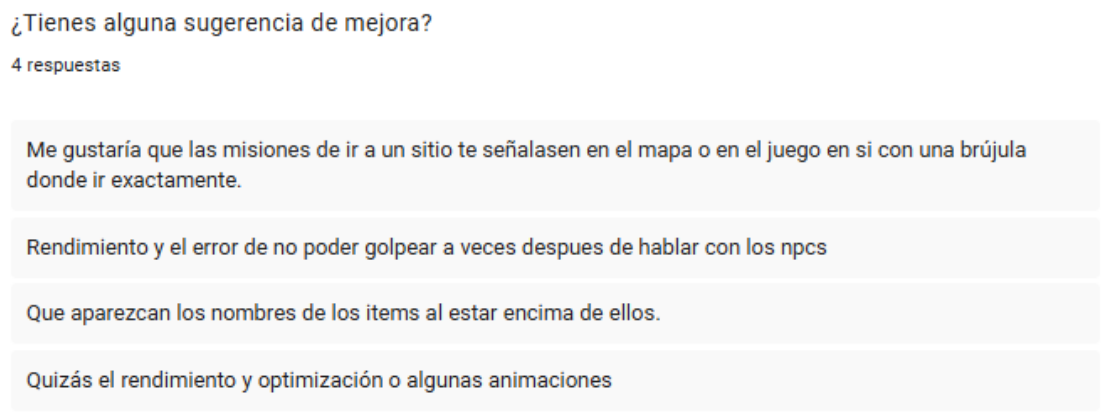


Figura 312: Sugerencias de mejora de los jugadores de prueba.

6

Conclusiones y líneas futuras

6.1. Conclusiones

El desarrollo de este proyecto ha supuesto un reto significativo, especialmente en sus fases iniciales, debido a la falta de experiencia previa en el diseño y la implementación de videojuegos completos, así como en el uso avanzado de herramientas como Unreal Engine [16] y su sistema de Blueprints [8]. No obstante, a lo largo del proceso se ha producido una evolución notable tanto en la comprensión de las mecánicas internas del motor como en la capacidad para estructurar sistemas complejos de forma coherente y funcional.

Los conocimientos adquiridos durante la carrera han resultado fundamentales para afrontar el proyecto con éxito. Habilidades transversales como la búsqueda y análisis de información técnica, la gestión del tiempo, la resolución de problemas y la capacidad de aprendizaje autónomo han permitido superar las dificultades iniciales y avanzar de manera progresiva hacia la consecución de un prototipo funcional y estable.

Como resultado, se ha logrado desarrollar un videojuego plenamente jugable que integra múltiples sistemas interconectados, tales como misiones, economía, combate, alineaciones, inventario, interfaz de usuario y narrativa reactiva. El proyecto no solo cumple los objetivos técnicos planteados inicialmente, sino que también ha permitido consolidar una comprensión más profunda del desarrollo de videojuegos, especialmente en lo relativo a la programación visual mediante Blueprints [8], que ha pasado de ser un elemento desconocido a convertirse en una herramienta comprensible y disfrutable durante el desarrollo.

En conjunto, el proyecto ha servido como una experiencia formativa completa, demostrando la viabilidad de aplicar los conocimientos adquiridos en el grado a un entorno práctico y creativo, y evidenciando la capacidad de adaptación y aprendizaje continuo ante tecnologías complejas.

6.2. Líneas futuras de desarrollo

A pesar de haber alcanzado un prototipo funcional y coherente, el proyecto presenta múltiples posibilidades de ampliación y mejora en futuras iteraciones. Una de las principales líneas de desarrollo futuro sería la expansión del contenido narrativo, profundizando en la historia del mundo de juego y en los personajes que lo habitan.

Durante el desarrollo del prototipo fue necesario simplificar y adaptar la narrativa a la escala reducida del proyecto, por lo que una versión más completa permitiría explorar tramas más complejas y un desarrollo narrativo más extenso.

Asimismo, podrían ampliarse las mecánicas existentes, incorporando un mayor número de misiones secundarias, enemigos, zonas explorables y objetos, así como un sistema de progresión más profundo para el personaje. En el ámbito técnico, también sería posible optimizar y refactorizar algunos sistemas para mejorar su rendimiento y escalabilidad, facilitando la incorporación de nuevo contenido sin afectar a la estabilidad del proyecto.

