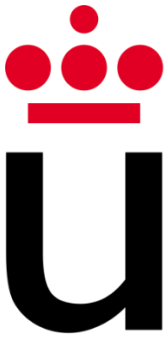


Universidad Rey Juan Carlos

**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA**

**GRADO EN DISEÑO Y DESARROLLO DE VIDEOJUEGOS**

**Curso Académico 2023/2024**

**Trabajo Fin de Grado**

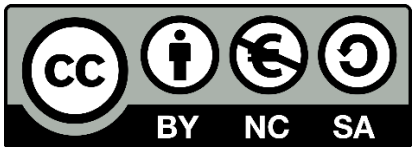
# DESARROLLO DE UN JUEGO ARENA DE BATALLA EN UNREAL ENGINE


**Autor:**

**Roberto Herencias Muñoz**

**Tutor:**

**Aarón Sújar Garrido**



*Agradecimientos a mi familia,*

*que siempre ha estado apoyándome en este largo proceso.*

*A mis amigos, que son la familia que elegimos,*

*en especial a Luis por compartir conmigo este proyecto.*

*A mi tutor Aarón por aceptar y guiar este proyecto,*

*y por su trato cercano y amable en cada reunión.*

# Resumen

Este TFG presenta el desarrollo de una demo técnica, realizada de manera conjunta con Luis Torres Valera, enfocada en el desarrollo de un videojuego de arena de batalla, similar al popular juego *League of Legends (Riot Games, 2009)*, donde los personajes, conocidos como "*fighters*" cuentan con habilidades y estadísticas que van mejorando tras cada ronda, mediante la compra de objetos que potencian dichas características.

El proyecto se centra en el apartado de programación y no tanto en el diseño. Se han creado diferentes modos de juego que permiten enfrentamientos en multijugador local, en modalidad 1vs1, 2vs2 y todos contra todos de 4 jugadores. Además, se ha incluido un modo de juego donde luchan entre sí dos Inteligencias Artificiales (IA).

La demo técnica ha sido desarrollada usando Unreal Engine, aprovechando su potente motor gráfico y el nivel de calidad que aporta a los proyectos. Uno de los pilares principales del proyecto ha sido la implementación del *plug-in* Gameplay Ability System (GAS) para gestionar las habilidades y estadísticas de los personajes, gracias a esto se ha conseguido una gran flexibilidad y escalabilidad en el desarrollo de las mecánicas de combate.

## Palabras Clave

Unreal Engine

Gameplay Ability System

Demo técnica

Videojuego

Multijugador local

Inteligencia artificial

Mecánicas de combate

Arena de batalla

# Abstract

This TFG presents the development of a technical demo, carried out jointly with Luis Torres Valera, focused on the creation of a battle arena video game, like the popular game *League of Legends (Riot Games, 2009)*, where the characters, known as "*fighters"*, have abilities and statistics that improve after each round through the purchase of items that enhance these characteristics.

The project focuses on the programming aspect rather than design. Different game modes have been created that allow for local multiplayer battles in 1vs1, 2vs2, and 4-player free-for-all modes. Additionally, a game mode where two Artificial Intelligences (AI) fight against each other has been included.

The technical demo has been developed using Unreal Engine, taking advantage of its powerful graphics engine and the quality it brings to projects. One of the main pillars of the project has been the implementation of the Gameplay Ability System (GAS) *plug-in* to manage the character's abilities and statistics. This has provided great flexibility and scalability in the development of combat mechanics.

## Keywords

Unreal Engine

Gameplay Ability System

Technical demo

Video game

Local multiplayer

Artificial intelligence

Combat mechanics

Battle arena

# Índice de contenidos

# Glosario

| | |
|---|---|
| ***Assets*** | Recursos digitales utilizados en el desarrollo de videojuegos, como modelos 3D, texturas, sonidos, etc. |
| ***Blueprints*** | En Unreal Engine, se trata de un sistema visual de desarrollo de videojuegos que permite crear lógica de juego mediante la conexión de nodos en un entorno gráfico sin necesidad de programación tradicional. |
| ***Buff*** | Mejora temporal de las habilidades o atributos de un personaje en un videojuego. |
| ***Bug*** | Error o falla en el *software* que provoca un funcionamiento incorrecto del programa. |
| ***Checklist*** | Lista de verificación utilizada para asegurarse de que se han completado todas las tareas o requisitos necesarios en un proceso. |
| ***Cooldown*** | Periodo de tiempo que debe esperar un jugador antes de poder usar una habilidad o acción nuevamente. |
| ***Delay*** | Retraso en la ejecución o respuesta de una acción o proceso. |
| ***Feedback*** | Retroalimentación que se recibe sobre el desempeño o resultados de una acción, proyecto o producto, utilizada para realizar mejoras o ajustes. |
| ***Framework*** | Estructura y conjunto de herramientas que facilita el desarrollo de aplicaciones al proporcionar soluciones predefinidas y estándares de trabajo. |
| ***Hardware*** | Componentes físicos de un sistema informático, como la CPU, la memoria, los dispositivos periféricos, etc. |

| | |
|---|---|
| ***Kiting*** | Estrategia de atacar a un enemigo mientras se mantiene fuera de alcance, moviéndose constantemente para evitar ser golpeado. |
| ***Marketplace*** | Plataforma en línea donde se compran y venden recursos digitales, como personajes y objetos para el desarrollo de videojuegos. |
| ***Matchmaking*** | Proceso de emparejar jugadores en un juego en línea en función de su destreza, preferencias y otros criterios que garanticen partidas equilibradas. |
| ***Minions*** | Personajes controlados por la inteligencia artificial en los videojuegos, generalmente enemigos o aliados menores. |
| ***MOBA*** | (Multiplayer Online Battle Arena) Género de videojuego en el que equipos de jugadores compiten en arenas con el objetivo de destruir la base del equipo contrario. |
| ***NavMesh*** | Malla de navegación utilizada en videojuegos para definir áreas de movimiento y permitir que los personajes controlados por IA naveguen eficientemente por el entorno. |
| ***NPC*** | Personaje no jugable controlado por inteligencia artificial en un videojuego, que interactúa con jugadores y contribuye a la narrativa o jugabilidad. |
| ***Online*** | Estado de conexión a internet que permite la interacción y comunicación en tiempo real entre dispositivos. |
| ***Pipeline*** | Secuencia de procesos o pasos necesarios para completar una tarea, comúnmente usada en desarrollo de *software* y gráficos. |
| ***Pitch*** | Propiedad del sonido que determina su altura, es decir, cuán grave o agudo es según la frecuencia de las vibraciones sonoras. |
| ***Plug-in*** | Componente de *software* que se añade a una aplicación principal para proporcionar funciones adicionales específicas. |

| | |
|---|---|
| ***Prompt*** | Instrucciones o entrada que se proporciona a un sistema o programa para que realice una acción específica o genere una respuesta. |
| ***RPG*** | (Role-Playing Game) Género de videojuego donde los jugadores asumen el rol de personajes en un mundo ficticio, desarrollando habilidades y completando misiones. |
| ***Script*** | Código escrito en un lenguaje de programación que automatiza tareas y define comportamientos en aplicaciones y videojuegos. |
| ***Software*** | Conjunto de aplicaciones y programas que permiten a un ordenador realizar tareas específicas. |
| ***Sprint*** | Periodo de tiempo definido en el desarrollo ágil de *software*, durante el cual se completan tareas específicas y se entrega una parte funcional del proyecto. |
| ***Tag*** | Etiqueta o marcador utilizado para categorizar y gestionar objetos, eventos, o elementos dentro de un juego. |
| ***Videojuego AAA*** | Juego de alta gama con una producción y presupuesto elevados, caracterizado por gráficos avanzados, jugabilidad compleja y extensa, y una narrativa elaborada. |

# Capítulo 1
# Introducción

## 1.1. Descripción del problema

Se plantea desarrollar una demo técnica con el motor de desarrollo Unreal Engine en su última versión 5.3, que dé como resultado un videojuego con diferentes modos de juego posibles, un conjunto de personajes con habilidades y estadísticas diferentes de entre los cuales poder elegir con quien jugar, así como un conjunto de objetos que mejoren los atributos de los personajes tras cada ronda de juego, y que podrán ser adquiridos mediante un sistema de economía, recompensando la realización de acciones durante las rondas de juego y según si se sale victorioso o derrotado de los combates. Las estadísticas de los personajes, habilidades y objetos se implementarán mediante el *plug-in* Gameplay Ability System (GAS). Además, se adaptará para ser jugado en multijugador local, por lo que es necesario abordar la sincronización de eventos que permitan manipular los menús de forma combinada a todos los jugadores, pudiendo elegir personaje o comprar objetos desde un menú común para todos. También se pretende dotar de comportamientos autónomos a una Inteligencia Artificial (IA) que pueda realizar combates de forma exitosa.

El proyecto se centra principalmente en los aspectos de programación y no en el diseño artístico. Por tanto, parte del material se ha diseñado a mano, pero otra parte de este ha sido seleccionada de la *Marketplace* de Unreal Engine, u obtenido de bancos gratuitos con licencia de uso, o mediante generación con inteligencia artificial.

El trabajo se ha realizado de manera conjunta con Luis Torres Valera, para solventar en parte la dificultad de enfrentarse al aprendizaje del uso de un motor que no se ha utilizado nunca. Permitiendo, además, lograr un resultado mucho mayor que el que podría lograrse en solitario, explorando más características del motor.

## 1.2. Motivación

Unreal Engine es una herramienta que no se ha tenido la oportunidad de emplear durante el desarrollo de las asignaturas impartidas en el grado, sin embargo, es un motor en auge en la industria del videojuego que cada vez es más usado por desarrolladores que no tienen un motor propio ya que cuenta con una amplia gama de posibilidades técnicas que aportan calidad al resultado final, y que está en constante desarrollo, recibiendo actualizaciones de manera frecuente para añadir nuevas funcionalidades y mejorar las ya desarrolladas en versiones anteriores [1].

Por otro lado, Unreal Engine se encuentra desarrollado en *C++* y es de código abierto, lo que permite que durante el desarrollo del trabajo pueda utilizarse programación en *C++* partiendo de la base de Unreal, y el lenguaje propio del motor desarrollado como *Blueprints*. *C++* es el lenguaje más estandarizado en el desarrollo de videojuegos, no obstante, durante las asignaturas impartidas en el grado, el lenguaje más utilizado ha sido *C#*.

En vistas laborales, este trabajo servirá para reforzar los conocimientos de *C++* y su uso aplicado a los videojuegos, así como aprender a utilizar con destreza el motor Unreal Engine.

También se ha decidido incluir el uso del *plug-in* Gameplay Ability System (GAS), un *framework* que permite el desarrollo de habilidades y atributos de personaje con un alto nivel de personalización. Este se encuentra implementado en *C++* y cuenta con un conjunto de clases que permite desarrollar juegos tipo *RPG* o *MOBA* donde los personajes comparten habilidades y atributos entre ellos, que pueden verse modificados situacionalmente por el nivel, efectos temporales, objetos equipados, mejoras, etc. Al tratarse de un *plug-in* desarrollado por Epic Games, es utilizado en alguno de sus títulos más conocidos, como puede ser *Fortnite (Epic Games, 2017)* donde todos los objetos que forman parte del videojuego comparten el atributo de poder recibir daño. Se considera que su aprendizaje igualmente es una herramienta útil para el futuro.

Realizar el proyecto de forma conjunta parte del interés común de ambos alumnos por aprender a utilizar Unreal Engine, pudiendo afrontar el reto en todas las áreas de desarrollo y no centrarse solo en el aprendizaje de un área en específico. Obteniendo un resultado de mayor envergadura que realizando el proyecto en solitario, y permitiendo explorar más características del motor.

Esto conlleva otra de las principales motivaciones de este trabajo, mejorar las habilidades blandas y las competencias de trabajo en equipo, así como el aprendizaje del correcto uso de herramientas colaborativas como GitHub Desktop o Trello, debido a que son aptitudes y aplicaciones altamente valoradas en el campo de los videojuegos a nivel laboral.

## 1.3. Objetivos

El objetivo principal del proyecto consiste:

- **La realización de un prototipo de juego con el motor de desarrollo Unreal Engine en su última versión 5.3.**

Se presentan a continuación los objetivos secundarios:

- **Aprender a utilizar el motor Unreal Engine con soltura.**
- **Aprender a programar en *Blueprints* y mejorar los conocimientos de *C++*.**
- **Uso del *framework* Gameplay Ability System (GAS) para desarrollar habilidades y atributos de personaje.**
- **Implementación de menús e interfaces de usuario funcionales para usarse de manera sincronizada por todos los jugadores en multijugador local.**
- **Implementación de Inteligencia Artificial (IA).**
- **Adquirir competencias de trabajo en equipo, y uso de herramientas colaborativas.**
- **Lanzamiento del videojuego.**

## 1.4. Planificación inicial

Se ha realizado una separación de tareas inicial para justificar el trabajo realizado por cada integrante, demostrando que el trabajo que se realizará será de manera individual, pero colaborando para que el resultado del proyecto pueda alcanzar una mayor extensión.

Ambos alumnos tendrán la oportunidad de adquirir experiencia con el *plug-in* Gameplay Ability System (GAS), pues la parte inicial, que actúa a modo de base del proyecto será común para ambos y trabajada simultáneamente. A continuación, se reparten el resto de las tareas de la siguiente forma:

| **Roberto Herencias Muñoz** | **Luis Torres Valera** |
|---|---|
| Personajes (Tanques / Luchadores / Asesinos) | Personajes (Tiradores / Magos) |
| Programación del Sistema de Rondas y Mejoras | Programación de la Tienda y Sistema de Vida General |
| Programación de las Habilidades de Movimiento | Programación de los Objetos de la Tienda |
| Diseño de Interfaz | Diseño de Nivel |
| Programación de Menús | Estructuración de Escenario |
| Sonidos y Música | Efectos Visuales y Partículas |
| IA (Personajes) | IA (Minions) |

*Ilustración 1 – Lista de tareas del proyecto repartidas entre los integrantes.*

El proyecto se plantea de forma iterativa, con *sprints* de 2 semanas cada uno, al final de los cuales se realiza una fase de *testing*, donde cada integrante del equipo testea los hitos conseguidos por el otro compañero, permitiendo a cada alumno adquirir noción de los apartados trabajados por el compañero. Este proceso se repite hasta llegar al producto final.

## 1.5. Estructura de la memoria

La memoria está organizada en tres partes principales:

- **Introducción:** En esta sección se proporciona una visión general del proyecto, incluyendo los objetivos, la motivación detrás de la creación del videojuego, y un resumen de la metodología utilizada. Se describe la relevancia del trabajo en el ámbito de los videojuegos y la programación.
- **Desarrollo del proyecto:** Esta sección constituye el núcleo de la memoria y se subdivide en varios capítulos que detallan el proceso de desarrollo del videojuego. Se abordan temas como la planificación del proyecto, la elección de herramientas y tecnologías, y la implementación de las diferentes funcionalidades del juego. Además, se discuten las dificultades encontradas durante el desarrollo, así como las estrategias y soluciones implementadas para superarlas. Entre los aspectos tratados se incluyen:
    - Implementación del Gameplay Ability System (GAS) para la gestión de habilidades y estadísticas de los personajes.
    - Desarrollo de modos de juego multijugador local y de inteligencia artificial junto con sus correspondientes interfaces.
    - Integración de los *assets* del Unreal Engine Marketplace.
    - Pruebas y ajustes realizados para asegurar el funcionamiento adecuado del juego.
- **Conclusiones:** En la sección final se presentan las conclusiones derivadas del trabajo realizado. Se evalúan los resultados obtenidos en relación con los objetivos planteados al inicio del proyecto. Además, se reflexiona sobre las posibles mejoras y futuras líneas de desarrollo, así como sobre el aprendizaje y las competencias adquiridas durante el proceso. Se destaca también la visibilidad del proyecto y su contribución al campo de la programación y desarrollo de videojuegos.

# Capítulo 2
# Metodología y Herramientas

## 2.1. Metodología

En el desarrollo del trabajo se ha seguido una metodología ágil de desarrollo basada en un modelo iterativo en espiral, implementando el método Kanban (*To Do, Doing, Done*) donde cada estudiante tiene asignadas unas tareas que deberán ser testeadas al finalizar cada sprint de 2 semanas.

El método en espiral [2] es un modelo de desarrollo de *software* que enfatiza en la iteración continua a través de cuatro fases principales: objetivos, análisis de riesgos, desarrollo y evaluación, junto con la planificación de la siguiente etapa. En cada ciclo el proyecto se va mejorando progresivamente, permitiendo adaptarse a cambios con retroalimentación constante, asegurando un desarrollo controlado y eficiente [3].

Kanban [4] se utiliza para mejorar la eficiencia y flexibilidad del flujo de trabajo, implementando un tablero donde visualizar las diferentes etapas del proceso, permitiendo identificar cuellos de botella, limitar el trabajo en progreso y mejorar continuamente el proceso de desarrollo [5].

Estas metodologías complementaron el enfoque de desarrollo ágil, proporcionando una estructura y flexibilidad que resulta en una gestión más efectiva y eficiente del proyecto.

Para materializar el tablero Kanban de forma que ambos integrantes del equipo de desarrollo tuviesen acceso al mismo, se ha utilizado la herramienta gratuita *Trello*[1].

[1] https://trello.com/

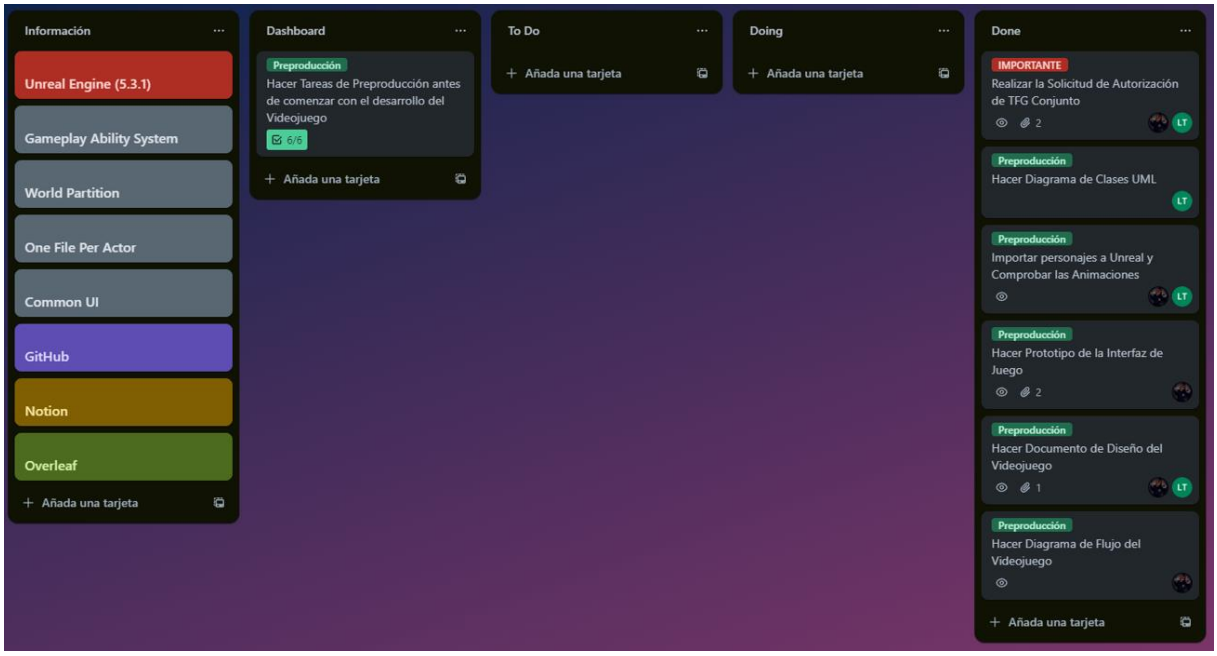



*Ilustración 2 – Tablero de Trello para la organización del proyecto*

## 2.2. Herramientas utilizadas

Durante la realización del proyecto se han utilizado diversas herramientas o aplicaciones que han facilitado la realización u obtención de materiales, el avance en el desarrollo, o la organización de los integrantes del equipo y las tareas asignadas a cada uno.

### *2.2.1. Trello*

Trello proporciona al usuario la posibilidad de crear columnas en las que almacenar tarjetas, y estas tarjetas a su vez pueden estar categorizadas en categorías mediante etiquetas y colores, contener descripciones, archivos, imágenes, documentos, *checklist*, etc., y la posibilidad de asignar personas encargadas de la realización de las tareas contenidas en la tarjeta. Por todo esto, es una herramienta muy útil a la hora de gestionar un trabajo en grupo.

La distribución elegida ha sido una columna con toda la información relevante del proyecto para mantenerse accesible en todo, otra columna *Dashboard* donde se mantienen las tarjetas sin desglosar, con un nombre enfocado en un objetivo más general, con *checklist* que almacenan los objetivos más específicos. Después las columnas de (*To Do, Doing,*

*Done*) donde se encuentran las tarjetas ya desglosadas en subobjetivos, asignados a un integrante del equipo en concreto y todo ello con etiquetas que categorizan la tarea dentro de un tipo: *Diseño, Programación, Música, Sonidos, Partículas, Animaciones, Bugs*, etc. Y una etiqueta de *Urgente* para las tareas que se necesitaban hacer apresuradamente.

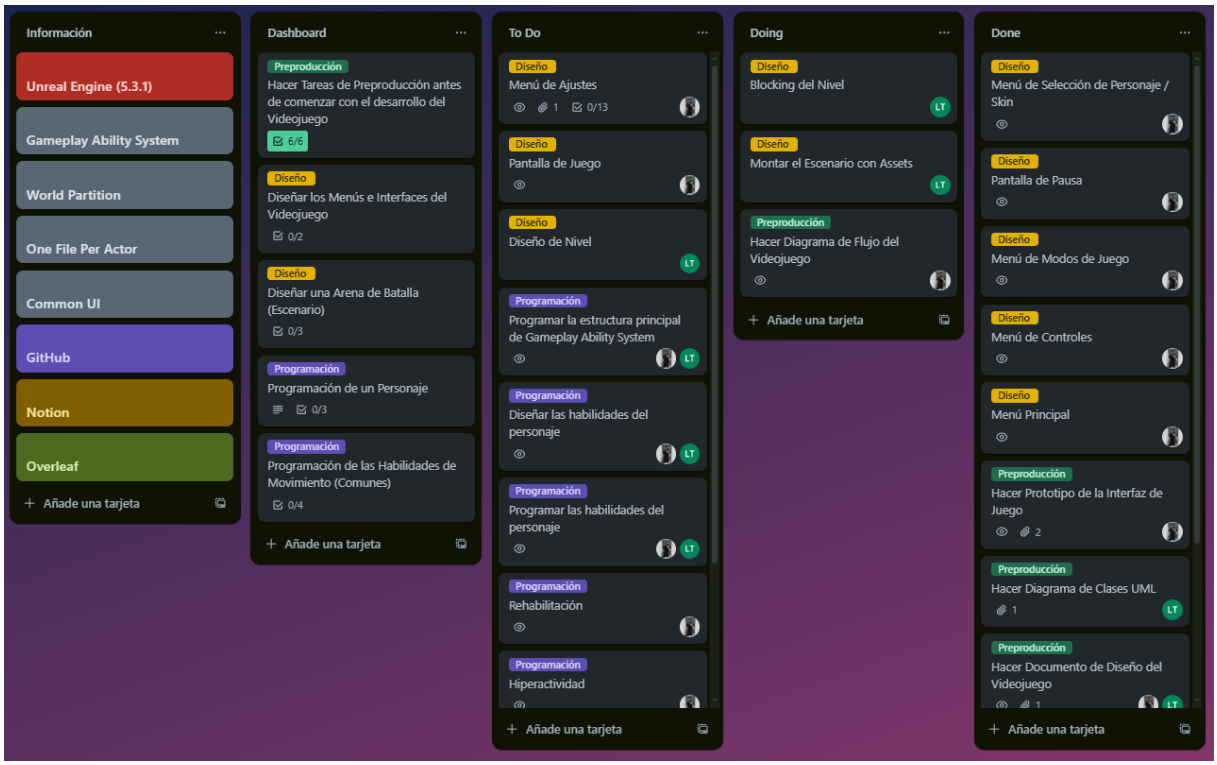



*Ilustración 3 – Tablero de Trello organizado por etiquetas*

### 2.2.2. Notion

Se trata de una aplicación web para gestionar proyectos mejorando la productividad y organización, combinando herramientas como notas, bases de datos, tareas, calendarios, etc. Notion[2] se utiliza habitualmente para colaborar en equipos y centralizar la información de manera estructurada y siendo accesible por todos los integrantes del equipo.

Se ha utilizado para desarrollar el Documento de Diseño de Videojuego, y para apuntar las ideas y las estadísticas que deben tener los personajes y objetos del juego.

[2] https://www.notion.so/es-la


*Ilustración 4 – Documento de Diseño de Videojuego en Notion*

Notion funciona mediante páginas, desde la página principal se colocan referencias a otras páginas, manteniendo el contenido de cada una de ellas separado pero accesible en todo momento.

### *2.2.3. GitHub Desktop*

GitHub[3] es una plataforma de desarrollo colaborativo que utiliza el sistema de control de versiones Git. Permite a los desarrolladores almacenar los proyectos en servidores remotos, pudiendo compartir el código fácilmente con colaboradores. Sus principales características son los repositorios, *issues* y *pull requests*.

[3] https://github.com

La herramienta GitHub Desktop simplifica el flujo de trabajo pues permite comprobar fácilmente los cambios en el proyecto antes de subirlo al servidor remoto, así como actualizar con un solo clic el código local con las actualizaciones que suba al servidor otro colaborador. Permite ver en todo momento el historial de versiones, y, además, permite solucionar incompatibilidades en las versiones, seleccionando a mano el código que debe permanecer en la versión que va a subirse al servidor.

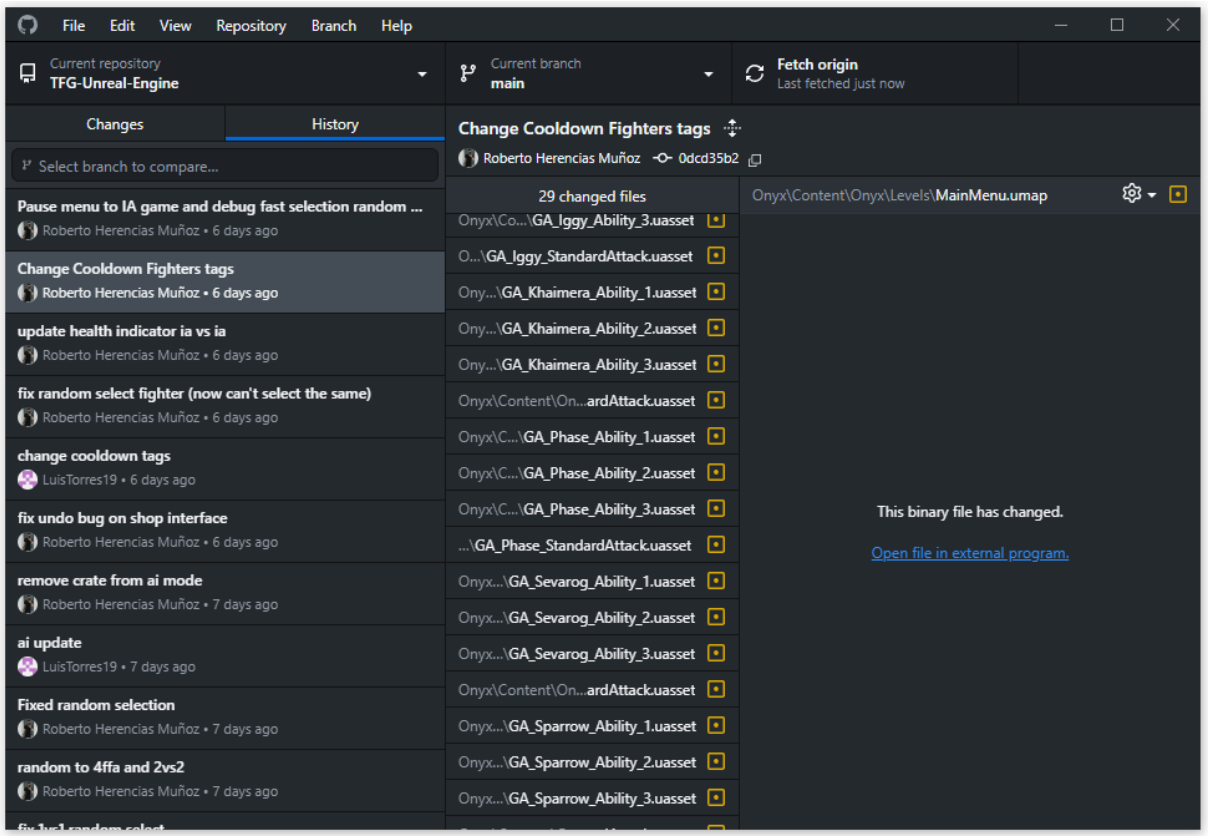



*Ilustración 5 – Historial de versiones de GitHub con los cambios realizados*

*Limitaciones*

Durante la realización del trabajo se ha descubierto una limitación interesante al trabajar en GitHub con un proyecto desarrollado en Unreal Engine. Debido a que Unreal Engine tiene la posibilidad de operar en *Blueprints* mediante nodos, sin necesidad de código, las actualizaciones de versión subidas al servidor de GitHub no pueden solucionar incompatibilidades cuando más de un integrante ha realizado cambios en un mismo *Blueprint*, lo cual dificulta el trabajo simultaneo sobre todo en la fase inicial del proyecto pues

hay muy pocos archivos y la mayoría de las personas necesitan realizar cambios en algún archivo común. Aun así, puede solventarse sin problema si existe comunicación en el equipo y se fijan tareas que no necesiten hacer cambios en los mismos *Blueprints*. Cuanto más avanzado está el proyecto, más difícil es coincidir en un mismo *Blueprint*, pero es interesante comentar la limitación para personas que tomen este documento como referencia para la realización de sus propios proyectos.

También es interesante mencionar que Unreal Engine resuelve satisfactoriamente los conflictos de edición de nivel, esto es gracias a One File Per Actor del sistema World Partition, que incluye Unreal Engine. Este crea un fichero diferente para cada uno de los actores del mapa para permitir a varias personas trabajar sobre el mismo editor de nivel o mapa, sin generar conflictos al subir GitHub los cambios, siempre y cuando no se actúe sobre el mismo actor en concreto.

Por otra parte, es necesario contar con la limitación base de GitHub de no permitir subidas de archivos de mucho tamaño, limitando estos a 100MB. Si se quieren incluir en el proyecto un conjunto de archivos extensos se deberá hacer de manera escalonada.

### *2.2.4. Discord*

Plataforma de comunicación que ofrece chat de texto, voz y vídeo. Permite la creación de servidores que pueden organizarse con diferentes chats y salas de llamada, conocidos como canales, donde tratar diferentes temas y actividades.

Se ha creado un servidor con canales para poder organizar la documentación utilizada, los tutoriales útiles, y posibles aportes, soluciones o ideas de los integrantes para ciertos apartados específicos del proyecto, además, se ha usado como forma de contacto estableciendo llamadas diarias.

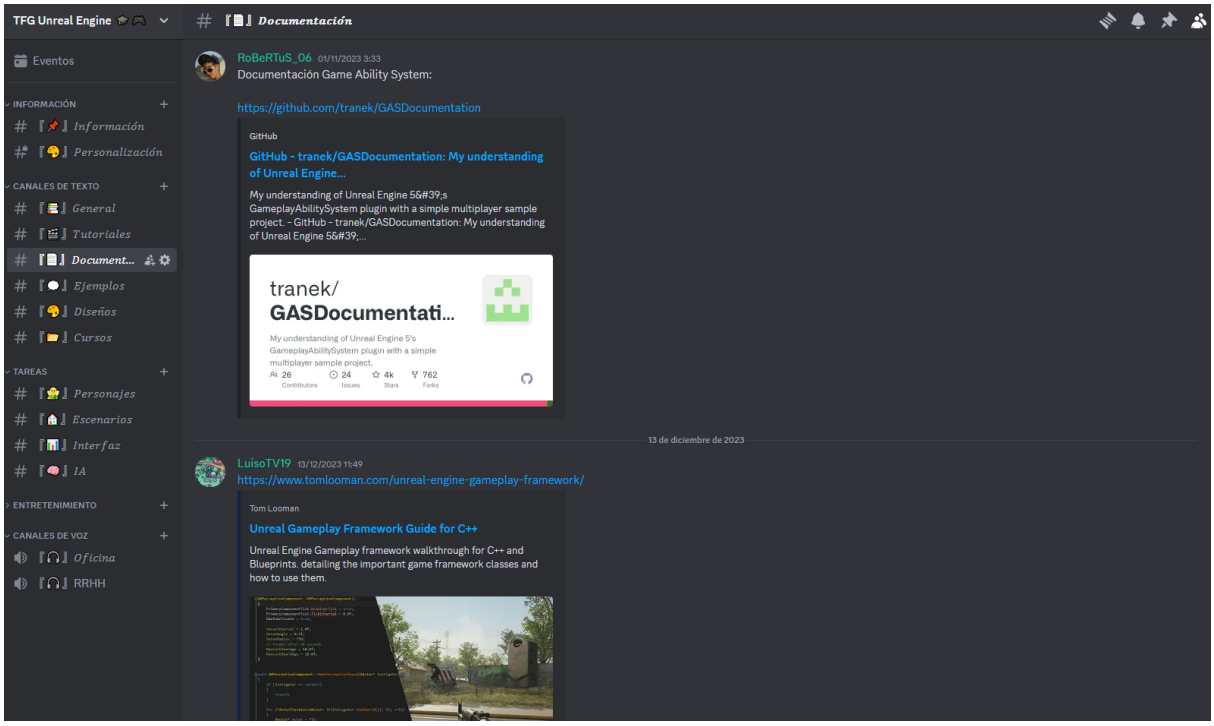



*Ilustración 6 – Servidor de Discord con canales para la documentación, tutoriales, cursos, etc.*

### 2.2.5. Hojas de Cálculo

Se han utilizado hojas de cálculo para representar gráficamente tablas con las que analizar, organizar y almacenar datos. Principalmente se han usado para el balance de los personajes, ajustando los daños, tiempos de enfriamiento de las habilidades, vida, duración de efectos, etc. Estás tablas pueden observarse en el **Anexo III**.

### 2.2.6. Photoshop

Photoshop es una herramienta de edición de imágenes desarrollada por *Adobe*, has sido utilizado para el diseño de las interfaces y sus componentes, estos pueden verse en el **Anexo II**.

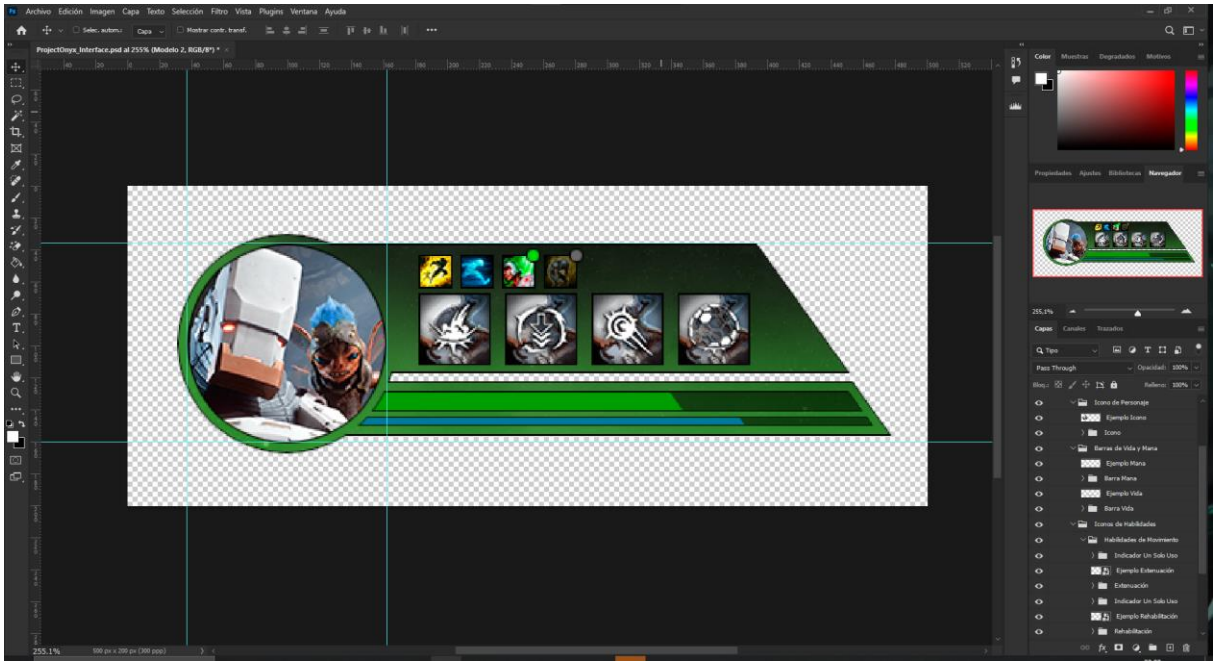


*Ilustración 7 – Photoshop realizándose el diseño de la interfaz de usuario in-game.*

También se ha utilizado para retocar imágenes generadas mediante inteligencia artificial. Como se ha comentado anteriormente en este documento, el enfoque del proyecto está centrado principalmente en la programación y no tanto en el diseño, por lo que se ha recurrido al uso de inteligencia artificial para la generación de algunas imágenes utilizadas en el proyecto, como por ejemplo los iconos de los objetos y las habilidades de movimiento, que también se encuentran disponibles en el **Anexo II**. La inteligencia artificial utilizada ha sido *Adobe Firefly*[4]*,* como *prompt* se ha solicitado imágenes para un videojuego estilo *MOBA*, similar a *League of Legends (Riot Games, 2009)* y se han introducido como referencias objetos e iconos de habilidades del videojuego.