Otra línea futura relevante sería la mejora del apartado visual y sonoro, incluyendo animaciones más elaboradas, efectos visuales avanzados y una banda sonora más dinámica, que contribuyan a una experiencia de juego más inmersiva.

En definitiva, el proyecto sienta una base sólida sobre la que construir un videojuego de mayor alcance, permitiendo que, en un contexto con más tiempo y recursos, pueda evolucionar hacia una experiencia más completa y ambiciosa tanto a nivel técnico como narrativo.

Referencias

[1] Adobe. (s.f.). *Adobe Photoshop* [Software]. https://www.adobe.com/products/photoshop.html
[2] Atlassian. (s.f.). Trello [Software]. https://trello.com/
[3] Bethesda Game Studios. (2011). The Elder Scrolls V: Skyrim [Videojuego]. Bethesda Softworks.
[4] Blender Foundation. (s. f.). Blender. https://www.blender.org/
[5] CD Projekt Red. (2015). *The Witcher 3: Wild Hunt* [Videojuego]. CD Projekt.
[6] Craiyon. (s. f.). Craiyon. https://www.craiyon.com/
[7] Donetonic. (s. f.). Metodología Waterfall vs metodología Agile. https://donetonic.com/es/metodologiawaterfall-vs-metodologia-agile/
[8] Epic Games. (s.f.). *Blueprints: Secuencias de comandos visuales en Unreal Engine [Documentación]*. https://dev.epicgames.com/documentation/es-es/unreal-engine/blueprints-visual-scripting-in-unreal-engine
[9] Epic Games. (s.f.). Chaos Destruction in Unreal Engine [Documentación]. https://dev.epicgames.com/documentation/en-us/unreal-engine/chaos-destruction-in-unreal-engine
[10] Epic Games. (s.f.). *Eventos* [Documentación]. https://dev.epicgames.com/documentation/en-us/unreal-engine/events-in-unreal-engine
[11] Epic Games. (s.f.). *Interfaces* [Documentación]. https://dev.epicgames.com/documentation/en-us/unreal-engine/interfaces-in-unreal-engine
[12] Epic Games. (s.f.). *Nav Mesh* [Documentación]. https://dev.epicgames.com/documentation/en-us/unreal-engine/basic-navigation-in-unreal-engine
[13] Epic Games. (s.f.). Niagara [Documentación]. https://dev.epicgames.com/documentation/es-es/unreal-engine/overview-of-niagara-effects-for-unreal-engine
[14] Epic Games. (s.f.). Plugins in Unreal Engine [Documentación]. https://dev.epicgames.com/documentation/en-us/unreal-engine/plugins-in-unreal-engine
[15] Epic Games. (s.f.). *Save Game Blueprints*. https://dev.epicgames.com/documentation/en-us/unreal-engine/saving-and-loading-your-game-in-unreal-engine
[16] Epic Games. (s.f.). Unreal Engine. https://dev.epicgames.com/es-ES
[17] Epic Games. (s.f.). *Widget Blueprints*. https://dev.epicgames.com/documentation/en-us/unreal-engine/widget-blueprints-in-umg-for-unreal-engine
[18] Fox, T. (2015). Undertale [Videojuego]. Toby Fox.
[19] Guerrilla Games. (2017). Horizon Zero Dawn [Videojuego]. Sony Interactive Entertainment.
[20] MelvinTang Games. (n.d.). *Home* [Canal de YouTube]. YouTube. https://www.youtube.com/watch?v=E6OSEktabos
[21] Meshy AI. (s. f.). Meshy AI. https://www.meshy.ai/
[22] Milanote. (s. f.). Milanote. https://www.milanote.com/
[23] Miro. (s. f.). Miro online whiteboard [Software]. https://miro.com/
[24] Mixamo. (s. f.). Mixamo. https://www.mixamo.com/
[25] Muse Group & contributors. (2026). Audacity: Free Audio Editor and Recorder [Software]. https://www.audacityteam.org/
[26] Nano Banana. (s. f.). Nano Banana. https://aistudio.google.com/models/gemini-2-5-flash-image
[27] NirnaethGameDev. (n.d.). Home [Canal de YouTube]. YouTube. https://www.youtube.com/watch?v=xMCYy5iVG54
[28] Project Sora. (2012). Kid Icarus: Uprising [Videojuego]. Nintendo.
[29] Rodin AI. (s. f.). Rodin AI (Hyper3D). https://hyper3d.ai/
[30] Sketchfab. (s. f.). Sketchfab. https://sketchfab.com/
[31] The Game Dev Cave. (n.d.). Home [Canal de YouTube]. YouTube. https://www.youtube.com/watch?v=1ICBWJ7srxQ
[32] Torvalds, L. (s.f.). *Git* [Software]. https://git-scm.com/
[33] VisualGPT. (s. f.). VisualGPT. https://visualgpt.ai/
[34] Visual Paradigm. (s.f.). Visual Paradigm [Software]. https://www.visual-paradigm.com/
[35] White, T. [GitHub, Inc.]. (s.f.). GitHub [Software]. https://github.com/