### *2.2.7. Vegas Pro*

Consiste en un programa de edición de vídeo, que también permite aplicar efectos a los sonidos, no se trata de un programa especializado en la edición de sonidos como puede ser *Audacity*, pero se ha utilizado este programa debido a la familiarización con el mismo.

[4] https://firefly.adobe.com

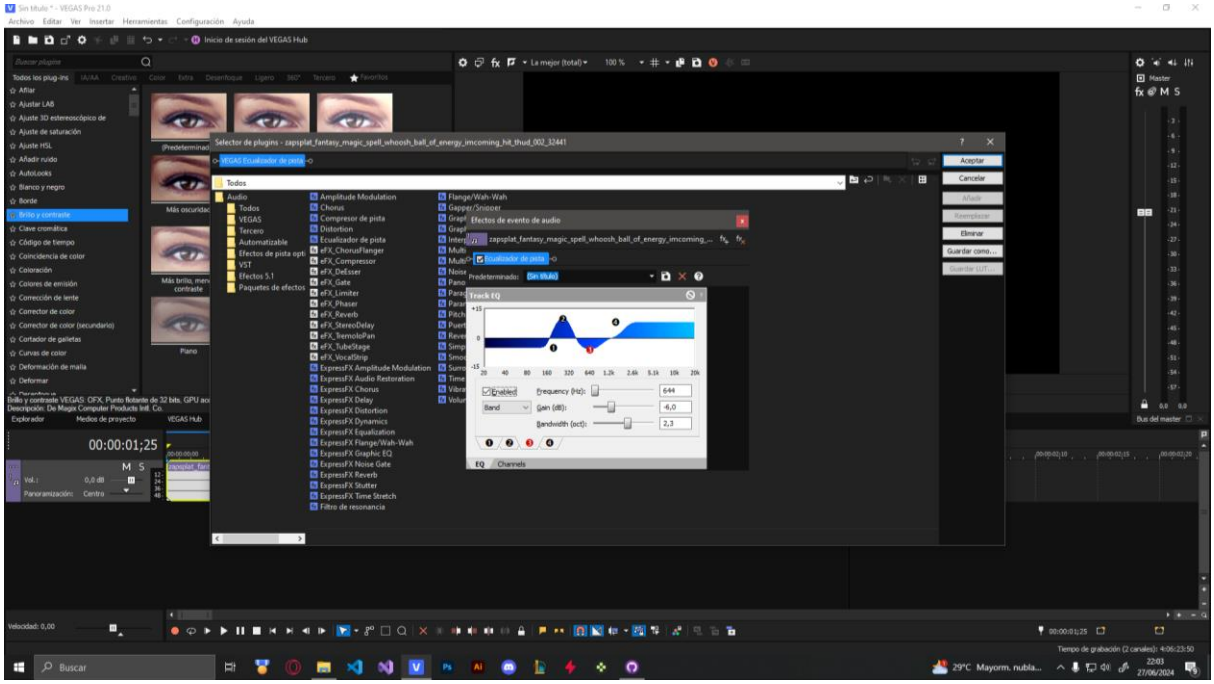


*Ilustración 8 – Vegas Pro en el apartado de efectos de sonido (Ecualización).*

La música y efectos de sonido se han seleccionado de bancos de sonidos con licencia de uso gratuita como *Pixabay* [5]o *Zapsplat*[6], y posteriormente han sido editados en el *pitch*, velocidad, ecualización, etc. Para lograr sonidos distintos a los originales y con varios usos del mismo sonido con distinta edición para distintas armas del juego.

### 2.2.8. *Unreal Engine*

Unreal Engine [6] es un motor de desarrollo de videojuegos creado por *Epic Games*, conocido por su capacidad para generar gráficos de alta fidelidad y su uso extensivo en la industria del entretenimiento. Se trata de un motor que cada vez se usa más y más por empresas desarrolladoras de *videojuegos AAA*, como *Fortnite (Epic Games, 2017), Sea of Thieves (Rare, 2018), Days Gone (SIE Bend Studio, 2019), Kingdom Hearts III (Square Enix, 2019)*, entre otros. Utiliza *C++* como lenguaje de programación principal, y entre sus características destacan el motor de renderizado en tiempo real, su sistema de físicas, las

[5] Music by Amir Firouzfard / Ambient Sound by Empress-Kathryne Nefertiti-Mumbi from Pixabay
[6] https://www.zapsplat.com

herramientas de animación y el editor visual *Blueprints*, que permite la creación de lógica sin necesidad de escribir código.

Unity [7] es un motor de Desarrollo de videojuegos desarrollado por *Unity Technologies*, que soporta múltiples plataformas y utiliza *C#* como lenguaje de programación principal.

*Comparativa entre Unreal Engine y Unity*

Para el desarrollo del proyecto se ha seleccionado Unreal Engine debido al gran auge de este en los últimos años, presentándose como una buena oportunidad de familiarizarse con el motor de cara a la incorporación laboral. Además, otro de los objetivos principales del proyecto es realizar un conjunto de habilidades y estadísticas de personaje eficiente, compleja y equilibrada, y una vez más Unreal Engine es más atractivo para lograr este hito, gracias al *plug-in* Gameplay Ability System (GAS) que incorpora el ya mencionado motor. Pero de igual manera, se ha realizado una comparativa de ambos motores para concluir el uso de uno u otro motor.

Gráficamente Unreal Engine destaca por su capacidad de producir gráficos de alta calidad, ideales para *videojuegos AAA* y proyectos que requieren de un nivel de detalle visual alto, ya que soporta iluminación y sombras realistas, un post-procesado avanzado y renderizado en tiempo real. Mientras que Unity es más adecuado para proyectos que no requieren de gráficos al nivel de un *videojuego AAA*, como pueden ser juegos de móvil, experiencias de realidad virtual (VR) o realidad aumentada (AR).

En cuanto a la comunidad y recursos de cada motor, ambos cuentan con una amplia comunidad activa, pudiendo encontrar fácilmente tutoriales o resolución de problemas en foros, documentación, recursos educativos, etc.

Ambos muestran un buen rendimiento en una variedad de plataformas como PC, consolas, móviles y VR. Sin embargo, Unreal Engine ofrece un rendimiento optimizado para consolas y ordenadores de alta gama.

Aunque la curva de aprendizaje de Unreal Engine puede parecer compleja, logra unos resultados mejor optimizados y de mayor calidad y realismo visual, que Unity [8].

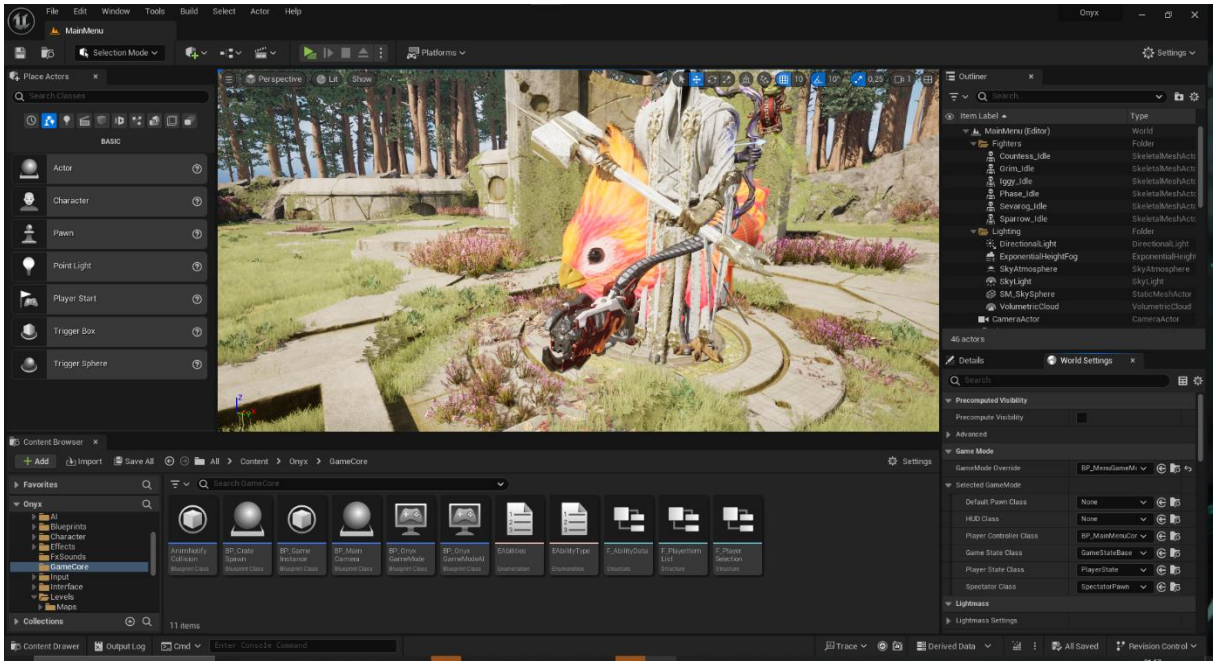


*Ilustración 9 – Unreal Engine 5.3 con el menú principal del proyecto abierto.*

### 2.2.9. *Visual Studio*

Se trata de un entorno de desarrollo integrado (IDE) creado por *Microsoft*. Ofrece herramientas para la edición de código, depuración, diseño de interfaces, gestión de bases de datos, etc. Compatible con múltiples lenguajes de programación como *C#, Python, C++*, y más. Existen bastantes (IDEs) pero se ha decidido utilizar este por su integración con Unreal Engine, ya que este genera automáticamente archivos de solución (.sln) que Visual Studio utiliza para abrir y gestionar el proyecto, además, se puede compilar y construir el proyecto de Unreal Engine directamente desde Visual Studio, lo que facilita el flujo de trabajo continuo.

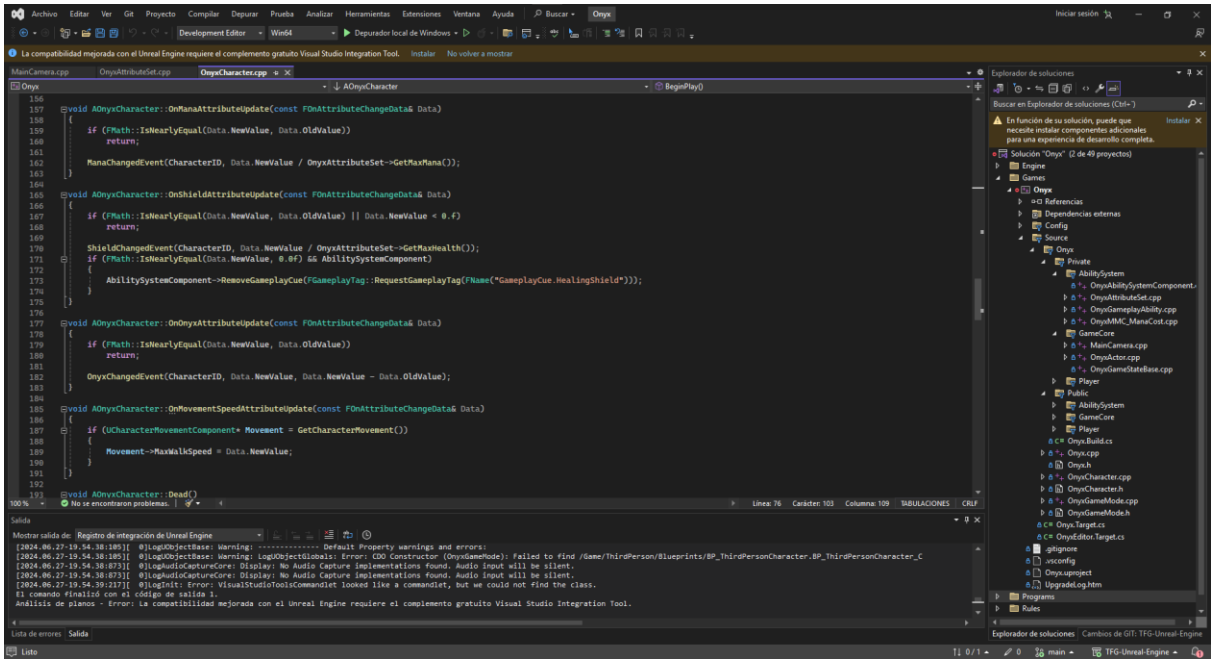


*Ilustración 10 – Visual Studio con el script "OnyxCharacter.cpp"*

### 2.2.10. Marketplace de Unreal Engine

La *Marketplace*[7] de Unreal Engine consiste en una plataforma digital de *Epic Games* donde desarrolladores y diseñadores pueden comprar, vender y compartir contenido y recursos para proyectos de Unreal Engine. Pueden encontrarse modelos 3D, animaciones, texturas, plug-ins, etc. Lo cual facilita el desarrollo de videojuegos con el motor.

Para este proyecto se han seleccionado de la Marketplace los personajes de *Paragon (Epic Games, 2016)*, pues son gratuitos y de uso libre en proyectos de Unreal Engine, y se proporciona a los desarrolladores los modelos 3D, junto con sus voces, animaciones, *skins*, partículas, etc. Todo el contenido se obtiene suelto, sin cohesión, y debe ser montado por el desarrollador.

[7] https://www.unrealengine.com/marketplace/en-US/store

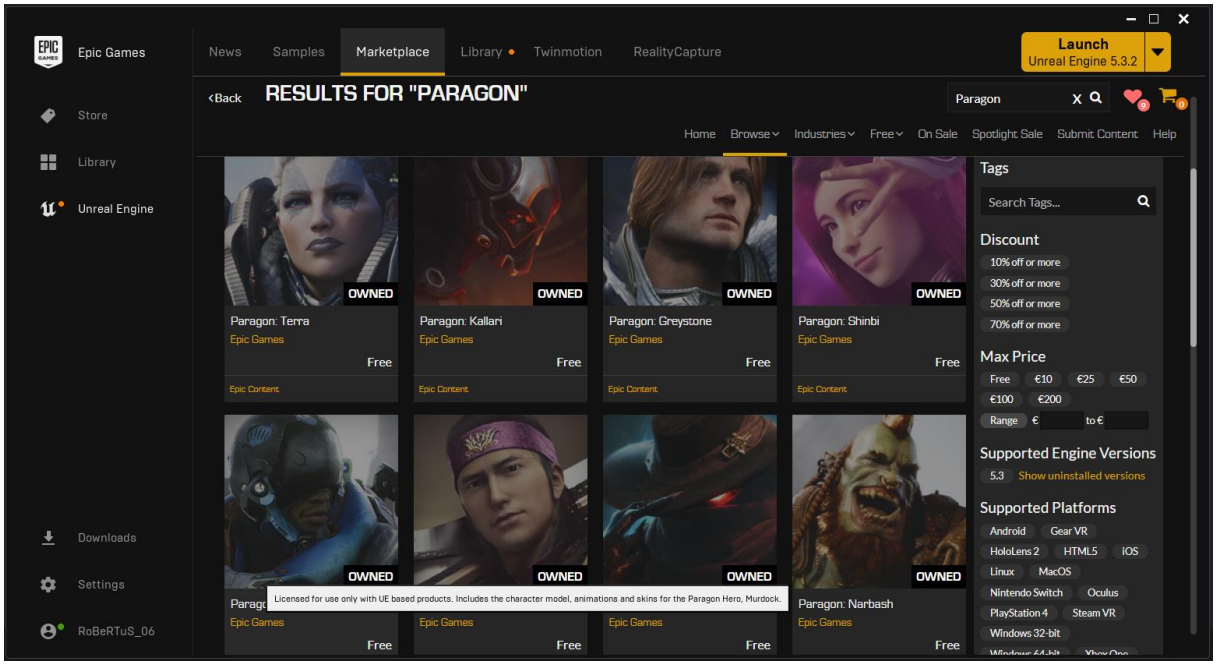



*Ilustración 11 – Marketplace de Unreal Engine, personajes de Paragon con licencia de uso gratuita.*

### 2.2.11. *Quixel Bridge*

*Quixel Bridge*[8] es una aplicación que permite a los usuarios explorar, descargar e importar directamente a Unreal Engine una amplia biblioteca de recursos de alta calidad de *Megascans*. Entre estos recursos se encuentran modelos 3D, superficies, materiales y texturas fotorrealistas, que mejoran la eficiencia y la calidad visual de los entornos y escenarios en proyectos realizados en el motor.

Se ha utilizado para obtener elementos 3D como cajas, rocas y texturas para las superficies de los escenarios.

[8] https://quixel.com/bridge

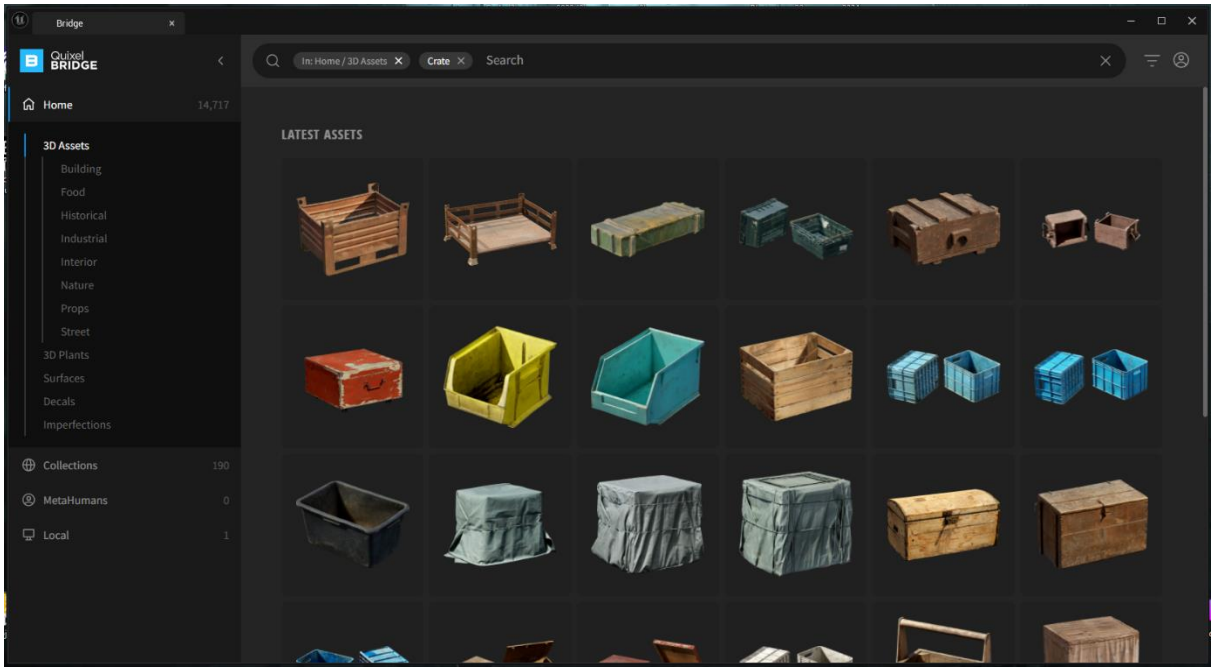



*Ilustración 12 – Quixel Bridge de Unreal Engine realizando la búsqueda de una caja para el proyecto.*

# Capítulo 3
# Estado del arte

## 3.1. Antecedentes

A lo largo de la historia ha habido una gran evolución en lo referente a los videojuegos, comenzando como simples juegos de arcade, hasta convertirse en complejas experiencias interactivas con gráficos de alta fidelidad y narrativas envolventes [9], gracias a los avances tecnológicos en *hardware* y *software*, y sobre todo gracias al desarrollo de motores de juego cada vez más potentes y accesibles [10].

### *3.1.1. Videojuegos de arena de batalla*

Los videojuegos de arena de batalla, como es el caso de este proyecto, han ganado popularidad desde inicios de los 2000, gracias a juegos como *Defense of the Ancients (IceFrog, 2003)* que sentaron las bases del género. Años más tarde aparecería *League of Legends (Riot Games, 2009)* convirtiéndose en uno de los videojuegos más jugados del mundo, e influenciando significativamente el desarrollo de juegos posteriores, pertenecientes al mismo género [11].

### *3.1.2. Motores de videojuego*

En cuanto a los motores de juego, Unreal Engine y Unity son dos de los principales motores de juego en la industria del desarrollo de videojuegos. Estos han facilitado la creación de juegos de calidad gracias a las herramientas y entornos de desarrollo robustos y accesibles que proporcionan.

Como ya se ha visto anteriormente en la comparativa entre Unreal Engine y Unity, el primero destaca por su capacidad para generar gráficos de alta fidelidad y por su potente motor de renderizado en tiempo real. Además, introduce en sus últimas versiones,

innovaciones como Nanite [12] y Lumen [13], que permiten crear entornos de alta calidad con iluminación dinámica. Sin embargo, Unity es más conocido por su flexibilidad y facilidad de uso. Soporta una amplia variedad de plataformas, lo cual hace el motor más atractivo para desarrolladores independientes o estudios pequeños [8].

Para el desarrollo de un juego de arena de batalla es crucial el desarrollo de sistemas de habilidades y estadísticas de personaje. El *plug-in* Gameplay Ability System (GAS) de Unreal Engine proporciona una infraestructura flexible para implementar estas habilidades y estadísticas, y su posterior equilibrado, con modificación de las estadísticas en tiempo real.

### *3.1.3. Tendencias actuales*

En la actualidad se puede observar una tendencia por desarrollar videojuegos que incluyen tecnologías avanzadas, como el trazado de rayos en tiempo real (RTX), experiencias inmersivas en realidad virtual (VR), etc. Pero si algo destaca por encima del resto, es la inclusión de inteligencias artificiales para crear comportamientos de personajes más realistas, y el desarrollo de infraestructuras capaces de soportar el juego en línea, con sistemas de *matchmaking* y de balance de equipos.

### *3.1.4. Conclusiones*

Como conclusión a los datos recopilados se ha decidido realizar la demo técnica con un enfoque de videojuego arena de batalla, con personajes con diferentes habilidades y estadísticas y, además, se ha optado por realizar una inteligencia artificial que controle personajes dentro del juego.

## 3.2. Referencias

Para realizar un proyecto es esencial inspirarse en referencias ya existentes en el mercado que compartan similitudes con los objetivos del trabajo que se pretende llevar a cabo. En este caso al tratarse de un videojuego con características cercanas a las habituales en un *MOBA*, pero con un toque más arcade, de combates rápidos entre varios

jugadores, en una posición de cámara cenital que se va acercando y alejando para permitir un encuadre de todos los elementos principales de la acción en pantalla, se ha encontrado inspiración en juegos como *League of Legends (Riot Games, 2009), DotA 2 (Valve Corporation, 2013), Smite (Hi-Rez Studios, 2014)*, pero también tiene similitudes con juegos como *Brawl Stars (Supercell, 2017), Godstrike (OverPowered Team, 2021)*, o con conocidos juegos de combates arcades como *Tekken 7 (Bandai Namco Games, 2015), Mortal Kombat 11 (NetherRealm Studios, 2019) y Street Fighter 6 (Capcom, 2023)*, que han servido como inspiración sobre todo para los menús de selección de personaje.

*Ilustración 13 – League of Legends.*

*Ilustración 14 – League of Legends (Arena Mode).*

*Ilustración 15 – DotA 2.*

*Ilustración 16 – Smite.*


*Ilustración 17 – Brawl Stars.*

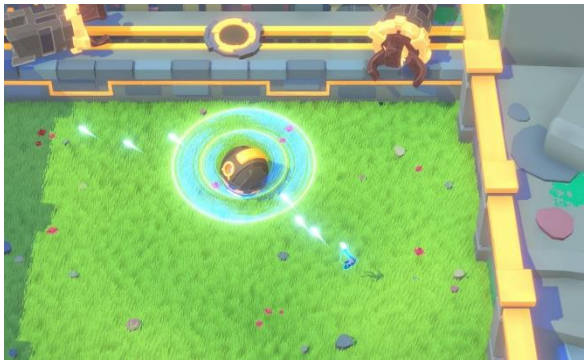


*Ilustración 18 – GodStrike.*


*Ilustración 19 – Tekken 7.*


*Ilustración 20 – Mortal Kombat 11.*


*Ilustración 21 – Street Fighter 6.*

# Capítulo 4
# Diseño del videojuego

## 4.1. Información general

### *4.1.1. Descripción*

***Onyx Project*** se trata de un videojuego de batallas entre jugadores en multijugador local que pueden enfrentarse en modos 1vs1, 2vs2 por equipos, o 4 todos contra todos. Las partidas constan de rondas de máximo 5 minutos de duración, o hasta que todos los jugadores, menos uno, tengan la barra de vida al cero. Al finalizar una ronda se restará un porcentaje de vida general a los jugadores que hayan perdido, y se hará un reparto de dinero entre todos los jugadores en función de su posición en la ronda (primero en morir, segundo en morir, ganador, etc.), pasarán a una interfaz de compra de objetos donde invertir el dinero en objetos que aportan mejoras a las estadísticas de los personajes. Esto se repite hasta que la vida general de todos los jugadores, menos uno, llega a cero y por tanto se proclama campeón de la partida.

El juego tiene una vista cenital o picada y la cámara enfoca la acción alejándose o acercándose para proporcionar una visión completa del mapa con todos los jugadores dentro del plano.

### *4.1.2. Público objetivo y plataformas*

Según el estudio de mercado este tipo de juegos suele estar orientado a un público joven de entre 15 y 30 años, que estén familiarizados con otros juegos de lucha, o juegos tipo *MOBA* o *RPG*, que busquen una experiencia de juego casual.

El proyecto está centrado en la plataforma de PC, pero es perfectamente compatible con la portabilidad a consolas como PlayStation, Xbox o Switch, ya que sus controles están pensados para mando incluso en PC.

### 4.1.3. Género

El proyecto pertenece al género de lucha 1vs1, por equipos o batalla campal, dentro del englobe arena de batalla, (género *MOBA*) pero sin la característica *online,* pues está pensado únicamente para multijugador local.

### 4.1.4. Características clave

Como puntos a destacar está la duración de las partidas, pues se ha enfocado en partidas rápidas y frenéticas. Los efectos visuales de partículas, gracias al desarrollo del proyecto en Unreal Engine, permite aportar calidad visual a las habilidades de los personajes de forma sencilla. También se subraya el tamaño de las arenas de batalla, mapas pequeños para tener en todo momento una visión total del entorno de juego.

## 4.2. Jugabilidad

Primero se selecciona el modo de juego, 1vs1, 2vs2 o 4 todos contra todos.

A continuación, cada jugador selecciona un personaje y combate a muerte haciendo uso de las habilidades de este.

La partida se divide en rondas, donde al inicio de cada una de ellas, todos los personajes tienen la vida y el maná al completo, junto con todas las habilidades disponibles, sin tiempo de enfriamiento pues no se han usado todavía.

Durante las rondas aparecerán en el mapa, en posiciones aleatorias, cajas que aportan dinero extra al jugador que las destruye. Esto busca fomentar la exploración de mapa y no simplemente la lucha constante.

Tras morir todos los personajes menos uno, se termina la ronda y se hace el reparto de dinero entre todos los jugadores (explicado en **Reparto de dinero**) y la substracción de vida general a los perdedores (explicado en **Sustracción de vida general**).

A continuación, se procede a comprar objetos en la tienda durante 1 minuto. Y apto seguido, una nueva ronda. Este proceso se repite hasta que todos los jugadores menos uno, tienen 0 de vida en sus barras de vida general.

### *4.2.1. Reparto de dinero*

El primer jugador en morir obtiene **1500 onyx** al finalizar la ronda, dinero que no es suficiente para la compra de ningún objeto en la primera fase de compra. El ganador obtiene **2000 onyx + bonificación según el porcentaje de vida**, es decir, si finaliza el combate con un 50% de vida, obtiene 3000 onyx (2000 onyx + 50% de 2000), la cantidad base es suficiente para comprar el objeto de mejora de movilidad, que se trata del único objeto que se encuentra a un precio de 2000 onyx, sin embargo, si acaba la ronda con un gran porcentaje de vida, puede ser suficiente para comprar otro, pues el precio de los objetos oscila entre 2500 y 3500 onyx.

En los combates de 4 jugadores, los dos jugadores centrales, es decir, el segundo y el tercero, obtienen **1750 onyx**.

*Bonificación*

Existe una bonificación por acumulación de derrotas, pensada para ajustar las posibilidades de victoria del perdedor.

Cada vez que un jugador queda **último** se **suma un 1 a su contador de derrotas**, cuando **gana** una ronda se **reinicia a 0 el contador**. Los jugadores **segundo y tercero suman un 0.5 a su contador de derrotas**. Cuando el **contador de derrotas es igual a 3** se **proporciona una bonificación** de dinero al final de la ronda al jugador y acto seguido se **reinicia a 0 el contador**, para no acumular bonificaciones.

Si la bonificación se obtiene quedando último, añadirá **6000 onyx** al jugador, por otro lado, si la bonificación se obtiene quedando segundo o tercero, como estás posiciones aportan algo más de dinero base y reciben menos sustracción de vida general, recibirá una bonificación menos cuantiosa, de **4500 onyx**.

### *4.2.2. Sustracción de vida general*

Ya se ha comentado el ajuste de posibilidad de victoria mediante la bonificación de dinero, pero también se ajusta la sustracción de vida general para mantener las posibilidades de victoria de todos los jugadores a lo largo de la partida.

Al finalizar la ronda, los jugadores perdedores ven como su vida general se reduce, y cuando esta llega a cero significa que han perdido el juego y no podrán seguir participando en las siguientes rondas.

Esta sustracción de vida se realiza de la siguiente manera: el **jugador que antes muere durante la ronda recibe una sustracción del 100% del valor base**, mientras que los **jugadores que terminan la ronda en segunda y tercera posición reciben una sustracción del 50% del valor base**.

El valor base se ajusta en función del número de rondas disputadas, en**tre la ronda 1 y la ronda 3 el valor base es de -10, entre la ronda 4 y la ronda 5 el valor base es de -20 y de la ronda 6 en adelante el valor base es de -30**.

Al ir incrementándose el valor en función del número de ronda, sumado al hecho de que el perdedor habitual recibe una bonificación de dinero con la que permitirse una mayor compra de objetos que le permita remontar, ajusta las posibilidades del jugador que va perdiendo, de remontar en valores de vida general al jugador que va ganando.

## 4.3. Personajes

Los personajes tienen habilidades con diferentes características, principalmente pueden clasificarse dentro de habilidades de daño: físico, mágico, o mixto; habilidades de mejora de estadísticas: mayor velocidad de movimiento, mayor velocidad de ataque, o mayor rango; y por último existen las habilidades de gestión de escudo. También pueden diferenciarse en función de su rango de ataque, existen los personajes melé, cuyos ataques son cuerpo a cuerpo, distancia corta; y los personajes de rango, que pueden atacar desde largas

distancias. Otra forma de categorizarlos es según su rol, un personaje puede ser Asesino, Luchador, Mago, Tanque, o Tirador.

Debido a que el proyecto está centrado principalmente en el apartado de desarrollo y no tanto en el apartado de diseño, no se han modelado personajes en 3D, se han cogido de la *Marketplace* de Unreal Engine, de forma gratuita con licencia de uso. Concretamente se han seleccionado personajes de *Paragon (Epic Games, 2016)*, estos paquetes de contenido incluyen los personajes, sus voces, animaciones, partículas y *skins* (igualmente es necesario adaptarlo al proyecto pues las animaciones no vienen montadas, las partículas no vienen aplicadas, sus habilidades no vienen incluidas y su sistema de movilidad no está actualizado a Unreal Engine 5.3).

Siguiendo la clasificación anteriormente mencionada los personajes del proyecto son:

- **Countess** – Asesino, melé, daño físico.
- **Grim** – Tanque, rango, daño físico, gestión de escudo y mejora de estadísticas.
- **Grux** – Tanque, melé, daño físico principalmente, pero alguna habilidad tiene daño mixto.
- **Iggy&Scorch** – Tirador, rango, daño mixto, gestión de escudo y mejora de estadísticas.
- **Khaimera** – Luchador, melé, daño físico principalmente, pero alguna habilidad tiene daño mixto, y mejora de estadísticas.
- **Phase** – Mago, rango, daño mágico y habilidades de inmovilización.
- **Sevarog** – Mago, melé, daño mágico y habilidades de inmovilización.
- **Sparrow** – Tirador, rango, daño físico.
- **Terra** – Tanque, melé, daño físico y gestión de escudo.

A continuación, se muestran atributos comunes a todos los personajes, cuyos valores cambian en función de la clase a la que pertenecen y en función del personaje específico en sí mismo. Dos personajes de una misma clase tendrán atributos similares, pero aun así los valores serán únicos en función de sus características.

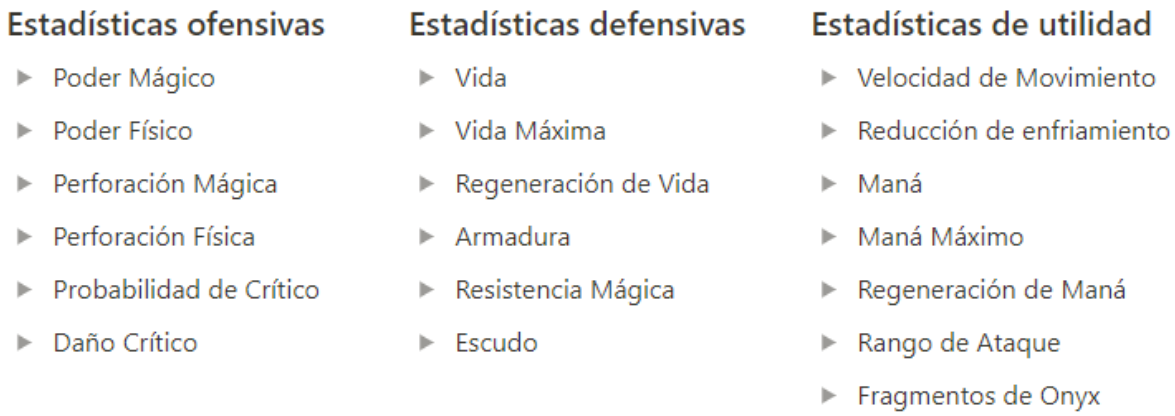



*Ilustración 22 – Lista de atributos comunes de los personajes.*

En el **Anexo III** pueden verse tablas detalladas de cada personaje con sus atributos iniciales, y las características de sus habilidades.

## 4.4. Habilidades de movimiento

Adicionalmente a las habilidades que posee cada personaje, existen en el juego cuatro habilidades de movimiento, comunes para todos los personajes. Permiten una mejora de estadísticas de forma temporal, que permita realizar estrategias diferentes a los jugadores.

- **HiperDash** – permite al jugador teletransportarse en un rango de 400 unidades, en la dirección que está apuntando. Para poder volver a usar esta habilidad tras su activación deberá pasar un tiempo de enfriamiento de 30 segundos.
- **HiperActivity** – aumenta la velocidad de movimiento del jugador en un 48,12% durante 15 segundos. Para poder volver a usar esta habilidad tras su activación deberá pasar un tiempo de enfriamiento de 60 segundos.
- **Healing** – Cura al jugador que usa la habilidad 220 unidades de vida durante los siguientes 10 segundos, además, añade un escudo de 285 unidades para optimizar la regeneración de vida. Esta habilidad tiene un único uso durante la ronda. En caso de no utilizarse no se amontona para la siguiente ronda, es decir, siempre se tiene un único uso por ronda.

- **Exhaust** – Reduce la velocidad de movimiento un 40% de todos los jugadores que se encuentren en un rango de 650 unidades alrededor del jugador, durante 8 segundos. Además, destruye cualquier escudo de forma instantánea. Esta habilidad tiene un único uso durante la ronda. En caso de no utilizarse no se amontona para la siguiente ronda, es decir, siempre tiene un único uso por ronda.

El jugador debe elegir el momento adecuado de usar cada habilidad de movimiento, por ejemplo, el *HiperDash* es bastante bueno para esquivar ataques fuertes del enemigo, la *HiperActivity* es buena para huir de un enemigo o perseguir con mayor facilidad, el *Healing* es útil para curarse cuando se tiene poca vida, y el *Exhaust* se puede utilizar de forma óptima en diferentes momentos de la ronda, puede contrarrestar un *HiperActivity*, puede utilizarse para evitar que un jugador huya durante la curación y además, romper su escudo, o puede utilizarse simplemente para romper el escudo de un jugador si esta acción puede ser determinante para obtener la victoria.

En el **Anexo III** puede verse la tabla de características de las habilidades de movimiento de forma detallada.

## 4.5. Objetos

Tras finalizar la ronda de combate se obtiene dinero, como se ha visto anteriormente en **Reparto de dinero** este dinero se puede invertir en comprar objetos de la tienda que incrementan las estadísticas base del personaje.

Se puede elegir entre ocho objetos diferentes, cada uno de ellos enfocado en la mejora de unos atributos concretos para un tipo de personaje en específico. Hay un objeto que mejora la probabilidad de infligir daño crítico, junto con una mejora de daño base, hay una versión para jugadores de daño físico y otra para jugadores de daño mágico. Hay un objeto para mejorar la penetración, uno en su versión física que mejora el daño físico e incrementa la penetración de armadura, y otro en su versión mágica que mejora el daño mágico y la

penetración de resistencia mágica. Se ha desarrollado un objeto de daño mixto y velocidad de movimiento que aumenta el daño físico y mágico base del personaje. También hay objetos pensados para aumentar las estadísticas de utilidad de los personajes, uno que mejora la velocidad de movimiento, regeneración de vida y regeneración de maná, pensado para ser comprado por todos los jugadores indistintamente de su rol. Y finalmente se han implementado objetos pensados para jugadores tanque, uno que mejora las estadísticas de armadura, resistencia mágica, velocidad de movimiento y vida máxima. Y otro que mejora la vida máxima, maná máximo y los daños físico y mágico del personaje enfocado para cualquier personaje que quiera mejorar sus estadísticas de vida y maná indistintamente del tipo de daño base que inflija este personaje.

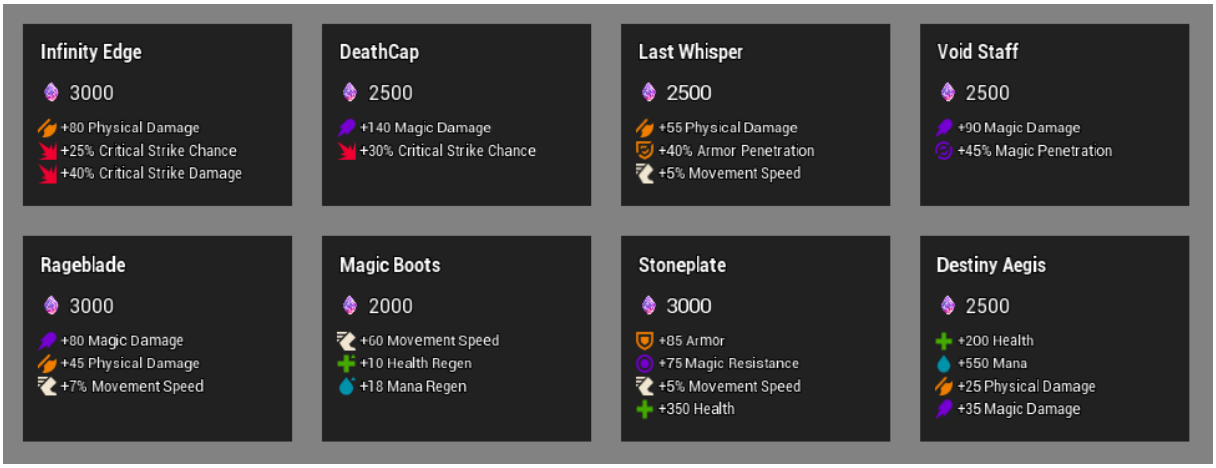



*Ilustración 23 – Objetos con sus correspondientes atributos.*

## 4.6. Inteligencia Artificial

Al inicio del proyecto la idea era realizar una inteligencia artificial diferente cada alumno. Uno se centraría en una inteligencia artificial aplicada a un personaje jugable, y el otro una aplicada a enemigos neutrales.

La primera se trataba de un robot escolta que actuase como guardián y defendiese al personaje, aplicando escudo, curando, recargando el arma, etc., según fuese necesario en cada momento. La segunda estaba enfocada en desarrollar un enemigo que, en cada ronda podría aparecer de forma aleatoria en el mapa, y al matarlo aportaría una mejora de

estadísticas al jugador que acabase con su vida. Este enemigo intentaría atacar a los jugadores, romper las cajas para hacerse con dinero que entregar después al jugador que lo mate, y algunas acciones más.

Esta idea fue descartada finalmente y se optó por implementar una única inteligencia artificial de manera conjunta pero más óptima y con mayor utilidad dentro del prototipo desarrollado.

Esta inteligencia artificial se aplica directamente a los personajes jugables, y se encarga de hacerlos pelear como si estuviesen controlados por un humano. Se ha añadido un modo IA. vs IA. al videojuego y con este modo se pretende enseñar a los jugadores las posibilidades de uso de cada personaje, pudiendo visualizar las habilidades en un combate real, y también es útil para el equipo de desarrollo pues permite obtener estadísticas de que personaje tiene mayor tasa de victoria contra otro, y de esta forma detectar personajes mal ajustados, con atributos mal calibrados.

Para realizar esta inteligencia artificial se han repartido las tareas de la siguiente forma: un alumno se ha encargado del árbol de comportamiento, y el otro alumno se ha encargado de los ajustes de tiempo de reacción de la IA, para que tenga un comportamiento más humano, se han incluido ciertos *delays* en algunos puntos del árbol, y también se han incluido *Environment Query System* (EQS) de Unreal Engine para añadir ciertos comportamientos concretos en condiciones específicas.

El esquema de comportamiento de la inteligencia artificial puede verse en **Anexo I**.

## 4.7. Controles

El juego está pensado para ser jugado con mandos, ya que se trata de un multijugador local, y al tener que apuntar y utilizar un gran número de habilidades, sería imposible que dos jugadores compartieran ratón y teclado, ratón porque solo hay uno, y teclado porque sería muy difícil asignar teclas a todas las habilidades de varios personajes en un solo teclado. Sin embargo, si en un futuro se desarrolla un modo de juego para un solo jugador,

si se podría adaptar fácilmente los controles para manejar el personaje y sus habilidades desde el teclado y ratón.

Los controles adaptados a mandos de diferentes plataformas pueden verse en **Anexo I**.

# Capítulo 5
# Desarrollo

## 5.1. Introducción

Antes de comenzar con el desarrollo del videojuego se ha realizado el documento de diseño del videojuego en Notion para sentar las bases del desarrollo. A continuación, se organizan las tareas de cada alumno y se determina la prioridad de cada una, se refleja en el tablero de Trello. Y como paso final, antes de entrar en el motor, se ha desarrollado una estructura inicial con diagramas de clases UML, y diagramas de flujo para la parte de diseño, que establezcan unas bases iniciales para comenzar con el proyecto. Estos diagramas pueden verse en **Anexo I**.

También se ha buscado una guía de convenciones recomendadas para la nomenclatura de los *assets* en Unreal Engine, y de esta forma mantener organizados los archivos del proyecto [14].

## 5.2. Configuración base

Tal como se puede observar en la **Ilustración 61** del **Anexo I**, que explica la arquitectura base de un videojuego en Unreal Engine, todo videojuego consta de clases propias como marco de desarrollo, y para la realización del proyecto se crean hijos heredados de estas clases para expandir y desarrollar las implementaciones específicas de nuestro videojuego [15].

### *5.2.1. Levels*

Un *Level* en Unreal Engine, es la unidad de contenido que agrupa diversos elementos y componentes que conforman una escena de juego, estos elementos pueden ser iluminación, actores, sonido, scripts, etc., que definen el comportamiento del entorno y de los personajes

dentro de la escena. Cada *Level* es un entorno de juego independiente que puede ser cargado, descargado o cambiado durante la ejecución del juego.

Se han creado *Levels* para cada parte del videojuego con características diferentes y para hacerlos más atractivos se ha diseñado un escenario diferente para cada uno de ellos.

***LV_MainMenu***

Está destinado a contener los menús, principal y de selección de modo de juego, junto con los ajustes, controles y todo lo que pertenece al flujo inicial del juego.

***LV_SelectFighter***

En este *Level* se muestra el menú de selección de personaje, con los personajes en pantalla para poder visualizar de forma más intuitiva, el personaje y la *skin* que está seleccionando cada jugador.

***LV_SelectFighterAI***

Igual que ***LV_SelectFighter,*** pero destinado al modo de juego de IA. vs IA., es necesario un *Level* diferente porque tiene clases diferentes que su homónimo sin AI.

***LV_Arena***

Es el *Level* de juego, destinado al escenario donde combatir, la lógica de movimiento de los personajes, habilidades, etc.

***LV_ArenaAI***

Igual que ***LV_Arena,*** pero destinado al modo de juego de IA. vs IA., es necesario para asignar los controladores de la inteligencia artificial a los personajes en lugar de los controladores destinados a los jugadores. Aprovechando que es necesario un *Level* diferente del resto de modos, se ha creado un escenario diferente sin tanto obstáculo que facilite el tránsito por la *NavMesh*.

***LV_ShopMenu***

Similar a ***LV_SelectFighter***, se trata de un *Level* destinado a mostrar un menú, pero en este caso de la tienda de objetos, tras un minuto vuelve a cargar el *Level* ***LV_Arena***.

***LV_GameResults***

Se muestra al finalizar la partida y sirve para visualizar el menú de resultados, con la tabla de posiciones y los botones para volver a jugar o regresar al menú principal. Según la opción elegida se vuelve a cargar el *Level* ***LV_Arena*** o ***LV_MainMenu*** respectivamente.

Cada *Level* contiene las clases, todas o algunas, que se muestran en la **Ilustración 61** del **Anexo I**. A continuación, se explican cada una de ellas.

### *5.2.2. GameMode*

Es la clase que define las reglas del juego y controla la lógica principal de este. Controla cómo funciona el juego en términos de lógica central. Determina qué tipo de controladores de jugadores se utilizan, las condiciones de victoria y derrota, gestiona el flujo de juego, etc., en otras palabras, define comportamientos esenciales del juego.

Se ha creado un *GameMode* diferente para cada *Level*.

***BP_MenuGameMode***

Crea un *Widget* que contiene la interfaz del menú y lo añade al *Viewport*, en función de si es la primera vez que se pasa por el ***LV_MainMenu***, es decir, se acaba de iniciar el juego, o se accede al menú desde la pausa, o el final de partida, el *GameMode* carga en el *Widget* el Menú Principal directamente o el Menú de “Pulsa un botón para continuar”. Además, asigna el sistema de *Inputs* al jugador 1, para que únicamente él pueda moverse por el menú.

***BP_SelectFighterGameMode***

Crea un *Widget* que contiene la interfaz de selección de personaje y la añade al *Viewport.* Comprueba en la *GameInstance* si el modo seleccionado es para 2 jugadores o para 4, y en función del número de jugadores muestra la interfaz correspondiente. Además, crea tantos *LocalPlayer* como jugadores haya, y asigna el sistema de *Inputs* a todos ellos, para poder seleccionar personaje de forma simultánea.

***BP_SelectFighterGameModeAI***

Igual que ***BP_SelectFighterGameMode,*** pero no debe consultar a la *GameInstance* para saber el número de jugadores pues el modo IA. vs IA. siempre deberá mostrar la interfaz de selección de 2 personajes. Tampoco es necesario crear *LocalPlayer* pues es suficiente con el jugador 1 para seleccionar los personajes que controlará la IA. Se asigna el sistema de *Inputs* al jugador 1 para seleccionar los personajes (primero el personaje de la IA1 y después el personaje de la IA2).

***BP_OnyxGameMode***

Comienza creando el *Widget* que contiene la interfaz de usuario (donde se muestran las barras de vida, maná, escudo, habilidades y sus tiempos de enfriamiento, etc.) y la añade al *Viewport*. A continuación, crea y posiciona los personajes en el mapa, inicializa la cámara, y crea tres *Widgets* que se añaden al *Viewport,* en el primero de ellos se carga la interfaz que muestra la cuenta atrás para iniciar el combate, en el siguiente se asigna la interfaz que aparece avisando de la aparición de una caja de dinero en algún punto del mapa, y finalmente en el último *Widget* se asigna el menú de pausa. Se hace así para que el menú de pausa quede por encima del resto de interfaces, ya que este menú tiene una capa *Blur* para distorsionar el fondo mientras es visible el menú. Además, este *GameMode* crea los *LocalPlayer* necesarios y les asigna el sistema de Inputs, asimismo, contiene toda la lógica de la partida, como las condiciones de victoria y derrota, traspaso de información a la interfaz de usuario, etc.

***BP_OnyxGameModeAI***

Igual que ***BP_OnyxGameMode,*** pero en este caso la lógica de la partida es menor pues se trata de un único combate, no es necesario hacer lógica de vida general, rondas, objetos comprados, etc. Y en cuanto a los *LocalPlayer* no es necesario crear ninguno, ni asignar sistema de *Inputs* a nadie, sin embargo, se ha dotado de sistema de *Inputs* al jugador 1 para poder pausar la partida e incluso regresar al menú principal en cualquier momento.

***BP_ShopGameMode***

Muy similar a ***BP_SelectFighterGameMode***, crea un *Widget* que contiene la interfaz de la tienda de objetos y la añade al *Viewport*. La interfaz varía en función del número de jugadores registrado en la *GameInstance*. Se crean tantos *LocalPlayer* como jugadores haya, y se asigna el sistema de *Inputs* a todos ellos, para poder moverse por la tienda de forma simultánea. Este *GameMode,* además, incluye la activación de un *Timer*, que pasado un minuto incrementa en uno el valor de la ronda y vuelve a cargar el *Level* ***LV_Arena***.

***BP_ResultGameMode***

Se encarga de crear un *Widget* con el menú de resultados y añadirlo al *Viewport*. El menú de resultados es diferente según el número de jugadores, para ello debe consultar la *GameInstance*. No es necesario crear *LocalPlayer* pues al tratarse de un menú, es controlado únicamente por el jugador 1. Se le asigna el sistema de *Inputs* al jugador 1. Este *GameMode* contiene lógica para mostrar los personajes en el escenario, a modo de podio. Resumidamente, la lógica solicita a la *GameInstance* la selección de personaje de cada jugador y también comprueba el orden de estos en el *array* de posiciones, con esta información asigna la *Mesh* y texturas del personaje seleccionado por el jugador correspondiente a la posición del podio que le pertenece.

### *5.2.3. GameState*

Es la clase que comparte información del estado actual del juego. Guarda datos del juego como el tiempo de partida, la puntuación actual, etc. Guarda una lista del estado actual de los jugadores de la partida. Es una herramienta más orientada al modo multijugador en línea, donde sea importante guardar la información referente a lo que está ocurriendo en la pantalla de cada jugador, pero no es el caso de este proyecto, por lo que se ha dejado con el *GameState* base sin aplicarle cambios, no ha sido necesario crear un hijo que herede del original.

### 5.2.4. PlayerController

Es el intermediario entre el jugador físico y el juego, se encarga de gestionar los *Inputs* y transmitírselos al personaje que posee, convirtiendo esos *Inputs* en acciones. A su vez, gestiona la interfaz para mostrarle el juego al jugador, es decir, el *PlayerController* es capaz de gestionar también cámaras y otros aspectos de la interfaz de usuario relacionados con el control del jugador.

Se ha creado un *PlayerController* diferente para cada *Level.*

***BP_MainMenuController***

Los menús controlados por un solo jugador, sin necesidad de sincronizar eventos de varios jugadores realizando *Inputs* al mismo tiempo, utilizan el sistema de navegación de botones básico de Unreal Engine, haciendo *focus* a los botones de la interfaz. Debido a esto, el ***BP_MainMenuController*** únicamente tiene la lógica para "*Any Key"* usada para registrar la pulsación de un botón cualquiera en el menú de "Pulsa un botón para continuar", y la lógica de retroceder en cualquier menú pulsando el botón de retroceso.

***BP_SelectFighterController***

Contiene toda la lógica para moverse por el menú de selección de personaje. En este caso no sirve el sistema de navegación de menús base de Unreal Engine, pues necesita recoger múltiples *Inputs* de diferentes jugadores de forma simultánea, son necesarios controles de movimiento, de selección, anulación, etc. Los *Inputs* ocasionan cambios en el menú, aplicando colores diferentes, personajes y *skins* diferentes en función del jugador que realiza el *Input*.

***BP_SelectFighterControllerAI***

Igual que ***BP_SelectFighterController,*** pero no necesita reconocer varios *Inputs* simultáneos ni tampoco necesita reconocer el jugador que los realiza, ya que el menú de selección de personaje para la IA es controlado únicamente por el jugador 1, y selecciona primero el personaje y *skin* de la IA1 y después, el personaje y *skin* de la IA2.

***BP_GameController***

Contiene solamente la lógica de activar y desactivar el menú de pausa asignada al *Input* correspondiente.

Como se puede apreciar no hay un *BP_PlayerControllerAI,* esto es debido a que la lógica de pausa está recogida directamente en los personajes junto al resto de *Inputs*, este *PlayerController* es necesario únicamente en el modo IA. vs IA. Si no fuese por el modo IA. vs IA. no sería necesario crear un *PlayerController* diferente de la clase base de Unreal Engine para el *Level* de juego.

***BP_ShopMenuController***

Similar a ***BP_SelectFighterController***, debe recoger los eventos de *Input* de todos los jugadores a la vez, y debe asignar colores a los botones en función del jugador que esté encima de dicho botón. Recoge lógica de movimiento, selección, anulación, cambio de modos (compra o venta), etc., además cada *Input* contiene la lógica correspondiente a su acción, si se compra un objeto debe guardarse que objeto es y sus características, y debe mostrarse en la interfaz.

***BP_GameResultController***

Realmente no es necesario pues no tiene ninguna lógica desarrollada, todos los botones del menú de selección pueden pulsarse mediante el sistema de navegación de menús base de Unreal Engine, y no es necesario tampoco un *Input* de retroceso, pues es un único menú que carga el *Level* ***LV_Arena*** o el ***LV_MainMenu*** según el botón seleccionado.

### *5.2.5. AI Controller*

Conecta una inteligencia artificial con el personaje que la posee. Se encarga de tomar decisiones y ejecutar acciones basadas en *scripts* y algoritmos de inteligencia artificial.

*AI Controller* utiliza herramientas como el *NavMesh* para la navegación e incorpora comportamientos complejos mediante árboles de comportamiento, Environment Query System (EQS) y otros sistemas de IA.

El proyecto consta de un *AI Controller* llamado ***BP_AIController***.

### *5.2.6. Actor / Pawn / Character*

Un *Actor* en Unreal Engine es cualquier objeto que pueda ser colocado o utilizado en un *Level*. Es la clase base para todos los objetos que puedan existir en el mundo del juego, incluyendo luces, cámaras, personajes, y elementos interactivos. Los actores pueden tener componentes como mallas o colisiones, y pueden ejecutar lógica a través de scripts.

*Pawn* es un tipo de *Actor* que representa una entidad que puede ser poseída y controlada por un jugador o inteligencia artificial. Los *Pawns* pueden moverse y realizar acciones en el mundo del juego. Son la base para cualquier tipo de personaje o vehículo en el juego.

*Character* es una subclase de *Pawn* que incluye funcionalidad específica para personajes humanoides o similares. Incluye un sistema de movimiento más avanzado y animaciones predeterminadas, lo que facilita la creación de personajes jugables y *NPCs* con movimientos complejos, como caminar, correr, saltar, atacar, etc.

Para el desarrollo del proyecto se han creado los siguientes actores directamente en *C++*, hijos de sus correspondientes clases base de Unreal Engine:

***OnyxCharacter***

Hijo de la clase *Character*, implementa la interfaz del Gameplay Ability System (GAS), contiene las clases ***Attribute Set***, y ***Ability System Component***, tiene la lista de habilidades y efectos de los personajes, además realiza toda la lógica de los eventos que se lanzan cuando el personaje recibe un cambio de vida, maná, escudo, daño, etc. Se encarga de cargar los efectos iniciales y los atributos de los personajes.

***OnyxFighter***

Hijo de ***OnyxCharacter***, tiene un *GiveAbilities()* en el *Begin Play* que otorga las habilidades a los personajes. Inicialmente estaba pensado para realizar aquí toda la lógica exclusiva de los personajes jugables, y el resto de los personajes que tuviesen características de Gameplay Ability System (GAS) como vida, escudo, etc., pero que no

fuesen jugables, como podrían ser los enemigos neutrales, heredasen directamente de ***OnyxCharacter*** dejando ***OnyxCharacter*** únicamente con la lógica básica de Gameplay Ability System (GAS), pero como finalmente solo se han desarrollado los personajes jugables, la lógica se ha programado directamente en ***OnyxCharacter*** y ***OnyxFighter*** ha quedo prácticamente en desuso.

***OnyxActor***

Es una clase hija de *Actor*, se ha creado para ser la clase padre de cualquier cosa que sea un actor y que necesite implementar Gameplay Ability System (GAS). Implementar la interfaz de Gameplay Ability System (GAS), tiene efectos iniciales, un *Attribute Set* y un *Gameplay Ability Component*.

Por ejemplo, para objetos destruibles que requieran de una barra de vida y la capacidad de recibir daño. No tiene lógica de personaje, no tiene movimiento, pero si el resto de las características.

A continuación, se han creado clases hijas de las anteriormente mencionadas para cosas aún más específicas. (Las que comienzan con BP_ son *Blueprints* a nivel del editor, mientras que las que su nombre no empieza con BP_ son clases de *C++*).

***BP_Fighter***

Hereda de ***OnyxFighter***, contiene toda la lógica principal de los personajes jugables, Inputs de movimiento y habilidades, cajas de colisión, *Decals* de los rangos de las habilidades, etc. Es a su vez el padre de los *Blueprints* más concretos de cada personaje, ***BP_Countess***, ***BP_Grim***, ***BP_Grux***, ***BP_Iggy***, etc.

***BP_Crate***

Hereda de ***OnyxActor***, contiene la lógica de los eventos de cambio de vida de las cajas de dinero y de daño recibido, para mostrar la vida restante y el daño en la interfaz, y el

evento que lanza una vez destruida, para iniciar la animación de partículas y otorgar un efecto que incrementa el dinero del jugador que la destruye.

***MainCamera***

Hereda de la clase *Actor* base de Unreal Engine, y contiene la lógica para la cámara principal del juego. Se trata de una cámara que pondera la media de las posiciones de los ***BP_Fighters*** dentro del *Level* de juego, y se posiciona en las coordenadas obtenidas, a continuación, se ajusta a la longitud del *SpringArm*, que aumenta o disminuye en función de cuán separados estén los personajes dentro del escenario de juego, manteniendo un intervalo máximo y mínimo que puede acercarse y alejarse la cámara para mantener en el encuadre a ambos personajes en todo momento.

### *5.2.7. GameInstance*

Es una clase que persiste durante la vida útil de una sesión de juego, incluso al cambiar entre *Levels*. Es por ello por lo que en la *GameInstance* se guardan todos aquellos valores que no se quieren perder al cambiar de *Level*. Actúa como un contenedor global para el estado del juego y almacena información que necesita ser accesible en todo momento, como datos de los jugadores, configuraciones del juego y procesos de guardado. A diferencia de otros objetos de juego que pueden ser destruidos y recreados al cargar nuevos *Levels*, la *GameInstance* permanece activa, facilitando la gestión de datos globales.

En la *GameInstance* se han creado variables para almacenar información importante como el número de jugadores, el número de equipos, los colores de cada jugador, el número de ronda, la vida general de los jugadores, los objetos que han comprado y que por tanto deben asignársele al comienzo de la siguiente ronda, su cantidad de onyx, etc.

También se han creado variables para recordar el tipo botones según el tipo de mando y la plataforma, que ha seleccionado el jugador que quiere ver en los menús. Y se recoge lista de *TextDecorators* [16] con los botones como puede verse en el atlas del **Anexo II**.

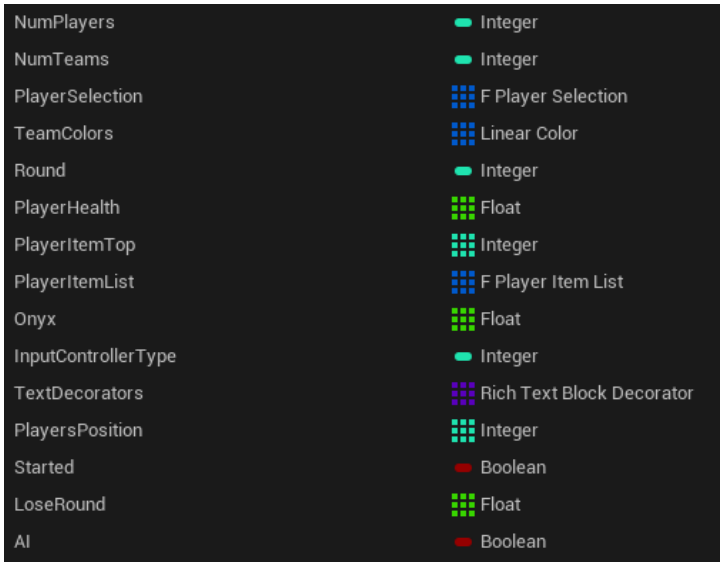



*Ilustración 24 – GameInstance con sus variables.*

La *GameInstance* también contiene eventos útiles: un evento para inicializar un sonido 2D, en el cual se carga la música de fondo al inicio del juego, y se mantiene activa entre *Levels*. Otro evento para recoger la selección de personaje y *skin* que ha realizado cada jugador, y almacenarlo en un *array* con la información de todos los jugadores. Y finalmente un evento para reiniciar todas las variables a sus valores iniciales, este evento se llama cada vez que se regresa al menú principal desde algún punto del videojuego.

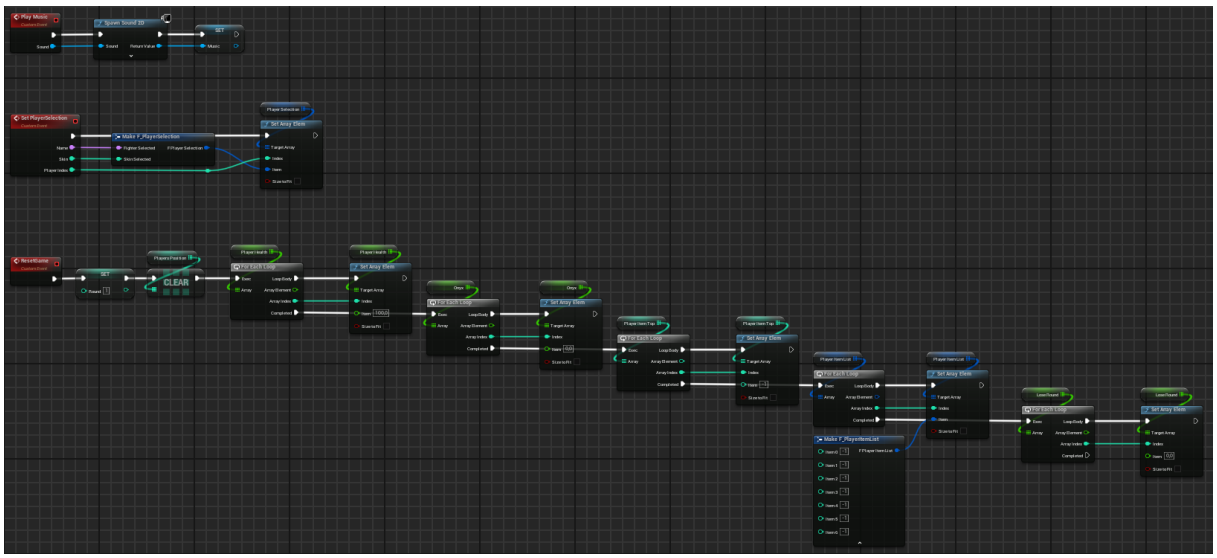


*Ilustración 25 – GameInstance con sus eventos.*

## 5.3. Gameplay Ability System base

La **Ilustración 62** del **Anexo I**, explica la arquitectura base del *plug-in* Gameplay Ability System (GAS), para adaptar el *plug-in* al proyecto y extender su funcionamiento, se crean hijos heredados de estas clases [17].

### *5.3.1. Ability System Component*

Es una clase que permite la implementación y gestión de habilidades y efectos en los personajes de un juego. Facilita la creación de habilidades complejas y maneja las estadísticas y atributos de los personajes proporcionando una estructura flexible y escalable para el desarrollo de mecánicas de juego avanzadas.

Se ha utilizado la clase base de Gameplay Ability System (GAS) pues no era necesario ampliar su lógica [18].

### *5.3.2. Attribute Set*

Clase utilizada para definir y gestionar los atributos de los personajes. Los atributos definidos en un *Attribute Set* pueden ser modificados por habilidades, efectos y otros mecanismos del juego, proporcionando una forma estructurada de controlar y actualizar las características del personaje durante el juego.

Se ha creado una clase hija de esta llamada ***OnyxAttributeSet***, que contiene todos los atributos de los personajes, la lista de atributos se encuentra en la **Ilustración 22**, además de los métodos *PreAttributeChange*, que recoge los cambios de valor en los atributos y los asigna, y *PostGameplayEffectExecute,* que recoge la lógica cuando se aplica un efecto que altera los valores de los atributos.

***PreAttributeChange***

Controla que los efectos de regeneración de vida y maná no superen los valores de vida y maná máximos, y para que reducción de escudo no permita valores por debajo de 0. El

control lo realiza comparando el nuevo valor con el máximo posible de dicho atributo y asignando uno u otro según corresponda.

***PostGameplayEffectExecute***

Esta lógica, necesita conocer el *Actor* objetivo que recibe el efecto y los atributos del *Actor* que inflige el efecto. Controla el daño infligido teniendo en cuenta el resto de los atributos defensivos, se tiene en cuenta la cantidad de escudo, para agotar este antes de comenzar a restar vida, se tiene en cuenta también la cantidad de armadura para contrarrestar el daño total de los efectos con poder físico, y la cantidad de resistencia mágica para reducir la cantidad de daño que aplica un efecto de poder mágico. Además, en caso de tener algún porcentaje de probabilidad de crítico, se lanza un *Random* en función del porcentaje, que indicará si el efecto aplica crítico o no, y en caso afirmativo, modifica el valor del daño multiplicándolo por el índice de potenciador de crítico que es de 175%. Tras aplicar todos los incrementos y reducciones necesarios, se cambia el valor de la vida, según el daño total recibido.

Además, se transfiere el daño total recibido a la interfaz mediante un evento a nivel de *Blueprint* que se llama al aplicar un daño, y tiene como valores de entrada el número de daño total infligido y el tipo de daño (1 – Daño a escudos, 2 – Daño físico, 3 – Daño mágico, y 4 – Daño crítico).

### *5.3.3. Gameplay Ability*

Clase que define una acción o conjunto de acciones que un personaje puede ejecutar en el juego. Estas habilidades pueden incluir ataques, hechizos, movimientos especiales, o cualquier otra acción que afecte el estado del juego. Permite a los desarrolladores especificar el comportamiento de la habilidad, como se activa, qué efectos tiene, y cómo interactúa con otros sistemas del juego.

Se ha creado una clase hija de esta, llamada ***OnyxGameplayAbility***, que contiene la lógica para comprobar los *cooldowns* de las habilidades, sus *tags* y el coste de maná,

asignarlo a una variable y aplicarlo al efecto. La *Tag* se utiliza para indicar la habilidad que está en *cooldown.*

***Optimizaciones***

Para el atributo de “Coste de Maná” de las habilidades, se ha creado un efecto común que aplica el coste a través de una *CustomCalculationClass* que toma el valor del coste de maná directamente de una variable de la propia habilidad, que puede cambiarse para cada una de ellas. Esto ahorra tener que crear un efecto de “Coste de Maná” diferente para cada habilidad, pues las habilidades vienen directamente con una casilla en el editor, donde solicita el coste como valor.

### *5.3.4. Gameplay Task*

Se trata de una unidad de trabajo o acción que puede ser ejecutada como parte del Gameplay Ability System (GAS). Estas tareas son fragmentos de comportamiento que se pueden ejecutar en paralelo o secuencialmente para realizar acciones específicas dentro de una habilidad. Permiten descomponer las habilidades complejas en componentes más pequeños y manejables, facilitando el desarrollo y la organización del comportamiento del juego.

Se han utilizado las *Tasks* base de Gameplay Ability System (GAS) como *AbilityTask_PlayMontageAndWait* que reproduce una animación y realizar lógica durante su ejecución o al finalizar; *AbilityTask_WaitGameplayEvent* que espera un evento para realizar una acción.

### *5.3.5. Gameplay Ability System en Actors*

***OnyxCharacter***

Incluye al *Character* el *AttributeSet,* el *AbilitySystemComponent* y el *GiveAbilities*, junto con la lógica del movimiento, colisiones, etc., propia de los *Characters*.

Se preparan variables para ser accesibles a nivel de editor, donde se puedan asignar los atributos iniciales de cada personaje con los correspondientes valores, una lista de efectos

iniciales (como la regeneración de vida constante, la regeneración de maná, etc.), y otra lista donde asignar las habilidades del personaje. Otra variable donde asignar la animación que debe reproducirse al morir.

Internamente la clase aplica un *ID* al *Character* y un *Tag* de equipo al que pertenece, lo cual permite realizar lógica para evitar que se pueda aplicar daño a los compañeros de equipo en el modo 2vs2.

También contiene métodos para gestionar la lógica después de morir, que se lanza cuando la vida llega a 0. Aquí se encarga de asignar el *Tag* "*Dead*" al personaje y evitar que pueda recibir efectos del resto de personajes que siguen jugando, reproduce la animación de muerte asignada en la variable, y al finalizar la animación desaparece del escenario el personaje.

Además, se generan los siguientes eventos para ser gestionados a nivel de *Blueprint*, útiles para transferir los valores asociados a estos a la interfaz.

Un evento para cuando cambia el valor de la vida, otro para cuando cambia el valor de maná, otro para cuando cambia el valor de escudo, y finalmente, otro para cuando cambia el valor de onyx, dinero del personaje. Estos eventos tienen como atributos de entrada el *ID* del *Character* que debe reflejar ese cambio, y el nuevo porcentaje o valor que debe mostrar en interfaz.

***OnyxActor***

Similar a ***OnyxCharacter,*** pero destinado a cualquier *Actor* que necesite implementar Gameplay Ability System (GAS) pero sin las capacidades de movimiento de un *Character*, es decir, para objetos estáticos del entorno que requieran de vida u otros atributos. Incluye al *Actor* el *AttributeSet* y el *AbilitySystemComponent*, y carga los atributos iniciales.

Se preparan variables para ser accesibles a nivel de editor, donde se puedan asignar los atributos iniciales con los correspondientes valores y una lista de efectos iniciales (como la regeneración de vida constante, la regeneración de maná, etc.).

Además, se generan los siguientes eventos para ser gestionados a nivel de *Blueprint*, útiles para transferir los valores asociados a estos a la interfaz, y para lanzar lógica en los *Blueprints* mediante *Nodes*.

Un evento para cuando cambia el valor de la vida, que tiene como atributo de entrada el nuevo porcentaje que debe mostrar la interfaz.

Otro evento para cuando muere, la vida llega a 0, que como atributo de entrada tiene un *Actor*, concretamente el actor que lo ha matado, es decir, el último *Actor* que ha golpeado el objeto que incorpora esta clase.

Finalmente, un evento que transfiere el daño recibido a la interfaz, este se llama cada vez que detectar un daño, y tiene como valores de entrada el número de daño recibido y el tipo de daño (1 – Daño a escudos, 2 – Daño físico, 3 – Daño mágico, y 4 – Daño crítico).

## 5.4. Personajes

Los personajes han sido obtenidos de la *Marketplace* de Unreal Engine, concretamente del paquete de personajes de *Paragon (Epic Games, 2016)* que tienen licencia de uso gratuito en proyectos de Unreal Engine, pues ambos son propiedad de Epic Games.

Aunque los personajes y sus animaciones no han sido diseñados, no venían plenamente montados y listos para usarse, ha sido necesario rehacer todo el sistema de *Inputs*, en nuestro propio *Character* ***BP_Fighter***, y las habilidades no venían montadas, además, la mayoría de las habilidades han sido reinventadas por nosotros, no son las mismas que tienen los personajes en el juego original de *Paragon*. Las animaciones y partículas tampoco vienen montadas, se dispone de todo el material necesario, pero hay que montar la animación uniendo fragmentos de animaciones de posiciones diferentes del personaje, en la línea de tiempo del sistema de animaciones de Unreal Engine. Y en esa misma línea de tiempo hay que colocar y ajustar la posición de los emisores de partículas, que una vez más, para una sola animación es necesario unir diferentes sistemas de partículas. En la siguiente ilustración puede verse la línea de tiempo necesaria para la animación de la habilidad 3 de

Grim, donde es necesario unir la animación de "*Ability_R_Start*", "*Ability_R_Fire*", y "*Ability_R_End*", y a continuación, añadir 7 sistemas de partículas, 3 archivos de voz, y 3 efectos de sonido.