Apéndice A. Manual de Instalación y Requerimientos

El presente apéndice describe los requisitos técnicos necesarios para la correcta ejecución del prototipo desarrollado, así como los pasos requeridos para la instalación del motor de desarrollo y las herramientas asociadas.

A.1. Requerimientos de software y hardware

Para el correcto funcionamiento del prototipo, el sistema deberá cumplir, como mínimo, las siguientes especificaciones:

Requerimientos mínimos de hardware:

- Procesador de 64 bits compatible con arquitectura x86-64.
- Memoria RAM mínima de 8 GB.
- Tarjeta gráfica compatible con DirectX 12 o Vulkan, con al menos 4 GB de memoria de vídeo.
- Espacio disponible en disco de al menos 30 GB.

Requerimientos de software:

- Sistema operativo Windows 10 o superior (64 bits).
- Controladores gráficos actualizados.
- Soporte para DirectX 12.
- Unreal Engine versión 5.6.

Para un rendimiento óptimo, se recomienda disponer de 16 GB de memoria RAM y una tarjeta gráfica de gama media o superior.

A.2. Instalación de Unreal Engine y dependencias

La instalación de Unreal Engine [16] se realiza mediante la plataforma **Epic Games Launcher**, disponible gratuitamente en el sitio web oficial de Epic Games.

El proceso de instalación consta de los siguientes pasos:

1. Descarga e instalación del Epic Games Launcher.
2. Inicio de sesión con una cuenta de Epic Games.
3. Acceso a la sección *Unreal Engine [16]* y selección de la versión 5.6.
4. Descarga e instalación del motor con los componentes por defecto.

Durante este proceso, se instalan automáticamente las dependencias necesarias para el correcto funcionamiento del editor, incluyendo librerías de compilación, soporte gráfico y herramientas básicas de depuración.

A.3. Instalación y uso de herramientas externas

Para el desarrollo del prototipo se han utilizado herramientas externas complementarias al motor Unreal Engine [16], orientadas a la creación y edición de recursos gráficos y sonoros.
Entre dichas herramientas se incluyen:

- Software de modelado y edición 3D como Blender.
- Editores de imagen para la creación de texturas y materiales.
- Herramientas de edición de audio para la gestión de efectos sonoros y música.
- Sistemas de control de versiones para la gestión del código y los activos del proyecto.

La instalación de estas herramientas se realiza de forma independiente, siguiendo las indicaciones proporcionadas por sus respectivos desarrolladores.

El proyecto puede ser descargado en la página itch.io en el siguiente enlace: https://orderframe.itch.io/order-frame-demo

Apéndice B. Documento de diseño de juego

Análisis de Juego

Order Frame es un ARPG (Action Role-Playing Game) de mundo abierto en tercera persona, centrado en la toma de decisiones morales. El núcleo del juego se fundamenta en un Sistema de Alineamiento Dinámico que registra las acciones del jugador (matar, interactuar) y las clasifica invisiblemente en tres categorías: Cielo, Tierra e Infierno. Este alineamiento, una vez elegido, no solo afecta la narrativa (reacciones de NPCs y misiones), sino que modifica las estadísticas base del personaje (salud, daño y velocidad) de forma permanente, creando una experiencia de juego reactiva.

El bucle central de juego combina la exploración no lineal de un mundo de grandes dimensiones, un sistema de combate avanzado y un complejo sistema de progresión y gestión de inventario.

Objetivo del Proyecto

El objetivo principal es diseñar e implementar un prototipo funcional de Order Frame utilizando Unreal Engine 5.6 [16] y su sistema de Blueprints [8], con una arquitectura de software modular y escalable.
Los objetivos específicos a nivel de diseño son:

1. Implementar un Sistema de Alineamiento Dinámico original que modifique las estadísticas (salud, daño, velocidad) y la relación con el entorno.

2. Crear una Arquitectura Modular basada en Blueprints [8], Interfaces [11] y Event Dispatchers [10] para asegurar la separación de responsabilidades, la escalabilidad y la mantenibilidad del código.

3. Integrar un Sistema de Combate Avanzado que incluya equipamiento dual de armas y pociones consumibles.

4. Implementar una Narrativa Reactiva mediante NPCs funcionales y un sistema de Persistencia (Save Game) que guarde el estado dinámico del mundo y el progreso.

Género

Principal: ARPG (Action Role-Playing Game).
Subgéneros: Mundo Abierto, Combate en Tercera Persona.

Plataforma

Principal: PC.

Audiencia objetivo

El juego está diseñado para personas que disfrutan de experiencias de mundo abierto (como *Skyrim* o *Horizon Zero Dawn*) con **narrativas reactivas** y **elecciones morales** significativas (como *Undertale* o *The Witcher 3: Wild Hunt*). El nivel de dificultad es medio-bajo, buscando ser un reto sin llegar a ser frustrante. Está pensado para jugadores que valoran la *agencia*, la *exploración* y las *consecuencias* a largo plazo de sus acciones.

Historia y Personajes

Historia

El protagonista es una entidad conocida como la "Llama del Juicio", que se manifiesta en tres identidades según el plano: Caelvorn (Cielo), Elandor (Tierra) y Azrakar (Infierno).

El protagonista ha sido desterrado del plano celestial tras cometer un acto de traición motivado por la codicia: el asesinato del portador legítimo de un arma divina para robarla. El arma sustraída (MoonSword) se convierte en el arma inicial del jugador, simbolizando su culpa.

El mundo principal se divide en tres planos de existencia (Cielo, Tierra e Infierno) que reflejan el conflicto moral.

Personaje Jugable

Imagen del personaje	Nombre/Identidad	Concepto Visual
 **Figura 313:** Personaje principal.	Caelvorn / Elandor / Azrakar	Diseño de dualidad física explícita, con una mitad del cuerpo y un ala de rasgos angelicales y la otra mitad con características demoníacas, actuando como metáfora visual del conflicto ético.

Tabla 20: Información personaje principal.

Personajes No Jugables (NPCs)

El mundo está poblado por NPCs amigables y hostiles que cumplen funciones:

- Comercio: Comerciantes de Cielo (Hals), Tierra (Alas) e Infierno (Nors), cada uno operando con su moneda regional.
- Narrativa/Misiones: Asignan misiones y ofrecen diálogos con elecciones que modifican el alineamiento.
- Modificadores: NPCs específicos que otorgan bendiciones o maldiciones (modificadores de estadísticas).
- Enemigos: Unidades estándar (Lizard, Wolf, Esqueleto) y el Jefe Final (Gluttony).

Descripción Gameplay

Order Frame es un ARPG que promueve la **exploración no lineal** y el **backtracking** asistido por la progresión del alineamiento y la obtención de armas y etiquetas (bendiciones y maldiciones).

Experiencia del jugador

El jugador avanza:

1. **Progresión por Habilidades:** Obtiene nuevas formas de moverse (ej. triple salto por poción o bendición de velocidad) y más daño al combatir.
2. **Narrativa Reactiva:** Las decisiones alteran el alineamiento, lo que cambia permanentemente las estadísticas y la relación con NPCs, afectando a los finales (final positivo/negativo).
3. **Domesticación:** Puede domesticar animales acompañantes (ej. Fenra y Skoll) que ayudan en combate.

Pautas para desarrollar el Gameplay

El desarrollo se priorizó bajo una metodología de **Vertical Slice** y **Arquitectura Modular**.

Pauta/Requisito	Descripción Funcional Clave
Movilidad Avanzada (RF1)	Implementar carrera, doble salto (tipo *flutter*) y rodar (*roll*) con *frames* de invulnerabilidad.
Combate Táctico (RF5, RF8)	Permitir equipamiento dual de espadas, uso de pociones en tiempo real y sistema de fijación de objetivos (*lock-on*).
Alineamiento Dinámico (RF12, RF13)	Registrar acciones morales para modificar estadísticas permanentes (vida, daño, velocidad) según afinidad (Cielo, Tierra, Infierno).
Gestión de Recursos (RF4, RF6)	Inventario por pestañas (Armas, Magia, Misiones), sistema de creación (Alquimia de pociones).
Mantenibilidad (RNF7, RNF8)	Utilizar una arquitectura modular (clases padre/hijo) con comunicación por *Interfaces* y *Event Dispatchers*.