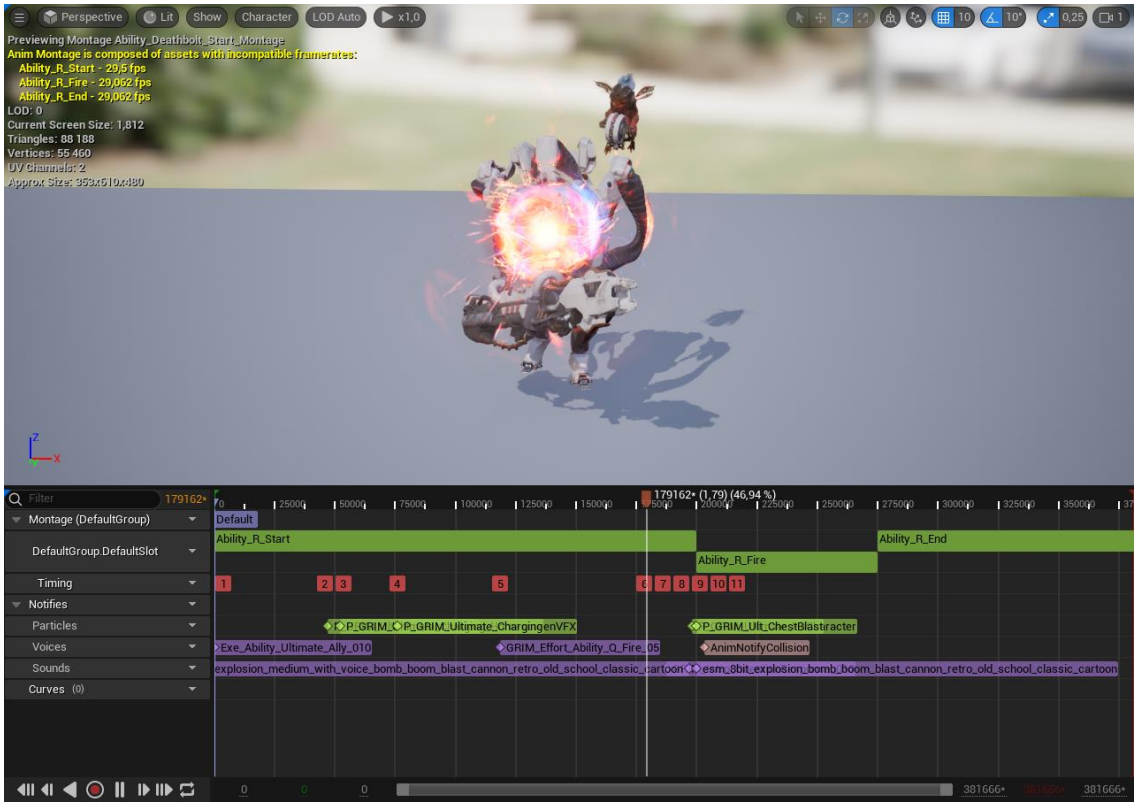



*Ilustración 26 – Ejemplo línea de tiempo de un AnimMontage de un personaje.*

***BP_Fighter*** al ser la clase padre de todos los *Characters* contiene la lógica principal de los personajes, además de todos los componentes necesarios para estos, pero vacíos.

Como lógica, tiene la programación del movimiento y el sistema de apuntado, los eventos que detectan un cambio de valor en los atributos del Gameplay Ability System, para transferir la información a la interfaz, los *Inputs* de pausa, movimiento, apuntado y lanzamiento de habilidades.

*Ilustración 27 – Inputs para el lanzamiento de habilidades.*

En la ilustración anterior se ve la lógica de los inputs de lanzamiento de habilidades, esta consiste en activar la *Decal* que indica el rango de la habilidad, mientras se mantiene pulsado el botón. Cuando el botón se suelta, si la habilidad no ha sido cancelada, se lanza la lógica de la habilidad y se desactiva la *Decal*.

Esta misma lógica se usa para las cuatro habilidades de movimiento.

Finalmente hay otro *Input* para cancelar el lanzamiento de habilidades, este cambia el valor del *booleano* que comprueba si una habilidad es cancelada o no, para que no se realice la lógica de la habilidad pulsada cuando se suelte el botón, además desactiva las *Decals* activas.

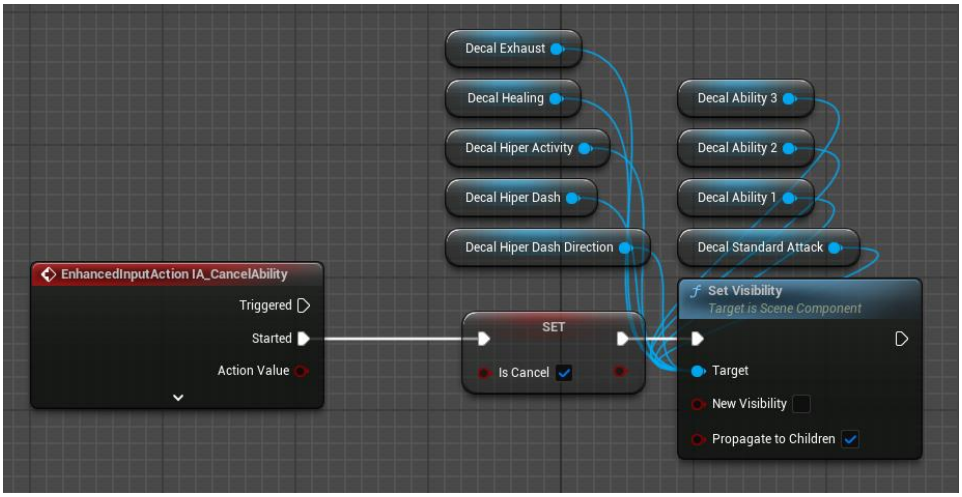



*Ilustración 28 – Input para cancelar el lanzamiento de las habilidades.*

Como componentes, tiene las *Decals* de cada habilidad, y los indicadores de interfaz, además, tiene atributos de editor para asignar los tipos de habilidad y su rango, y los componentes del Gameplay Ability System (GAS), como atributos iniciales, efectos iniciales, etc [19]. Para asignar el tipo de habilidad y su rango, se ha creado un *Enumerator* con el tipo de las habilidades, diferenciando entre habilidades de daño, habilidades de escudo, y habilidades de *Buff*. Y una estructura de datos con el *Enumerator* y una variable número para el rango.

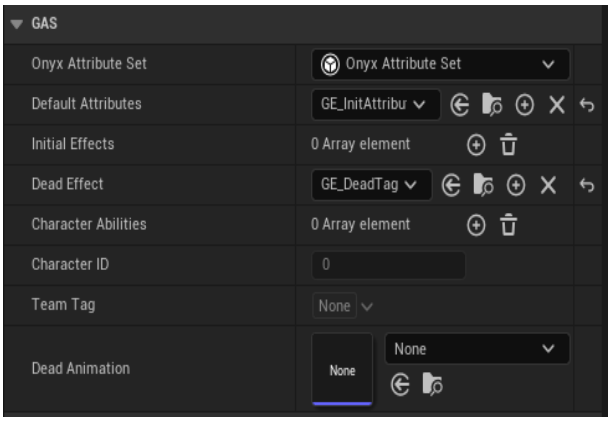



*Ilustración 29 – Atributos Gameplay Ability System.*

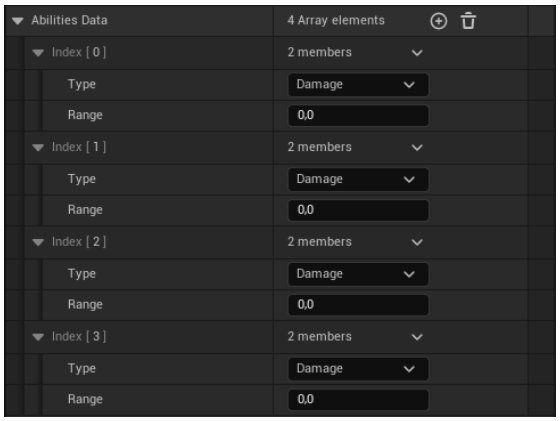



*Ilustración 30 – Tipos de habilidad y su rango.*

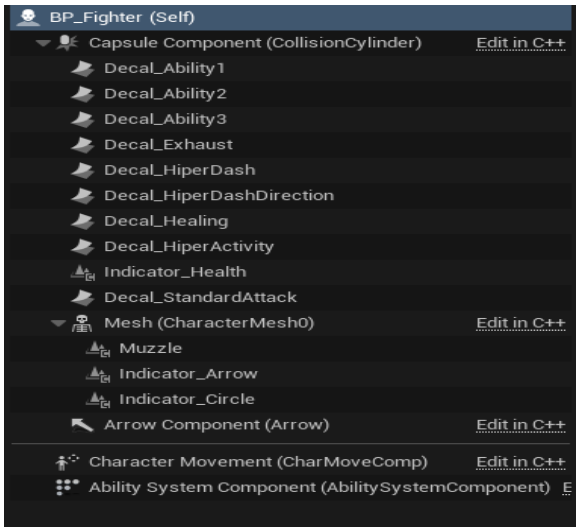



*Ilustración 31 – Componentes del BP_Fighter.*

Los personajes que tienen una habilidad de *Buff* que afecta a los rangos de sus habilidades, tiene un componente *Decal* para cada rango, y en su *Blueprint* tiene un evento

que es llamado desde la habilidad, para cambiar la *Decal* visible, al terminar la habilidad se vuelve a cambiar la *Decal*.

Por cada personaje se ha creado un nuevo *Character* hijo de ***BP_Fighter***, y se le ha asignado a cada componente los valores correspondientes (*Decals*, *AnimBlueprint*, etc).

Cada uno tiene un *set* de atributos único, con valores diferentes de vida, maná, resistencias, velocidad de movimiento, daño, etc. Cada personaje cuenta con una habilidad de ataque básico, sin *Cooldown* y sin coste de maná, y tres habilidades que pueden ser de cualquiera de los tres tipos que contiene el *Enumerator* previamente mencionado.

Estas habilidades son Gameplay Abilities que deben asignarse en el *Character* en la lista de habilidades.

Además, es necesario crear *Sphere Collision* o *Box Collision,* según convenga, para cada habilidad y añadirlas como componente a los *Characters*. Para comprobar las colisiones se crea un evento dentro de cada personaje con un *switch* de habilidades, según la habilidad utilizada, se llama al evento con la correspondiente *Collision* como atributo de entrada, para realizar la lógica de comprobar los *Actors* con los que colisiona la *Sphere* o *Box*, y de todos los *Actors* elimina aquellos que no implementan Gameplay Ability System (GAS), y a sí mismo, y lanza un *Gameplay Event* de daño.

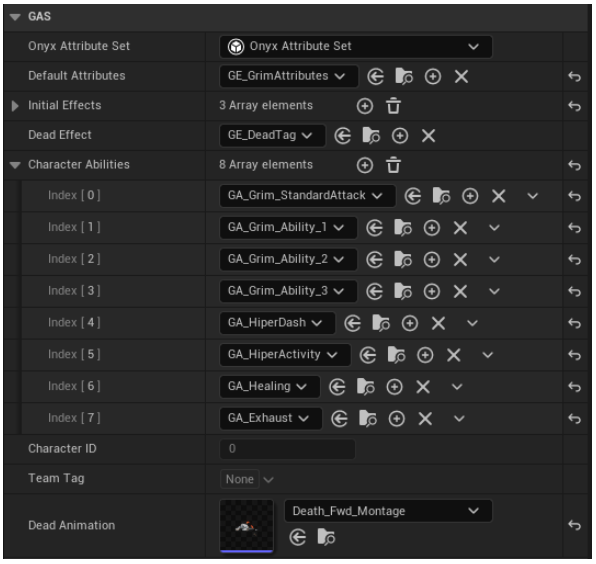



*Ilustración 32 – Ejemplo atributos (GAS)*

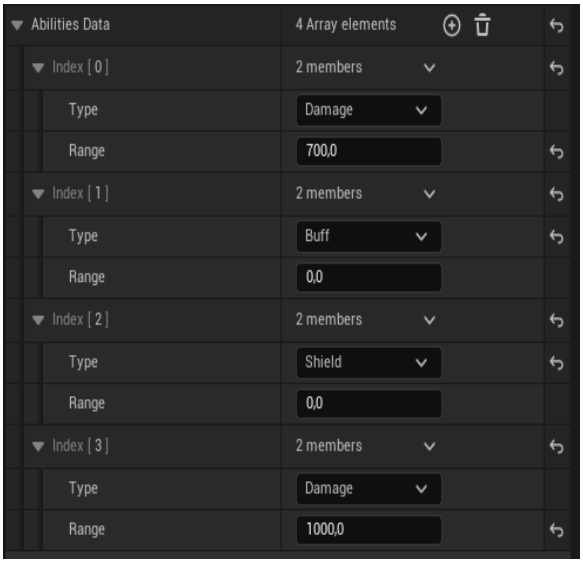



*Ilustración 33 – Ejemplo tipos y rango.*

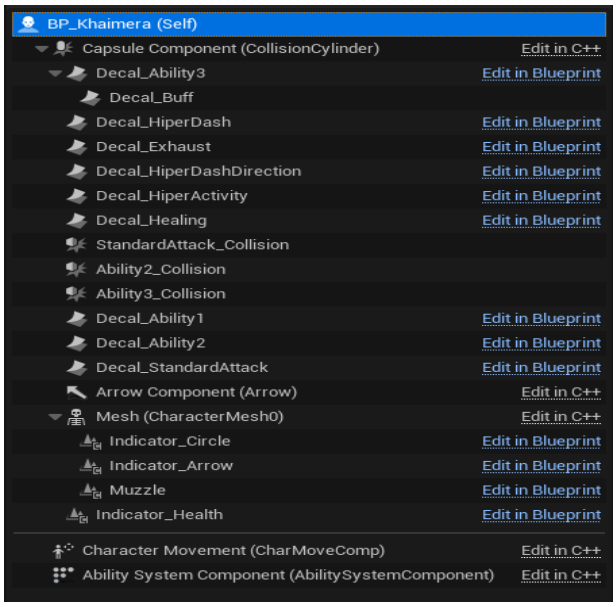



*Ilustración 34 – Ejemplo componentes de un Character (Con Decal Buff y Box Collisions).*

El *Gameplay Event* de daño, es un componente que tienen todas las *Gameplay Ability* de las habilidades. En concreto, cada habilidad tiene un *Gameplay Ability* donde se realiza su lógica, estos comienzan reproduciendo la animación correspondiente, y cuando la animación termina se finaliza la habilidad. Mientras la animación se está reproduciendo, se espera recibir el *Gameplay Event* de daño antes mencionado, obteniendo quien es el propietario de la habilidad para poder consultar sus atributos en el *Ability System Component*, y quienes

son la lista de objetivos a los cuales se deben aplicar los efectos (aquí comprueba los *Tags* de equipo de los objetivos, y si alguno tiene el mismo *Tag* que él, le elimina de la lista de objetivos, para evitar dañar compañeros en el modo 2vs2), a continuación se realiza la lógica necesaria para los efectos, por ejemplo, incrementar según el valor del daño físico del personaje que tira la habilidad, o comprobar si es crítico o no, y aplicar el bonus de daño por crítico, etc., y finalmente se aplica el efecto al objetivo u objetivos.

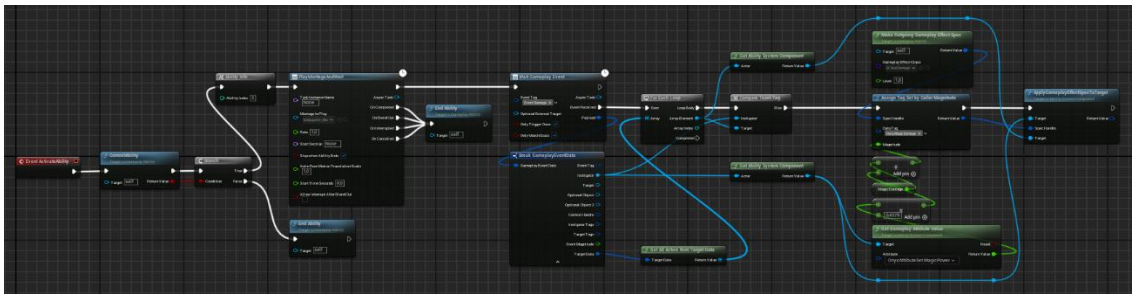


*Ilustración 35 – Ejemplo del Gameplay Ability de una habilidad.*

En el ejemplo de la ilustración se trata de una habilidad que aplica un efecto de daño, calculado en función de un daño base más un 43,75% del valor del atributo de daño del personaje.

### *5.4.1. Roles*

La tabla de estadísticas de cada personaje, con el ajuste de los daños de cada habilidad puede verse en **Anexo III**.

A continuación, se realiza un desglose de los personajes según su rol.

***Asesinos***

Personajes con poca vida, pero mucha velocidad de movimiento y gran cantidad de daño instantáneo. Se ha creado el *Character* ***BP_Countess***.

***Luchadores***

Personajes con un buen equilibrio entre vida, velocidad de movimiento y daño. Se ha creado el *Character* ***BP_Khaimera.***

***Tanques***

Personajes con mucha vida, que además suelen tener habilidades de gestión de escudo, pero su velocidad de movimiento no suele ser muy alta y el daño es equilibrado. Se han creado los *Characters* ***BP_Grim***, ***BP_Grux***, y ***BP_Terra***.

***Magos***

Personajes con daño mágico, equilibrados, pueden ser a distancia o melés, similares a los luchadores, pero con poder mágico. Y suelen destacar más sus habilidades que sus ataques básicos. Se han creado los *Characters* ***BP_Phase***, y ***BP_Sevarog***.

***Tiradores***

Personajes con ataques a distancia, pueden tener daño físico o mixto, y destacan principalmente por su ataque básico que pueden utilizar constantemente desde lejos aplicando mucho daño. Suelen tener buena velocidad de movimiento para huir de los personajes melés, pero a cambio, tienen poca vida. Se han creado los *Characters* ***BP_Iggy***, y ***BP_Sparrow***.

### *5.4.2. BP_Projectile*

Para los personajes de rango se ha creado un *Actor script* que genera un proyectil en la posición del jugador y se propaga a una velocidad indicada, en la dirección del vector *Forward* del personaje, en un rango indicado. Cuando recorre esa distancia se destruye, pero si colisiona con algún *Actor* del tipo ***BP_Fighter*** durante el recorrido, se destruye automáticamente y aplica un efecto de daño, con los atributos del personaje que ha lanzado el proyectil.

### *5.4.3. BP_Crate*

Se ha creado un *Actor* hijo de ***OnyxActor***, que por tanto cuenta con Gameplay Ability System (GAS). Se ha incluido en el *Actor* la *Mesh* de una caja, obtenida del *Quixel Bridge*[9]

[9] https://quixel.com/bridge

de Unreal Engine, se le ha aplicado una *Box Collision* y un ***Widget* indicador de estadísticas**.

Se le ha asignado unos atributos iniciales al ***Attribute Set***, con los valores de vida y vida máxima; y un efecto inicial de regeneración de vida, a la lista de efectos iniciales.

Se ha programado la lógica de los eventos del ***OnyxActor*** para cuando recibe daño, cuando detecta un cambio de vida y para cuando muere.

El cambio de vida, lo notifica al ***Widget* indicador de estadísticas** para mostrarlo por interfaz. El evento al recibir daño activa el ***Overlay* de daño** y genera un ***Widget* indicador de daño**. Y el evento de muerte, aplica un efecto que incrementa el valor del atributo *Onyx* del personaje que ha golpeado por última vez la caja, y activa un emisor de partículas y un sonido 2D para retroalimentar visualmente la destrucción de la caja [20].

***BP_CrateSpawn***

Se ha creado un *Actor* de tipo *script*, que recoge la lógica para que aparezcan cajas en el mapa. Se han asignado por el escenario varios *Target Point* en diferentes localizaciones, y desde el *script* se selecciona aleatoriamente uno de ellos para generar una caja en esa localización. Aparece una caja nada más comenzar el combate y se intenta generar una nueva cada 30 segundos, si aún sigue habiendo una caja en el escenario no se generan más, si la anterior caja ha sido destruida cuando el *Timer* llama al evento, se genera una nueva caja en una nueva localización aleatoria del conjunto de *Target Points*.

Para comprobar si sigue habiendo cajas en el mapa, se hace un *Get All Actors of Class BP_Crate* y si el *Length* es mayor que cero, no se realiza la lógica.

Cuando aparece una nueva caja, se activa la interfaz que avisa de una nueva caja generándose, y pasados 5 segundos se vuelve a desactivar.

## 5.5. Lógica de Partida

Se procede a explicar la lógica dentro del ***BP_OnyxGameMode*** que contiene todos los eventos que ocurren durante la partida. En concreto, se explicará la gestión del sistema de rondas y el otorgamiento de bonificaciones.

### *5.5.1. Sistema de Rondas*

Al iniciar el combate, se solicita a la *GameInstance* el número de la ronda, se asigna al texto que indica la ronda en la interfaz, y a continuación, se inicia un *Timer* con una duración de 5 minutos [21], este determina el tiempo que dura la ronda como máximo. Cuando el tiempo se acaba, se comprueba la vida de cada jugador, y se almacenan en un *array* ordenado según el *ID* de jugador. Después, se añaden al *array* de orden de los jugadores, sobre el que se aplica la lógica de sustracción de vida general, aumento de onyx, se aplican las bonificaciones, etc. Para añadirlos a este *array* se añade en orden de perdedor a ganador, el primero en añadirse es el *Index* del que menor valor tiene en su celda, es decir, el primer perdedor es el que menos vida tiene cuando acaba el tiempo, y así sucesivamente hasta rellenar el *array* de posiciones con tantas celdas como jugadores haya en la partida.

En el caso de los combates 2vs2, la vida que se comprueba para determinar el ganador es la del equipo, es decir, suma la vida de los dos jugadores, pero después, el *array* de posiciones de los jugadores se hace individualmente para cada jugador.

*Aquí surge un problema, y es que el juego contempla la posibilidad de un primero, un segundo, un tercero y un cuarto, y así se reparten las bonificaciones, sin embargo, en combates por equipos, puede darse el caso de que luchen 1vs1, mientras sus respectivos compañeros simplemente deambulen por el mapa sin recibir daño, al finalizar la ronda por tiempo, ambos jugadores tendrían la vida al máximo y por tanto a ambos le corresponde la bonificación de ser primeros, esto está contemplado y solucionado en la lógica*.

A continuación, se llama al evento de terminar ronda.

Otra forma posible de que acabe la partida es cuando todos los jugadores, menos uno, han muerto, por tanto, cada vez que un jugador lanza su evento de muerte, se le añade al *array* de posicionamiento de los jugadores, cuando el *array* tiene tamaño igual al número de jugadores menos uno, se entiende que solo queda un jugador en la partida, se le añade como ganador, y se lanza igualmente el evento de terminar ronda. En los combates por equipos se lanza el evento de terminar ronda cuando los dos jugadores en un equipo han muerto, independientemente de que sigan con vida uno o dos jugadores del equipo contrario.

El evento de terminar ronda reparte el dinero a los jugadores y resta parte de su vida general en función de su posición en el *array*. Mientras la vida general de al menos dos jugadores sea distinta de cero, se procede a abrir el menú de tienda de objetos. Si solamente un jugador tiene vida general mayor que cero, se abre el menú de resultados de la partida.

### *5.5.2. Añadir dinero y Restar vida general (Bonificaciones)*

En cuanto a las bonificaciones, se aplican a los valores base de los eventos de sustracción de vida general y del reparto de onyx.

La sustracción de vida general va aumentando tras varias rondas, para facilitar las remontadas por parte del jugador que va perdiendo. De ronda 1 a ronda 3 resta 10, de ronda 4 a ronda 5 resta 20, y de ronda 6 en adelante resta 30. Además, en el modo de 4 jugadores, solo se le resta el valor base al primero en morir, el segundo y tercero reciben solo el 50% del valor como sustracción de vida.

En cuanto al reparto de onyx, de base añade 2000 al ganador, 1500 al perdedor, y 1750 al segundo y tercero. Sin embargo, existe una bonificación para el ganador que le aumentar el valor base en un porcentaje correspondiente al porcentaje de vida con el que termina la partida, obteniendo 3000 onyx si gana con el 50% de vida, por ejemplo.

También hay una bonificación destinada a los perdedores recurrentes. Al perder se suma 1 al contador de rondas perdidas (al quedar segundo o tercero se suma 0,5 al contador de

rondas perdidas). Cuando el contador es igual a 3 se otorga al jugador un bonus de dinero y se reinicia el contador a 0 (también se reinicia a 0 cuando se gana una ronda). El bonus es de 6000 onyx para el jugador que obtiene la bonificación habiendo perdido, y de 4500 para los jugadores que obtienen la bonificación quedando segundos o terceros en 6 combates seguidos (6 porque suma solamente 0,5 al contador estas semiderrotas).

El nuevo valor de vida general se asigna a la variable que recoge esta información en la *GameInstance*.

Los onyx se añaden al jugador, incrementando el valor del atributo mediante un efecto.

## 5.6. Diseño de Interfaces

En **Anexo I** puede verse el diagrama de flujo del videojuego, con sus correspondientes interfaces, para poder ir cargando cada interfaz sobre el *Widget* asignado al *Viewport* de los diferentes *GameModes* se ha creado un *Widget* con un *CommonActivatableWidget* como único componente y los eventos *PushMenuOverlay, RemoveMenuOverlay* y *ClearMenuOverlay*. Cuando se crea un *Widget* en el *GameMode* es realmente un *Widget* de este tipo, y mediante el método *PushMenuOverlay* se asigna al *CommonActivatableWidget* el *Widget* de interfaz deseado. Gracias a esto, cuando se quiere pasar de una interfaz a otra, por ejemplo, del menú principal, al menú de selección de modo de juego, solo es necesario hacer un *RemoveMenuOverlay* y asignar el nuevo menú al pulsar el botón correspondiente [22].

Se ha creado un *Widget* para el diseño de los botones, y poder añadirlos directamente en los menús. Para ello se ha utilizado el material de la **Ilustración 65** del **Anexo II**. Se ha creado una animación para cuando se posiciona sobre el botón y se han creado eventos *Hover* y *Unhover* que activan esta animación al derecho y al revés. También se ha asignado un método que permite introducir un valor de texto desde el editor y se lo asigna al botón.

Se ha creado otro botón de forma cuadrada, con la misma lógica que el anterior, pero en lugar de texto, permite asignar una imagen, este se utiliza en el menú de selección de

personaje [23]. Además, añade 4 marcos ocultos, que se activan cuando los jugadores se posicionan sobre ellos. Hay 4 marcos porque los 4 jugadores pueden situarse sobre el mismo botón. Cuando se hace el *Hover* y *Unhover*, se hace también el *ActiveColor* y *DesactiveColor* que asigna el color del jugador al marco del botón [24].

### 5.6.1. Controles

Para el menú de controles se han diseñado imágenes explicativas que pueden verse en **Anexo I**, según la plataforma seleccionada, se mostrarán los botones de los menús según corresponda con dicho mando. Para esto se han creado los textos de las interfaces en formato *RichText* al cual se le pueden asignar *Decorator Class* con una *Data Table* del tipo *RichImageRow*, con todos los iconos.

Cuando un texto es del tipo *RichText* y tiene asociada la *Decorator Class*, puede hacerse referencia a los *ID* de las imágenes de la *Data Table* para decorar los textos. Solo es necesario incluir en el texto ***<img id="idFoto"/>*** donde *idFoto* es el nombre de la fila en la *Data Table* [16].

Para poder cambiar entre plataformas, se han creado tantos *Decorartor Class* como plataformas hay, y a cada *Decorator* se le ha asignado una *Data Table* con los iconos de los botones correspondientes. Los textos tienen un *idFoto* concreto, como "*select*", "*decline*", etc. Las *Data Table* de cada plataforma se han hecho asignando los mismos *IDs* a los mismos botones, y cuando la plataforma cambie simplemente se cambia el *Decorator* asignado al texto y los iconos se ajustan al mismo *ID.*

### 5.6.2. Interfaz In-Game

Se ha diseñado en *Photoshop* todos los *Assets* necesarios para la interfaz, estos *Assets* pueden verse en **Anexo II**. El prototipo de la interfaz puede verse en **Anexo I**.

Ya en Unreal Engine se ha creado un *Widget* para la interfaz y dentro de él, se han creado componentes de interfaz del tipo imagen para los iconos de personaje, los iconos de habilidades y de los objetos. Una variable texto asociada a un *Timer* que actualiza el valor

del texto que sirve como temporizador de la partida. Barras de progreso que indican la vida, maná y escudo de los personajes y, además, se han puesto barras de progreso vacías sobre las imágenes de las habilidades, junto con un texto oculto. Cuando una habilidad está en *Cooldown* se asigna el tiempo al texto y a la barra de progreso, y se va vaciando según va quedando menos para volver a tener disponible la habilidad.

Se ha creado un material que actúa como barra de progreso circular, para las barras de vida general de los personajes [25] [26].

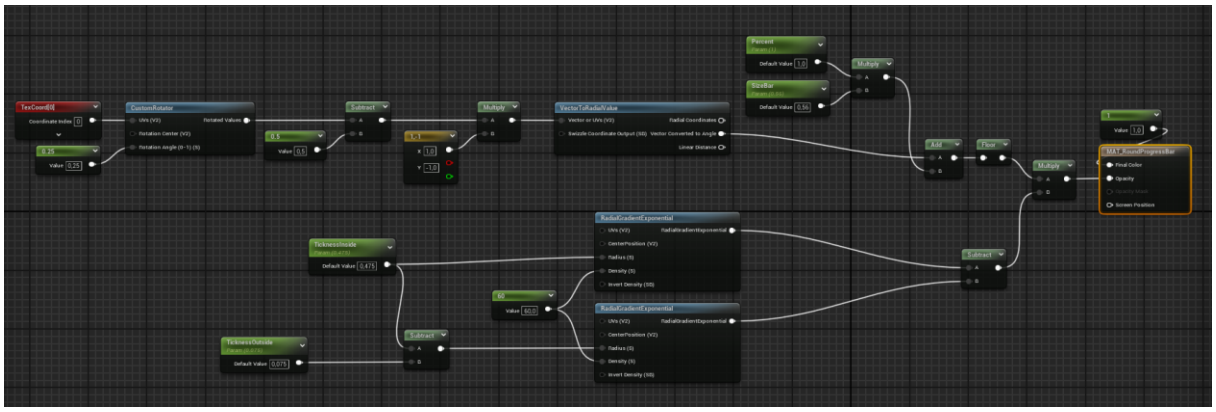


*Ilustración 36 – Material para la Barra de Progreso Circular de la vida general.*

Y un material con máscara circular para que el icono de los personajes no sea cuadrado, además, otro con máscara semicircular, para el modo 2vs2 [27].

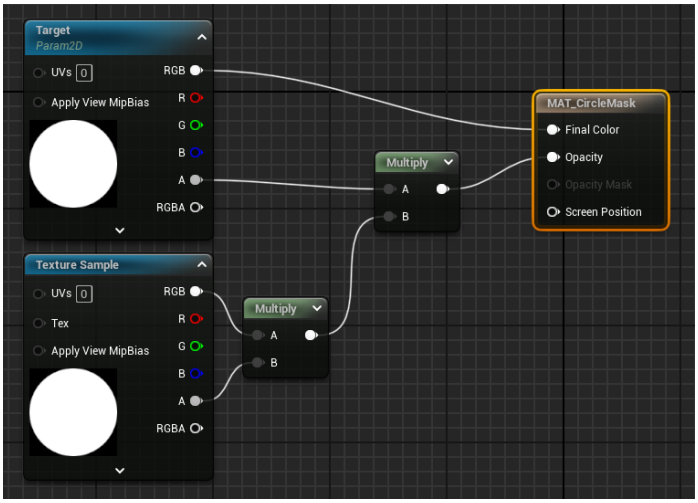



*Ilustración 37 – Material con máscara circular para iconos de personaje.*

### 5.6.3. Tienda de objetos

Para los datos del jugador, como su dinero, y los objetos ya comprados, se solicita dicha información a la *GameInstance* y se asigna a los componentes de la interfaz destinados para ello. Además, la lista de objetos ya comprados de cada jugador tiene oculto un marco sobre cada objeto, que se hace visible mediante eventos *Hover* y *Unhover* cuando el jugador se encuentra en modo vender, y por tanto pasa a moverse por su propia lista de objetos y no por los botones de la tienda.

También se han asignado casillas de texto en la interfaz, ocultas de forma inicial, que se cambia el texto según la información que deba mostrar y se activa temporalmente mediante una animación de aparición y desvanecimiento. Estos textos muestran por ejemplo si el jugador ha obtenido un bonus por derrotas acumuladas, si el dinero no es suficiente, si ya está muerto, o si su lista de objetos ya está llena.

Los botones de compra se explican en el siguiente apartado.

## 5.7. Lógica de Interfaces

### 5.7.1. Menú de Tienda de Objetos (Generación Autónoma de botones)

Se ha creado un *Widget* con la estructura del botón, añadiendo un fondo, una imagen donde aparecerá el icono del objeto, un texto para el nombre, un texto para el precio, otro con la descripción de las estadísticas que mejora, y 4 marcos, que posteriormente se pondrán del color del jugador que se coloque sobre el botón, tiene 4 marcos, porque puede darse el caso de que los 4 jugadores se posicionen sobre el mismo botón. Al igual que los botones base del juego y los de selección de personaje se han creado eventos de *Hover* y *Unhover* con una animación que agranda y achica el botón, y los eventos *ActiveColor* y *DesactiveColor* para pintar los marcos del color del jugador u ocultarlo.

Para cargar la información de los objetos en los botones, se crea una variable del tipo *F_GameItems* una estructura de datos previamente creada que contiene los campos:

nombre, descripción, icono, efecto, y precio. Con estos datos se editan los valores de los componentes del botón.

Se crea una *Data Table DT_GameItems* con la estructura *F_GameItems,* esta *Data Table* contiene todos los objetos del juego, con su información.

A continuación, se crea un nuevo *Widget* con una *HorizontalBox* vacía. Y se crea el evento "*GenerateButtonsItems*" que tiene como variable de entrada un *array* del tipo *F_GameItems* y mediante un *for each loop* crea tantos *Widget* del botón como objetos hay. Cada vez que crea un botón lo añade a un *array* de botones y lo convierte en hijo del *HorizontalBox*.

Finalmente, en la propia interfaz del menú de tienda de objetos, se crea un *VerticalBox* vacío donde se desea la tienda (en el centro de la pantalla en este caso). Y se crea un evento que coge los objetos de la *Data Table DT_GameItems*, los introduce en un *array* y crea un *Widget* de la *HorizontalBox*, al cual se le asigna como entrada el array y se llama a su evento "*GenerateButtonsItems*". Finalmente se convierte el *HorizontalBox* resultante, como hijo del *VerticalBox* del menú, y se almacena en una variable, el array de botones que devuelve el *Widget* de *HorizontalBox* para poder realizar lógica con los botones. Con esto ya se crean tantos botones como objetos haya en el juego de forma automática, si se añaden más botones a la *Data Table*, se generan más botones de forma autónoma [28].

***Optimizaciones***

Para que la *VerticalBox* no sea de una sola fila muy larga en caso de haber muchos objetos, se ha asignado un máximo de objetos por fila, cuando la *HorizontalBox* tiene ese número de botones, la añade a la *VerticalBox* y crea otra nueva *HorizontalBox*.

La *VerticalBox*, almacena múltiples *HorizontalBox* de X botones. (En el caso de este proyecto, 2 filas de 4 botones cada una, como puede verse en la **Ilustración 81** del **Anexo II**.

Para poder añadir los iconos a las descripciones de los objetos, indicando los atributos que mejora, el texto es de formato *RichText* al igual que se hace con los iconos de los botones de los mandos de las diferentes plataformas [16].

### *5.7.2. Interfaz In-Game*

#### *Estadísticas del jugador*

Para transferir los datos de vida, maná y escudo a la interfaz In-Game, es necesario transmitir la información desde el ***BP_Fighter***. En los eventos de cambio de vida, cambio de maná y cambio de escudo del ***BP_Fighter*** se manda la información al ***BP_OnyxGameMode*** a eventos creados, con el *ID* del jugador y el porcentaje de la estadística que corresponda, como atributos de entrada. Desde el *BP_OnyxGameMode* se manda la información a una ***Blueprint Interface*** que tiene eventos (*READ-ONLY)* que sirven como puente de información entre *Blueprints* e interfaces de tipo *Widget*.

Ya en la interfaz se lanzan los eventos de la *Blueprint Interface* y con sus atributos de entrada, se cambian los porcentajes de las barras de progreso que sirven como barras de vida, maná y escudo. Para saber al jugador que hay que cambiarle la barra se usa el *ID* del jugador, y mediante un *switch* se aplica la lógica a la barra de progreso que corresponda.

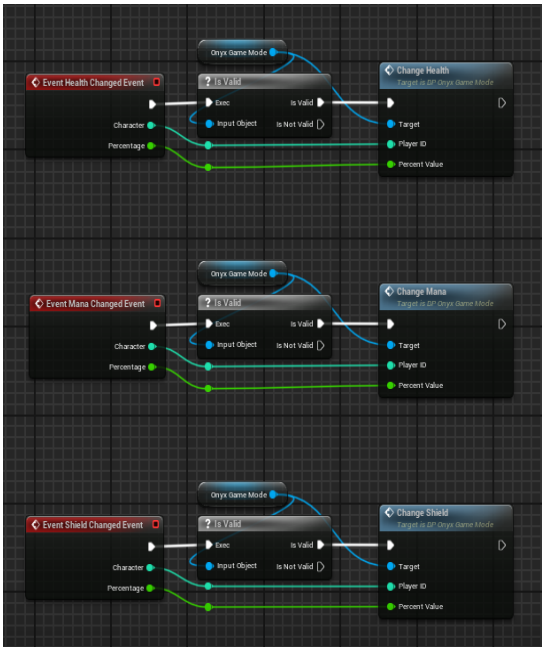



*Ilustración 38 – Eventos en BP_Fighter.*

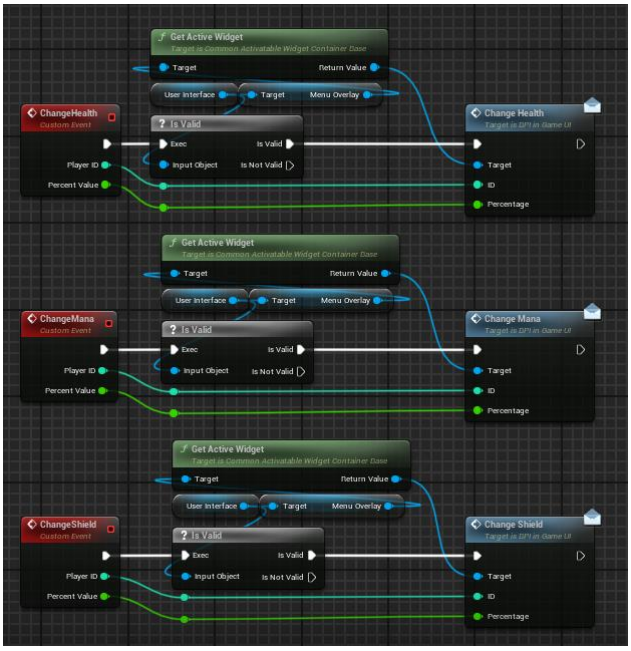



*Ilustración 39 – Eventos en BP_OnyxGameMode.*

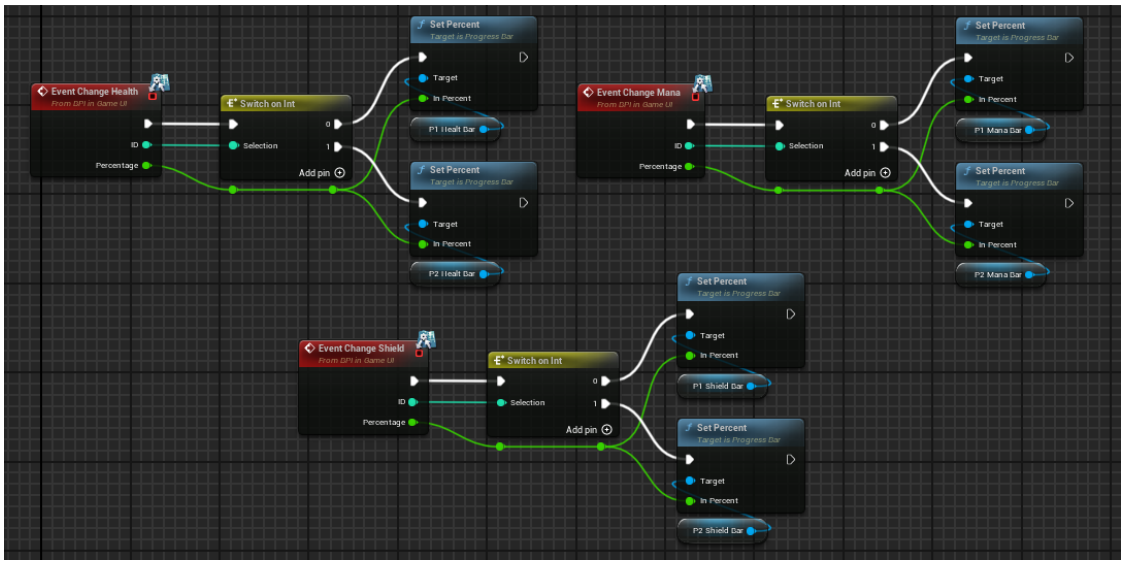



*Ilustración 40 – Eventos de Blueprint Interface dentro del Widget interfaz In-Game.*

Para el cambio de iconos, tanto de jugador como de habilidades, que se asignan automáticamente según el personaje seleccionado, se ha creado una estructura de datos que contiene una textura 2D para icono de personaje, y un *array* de texturas 2D, 4 iconos, uno para cada habilidad. A continuación, se ha creado una *Data Table* con la estructura de datos, y se creado una fila por cada personaje, asignando sus iconos donde toca.

Desde la interfaz se hace una función que asigna a cada componente la imagen que le corresponde, según el personaje seleccionado, visitando la fila de la *Data Table* que sea necesaria según la información de personaje seleccionado que contiene la *GameInstance*. Además, para que esta función se pueda aplicar exactamente igual en la interfaz de 1vs1, en la 2vs2, y en la de 4 jugadores todos contra todos, se ha creado una librería de funciones, donde se ha almacenado la función que se encarga de cambiar los iconos de personaje y habilidades, y la función que se encarga de cambiar los iconos de los objetos comprados en la tienda.

***Cooldown en habilidades***

En ***BP_Fighter*** se ha creado un nuevo evento en el *Blueprint* que tiene como entrada un atributo con el *Index* de la habilidad y otro atributo con el *Cooldown*. Esos datos se le mandan a la interfaz *In-Game* mediante el *BP_OnyxGameMode* y la *Blueprint Interface* igual que las **estadísticas del jugador**.

El evento se llama desde las propias *Gameplay Abilities* de las habilidades, antes de lanzar los efectos y la lógica adecuada, se hace un "*Cast to…*" a ***BP_Fighter*** y se llama al evento, asignando a mano el *Index* de la habilidad que corresponda y cogiendo el *Cooldown* de su propia variable *Cooldown* que tienen todas las *Gameplay Ability* de forma base.

Una vez en la interfaz, se realiza un *switch* de la variable *Index* de la habilidad, y según cual sea, se aplica la lógica al icono de habilidad que se solicite.

Se pone visible el texto para los números del tiempo restante, y la barra de progreso sobre el icono, se activa un *Timer* con el valor del atributo *Cooldown* y cada 0,1 segundos se cambia el valor de ambos componentes para ir mostrando progresivamente el tiempo restante para recuperar la habilidad.

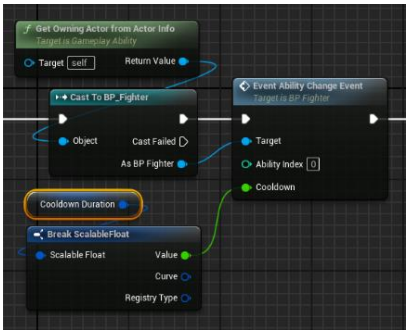



*Ilustración 41 – Gameplay Ability.*

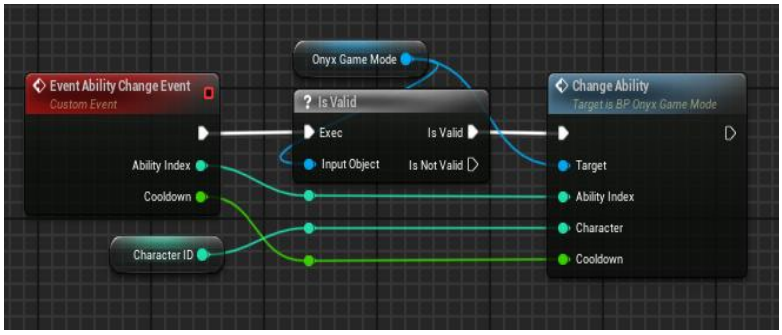



*Ilustración 42 – Evento en BP_Fighter.*


*Ilustración 43 – Cooldown en las habilidades de la interfaz.*

## 5.7.3. Indicadores dentro del personaje

### Indicadores de apuntado

El indicador de apuntado puede verse en la **Ilustración 67** del **Anexo II**. Se ha creado un *Widget* para la flecha y otro para el aro, y se han añadido ambos *Widgets* al ***BP_Fighter***, al formar parte del *Actor* al igual que la *Mesh*, la *Collision*, etc, rota junto con él, por tanto, la flecha apunta en la misma dirección que el vector *Forward* del personaje. Para que el aro este siempre girando sobre el personaje se le ha asignado una rotación local en el *Event Tick* que se reproduce de forma constante. Cuando se crean los personajes en la arena de combate, se le asigna un tinte por encima del color del jugador.

### *Indicadores de estadísticas*

Utiliza los mismos eventos que las estadísticas de la interfaz, pero en este caso no es necesario transmitir la información a través de un *Blueprint Interface* pues los indicadores son atributos del propio *Actor* que contiene los eventos de cambio de vida, maná y escudo.

Se crea un *Widget* y se diseña una barra de vida, maná y escudo semicircular, utilizando el mismo material que las barras de vida general de la interfaz *In-Game*, se crean los eventos que se encargan de cambiar el porcentaje de la barra, en función de un atributo de entrada. Se añade el *Widget* a los componentes de ***BP_Fighter***, y se le desactiva la rotación para que siempre se mantenga en la parte inferior del personaje, en el exterior del indicador de apuntado (no rota junto con el vector *Forward)*. Cada vez que se lanza el evento de cambio de vida, maná, y escudo, se transfiere el nuevo porcentaje al evento correspondiente del *Widget.*


*Ilustración 44 – Indicadores de estadísticas y el indicador de apuntado.*

### *Indicadores de daño*

Se ha creado un *Widget* que genera números que flotan sobre el personaje. Cuando un jugador recibe daño y salta el evento, se aplica la siguiente lógica, se cambia el *Overlay* del personaje a un *Overlay* rojo que indica daño, y pasado 0,1 segundos, se vuelve a asignar el *Overlay* que tenía anteriormente, si no tenía ninguno se asigna *None.* Se obtienen las

coordenadas de la cabeza del personaje accediendo a su esqueleto, y en esa posición se genera un *Actor* de números flotantes que necesita como atributos de entrada el número de daño recibido y el tipo (1 – Daño a escudos, 2 – Daño físico, 3 – Daño mágico, y 4 – Daño crítico).

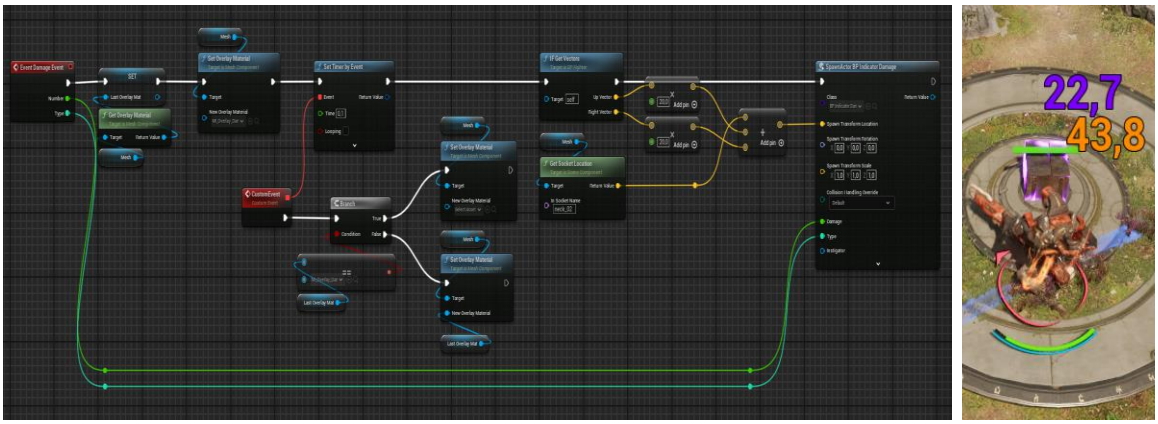



*Ilustración 45 – Lógica para generar indicadores de daño.*

El *Widget* contiene únicamente un texto centrado y vacío, y una animación que hace que el texto comience a subir y hacerse cada vez más grande, mientras su opacidad se va disminuyendo, hasta que finalmente se desvanece. Adicionalmente contiene un evento que cambia el color del texto según un atributo de entrada.

El *Actor* de números flotantes contiene un componente *Widget*, concretamente el *Widget* de números flotantes. Cuando el evento de recibir daño de ***BP_Fighter*** genera un nuevo *Actor* de este tipo, se lanza la lógica de, hacer que ese texto se mueva dentro de un rango, para evitar que dos daños muy seguidos se montasen uno sobre otro, es decir, ambos se generan cerca de la cabeza del personaje y ambos suben hacía el cielo para terminar desvaneciéndose, pero el recorrido que hacen en su ascensión es en una dirección de entre un rango *Random*. Se genera un *TimeLine* y pasado el tiempo se destruye el *Actor* (después de desvanecerse y quedarse oculto, se destruye) [29].

La mayor dificultad de este *Actor* reside en transferir la información, pues es necesario transferir desde el evento de daño del ***BP_Fighter*** al *Actor*, la posición del cuello del personaje, y posteriormente desde el *Actor* es necesario transferir al *Widget* el número y el

tipo del daño, para cambiar los valores del texto. Por tanto, son necesarias dos *Blueprint Interface*.

***Indicadores de onyx***

Junta la lógica de los indicadores de estadísticas y de los indicadores de daño. Con la primera parte de la lógica notifica el cambio a la interfaz para mostrar el nuevo número de onyx en el texto conveniente de la interfaz, y con la segunda parte de la lógica genera un *Actor* de números flotantes para notificar la cantidad de onyx ganada, por ejemplo (+500) cuando destruye una caja.

### *5.7.4. Cuenta atrás inicio de partida*

Se ha creado un *Widget* que contiene un *Blur* y un texto central donde se ponen los números de la cuenta atrás. Se le ha asignado una animación al *Blur* para que pasados 3 segundos comience a desvanecerse, dejando la pantalla sin desenfoque.

Dentro del ***BP_OnyxGameMode*** se inicializa el *Widget* al comienzo de la partida y se activa la animación. Se desactivan los *Inputs* de los jugadores y se crea un *Timer* que lanza un evento pasado 1 segundo, y se activa la casilla de *Looping* para que se repita constantemente. El evento decrementa el valor de un contador que inicia en 3, y asigna el valor de ese contador al texto del *Widget* y reproduce un sonido. Cuando el valor del contador es 0, se cambia el valor del texto a "*FIGHT*" y se reproduce un sonido que da inicio la partida. La siguiente vez que se lance el evento, como el contador tendrá valor -1, se limpiara el intervalo del *Timer* para que no siga ejecutándose y se devuelven los *Inputs* a los jugadores [30].

## 5.8. Lógica de Interfaces de uso simultaneo

Se procede a explicar la lógica dentro de los *GameControllers* de los menús de selección de personaje ***BP_SelectFighterController***, y del menú de tienda de objetos

***BP_ShopMenuController*** que contienen la lógica compleja de movimiento simultaneo de todos los personajes por los botones.

Para ello se añaden al ***InputMappingContext*** las ***InputsActions*** correspondientes a la navegación por los menús y se les asignan las teclas y botones deseados.

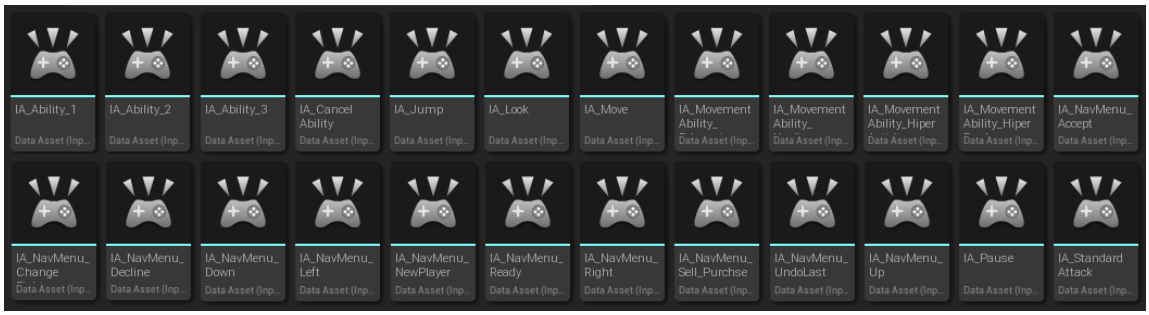



*Ilustración 46 – Input Actions del videojuego.*

## *5.8.1. Menú de Selección de Personaje*

Se realiza la lógica para cada ***InputAction*** navegación en todas las direcciones, selección, cambio de personaje, indicar que estás listo, y retroceder al menú.

***Retroceso***

Retroceder al menú simplemente necesita abrir el *Level* del menú principal.

***Navegación***

La navegación solicita al *Controller* el *ID* del jugador, y en función del jugador que sea, incrementa o decrementa el valor de la variable *Index* de ese jugador. Se coge el botón correspondiente al valor del *Index* del jugador del conjunto de botones que contiene el *array* de botones y se le ejecuta el evento *Hover* y *ActiveColor* con el color correspondiente al jugador. El anterior botón donde se encontraba el jugador se le hace un *DesactiveColor* y un *Unhover* en caso de que ningún otro *Index* de jugador se encuentre en el mismo botón.

Si el *Index* se sale de los límites del *array* se reposiciona en la posición final o inicial, según corresponda para hacer un bucle de movimiento por los botones. Cuando un botón ya está seleccionado por otro jugador, se continúa incrementando el *Index* hasta posicionarse en un personaje que no esté seleccionado, mediante un *while loop.* La navegación hacia arriba o abajo permite acceder al botón de personaje *Random* o regresar al *array* de

personajes, en la misma posición que estabas en caso de que sea accesible, si ese personaje ya está seleccionado se incrementará el *Index*.

Cada vez que se está sobre el botón de un personaje concreto, se carga la *Mesh* de tal personaje en los *Actors* que actúan como visualizadores de la selección, para una mayor retroalimentación visual [31]. Puede verse en la captura del menú de selección del **Anexo II**.

Cuando ya se ha realizado una selección de personaje y es visible el menú de selección de *skin* los botones de navegación pasan a cambiar la *Mesh* del *Actor,* pero esta vez entre el conjunto de skins del personaje concreto, el conjunto de *skin* actúa similar al *array* de botones, pues al llegar a la última y pulsar el botón de avanzar, regresa a la primera para mantener un bucle.

***Selección***

Cuando se selecciona un personaje se hace *Unhover* y marca como seleccionado en una variable que la navegación consulta. Se ocultan los marcos, pues ya no es navegable ese botón y se tinta del color del jugador que lo ha seleccionado para dar retroalimentación visual. Se notifica a la *GameInstance* la selección del jugador para que almacene el personaje y se activa el menú de selección de *skin*.

No permite seleccionar un personaje ya seleccionado. Puede darse el caso de estar ambos jugadores en la misma casilla, y uno de ellos seleccionar, el otro jugador no podrá seleccionar, aunque se encuentre sobre el botón.

Además, *Random* no puede seleccionar personajes ya seleccionados.

***Cambiar de personaje***

Se ha asignado un *Input* que permite deseleccionar el personaje, retirar de la *GameInstance* la anterior selección, oculta el menú de selección de skin, devuelve los *Inputs* de navegación a los botones de personaje, vuelve a marcar el personaje como no seleccionado, lo quita el tinte del botón, y hace *Hover* sobre el *Index* que se encontraba cuando seleccionó anteriormente.

***Indicar Listo***

La lógica del *Input* solo se realiza cuando el jugador se encuentra escogiendo *skin*, es decir, cuando el menú de selección de *skin* está activo. Asigna la skin seleccionada a la *GameInstance*, retira los *Inputs* al jugador, y pone un texto en pantalla indicando que está listo, a la espera de que el resto de los jugadores seleccionen sus personajes.

Cuando todos los jugadores están listos, se abre el *Level* de juego y comienza la batalla.

### *5.8.2. Menú de Tienda de Objetos*

Se realiza la lógica para cada ***InputAction*** navegación en todas las direcciones, selección, deshacer ultima compra, cambiar entre modo de venta y modo de compra, y activar o desactivar menú de pausa.

***Pausa***

El menú de pausa tiene la misma lógica que en el ***BP_Fighter***, con la única diferencia de que en este caso no hay personajes de fondo, en este caso hay un menú. Pausa paraliza el juego, *Timer* incluido, lo cual es suficiente en el combate, y mantiene activos los *Inputs* para moverse por el menú de pausa [32] [33]. Esta última característica ocasiona un problema en este caso pues lo que hay debajo del menú de pausa, también es un menú, pero de compra de objetos, como los *Inputs* no se desactivan para los menús, puede seguir comprando y moviéndose por la tienda, mientras realiza acciones en el menú, para arreglar esto se asigna una variable *booleana* a un *Branch* al inicio de todos los ***InputsActions*** que comprueba que la pausa no esté activa para permitir los *Inputs* correspondientes a la lógica de la tienda.

***Navegación***

La navegación hacía la izquierda o derecha, se realiza igual que la de selección de personaje, se solicita al *Controller* el *ID* del jugador, para saber que *Index* se debe tener en cuenta, y se recorre el array de botones de los objetos ejecutando los métodos *Hover* y *Unhover*, y *ActiveColor* y *DesactiveColor*, cuando corresponda. Si el *Index* de cualquier otro

jugador es igual que el del jugador que abandona el botón, no se realiza el *Unhover*, pues alguien seguirá en el botón cuando se cambie el *focus* del jugador que realiza el *Input*.

Como se ha asignado un limitador de 4 objetos por fila, el *Input* de navegación hacía arriba o abajo, incrementa o reduce el *Index* en 4 unidades, y cuando se sale de los límites del *array* lo reposiciona donde corresponde, comprobando el resto de intentar incrementar o decrementar 4, entre la longitud del *array*, para de esta forma, hacer un bucle de la columna de objetos.

Cuando se cambia el modo de compra a modo venta, se tiene en cuenta el *Index* de venta, que recorre el *array* de objetos propios. Los huecos que aún no contienen ningún objeto se asignan con valor -1, y para no visitarlos, hay disponible una variable *IndexTop* que recoge el número de objetos, o, en otras palabras, el tope hasta donde es necesario visitar el *array*. En este caso siempre se hace *Unhover* pues cada jugador accede a su propio menú de objetos, y por esa misma razón, no es necesario que el marco tenga color, se asigna un color neutral (blanco) al marco.

***Selección***

Cuando se compra un objeto se resta el importe de la cantidad de onyx del jugador, se añade el efecto de dicho objeto a un *array* de efectos en la *GameInstance* para cargarlos como efectos iniciales en la próxima ronda de combate, se aumenta en 1 el valor del *IndexTop* y se asigna el icono del objeto a la posición correspondiente de la lista de objetos propios del jugador.


*Ilustración 47 – Efectos de los objetos, contienen la mejora de atributos que corresponda.*

Se almacena el precio del objeto comprado en una variable, por si el jugador decide deshacer la compra, poder devolverle el importe.

Si el jugador intenta comprar con la lista de objetos llena o sin tener dinero suficiente, salen las notificaciones de texto en pantalla, como se vio en el apartado de diseño.

En caso de estar en el modo vender, selección sumaría el importe correspondiente (70% del coste del objeto) al contador de onyx, se quita el efecto de dicho objeto del *array* de efectos en la *GameInstance*, y se reduce en 1 el valor del *IndexTop*, además se retira el icono del objeto de la lista de objetos propios, y se recolocan los objetos restantes, para no dejar huecos vacíos entre objetos.

***Deshacer última compra***

Se devuelve el importe de la anterior compra, reflejado en la variable, se retira el efecto del *array* de la *GameInstance,* se resta 1 al *IndexTop* y se retira el último icono de la lista de objetos propios.

***Cambiar entre modo Vender o Comprar***

Cambia el valor de la variable *booleana* que notifica a los *Inputs* de navegación que está en un modo u otro para usar el *Index* oportuno.

Cuando se activa el modo vender se hace el *DesactiveColor* y el *Unhover* (si es necesario), del botón de objetos donde se encuentre el jugador, y se activa el marco de la lista de objetos propios en el objeto del *Index* de venta.

Cuando se activa el modo comprar se desactiva el marco de la lista de objetos propios y se hace *Hover* y *ActiveColor* del botón de objetos donde se encuentre el jugador según su *Index* de compra.

El modo venta no se activa cuando el *IndexTop* del jugador es 0.

## 5.9. Habilidades de Movimiento

Cada una de ellas es una clase ***Gameplay Ability*** con lógica en su interior, que aplica **efectos** a los personajes, y activa ***GameplayCues*** [19]. Su lógica correspondiente se explica a continuación para cada habilidad.

### 5.9.1. HiperDash

Utiliza un *Line Trace By Channel* para trazar una línea entre dos posiciones dadas en forma de coordenadas, se utiliza la posición del propio personaje que lanza la habilidad como coordenada de inicio, y como coordenada final se utiliza la posición del personaje más un incremento de 400 unidades en la dirección del vector *Forward* del personaje. Si es posible teletransportarse a esta ubicación reposiciona al personaje en esta localización, si no es posible, encuentra el punto más lejano dentro de las 400 unidades a las que si puede teletransportarse y lo posiciona ahí [34]. Y a continuación aplica la *GameplayCue* correspondiente.

### 5.9.2. HiperActivity

Aplica un efecto al personaje que activa la habilidad con un incremento del atributo de velocidad de movimiento de forma temporal, junto a este efecto, se llama a la *GameplayCue* correspondiente.

### 5.9.3. Healing

Aplica un efecto al personaje que activa la habilidad con un incremento del atributo de escudo, y a su vez, otro efecto con un incremento temporal de la regeneración de vida base, para curarse más durante el tiempo que dura la habilidad. Junto a cada efecto, se llama a las *GameplayCues* correspondientes.

Una vez utilizado, se cambia la variable *Cooldown* a *-1*, para garantizar que esta habilidad solo se tiene una vez por partida, y una vez tirada no vuelve a estar disponible.

### 5.9.4. Exhaust

Se comprueba los *Actors* en un rango de 500 unidades mediante un *Sphere Overlap Actors*, este nodo devuelve un *array* de *Actors*. A continuación, se realiza un *for loop* para cada *Actor* que cumpla la condición de ser un ***BP_Fighter***, esto se comprueba mediante un "***Cast To…***". A todos aquellos *Actors* del tipo ***BP_Fighter*** que se encuentren dentro del

rango, se les aplica un efecto que reduce el valor del atributo de velocidad de movimiento durante un tiempo determinado, y otro efecto que destruye al instante cualquier escudo que puedan tener activo (cambia el atributo de escudo por un 0).

Al igual que ***Healing*** es una habilidad de un solo uso, por ello, se cambia el valor de *Cooldown* a *-1*.

## 5.10. Efectos Visuales y Sonoros

### *5.10.1. Partículas y Overlays*

Las partículas y los *Overlays* son aplicados a los personajes para destacar visualmente un efecto sin retroalimentación visual aparente. Concretamente se añaden partículas y *Overlays* cuando se activa una habilidad que mejora estadísticas en el personaje, en las habilidades de movimiento, y en los eventos de daños recibido, además se ha añadido un efecto visual llamativo a las cajas de onyx, para captar la atención de los jugadores. Para lanzar estas partículas y *Overlays* (y los sonidos) junto con los efectos, se hace uso de las ***GameplayCues***, estas reproducen partículas, sonidos, animaciones, etc., mientras el efecto está activo y detiene todo cuando finaliza el efecto. Se han creado tantas ***GameplayCues*** como efectos se querían retroalimentar visualmente, y en cada *Cue* se ha desarrollado la lógica deseada.

Las partículas se han creado mediante el ***Cascade Particle System*** de Unreal Engine.

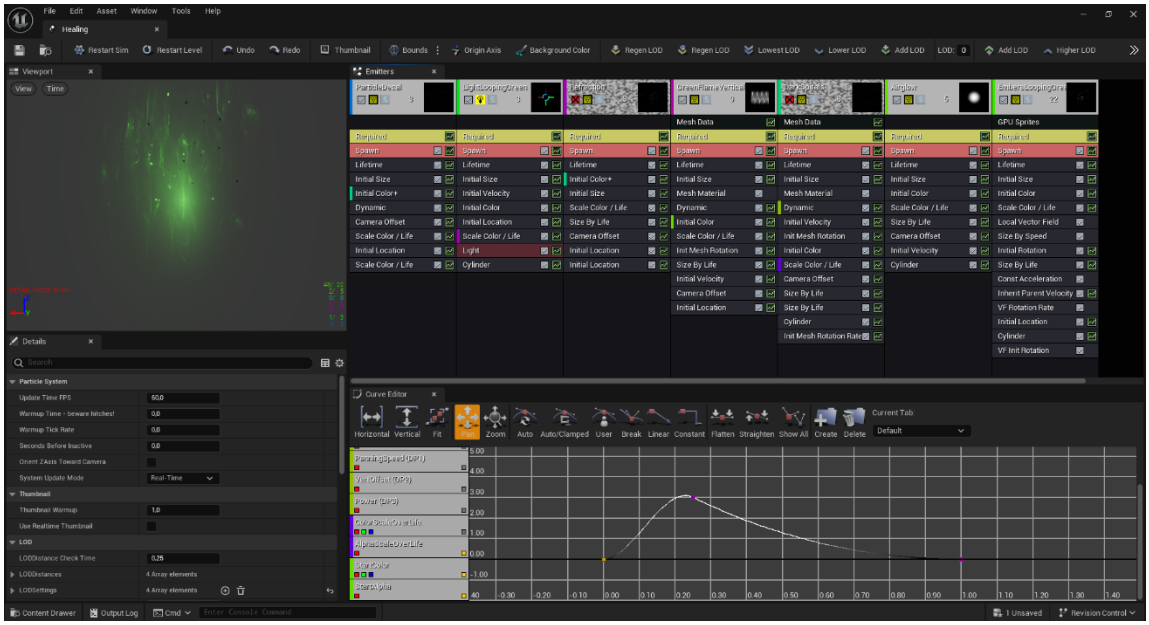


*Ilustración 48 – Cascade Particle System (Ejemplo partículas para Healing)*

Y los *Overlays* se han creado mediante un material, e instancias de este con diferentes colores para diferentes efectos. Se ha configurado el *Blend Mode* del material en *Translucent* y se ha implementado la lógica para que el material emita luz en los bordes y se mueva con una textura de nubes en el interior. Se aplica al "*Overlay Material*" dentro de la categoría *Rendering* del *Blueprint* del personaje [35].

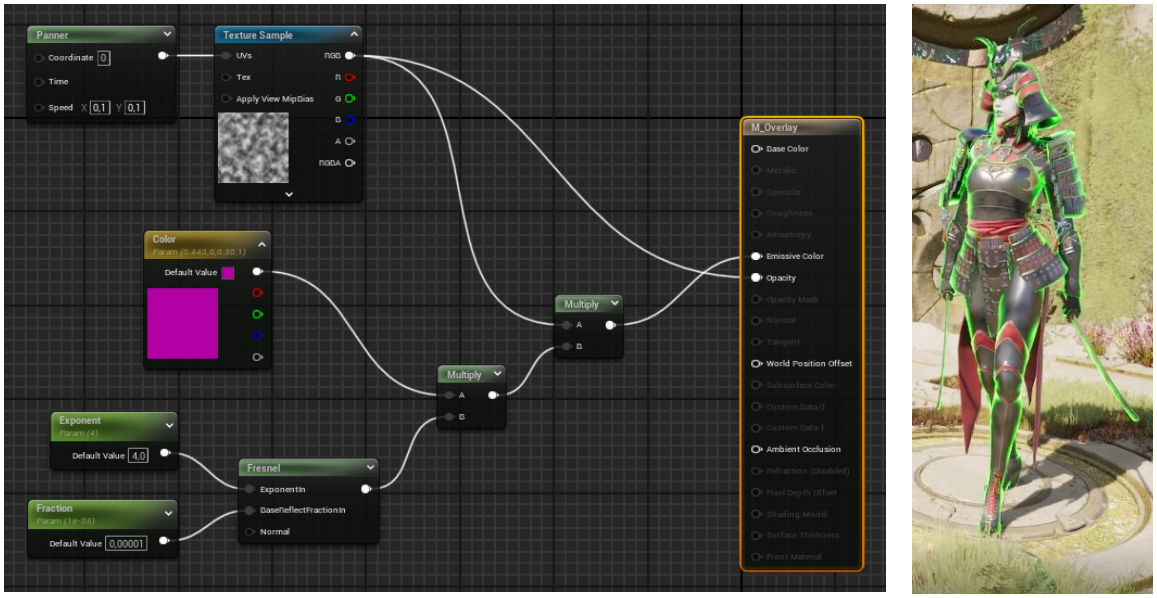



*Ilustración 49 – Material para los Overlays (Ejemplo verde).*


*Ilustración 50 – Instancias del Material Overlays para cada efecto.*

### *5.10.2. Decals*

Se han creado *Decals* de Unreal Engine, con máscaras que pueden observarse en **Ilustración 66** del **Anexo II**, con las máscaras se han creado texturas y con las texturas se han generado los materiales de las *Decals*, aplicando el *Blend Mode* del material en *Translucent* y asignando la textura al *Opacity* del material.

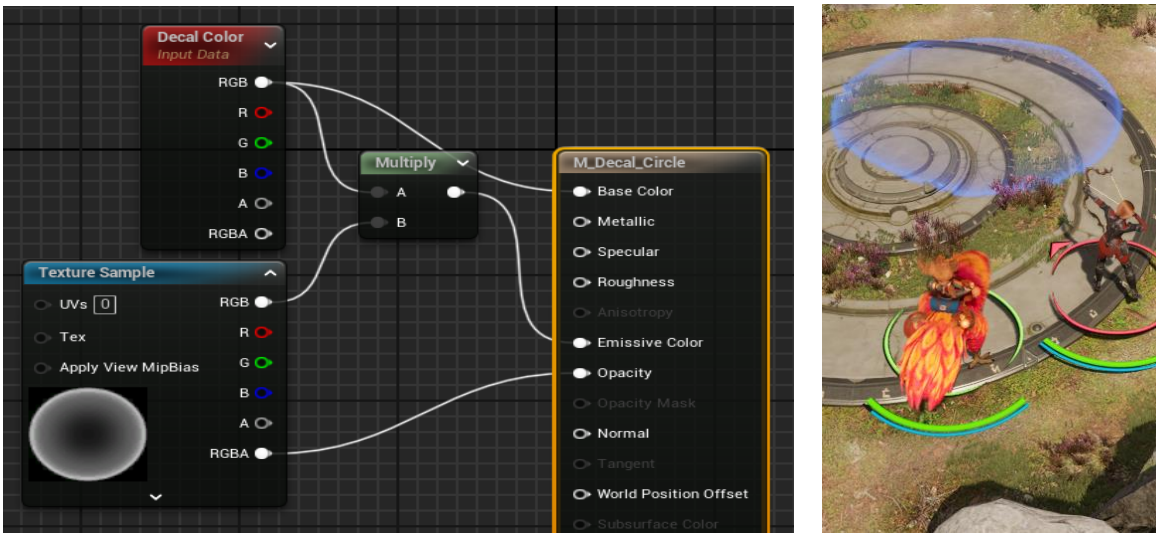



*Ilustración 51 – Material para las Decals (Ejemplo circular).*

Estas *Decals* indican el rango de ataque de las habilidades de los personajes. Se les ha asignado un tono azul, para que se pueda apreciar sobre cualquier parte del escenario.

Como ya se ha comentado en el apartado de **Personajes** estás aparecen cuando la habilidad va a ser lanzada (al mantener el botón pulsado), cuando el botón se suelta, o bien cuando se cancela la habilidad con el botón de cancelas habilidades, la *Decal* desaparece.

### 5.10.3. Sonidos

Todos los efectos de sonido han sido obtenidos de un banco de sonidos gratuito con licencia de uso, *Zapsplat*[10], y posteriormente retocados en valores de *pitch*, ecualización, velocidad, etc., tal como se comentó en el apartado de ***Herramientas utilizadas***. De esta forma es posible reproducir sonidos medianamente diferentes, pero similares, en cada habilidad, por ejemplo, varios sonidos diferentes de corte del viento con la espada, obtenidos de un mismo efecto de sonido base.

Se han aplicado sonidos a todas las habilidades de los personajes, a las habilidades de movimiento, a la cuenta atrás de inicio de partida, y a las cajas de onyx una vez son destruidas y otorgan el dinero a los personajes.

A nivel de interfaz y menús, se ha aplicado sonidos a cualquier navegación por los menús, al hecho de clicar un botón o aceptar una selección, y a su contrario, es decir, cuando se vuelve atrás en los menús o se deselecciona.

### 5.10.4. Música de fondo

Se ha aplicado una música de fondo en un volumen que no sea demasiado estridente, junto con un sonido ambiental de pájaros piando, y ruido de árboles. Ambos sonidos han sido adquiridos de un banco de sonidos gratuito con licencia de uso, *Pixabay*[11].

Para lograr que ambos se mantengan en reproducción entre *Levels* se han inicializado en el *GameMode* del menú principal y se ha recogido en una variable dentro de la *GameInstance* como ya se comentó en el apartado de ***GameInstance***. Además, se ha indicado que estos sonidos deben reproducirse en bucle para mantenerse constantemente, y se ha retocado para que el inicio y final coincidan lo mejor posible y no se note el reinicio.

[10] https://www.zapsplat.com
[11] Music by Amir Firouzfard / Ambient Sound by Empress-Kathryne Nefertiti-Mumbi from Pixabay

## 5.11.Inteligencia Artificial

Tal como se ha explicado en el apartado de **Diseño del videojuego** referente a la **Inteligencia Artificial**, al inicio del proyecto se ideó una forma de incluir IA en el videojuego, que finalmente se ha sustituido por otra.

La inteligencia artificial desarrollada se trata de una única IA realizada de forma conjunta entre ambos alumnos, destinada a controlar los personajes jugables y combatir entre ellas. Su enfoque es que sirva para enseñar a los jugadores a utilizar los personajes y sus habilidades con un ejemplo de combate real, y también es útil a nivel de desarrollo pues permite a los desarrolladores analizar combates entre personajes, con estadísticas de tasa de victoria para concluir si un personaje necesita ajustar sus daños y realizar el balance del juego.

La inteligencia artificial consta de un *Behaviour Tree* o árbol de comportamiento que está esquematizado en la **Ilustración 63** del **Anexo I**, sin embargo, se han aplicado algunas ramas más, puede observarse el árbol completo en **Ilustración 64** del **Anexo I**, y de varias Environment Query System (EQS) para mejorar su comportamiento con acciones concretas.

La inteligencia artificial cuenta con un *Blackboard* donde almacena variables que están a disposición del *AI Controller*. Estos datos se actualizan o se usan, en tareas del *Behaviour Tree*. Mediante la variable *SelfActor* se reconoce a sí mismo, *Enemy* reconoce al enemigo (al ser un combate 1vs1 el enemigo es el otro *Actor* que no sea uno mismo), también recoge el *AbilitySystemComponent* propio en una variable, y el del enemigo en otra. Finalmente contiene 2 variables con vectores de posición y 3 valores numéricos con rangos.

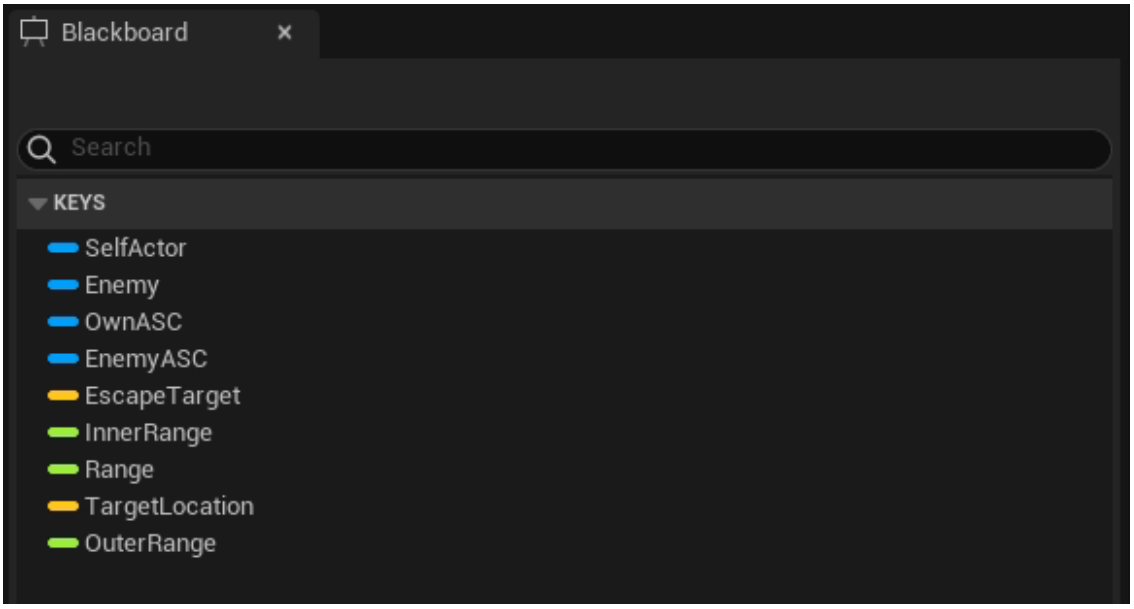



*Ilustración 52 – Blackboard de la Inteligencia Artificial.*

Una vez implementado el *Behaviour Tree* se ha mejorado añadiendo ciertos *Delays* a las acciones para generar un combate con unos rangos de reacción más humanos. Y se han añadido cuatro Environment Query System (EQS), una para escapar del enemigo cuando se tienen todas las habilidades en *cooldown* y alejarse el rango suficiente para poder volver a atacarle desde máximo rango la próxima vez que se tenga disponible una habilidad, lo que se conoce como *Kitting*. Esta segunda parte es la que se gestiona desde la otra (EQS), encargada de buscar combate desde el rango máximo de las habilidades.

La tercera (EQS) es una derivación de la segunda, busca mantenerse en movimiento mientras ataca, para no estar nunca realizando ataques en estático, utiliza la misma lógica, intenta moverse a un punto alejado del enemigo, pero a la vez siga manteniéndose dentro del rango, para que el ataque se efectúe con éxito.

La cuarta (EQS) es una derivación de la primera, busca esquivar ataques del enemigo, moviéndose a una posición separada de él, cuando detecta que el enemigo está realizando algún ataque.

Las (EQS) crean un mapa de puntos conocido como *Donut*, y a esos puntos se les asigna un valor en base a una serie de condiciones.

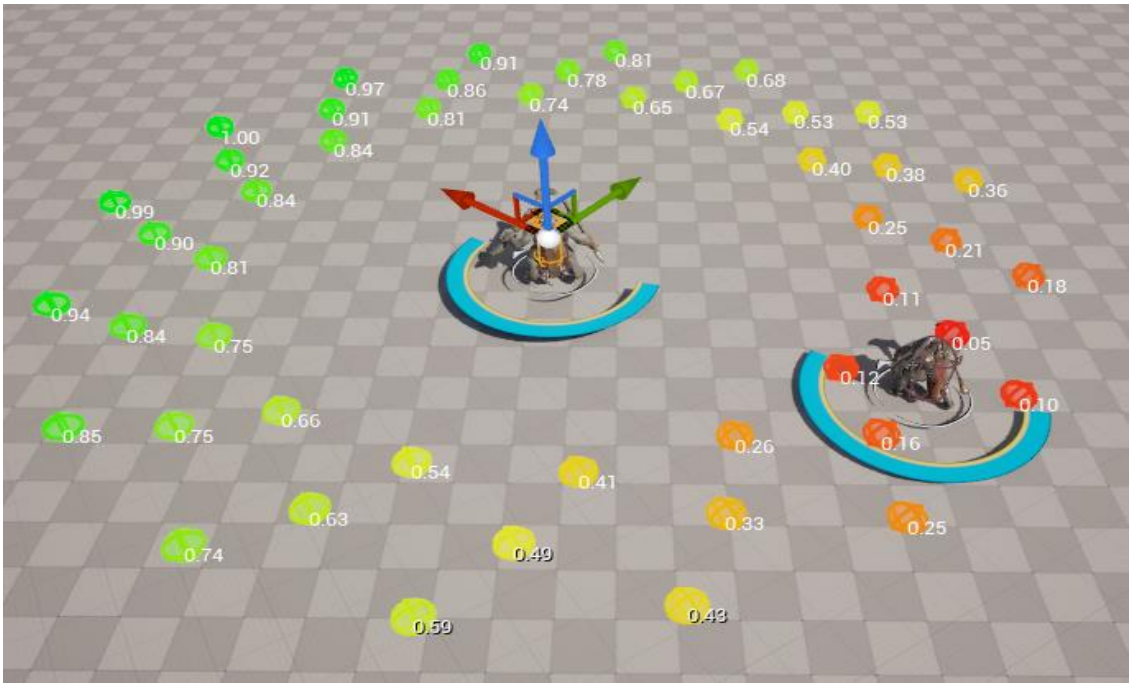



*Ilustración 53 – Environment Query System (EQS) para escapar.*

### 5.11.1. EQS_FindCombatLocation

Genera el mapa de puntos alrededor del enemigo, con un rango de distancia entre el rango de ataque de *SelfActor* y un rango menor (más cercano al enemigo).

Asigna valores a los puntos en función de la distancia que hay desde el enemigo, lugar donde se genera el mapa de puntos, y el *Actor* que ejecuta la (EQS) y cuanto más corta es la distancia, más valor les da a los puntos, por tanto, los de mayor valor son los más cercanos a sí mismo.

Después hace un *PathExist* que descarta los puntos a los que no puede ir, por ejemplo, porque estén fuera de la *NavMesh*. Finalmente escoge un punto *Random* de entre el 25% de los mejores puntos.

### 5.11.2. EQS_FindEscapeLocation

Genera el mapa de puntos alrededor de sí mismo, con un rango de distancia entre el rango de ataque propio, y un rango mayor (se aleja más allá de su propio rango con respecto al enemigo).

Asigna valores a los puntos en función de lo alejados que estén del enemigo, para mantenerse lo más alejado posible a la espera de recuperar alguna habilidad, pero lo suficientemente cerca para atacarle nada más la recupere, lo que se conoce comúnmente como *Kiting* en los videojuegos. Escoge un punto *Random* de entre el 5% de los mejores puntos.

El comportamiento de esta (EQS) es el que puede observarse en el ejemplo de la **Ilustración 27**.

# Capítulo 6
# Validación

## 6.1. Resultado final

Finalmente se ha logrado realizar un videojuego con cuatro modos de juego diferentes, para 2 o 4 jugadores en multijugador local, pudiendo elegir entre jugar 1vs1, 2vs2 realizando dos equipos, o 4 jugadores todos contra todos. Además, se han incluido con éxito nueve personajes seleccionables, con *skins* variadas, y con cuatro habilidades diferentes cada uno de ellos. Cada personaje atributos únicos que pueden verse modificados mediante la compra de objetos. El máximo de objetos permitidos a cada jugador es siete, que pueden comprar en la tienda de objetos al finalizar cada ronda de juego. También tienen permitido vender objetos comprados anteriormente, por el 70% de su coste. Los objetos disponibles en la tienda son ocho, cada uno enfocado en mejorar unas estadísticas concretas. Hay objetos de daño físico, objetos de daño mágico, objetos que proporcionan armadura, resistencia mágica, más vida, etc. También, se han incluido cuatro habilidades de movimiento comunes para todos los personajes, que aportan un pequeño teletransporte, velocidad de movimiento, curación y escudo, y ralentización al adversario combinado con rotura de los escudos activos.

Los menús iniciales son controlados únicamente por el jugador 1, el menú de pausa es controlado por el jugador que ha pausado el juego únicamente, y finalmente los menús de selección de personaje y de la tienda de objetos, permiten el movimiento simultaneo de todos los jugadores.

Para rematar el proyecto y como herramienta de ajuste de los daños de los personajes, se ha realizado una inteligencia artificial capaz de combatir, y se ha creado un modo de juego de IA. vs IA. donde los jugadores pueden aprender las habilidades, y los

desarrolladores podemos obtener datos estadísticos de la tasa de victoria de cada personaje.

En el **Anexo II** pueden verse capturas de pantalla del videojuego en funcionamiento que validan el correcto funcionamiento de este.

Para finalizar se ha subido la versión final del videojuego a la plataforma itch.io, en formato descargable, acompañada de una descripción del proyecto, explicación de los controles, capturas de pantalla del videojuego en ejecución y los créditos correspondientes a los propietarios de los materiales de terceros utilizados en el trabajo.

Como el juego ocupa más de 1GB que es el tamaño que permite itch.io de forma base, ha sido necesario utilizar herramienta de líneas de comando *Butler*, que está integrada en el *pipeline* automático de itch.io, permitiendo subidas de archivos de hasta 30GB.

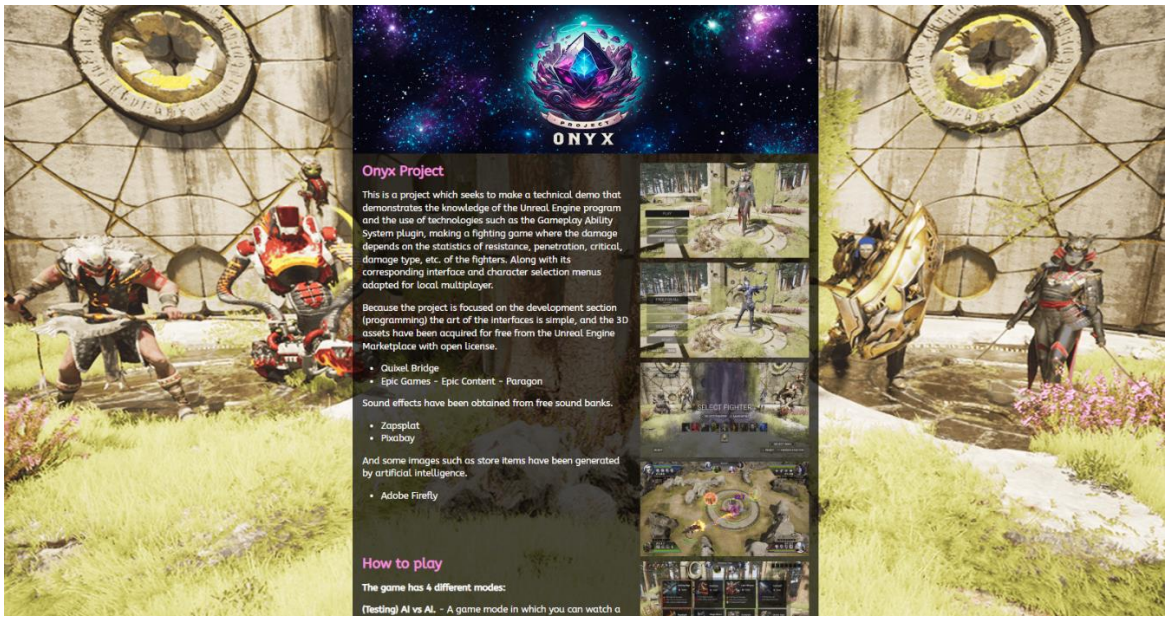



*Ilustración 54 – Proyecto subido a itch.io en forma de descargable.*

Debido al intervalo de tiempo tan limitado desde la subida del proyecto, hasta la presentación de este, no ha sido suficiente para obtener un gran *feedback,* pero se ha recomendado probar el videojuego a los amigos cercanos y han dado su opinión y recomendaciones para futuras versiones, que han sido recogidas en el apartado **líneas futuras**, de las **conclusiones**.

# Capítulo 7
# Conclusiones

Los resultados obtenidos demuestran la viabilidad de desarrollar un videojuego funcional y atractivo utilizando recursos gratuitos disponibles en el Unreal Engine *Marketplace*, además de resaltar las capacidades del motor Unreal Engine para crear experiencias de juego complejas y dinámicas. El proyecto también destaca la importancia de la programación en el desarrollo de videojuegos, mostrando cómo una buena arquitectura de *software* puede facilitar la implementación de diversas modalidades de juego y comportamientos de IA.

Este trabajo proporciona una base sólida para futuros proyectos desarrollados y estudios en el campo de los videojuegos, ofreciendo tanto una herramienta de entretenimiento como una plataforma de aprendizaje sobre programación y desarrollo en Unreal Engine.

## 7.1. Logros alcanzados

Se procede a comparar los objetivos propuestos al inicio del documento con los logros alcanzados durante el desarrollo del proyecto, para determinar así cuan satisfactorio es el resultado final obtenido.

El objetivo principal del proyecto era **desarrollar un prototipo de juego con el motor de desarrollo Unreal Engine en su última versión 5.3**, junto con el **uso del *plug-in* Gameplay Ability System (GAS)**, estos objetivos han sido logrados considerablemente con la propuesta presentada, pues se puede elegir entre cuatro modos de juego diferente, y con la posibilidad de elegir entre nueve personajes con una pack de habilidades y características diferentes, además, existen cuatro habilidades de movimiento comunes a todos los personajes, y una pila de ocho objetos con estadísticas diferentes entre los que elegir.

También se ha logrado **adaptar correctamente el prototipo a multijugador local**, **permitiendo a los jugadores elegir el personaje deseado de forma simultánea en el menú de selección de personajes, así como comprar objetos en la tienda también de manera síncrona**.

**Se ha desarrollado una Inteligencia Artificial (IA)** capaz de tomar decisiones, suficientes para realizar un combate coherente y con un resultado lucrativo de cara a mantener el juego balanceado, gracias a los datos obtenidos, como porcentaje de victoria de cada personaje.

La envergadura del resultado obtenido junto con su calidad sirve de abal para dar por conseguidos en buena medida los objetivos de **aprender a utilizar el motor Unreal Engine con soltura, aprender a programar en *Blueprints* y mejorar los conocimientos de *C*++,** y, por otro lado, también **demuestra una buena compenetración del equipo y un correcto uso de las herramientas colaborativas**.

Finalmente, el juego se ha compilado en forma de ejecutable y **se ha subido a la plataforma Itch.io de forma pública**.

## 7.2. Lecciones aprendidas

El desarrollo de la demo técnica desde su planificación hasta su resultado final ha supuesto una serie de desafíos que no habían sido previsto en un principio, lo cual ha propiciado una experiencia de aprendizaje muy interesante que ha permitido profundizar más de lo esperado en comprender el funcionamiento de un programa tan complejo como es Unreal Engine, y su consiguiente familiarización con el *software*.

Además, el hecho de realizar el proyecto de manera conjunta con un compañero, sumado al uso de herramientas colaborativas con su conveniente reparto de tareas, también es destacable como un refuerzo de las habilidades blandas, experimentando un caso real de trabajo en equipo, comunicación, resolución de problemas, etc. También ha permitido

ampliar los conocimientos en el uso de las herramientas colaborativas, como GitHub, muy presentes en el ámbito profesional.

Adquirir estos conocimientos se percibe como un punto favorable a la hora de trabajar profesionalmente en futuros proyectos a nivel laboral, proyectos con carácter comercial, de gran calidad y con estricta organización dentro del equipo de diseñadores y desarrolladores.

## 7.3. Impacto del proyecto

Se ha tenido en cuenta como el proyecto puede aportar ayuda en el ámbito académico o profesional del desarrollo de videojuegos, en función de las complicaciones que han ido surgiendo durante su realización y en vista de la poca información que había en foros de programadores para dar solución a las mismas.

Los principales problemas que se han encontrado han sido:

- **La falta de ejemplos claros y amplios de videojuegos realizados con el *plug-in* Gameplay Ability System**, donde hubiese varios personajes con diferentes características interactuando entre sí.
- **La adaptación de una interfaz capaz de manejar la mayoría de los datos resultantes del Gameplay Ability System**, teniendo que informar de la cantidad y tipo de daño, los tiempos de enfriamiento de las habilidades, estadísticas que se ven alteradas con cada objeto, etc.
- **La implementación de un menú de selección de personajes o de tienda donde comprar objetos, que sea funcional para varios jugadores actuando simultáneamente en multijugador local.**

Estos puntos han sido conseguidos y explicados en este documento para facilitar la realización de futuros proyectos similares, además, gracias a que la memoria se encuentra bajo licencia *CC BY-NC-SA 4.0* se espera que sirva como iniciación para cualquier persona interesada en el tema.

## 7.4. Líneas futuras

Durante la planificación y la realización del proyecto han surgido ideas atractivas pero que finalmente no han podido ser implementadas por falta de tiempo o por ser demasiado ambiciosas. Además, han llegado nuevas ideas por parte de personas que han probado el videojuego después de su publicación que también merecen la pena ser comentadas. A continuación, se resumen las propuestas más llamativas, que podrían mejorar notablemente el proyecto en futuras actualizaciones.

- **La implementación de un modo de juego en solitario** que permita al jugador invertir tiempo de juego sin necesidad de tener un compañero físicamente con él. Esto se puede conseguir de varias formas:
    - **Un modo de juego estilo supervivencia**, donde el jugador seleccione un personaje con el cual deba defender un objetivo el máximo tiempo posible sin ser destruido. Se generaría oleadas de *minions* enemigos controlados por IA con el objetivo de matar al personaje y destruir el objetivo que este defiende. Tras cada ronda el jugador podría comprar objetos que mejoren las estadísticas de su personaje para sobrevivir más fácilmente a la siguiente horda. Si el jugador muere tendría una penalización de tiempo para reaparecer que permitiría a la IA dañar el objetivo. El juego terminaría cuando la vida del objetivo llegue a cero y se mostraría al jugador el número de rondas que ha conseguido superar y el tiempo de juego.
    - **Un modo de juego de combate 1vsIA**, igual que los actuales modos de juego, pero en este caso el jugador seleccionaría un personaje con el que jugar contra otro personaje controlado por una IA.
    - **Un modo de juego *Online***, mediante la implementación de un servidor ampliar la posibilidad de jugar con amigos a la modalidad *Online* y no únicamente multijugador local.

- **La incorporación de enemigos neutrales en ciertas rondas**, estos enemigos aparecerían en rondas aleatorias deambulando por el mapa, pudiendo tener un carácter neutral, un carácter agresivo y por tanto atacar a los jugadores de manera indiscriminada, o un carácter explorador e intentar destruir las cajas de dinero para de esta forma aumentar su botín. Cuando un jugador logre matar a este personaje controlado por la IA, obtendría algún tipo de mejora (vida, maná, daño, etc) además de una recompensa monetaria que podría verse incrementada con el valor del botín en caso de tratarse de un personaje con carácter explorador.
- **La obtención de mejoras tras varias rondas finalizadas**, similar a los objetos, pero sin necesidad de invertir dinero en su obtención y además adquirible por todos los jugadores en la misma ronda. Se mostrarían a cada jugador tres mejoras aleatorias de entre las cuales podría seleccionar una, que mejore las estadísticas de su personaje en algún aspecto.
- **Desarrollo de nuevos personajes y objetos** para futuras actualizaciones además de ir balanceando periódicamente los actuales para intentar mantener los combates lo más igualados posible.
- **Diseñar nuevos mapas** y que cada ronda se dispute en un mapa distinto elegido de manera aleatoria de entre el conjunto de mapas disponibles.
- **La implementación de un menú de ajustes funcional** donde poder cambiar valores de calidad de imagen, resolución, volumen de los efectos de sonidos y la música, asignación de controles, etc.

Bibliografía

[1] DEV, Libro Blanco del Desarrollo Español de Videojuegos 2023, 2023.

[2] RyteWiki, «Modelo en Espiral,» RiteWiki, 2024. [En línea]. Available: https://es.ryte.com/wiki/Modelo_en_Espiral.

[3] J. Roche, «¿Qué es el desarrollo en Espiral?,» Deloitte, 2024. [En línea]. Available: https://www2.deloitte.com/es/es/pages/technology/articles/que-es-el-desarrollo-en-espiral.html.

[4] J. Garzas, «Javiergarzas,» 2024. [En línea]. Available: https://www.javiergarzas.com.

[5] L. Gilibets, «Qué es la metodología Kanban y cómo utilizarla,» iebschool, 12 Enero 2023. [En línea]. Available: https://www.iebschool.com/blog/metodologia-kanban-agile-scrum/.

[6] U. Engine, «Unreal Engine Games,» 2024. [En línea]. Available: https://www.unrealengine.com/es-ES/uses/games.

[7] Unity, «Unity Games,» 2024. [En línea]. Available: https://unity.com/es/games.

[8] J. G. Gomilla, «¿Qué es mejor Unity o Unreal Engine?,» Frogamesformacion, 2024. [En línea]. Available: https://cursos.frogamesformacion.com/pages/blog/que-es-mejor-unity-o-unreal-engine.

[9] A. Higuera, «Del puñado de pixeles a confundir la realidad con la ficcion asi han mejorado los graficos de los videojuegos,» 20minutos, 30 Junio 2022. [En línea]. Available: https://www.20minutos.es/tecnologia/actualidad/del-punado-de-pixeles-a-confundir-la-realidad-con-la-ficcion-asi-han-mejorado-los-graficos-de-los-videojuegos-5023308/.

[10] F. d. d. Barcelona, «Historia de los videojuegos,» Retro Informática, 2024. [En línea]. Available: https://www.fib.upc.edu/retro-informatica/historia/videojocs.html.

[11] M. Terreno, «League of Legends: esport y fenómeno cultural,» infobae, 14 Julio 2023. [En línea]. Available: https://www.infobae.com/malditos-nerds/2023/07/14/league-of-legends-esport-y-fenomeno-cultural/.

[12] E. Games, «Geometría virtualizada Nanite,» Epic Games, 2024. [En línea]. Available: https://dev.epicgames.com/documentation/es-es/unreal-engine/nanite-virtualized-geometry-in-unreal-engine.

[13] E. Games, «Iluminación global y reflejos de Lumen,» Epic Games, 2024. [En línea]. Available: https://dev.epicgames.com/documentation/es-es/unreal-engine/lumen-global-illumination-and-reflections-in-unreal-engine.

[14] U. Engine, «Recommended Asset Naming Conventions [Documentación],» 2024. [En línea]. Available: https://docs.unrealengine.com/4.27/en-US/ProductionPipelines/AssetNaming/.

[15] T. Looman, «Unreal Gameplay Framework Guide for C++ [Documentación],» 19 Septiembre 2023. [En línea]. Available: https://www.tomlooman.com/unreal-engine-gameplay-framework/.

[16] R. Laley, «How to... Inline Images in Text [Video],» 31 Agosto 2021. [En línea]. Available: https://youtu.be/mTHjWAH2KkQ?si=w5OpXzPZsm6OqkNH.

[17] Tranek, «GASDocumentation [Repositorio de GitHub],» 2023. [En línea]. Available: https://github.com/tranek/GASDocumentation.

[18] V. Cantão, «A Gameplay Framework with GAS based on Risk of Rain 2 [Documentación],» 29 Enero 2023. [En línea]. Available: https://www.vitorcantao.com/post/gas-gameplay-framework/.

[19] T. G. D. Cave, «Gameplay Ability System Unreal Engine [Lista de Reproducción],» 16 Diciembre 2023. [En línea]. Available: https://www.youtube.com/playlist?list=PLoReGgpfex3woa35rnoXRyF9N3_p7QVQ2.

[20] Chankulovski, «Press "E" to Interact | No Casting | Unreal Engine 5 Tutorial [Video],» 26 Mayo 2024. [En línea]. Available: https://youtu.be/UVE6vnpjMwA?si=9qMnFuKgjsp_6JEG.

[21] S. Fansi, «How to make TIMER in unreal engine 4 and 5 [Video],» 1 Diciembre 2021. [En línea]. Available: https://youtu.be/rnmfKnN7bhA?si=b_aXyNFPPgxsqviM.

[22] T. G. D. Cave, «The Common UI Plugin is AMAZING! This is How it Works [Video],» 26 Julio 2023. [En línea]. Available: https://youtu.be/NJq_eLIX9G4?si=35UkcTiTErBWVVfJ.

[23] UntitledProyectX, «UNREAL ENGINE 5 | How to make ANIMATED widget button easy [Video],» 17 Agosto 2023. [En línea]. Available: https://youtu.be/pTTHen7vhJ8?si=R8hkXXuhemdpB0Gx.

[24] hawaiifilmschool, «Super Basic Intro to the Common UI: Common Activatable Widget [Video],» 14 Abril 2023. [En línea]. Available: https://youtu.be/IM8JvmxDe9M?si=GUDHA3LFvBWXQnY_.

[25] T. G. D. Cave, «How to Make Circular Progress bars in Unreal Engine (Round HP bars) [Video],» 6 Septiembre 2023. [En línea]. Available: https://youtu.be/QH8uV0WvfCw?si=riJl381ERlHlmQmm.

[26] NiceShadow, «Round / Radial Progress Bar - Unreal Engine 5 Tutorial [UE5] [Video],» 12 Junio 2022. [En línea]. Available: https://youtu.be/BgOAbAdi8f0?si=XxVwC6rEfbCLkGnG.

[27] D3kryption, «Unreal Engine 5 - UMG / UI - Circle Mask in 2~ minutes | D3kryption [Video],» 26 Abril 2022. [En línea]. Available: https://youtu.be/SHbcTe_lec0?si=joqTnOCK7ZZSrXwT.

[28] UnrealGaimeDev, «[Tutorial Series] Ability/Skill Tree System [Lista de Reproducción],» 9 Marzo 2017. [En línea]. Available: https://www.youtube.com/playlist?list=PLmKKTERcjTPLIW5fPHRaFcl1hfRb0okbz.

[29] MizzoFrizzo, «How To Set Up A Floating Damage Indicator - Unreal Engine 5 Tutorial [Video],» 24 Enero 2024. [En línea]. Available: https://youtu.be/qBDnJTb9Clo?si=tGuOmidwyM_3CygU.

[30] G. Games, «How to Make a Game Start Countdown in Unreal Engine 5 [Video],» 12 Septiembre 2023. [En línea]. Available: https://youtu.be/Qaf-vmTdQHc?si=rX9ljuQ6HDFku903.

[31] U. A. W. Alireza, «Switching Objects in Unreal Engine 5 Using Blueprints [Video],» 4 Agosto 2022. [En línea]. Available: https://youtu.be/-IzDYXhJg4M?si=nlH6UELIbxtOIWjZ.

[32] GomVoDeveloper, «Unreal Engine 4: Mover MENÚ Widgets con GAMEPAD/TECLADO [Video],» 3 Mayo 2022. [En línea]. Available: https://youtu.be/68vwEzX-ltE?si=H_LJ58XaS-42BkaO.

[33] G. GameDev, «COMO CREAR/HACER UN MENU DE PAUSA EN UNREAL ENGINE 5 UE5 [Video],» 5 Marzo 2023. [En línea]. Available: https://youtu.be/MR9n3tvQOQU?si=5wmUlPws7AWAik4f.

[34] C. C. G. Studios, «Unreal Engine 5 Tutorial - Directional Dash (Request) #unreal #unrealengine #unrealengine5 [Video],» 28 Mayo 2022. [En línea]. Available: https://youtu.be/K8FqqOjdCaw?si=VILXGqhx4Q-ieVxe.

[35] LeafBranchGames, «Translucent Overlay Material - Unreal Engine 5 tutorial [Video],» 20 Diciembre 2022. [En línea]. Available: https://youtu.be/0wGZJzNR7Ww?si=Kuf_3tTCQYdur2nG.

[36] C. Coronado, «Unreal Engine 5 de 0 a DIOS [Curso en línea],» 2023. [En línea]. Available: https://www.udemy.com/course/unreal-engine-5-de-0-a-dios/.

[37] C. Coronado, «Desarrollo de juegos con Unreal Engine de 0 a profesional,» 2023. [En línea]. Available: https://www.udemy.com/course/desarrollo-de-juegos-con-unreal-engine-4-de-0-a-profesional/.

[38] C. Coronado, «Programar Blueprints en Unreal Engine de 0 a profesional,» 2023. [En línea]. Available: https://www.udemy.com/course/blueprints-unreal/.

[39] E. Games, «World Partition [Documentación],» 2024. [En línea]. Available: https://dev.epicgames.com/documentation/es-es/unreal-engine/world-partition-in-unreal-engine?application_version=5.2.

[40] E. Games, «One File Per Actor [Documentación],» 2024. [En línea]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/one-file-per-actor-in-unreal-engine?application_version=5.2.

[41] E. Games, «Gameplay Ability System [Documentación],» 2024. [En línea]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/gameplay-ability-system-for-unreal-engine?application_version=5.2.

[42] E. Games, «Common UI [Documentación],» 2024. [En línea]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/common-ui-plugin-for-advanced-user-interfaces-in-unreal-engine?application_version=5.0.

[43] U. Engine, «Scalability Reference [Documentación],» 2024. [En línea]. Available: https://docs.unrealengine.com/4.26/en-US/TestingAndOptimization/PerformanceAndProfiling/Scalability/ScalabilityReference/.

[44] Paragon, 2024. [En línea]. Available: https://paragon.netmarble.com/es/hero.

[45] Predecessorgame, 2024. [En línea]. Available: https://www.predecessorgame.com/heroes.

[46] U. E. Forums, 2024. [En línea]. Available: https://forums.unrealengine.com/.

[47] Reddit, 2024. [En línea]. Available: https://www.reddit.com.

[48] Stackoverflow, 2024. [En línea]. Available: https://es.stackoverflow.com.

[49] F. Angel, «UE4 Gameplay Ability System Practical Example [Lista de Reproducción],» 16 Marzo 2022. [En línea]. Available: https://www.youtube.com/playlist?list=PLeEXbS_TaXrAbfoPYSNROqe1fDQfQHTfo.

[50] M. Aspland, «Character Select From Main Menu - Unreal Engine 4 Tutorial [Video],» 26 Marzo 2021. [En línea]. Available: https://youtu.be/ziAkrTv59TA?si=diZ8bOMzYHld7bJR.

[51] M. Aspland, «Random Loot Drops When Enemy Dies - Unreal Engine Tutorial [Video],» 9 Julio 2021. [En línea]. Available: https://youtu.be/oL_KGOAYIk4?si=eTNxpApmGv3viY9Z.

[52] T. G. D. Cave, «Changing UI Icons Based in Input Controller in Unreal Engine [Video],» 2 Agosto 2023. [En línea]. Available: https://youtu.be/0LTFdHq14jw?si=or-rr_pUjY-rZr5N.

[53] CodeLikeMe, «Unreal Adventure Tutorial Series [Lista de Reproducción],» 28 Junio 2021. [En línea]. Available: https://www.youtube.com/playlist?list=PLNTm9yU0zou5mxzKt2DUIE1QJSkGo7tvc.

[54] F. Dev, «How To Dash In 2 Minutes | Unreal Engine 5 Tutorial [Video],» 9 Junio 2023. [En línea]. Available: https://youtu.be/rF_oIaDvtJ4?si=3urwz8NPDYtD-xl8.

[55] O. Developers, «UNREAL ENGINE 5 Empezar Con La Interfaz De Usuario (UI I)Tutoriales "Español" [Video],» 13 Abril 2022. [En línea]. Available: https://youtu.be/b6rYk76_ZZA?si=U5LW2V1Irv_X9mMu.

[56] D. Enabled, «UE4 Multiplayer [Lista de Reproducción],» 16 Noviembre 2020. [En línea]. Available: https://www.youtube.com/playlist?list=PL9z3tc0RL6Z7rISWe-r5DyPopRPJWaITF.

[57] Foxcoder, «Unreal UI Tutorial Part 1, Demo and Preview for User Widget, Common UI, Material, and C++ code. [Video],» 9 Junio 2023. [En línea]. Available: https://youtu.be/6X3OD5Bi8pU?si=ph6kzs7IvU5-rY8V.

[58] GameDevRaw, «Make a Multiplayer Game in Unreal Engine 5 - Character Selection - Unreal Beginner Tutorial # 16 [Video],» 1 Julio 2022. [En línea]. Available: https://youtu.be/9f-feH2gP-o?si=CEV1XKLVX9vKoHFw.

[59] G. Games, «Unreal Engine 5 RPG Tutorial Series [Lista de Reproducción],» 20 Febrero 2023. [En línea]. Available: https://www.youtube.com/playlist?list=PLiSlOaRBfgkcPAhYpGps16PT_9f28amXi.

[60] G. Games, «How to Make a Simple Damage Indicator in Unreal Engine 5,» 12 Enero 2023. [En línea]. Available: https://youtu.be/6-OQiKK6W1I?si=OooNPyewOU-1fCbP.

[61] R. L. Games, «Unreal Engine 4 Local Multiplayer Tutorial Part 1 - Spawning Players [Video],» 18 Agosto 2017. [En línea]. Available: https://youtu.be/PpGdOCvWk74?si=tS3Hu3kkoYCmyiy_.

[62] R. L. Games, «Unreal Engine 4 Local Multiplayer Tutorial Part 2 - Top Down Tracking Camera [Video],» 20 Agosto 2017. [En línea]. Available: https://youtu.be/FJKk92PV8wU?si=KTBgVVpR4uEelCby.

[63] Haez, «Health Pickup - Unreal Engine 5 Tutorial [Video],» 20 Diciembre 2023. [En línea]. Available: https://youtu.be/uWzArAlWQVg?si=PdvEnqkYIxPLjphG.

[64] U. E. JOURNEY, «UE5 Fighting Game Tutorial: Local Multiplayer Game With 2 Gamepads | TrueFGE & Unreal Engine 5 [Video],» 5 Septiembre 2023. [En línea]. Available: https://youtu.be/oonRBOBJcIo?si=5zdA1TtuUmWEv7fP.

[65] LexoDevs, «SISTEMA de GUARDADO y PERSISTENCIA de NIVEL - APRENDIENDO BIEN - UNREAL ENGINE 5.3 [Video],» 15 Febrero 2024. [En línea]. Available: https://youtu.be/wZRS9f5ZPlk?si=rce9hQvLxvzj6CjK.

[66] LuisCanary, «Start Menu en Unreal Engine 5/Main Menu/ Facil y Sencillo para 3D y 2D [Video],» 26 Junio 2023. [En línea]. Available: https://youtu.be/YNkXJdr7LPA?si=CKuPz5sMt34BKAqf.

[67] [En línea]. Available: https://youtu.be/K8FqqOjdCaw?si=_ZSYtUOc-I3YkzLu.

# Ludografía

Bandai Namco Games. (2015). ***Tekken 7*** [Videojuego]. Bandai Namco Games. PC, Consolas.

Blizzard Entertainment. (2004). ***World of Warcraft*** [Videojuego]. Blizzard Entertainment. PC.

Epic Games. (2016). ***Paragon*** [Videojuego]. Epic Games. PC, Consolas.

HABBY PTE. LTD. (2019). ***Archero*** [Videojuego]. HABBY PTE. LTD. Móvil.

Hi-Rez Studios. (2014). ***Smite*** [Videojuego]. Hi-Rez Studios. PC, Consolas.

NetherRealm Studios. (2019). ***Mortal Kombat 11*** [Videojuego]. Warner Bros. Interactive Entertainment. PC, Consolas.

OverPowered Team. (2021). ***Godstrike*** [Videojuego]. Freedom Games LLC. PC.

Riot Games. (2009). ***League of Legends*** [Videojuego]. Riot Games. PC.

Supercell. (2017). ***Brawl Stars*** [Videojuego]. Supercell. Móvil.

TiMi Studios. (2016). ***Arena of Valor*** [Videojuego]. Level Infinite. Móvil.

Valve Corporation. (2013). ***Dota 2*** [Videojuego]. Valve Corporation. PC.

Capcom. (2023). ***Street Fighter 6*** [Videojuego]. Capcom. PC, Consolas.

# Anexo I

# Controles y Diagramas

A continuación, se muestran los controles del videojuego adaptados a los mandos de diferentes plataformas. En el menú de controles es posible seleccionar el tipo de mando que se está utilizando, y según el seleccionado, se mostrará al usuario una de las siguientes imágenes explicando los controles. Además, se adaptarán todas las interfaces y menús al tipo de botones del mando seleccionado, utilizando los iconos del atlas que se encuentra en **Anexo II**.

También se puede observar seguidamente, el prototipo inicial de la interfaz *in-game* y los diagramas realizados para el flujo del videojuego y los comportamientos de la inteligencia artificial.

El flujo del videojuego muestra el recorrido por los diferentes menús y las conexiones entre ellos, un croquis inicial de diseño y desarrollo.

El comportamiento de la inteligencia artificial se ha desarrollado en forma de Árbol de Comportamiento y Environment Query System (EQS) de Unreal Engine, para su realización se utilizó este esquema a modo de guía.

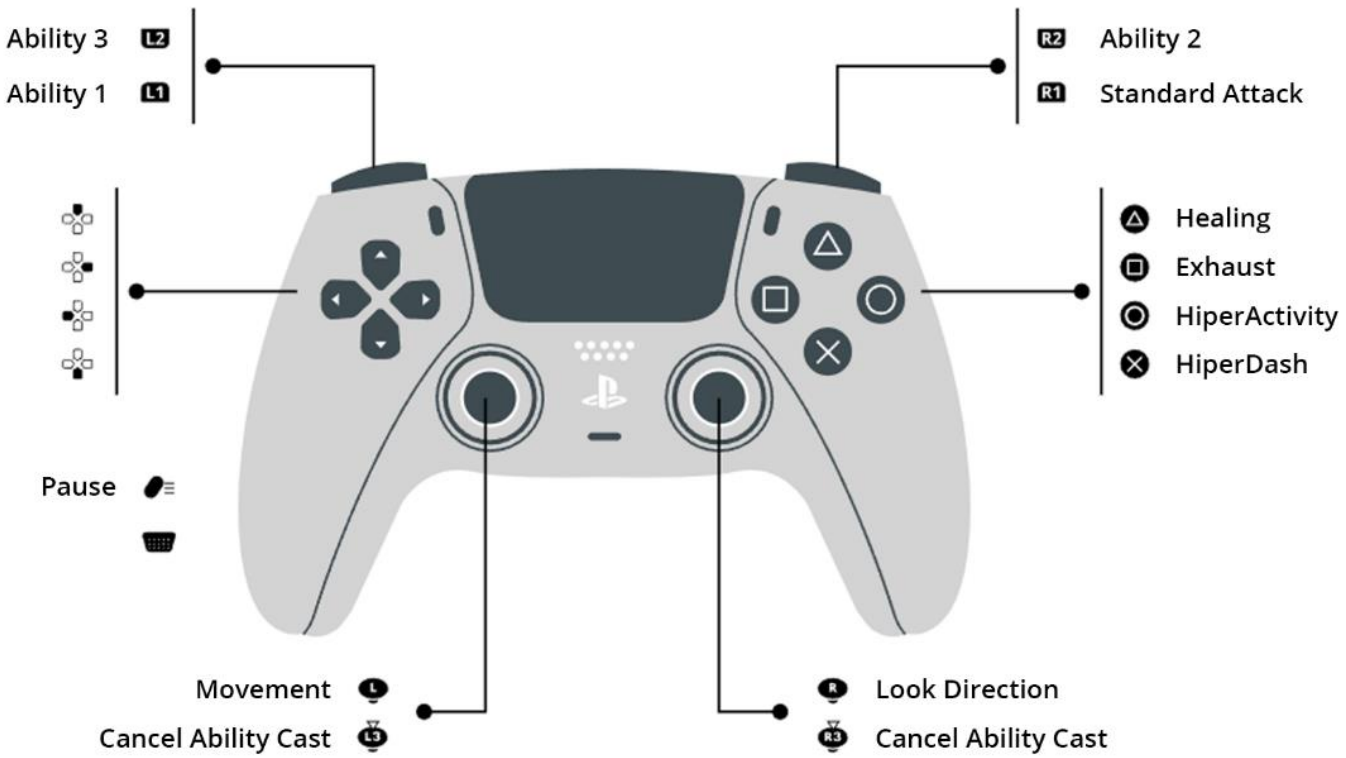



*Ilustración 55 – Controles en un mando de PlayStation.*

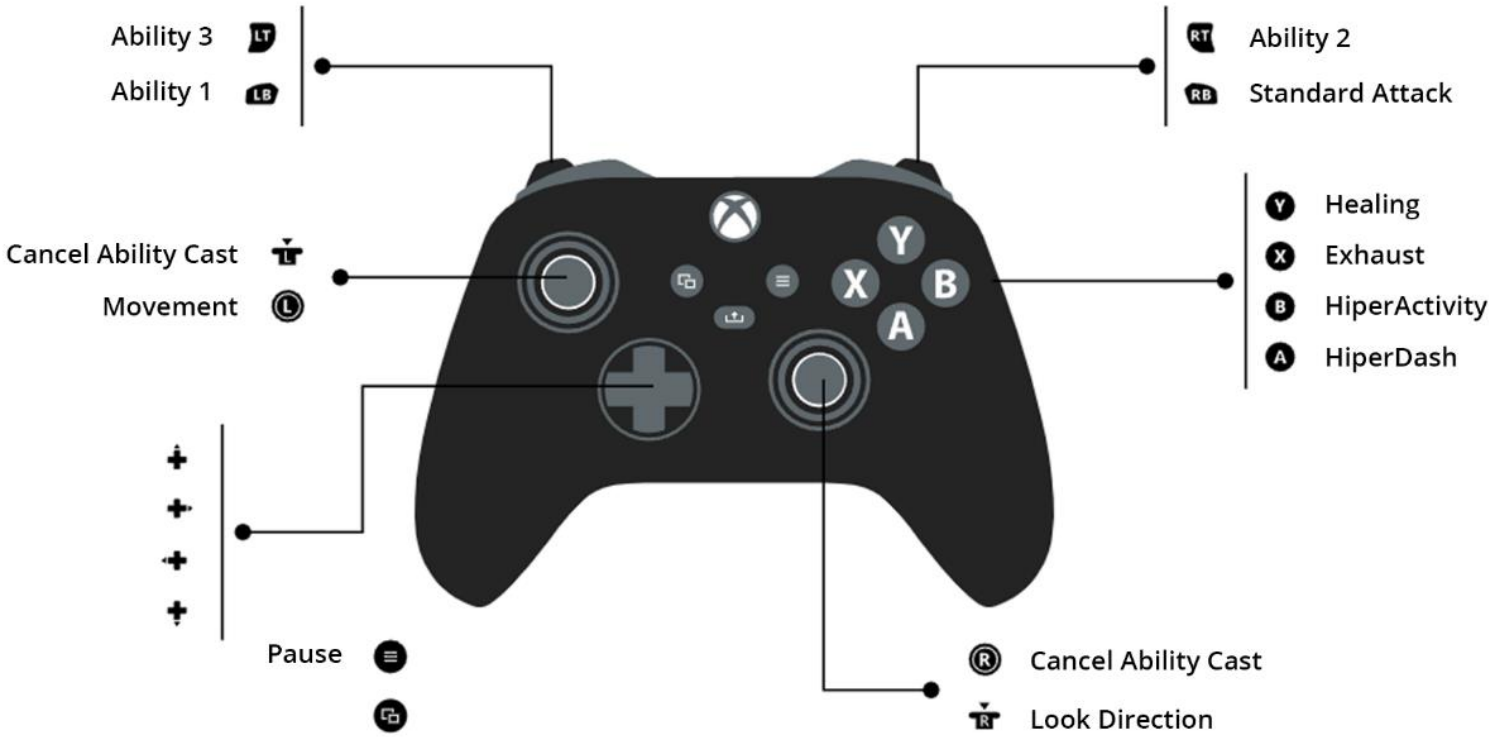



*Ilustración 56 – Controles en un mando de Xbox.*

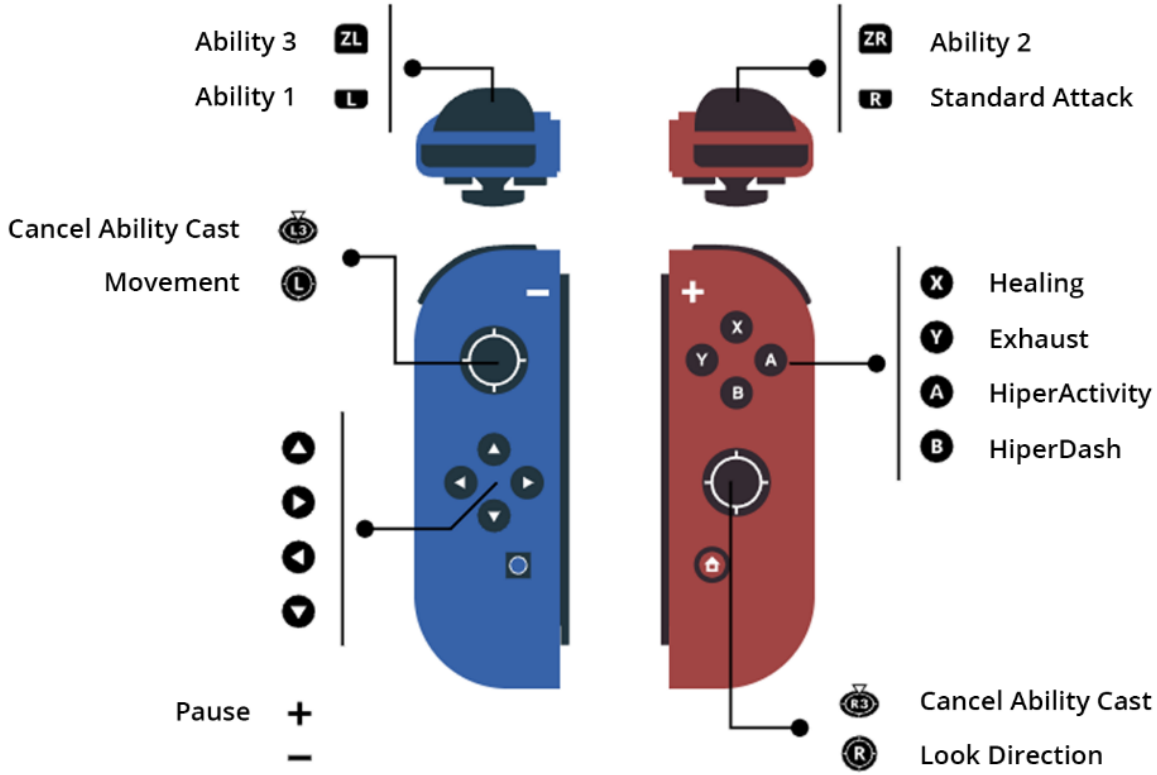



*Ilustración 57 – Controles en un mando de Switch.*

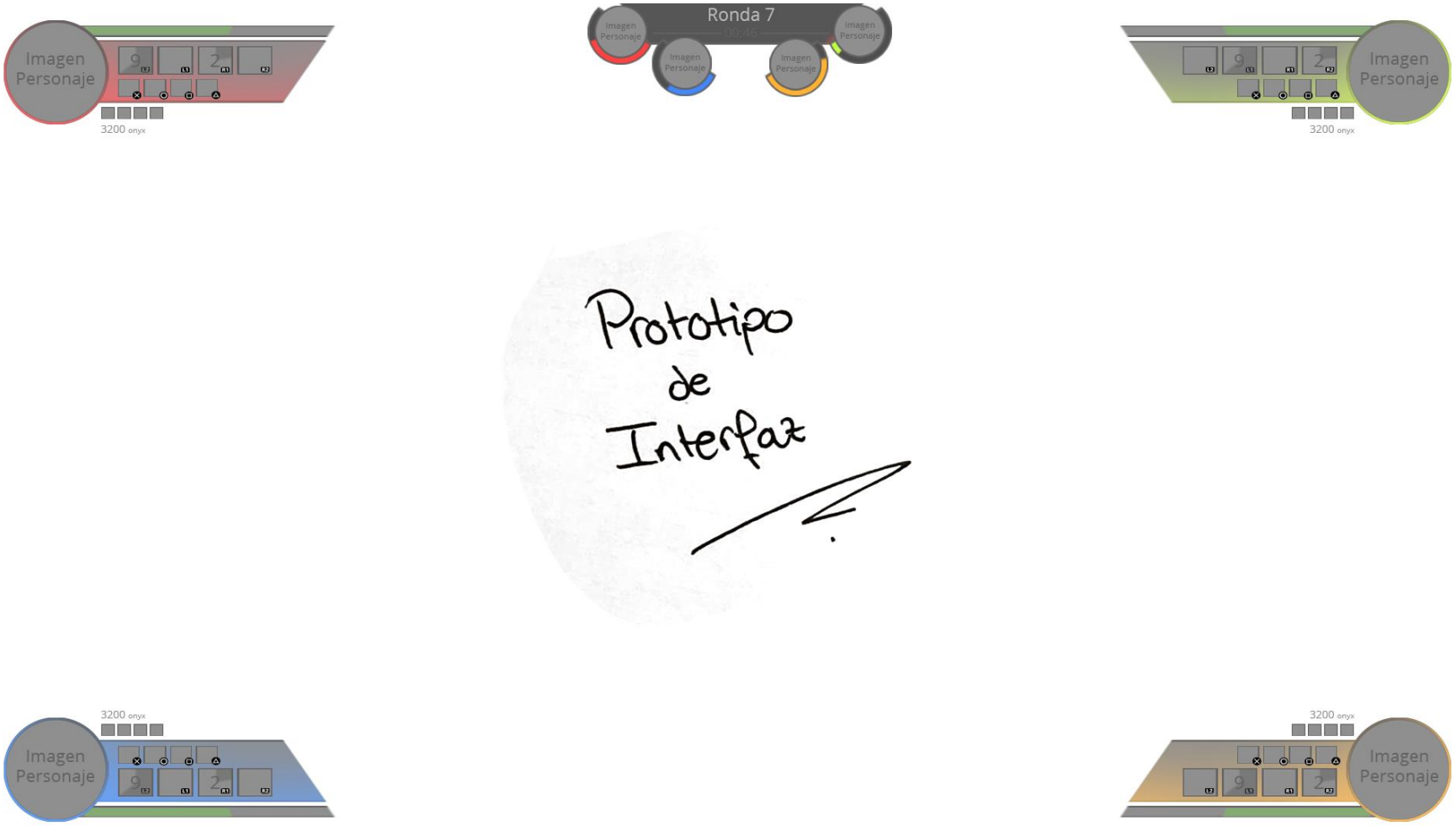



*Ilustración 58 – Prototipo inicial de la interfaz in-game.*

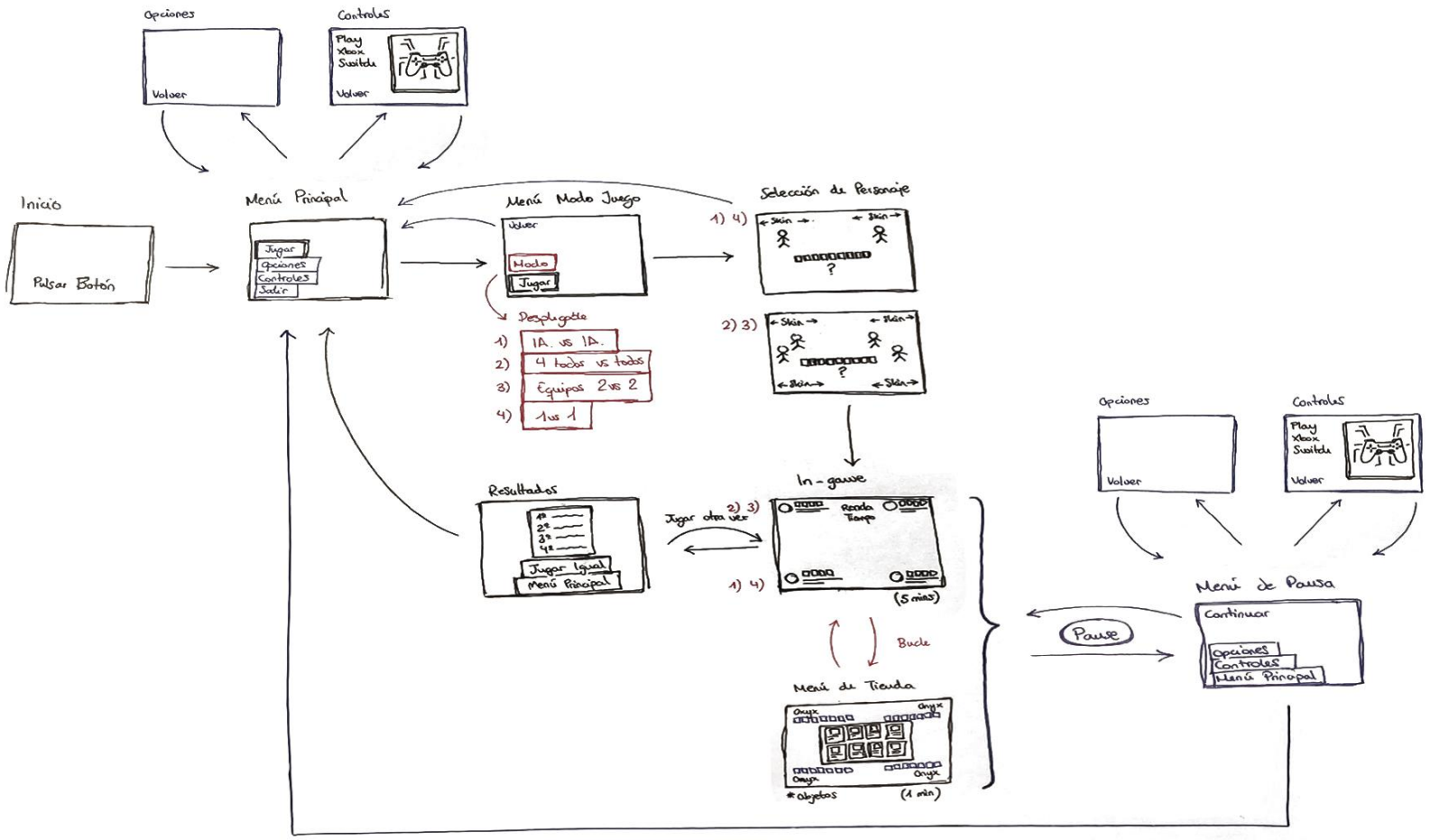



*Ilustración 59 – Diagrama de flujo del videojuego (Boceto).*

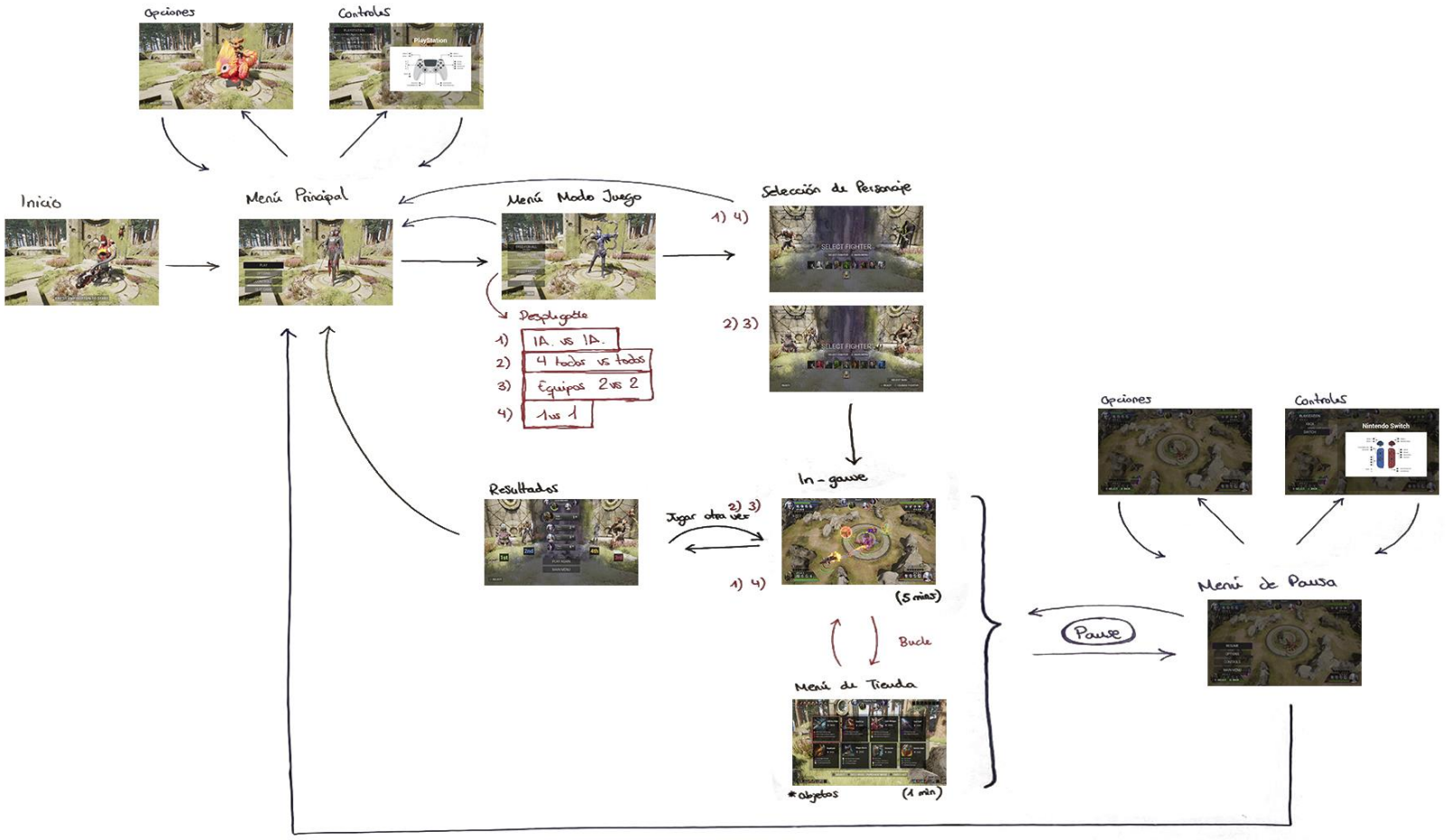



*Ilustración 60 – Diagrama de flujo del videojuego (Resultado).*

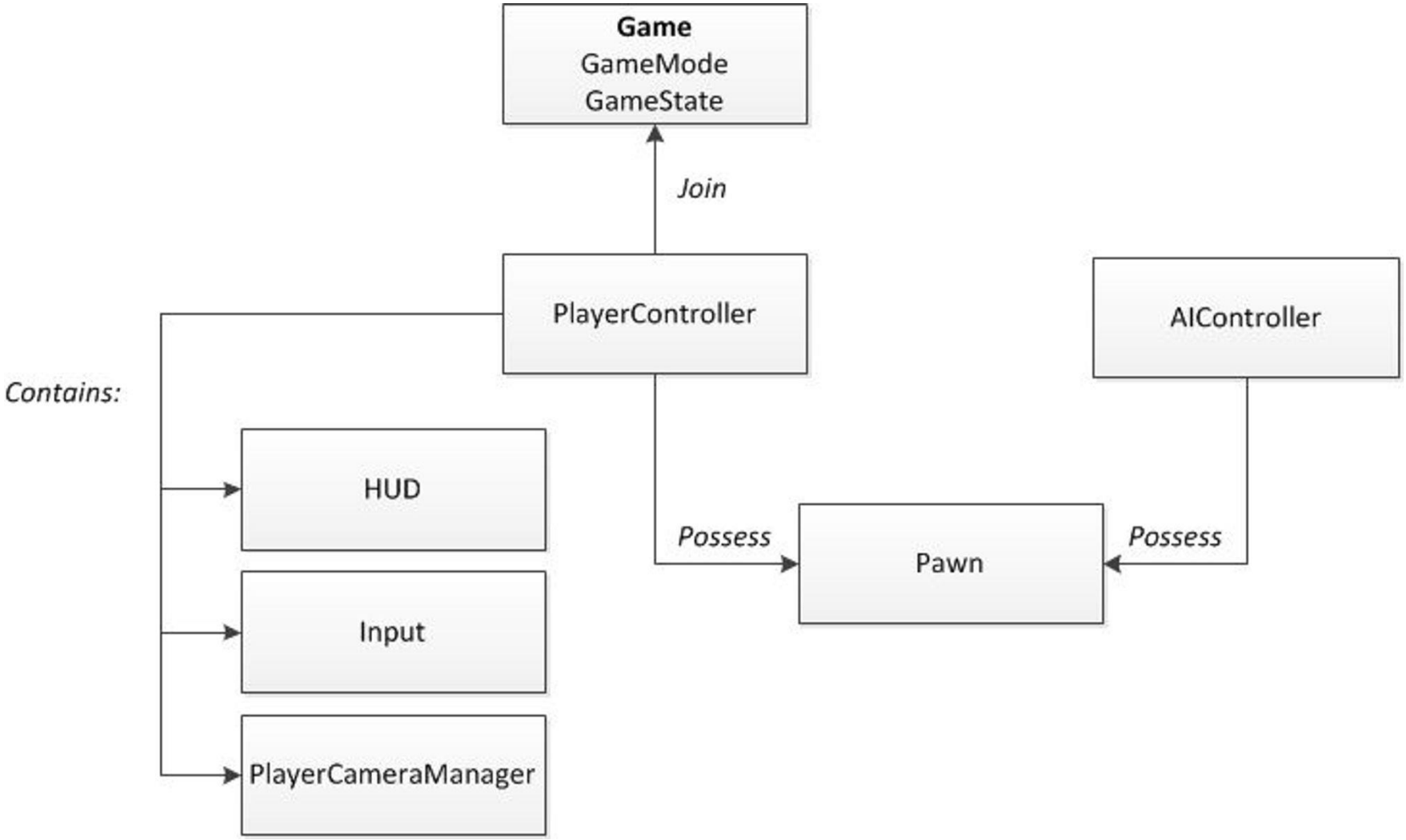



*Ilustración 61 – Diagrama de clases UML de la arquitectura base del juego en Unreal Engine.*

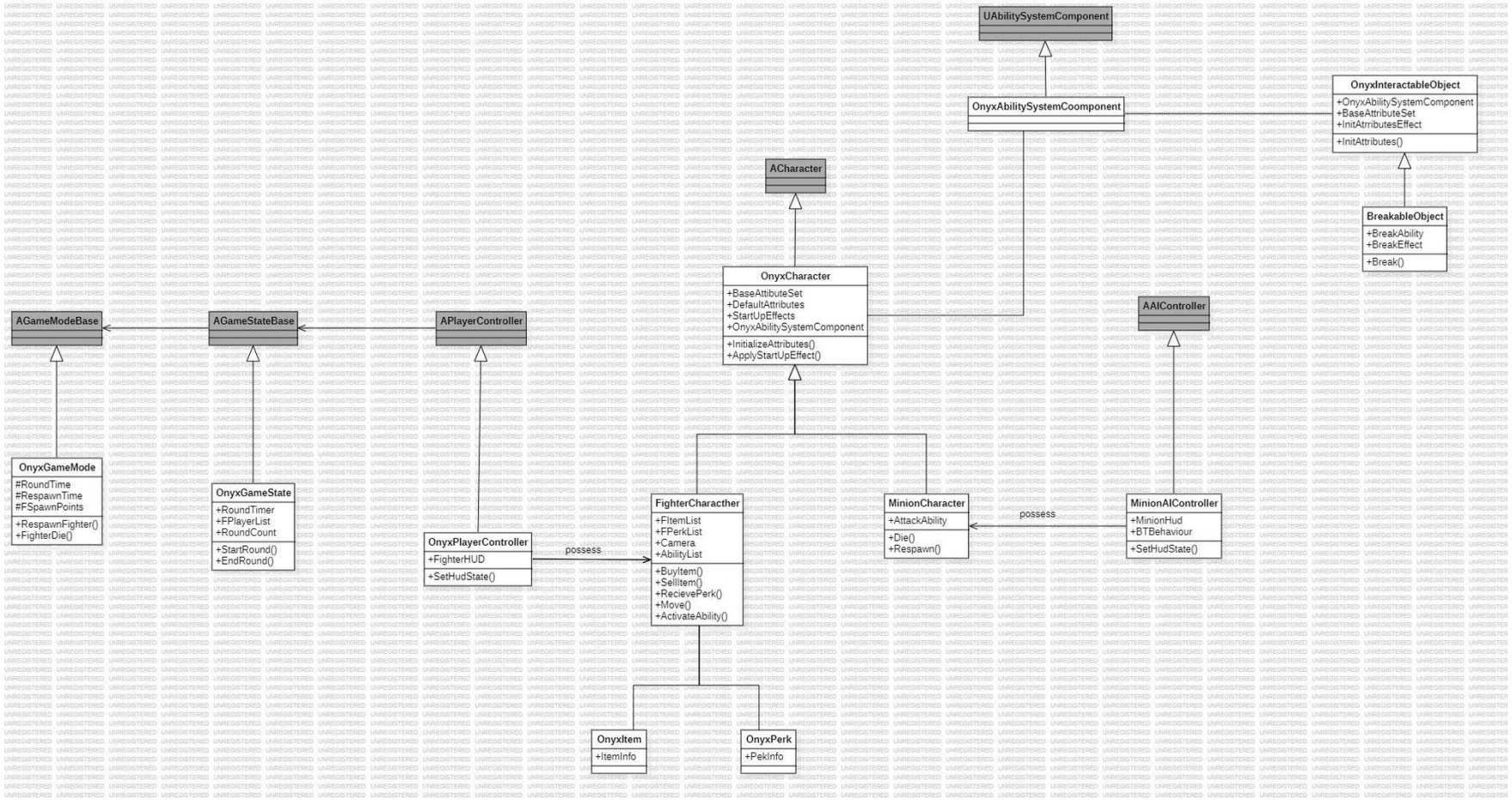



*Ilustración 62 – Diagrama de clases UML de la arquitectura del Gameplay Ability System (GAS).*

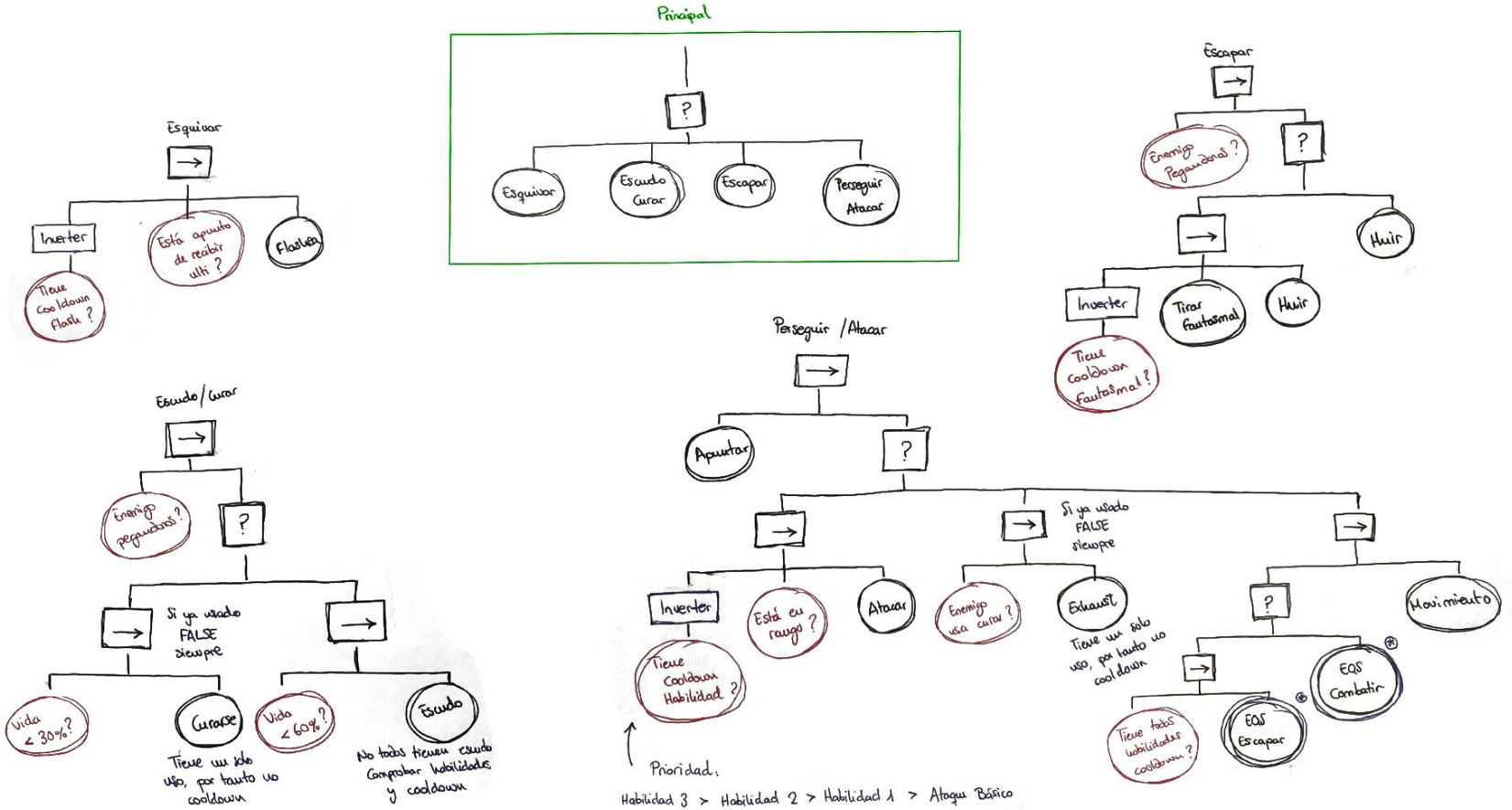



*Ilustración 63 – Esquema de comportamiento de la Inteligencia Artificial (IA).*

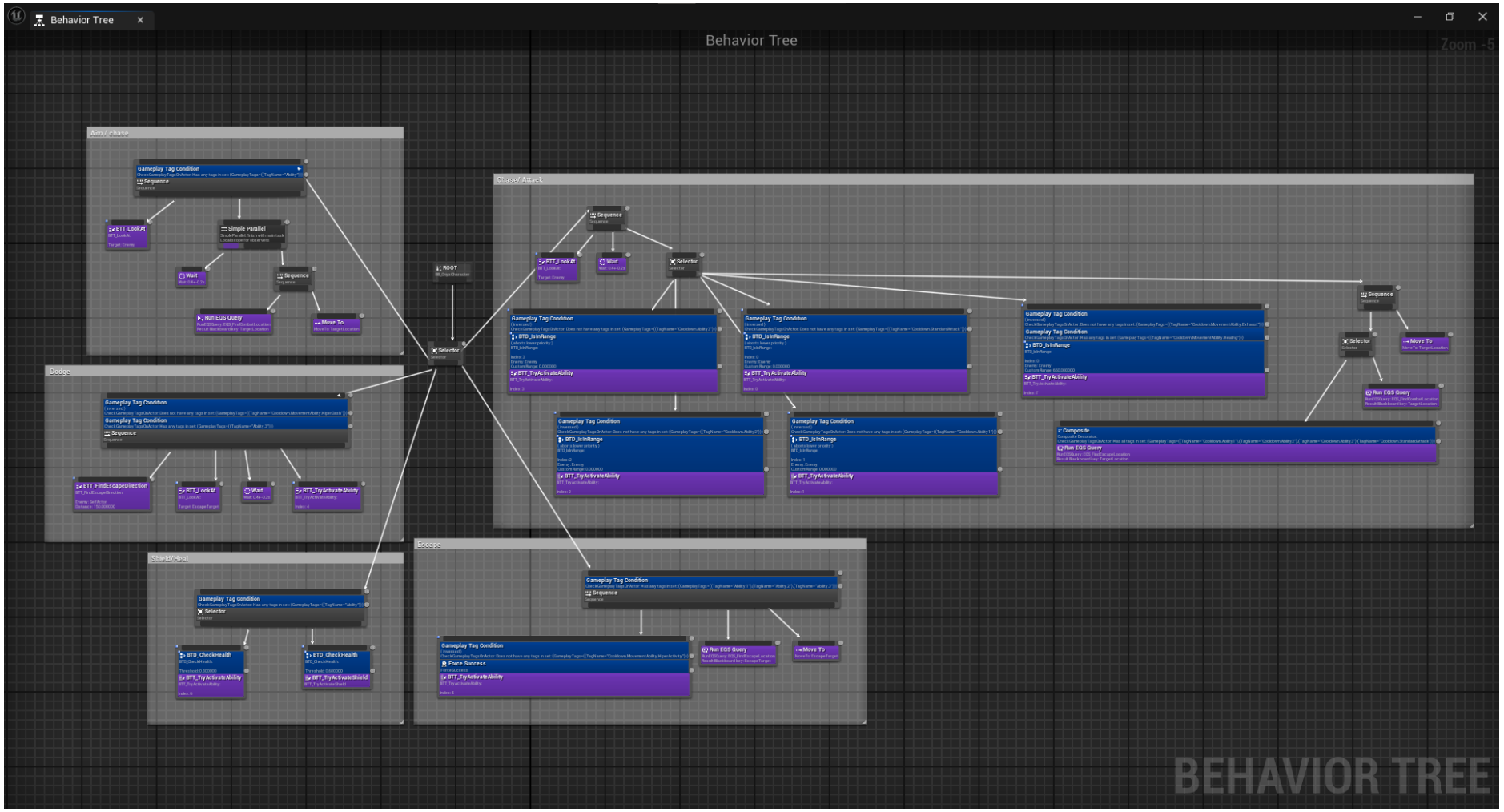



*Ilustración 64 – Behaviour Tree implementado en Unreal Engine.*

# Anexo II

# Galería de imágenes

A continuación, se adjuntan los atlas de recursos utilizados en la interfaz del videojuego. Se puede observar un atlas de iconos utilizado para los controles, y las características de los objetos de la tienda, también se adjunta un atlas para los botones de los menús, un conjunto de máscaras utilizadas para la creación de *Decals* para indicar los rangos de las habilidades de los personajes, y para la generación de barras de progreso usadas para la generación de las barras de vida, maná y escudo, también hay máscaras para realizar un recorte con forma circular de los iconos de los personajes. Finalmente se adjuntan también los elementos que conforman la interfaz del usuario, así como los iconos de personaje, habilidades, habilidades de movimiento, y objetos de la tienda.

Seguidamente pueden encontrarse capturas de pantalla *in-game* que muestran el resultado final de la interfaz de usuario y los menús del videojuego.

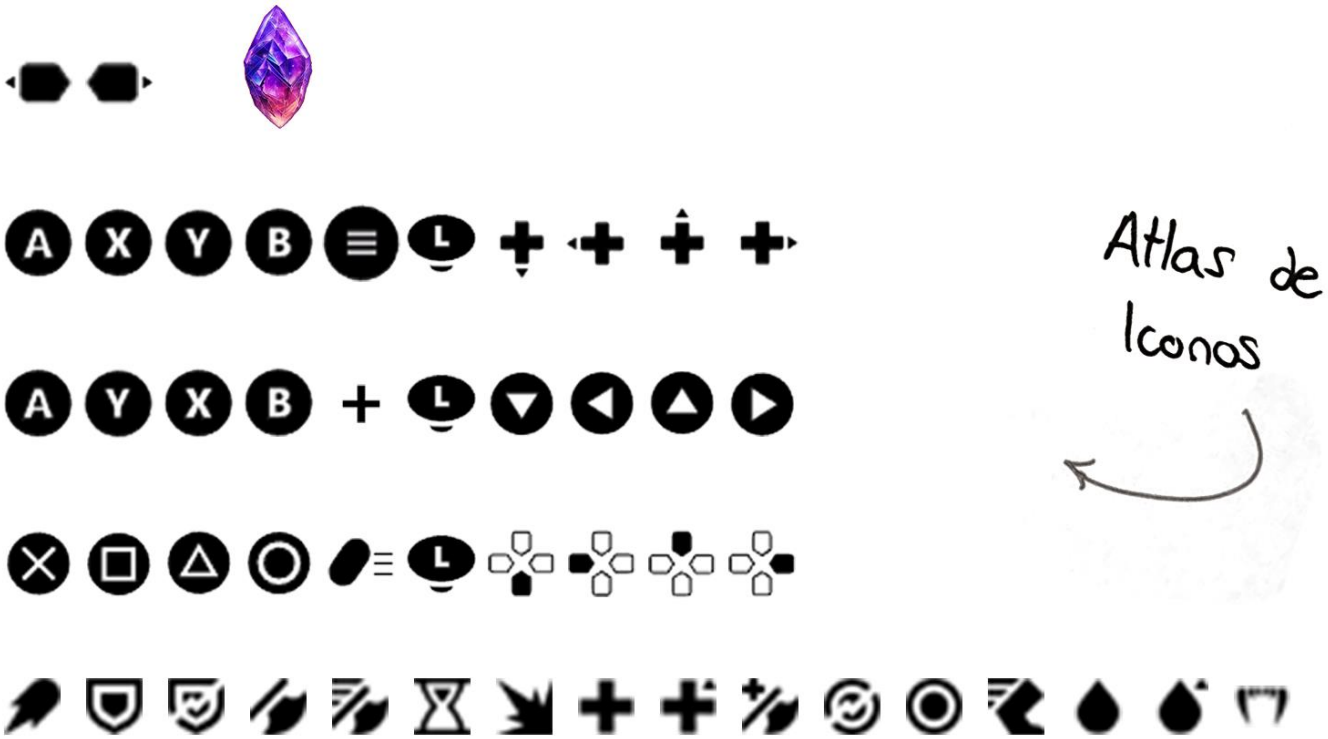



*Ilustración 65 – Atlas de iconos para los controles y las estadísticas de los objetos de la tienda.*

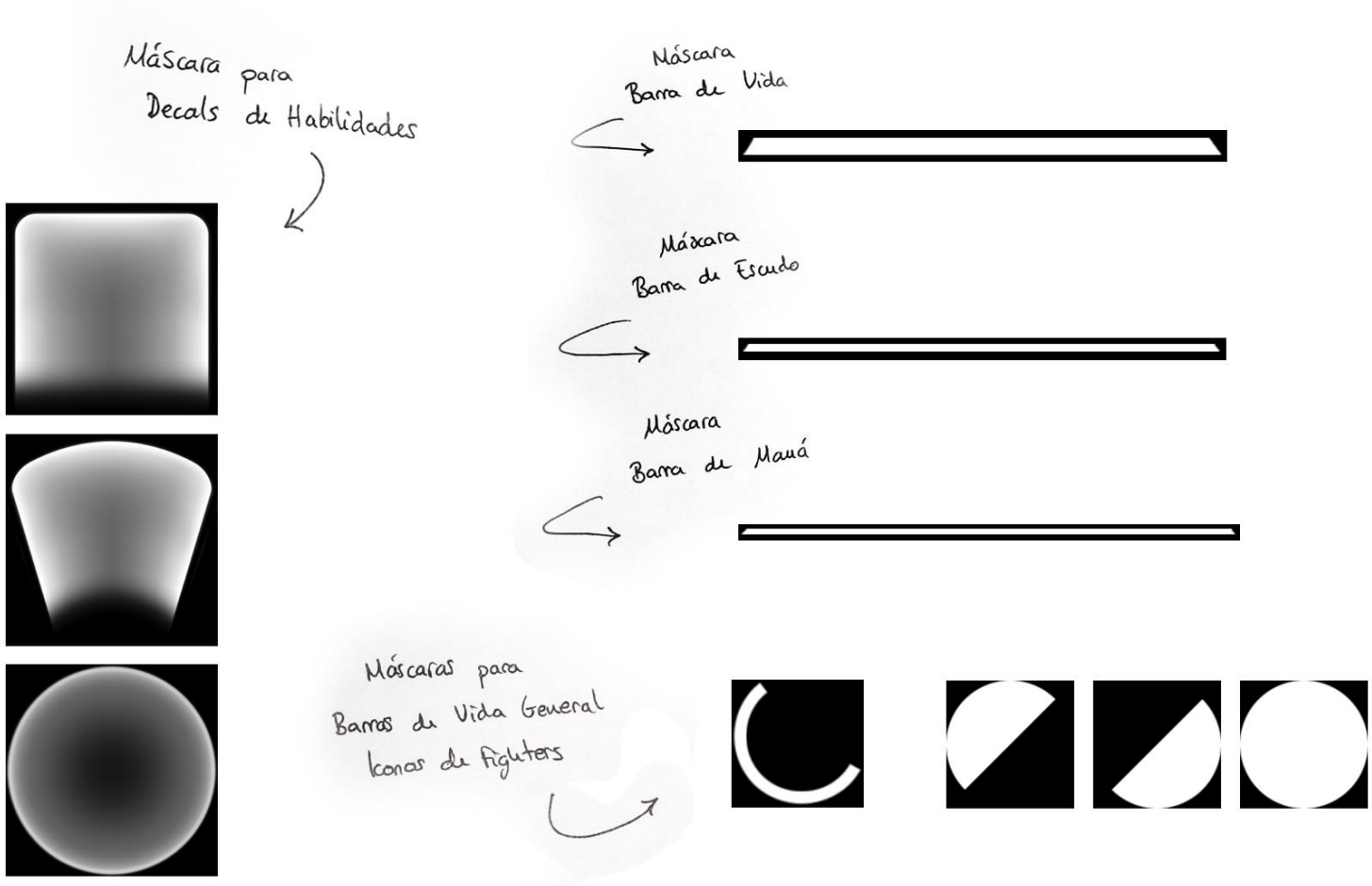



*Ilustración 66 – Máscaras de recorte para Decals de rango de habilidades y barras de vida, maná y escudo.*

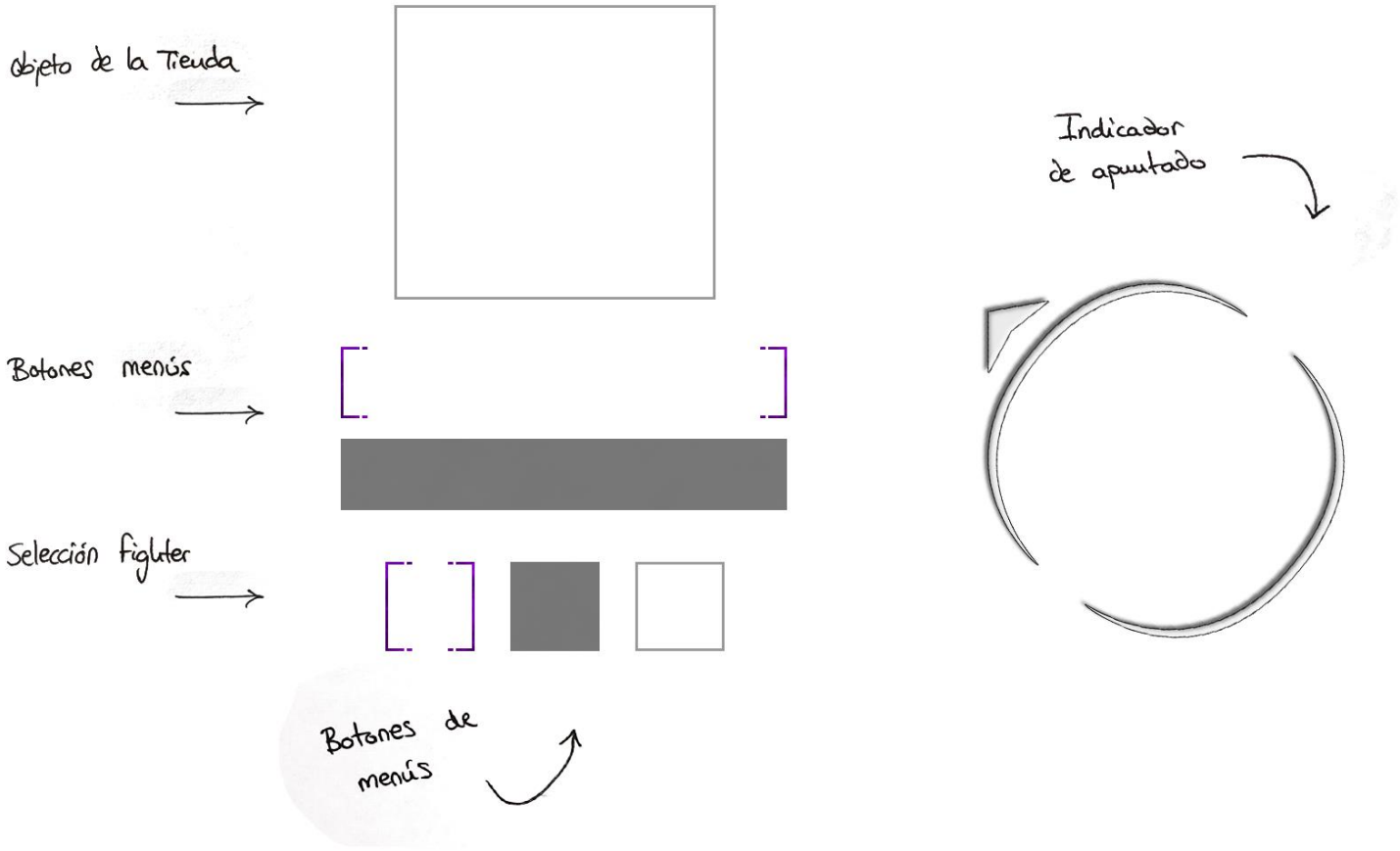



*Ilustración 67 – Altas de botones para los menús e indicador de apuntado de los personajes.*


*Ilustración 68 – Background del menú de selección de personaje (2 personas).*


*Ilustración 69 – Background del menú de selección de personaje (4 personas).*

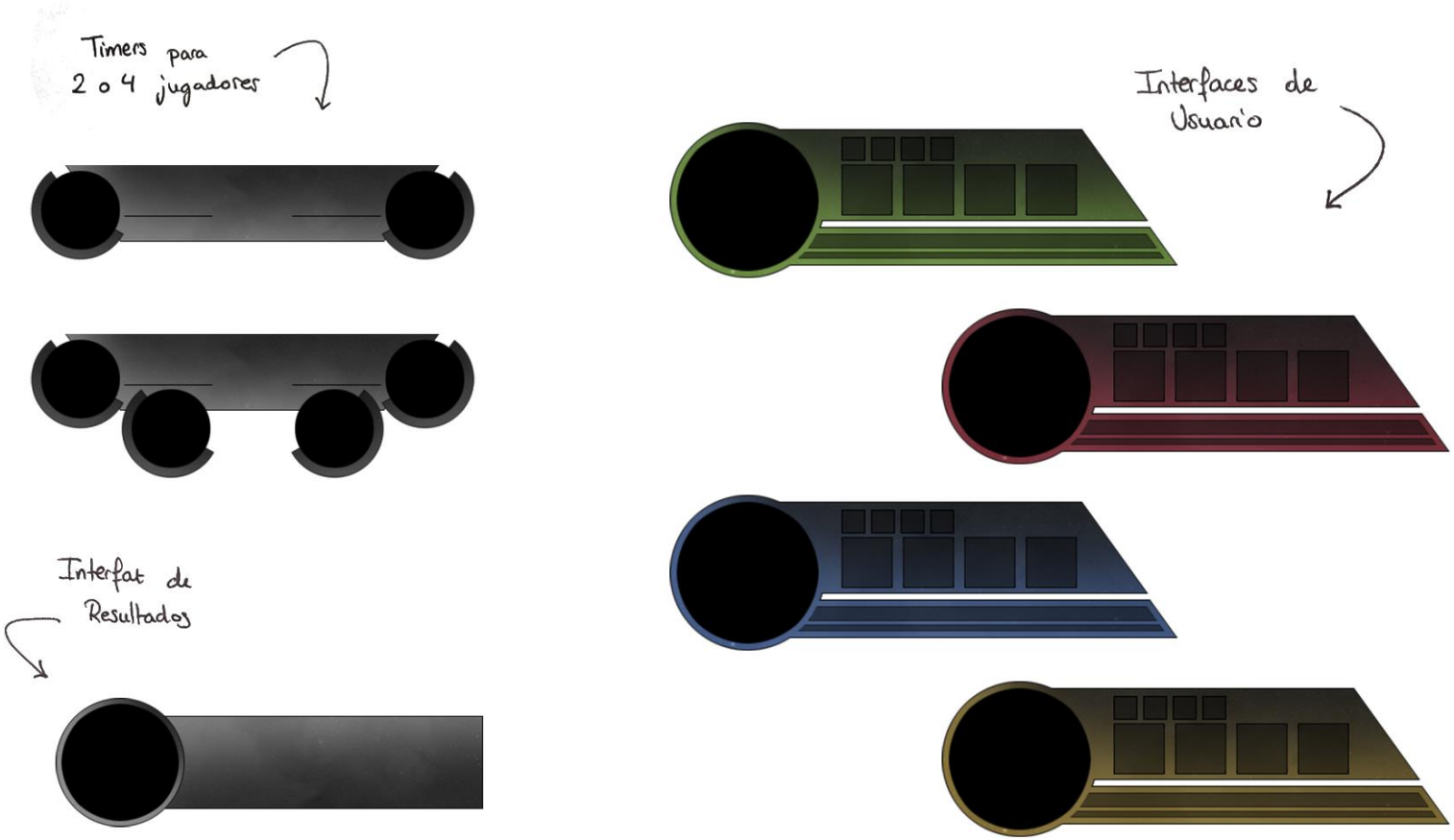



*Ilustración 70 – Componentes de la interfaz in-game.*

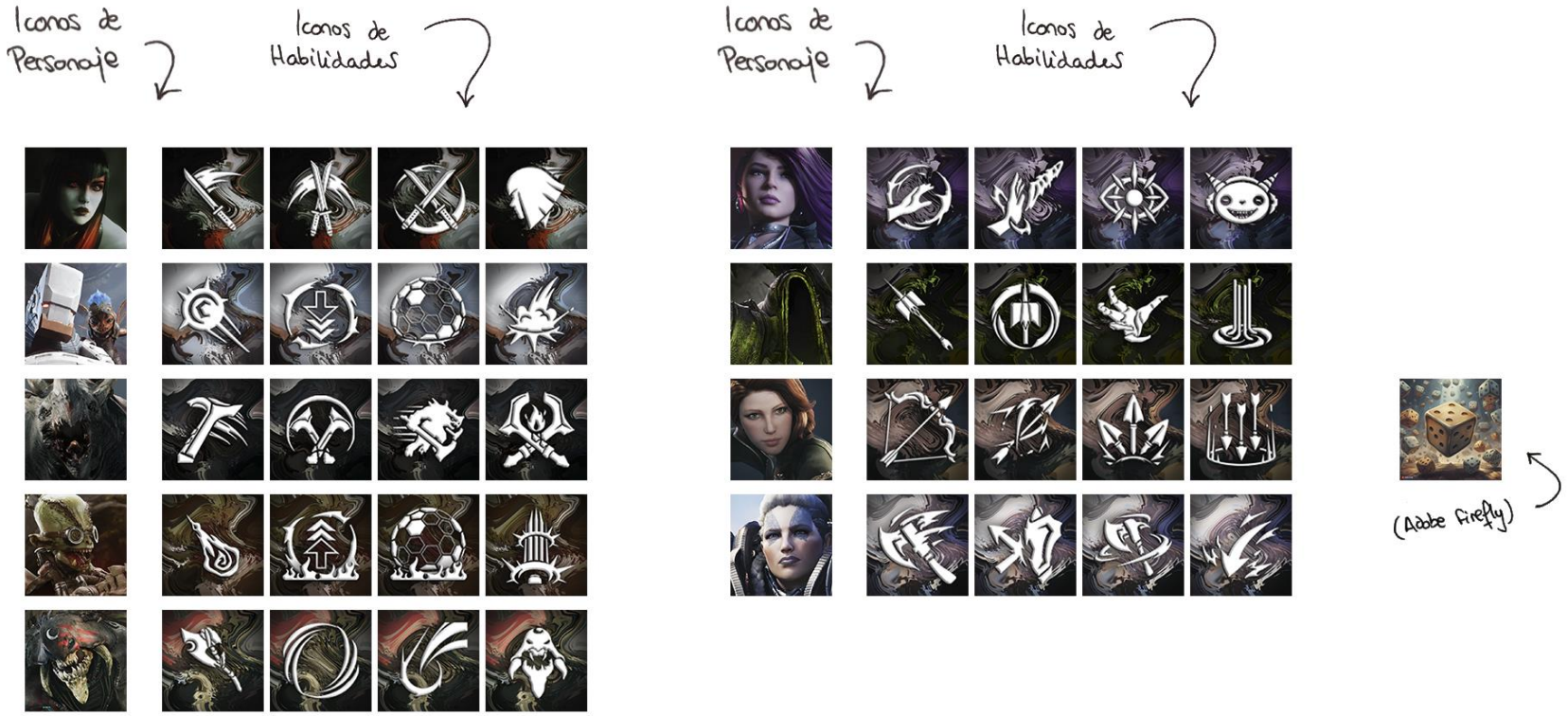



*Ilustración 71 – Iconos de personaje e iconos de sus correspondientes habilidades. (Icono de personaje random creado por la IA Adobe Firefly[12]).*

[12] https://firefly.adobe.com

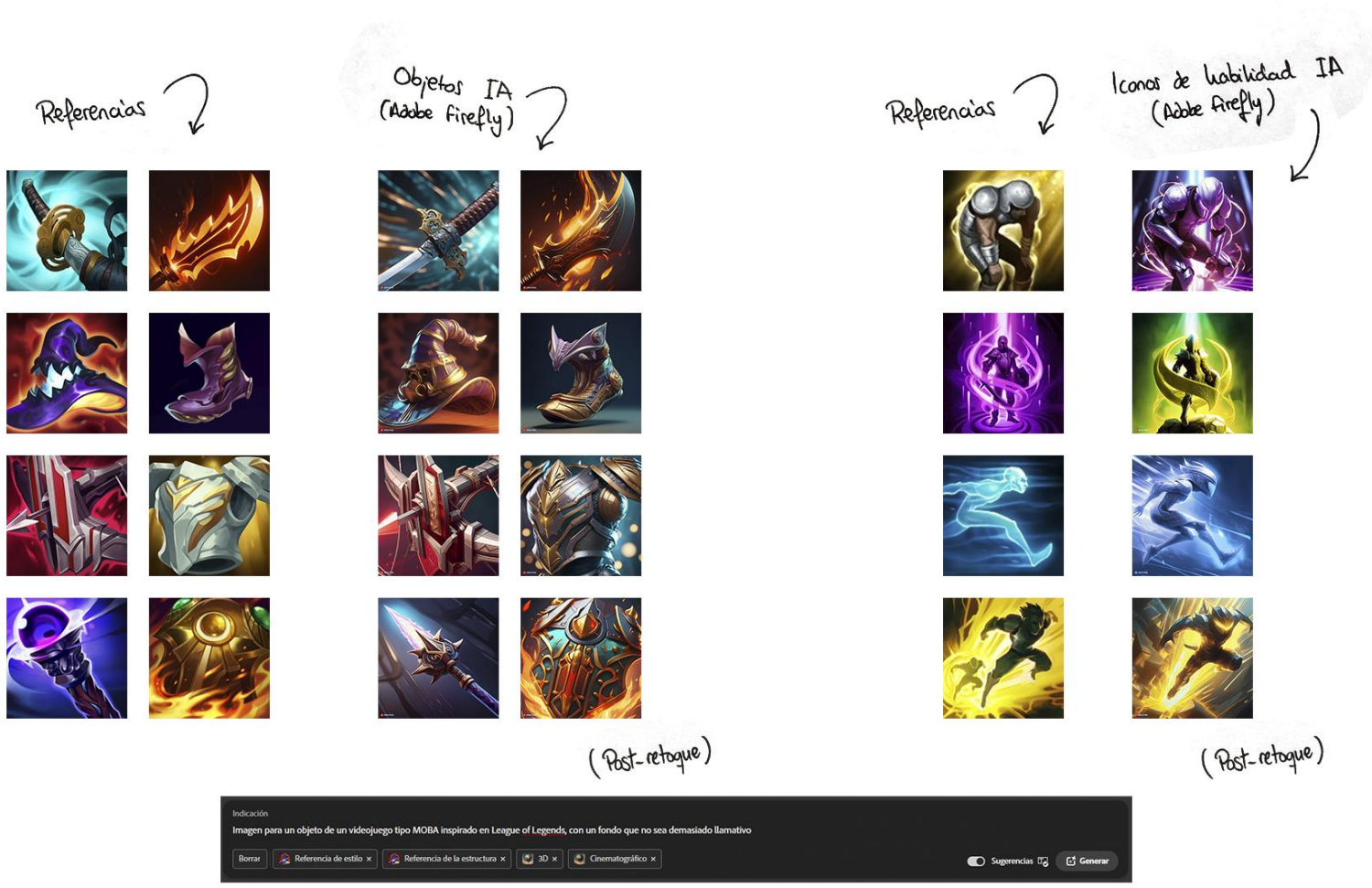



*Ilustración 72 – Referencias e iconos de objetos y habilidades de movimiento creados por la IA Adobe Firefly [13](se adjunta prompt).*

[13] https://firefly.adobe.com

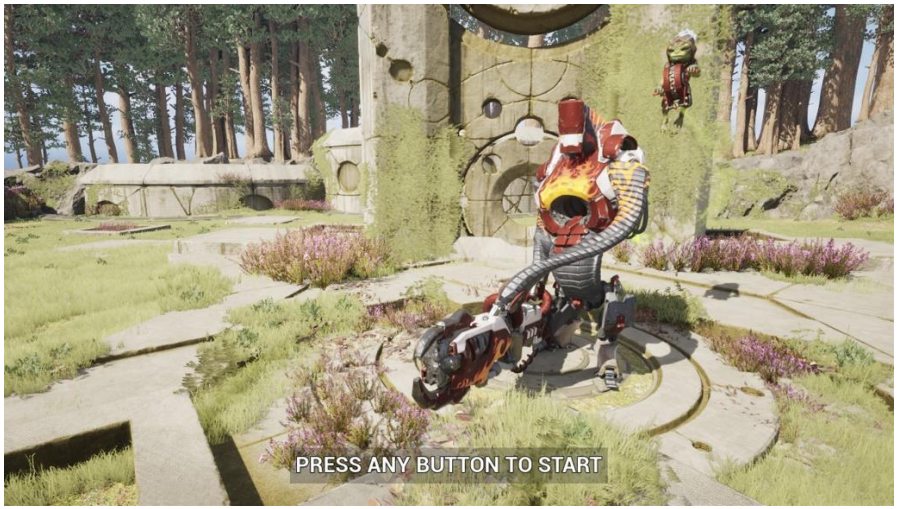



*Ilustración 73 – Menú de inicio.*


*Ilustración 74 – Menú principal.*

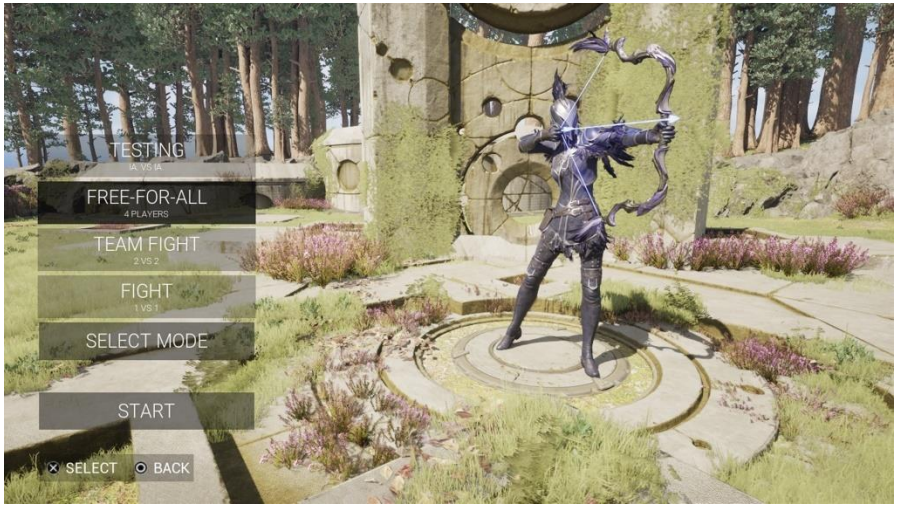



*Ilustración 75 – Menú de selección de modo de juego.*


*Ilustración 76 – Menú de controles.*


*Ilustración 77 – Menú de selección de personaje (2 jugadores).*


*Ilustración 78 – Menú de selección de personaje (4 jugadores).*


*Ilustración 79 – In-game combate de todos contra todos (4 jugadores).*

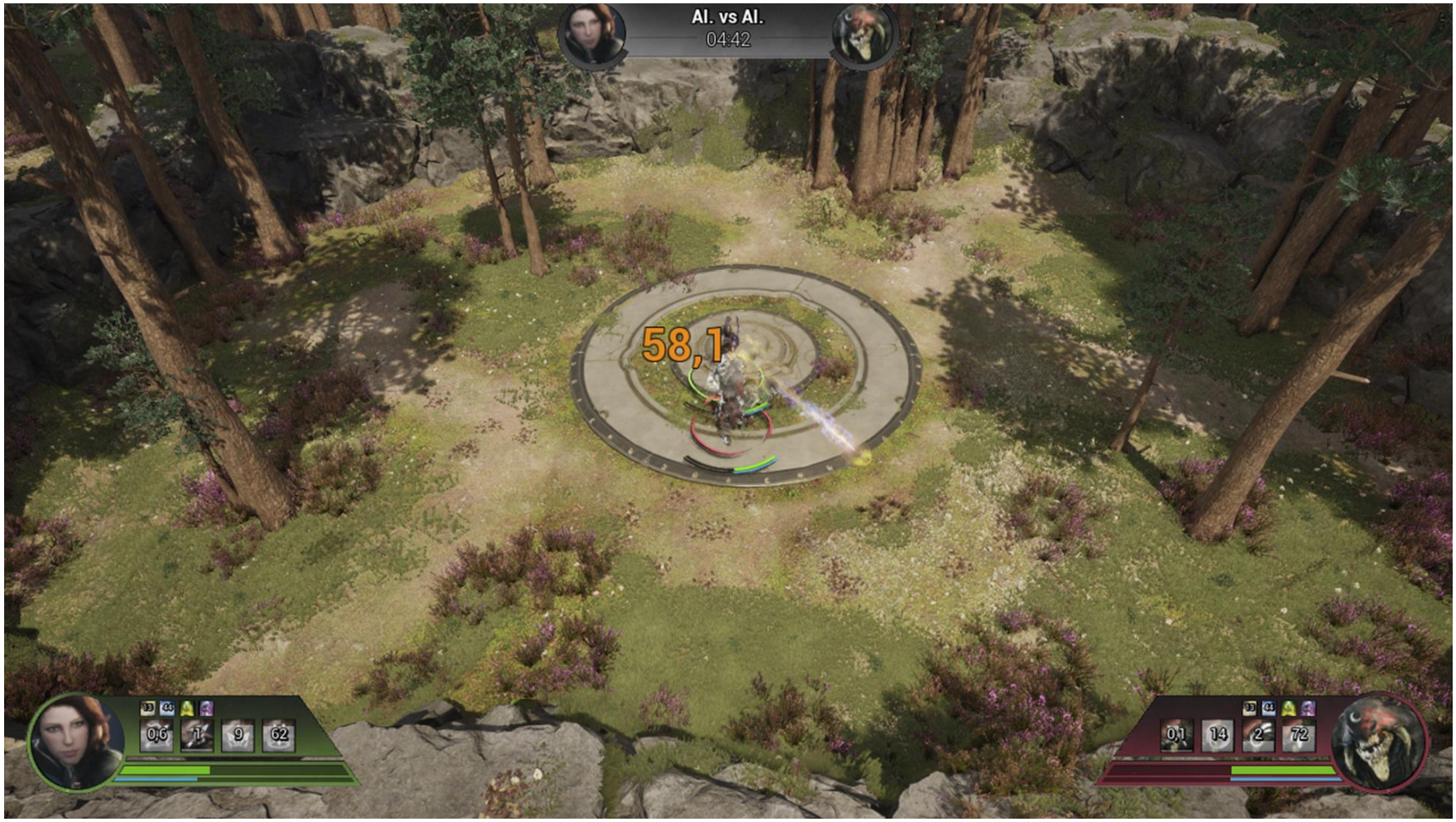



*Ilustración 80 – In-game combate de IA. contra IA.*

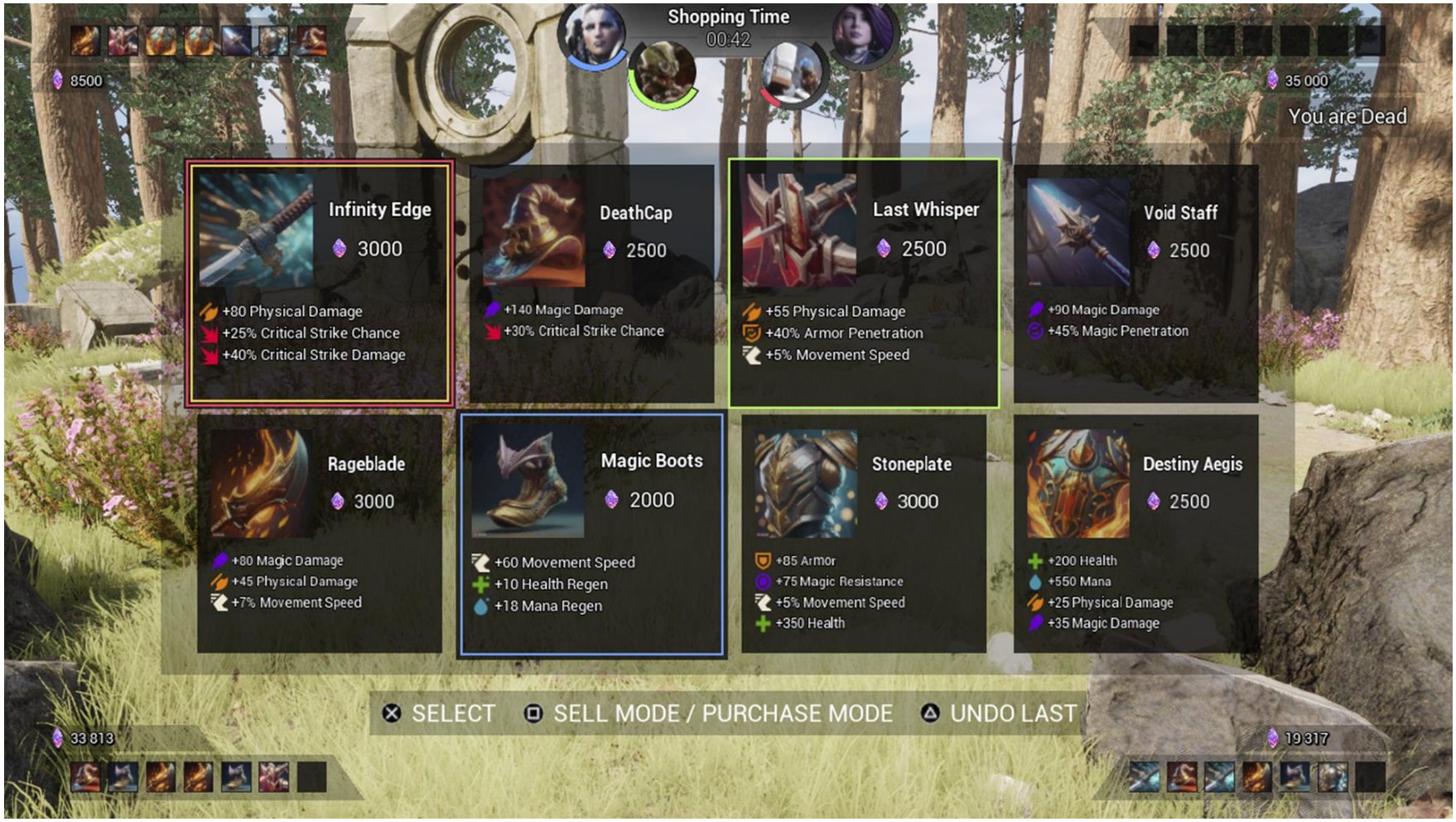



*Ilustración 81 – Menú de tienda para comprar objetos (4 jugadores).*


*Ilustración 82 – Menú de pausa.*


*Ilustración 83 – Menú de controles en pausa.*