Tabla 21: Pautas de desarrollo de Order Frame.

Objetivos de juego y Recompensas

Objetivo	Recompensa Principal	Penalización
Principal: Derrotar al Jefe Final (Gluttony).	Desbloqueo del portal de escape (cierre de la demo).	Pantalla de *Game Over* y reinicio al último punto de guardado.
Progresión: Explorar y completar misiones.	Nueva información narrativa, monedas y recursos.	Falta de preparación en las batallas.
Alineamiento: Alcanzar umbral de 100 puntos en una alineación.	Mejora permanente de estadísticas base y desbloqueo de la alineación en la interfaz.	-
Misiones Secundarias	Recompensas monetarias y de alineamiento.	-

Tabla 22: Objetivos y recompensas de Order Frame.

Mecánicas de Juego

Sistema	Mecánica	Descripción
Movimiento y Control	Evasión (*Roll*)	Concede *frames* de invulnerabilidad temporal.
	Doble Salto (*Flutter*)	Movilidad avanzada para acceder a nuevas zonas.
Combate	Equipamiento Dual	Permite llevar dos espadas y alternarlas dinámicamente.
	Fijación de Objetivo (*Lock-on*)	Centra cámara y ataques sobre un enemigo específico.
Progresión/Estadísticas	Bendiciones/Maldiciones	Etiquetas que actúan como modificadores.
	Pociones	Consumibles con *buffs* temporales.
	Alineación	Elección tras llegar a un umbral de puntos que otorga *buffs* permanentes.
Inventario/Economía	Creación (Alquimia)	Combinación de dos ingredientes para crear pociones.
	Monedas Regionales	**Hals** (Cielo), **Alas** (Tierra), **Nors** (Infierno). Se usan en tiendas específicas.
Interacción	NPC Talk (Interface)	Diálogos simples o con opciones que modifican el alineamiento.

Tabla 23: Mecánicas principales de Order Frame.

Controles

Botón/Input	Acción que realiza
WASD	Moverse (Movimiento Frontal/Lateral).
Espacio (Space Bar)	Salto (Doble Salto/Flutter).
Click Izquierdo (Mouse)	Ataque si espada está equipada.
Click Derecho (Mouse)	Mover cámara / desequipar de un slot en el inventario.
Ctrl Izquierdo	Rodar (Roll).
Mayús Izquierdo (Shift)	Correr (Sprint).
I	Abrir/Cerrar Inventario.
M	Abrir Mapa.
R o F	Interactuar con objetos / NPCs.
C	Fijación de Objetivo (*Lock-on*).

Tabla 24: Controles de Order Frame.

Estética del Juego e interfaz de usuario

Estética y Dirección Artística

El juego adopta una estética de **Fantasía Oscura** con una estructura conceptual basada en los planos de existencia (Cielo, Tierra e Infierno), cada uno con un lenguaje visual distintivo. Los modelos 3D son generados mediante un **flujo de trabajo híbrido** (Rodin AI, Mixamo) y los gráficos 2D (sprites/iconografía) son dibujados a mano (Photoshop) para mantener coherencia.

Interfaz de Usuario (HUD)

El diseño de la interfaz es **minimalista** para preservar la inmersión, centrándose en la información crítica:

- **Vida:** Sistema de **corazones segmentados**. 6 corazones base, cada uno con 4 impactos (24 puntos de vida totales base). Los corazones se implementan como *Widget Blueprints* individuales para una actualización dinámica y visualmente clara de la salud y el alineamiento.
- **Equipamiento Rápido:** Dos slots en la parte inferior izquierda para las armas equipadas (toggle con Q y E).
- **Notificaciones:** El tutorial se muestra en la esquina superior derecha, notificación de animal encontrado arriba en el centro y al recoger objeto sale una notificación a la izquierda de la pantalla bajo los corazones.
- **Mapas/Menús:** Se implementan mediante *Widget Blueprints* dinámicos (*bindings*) que actualizan la información en tiempo real.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA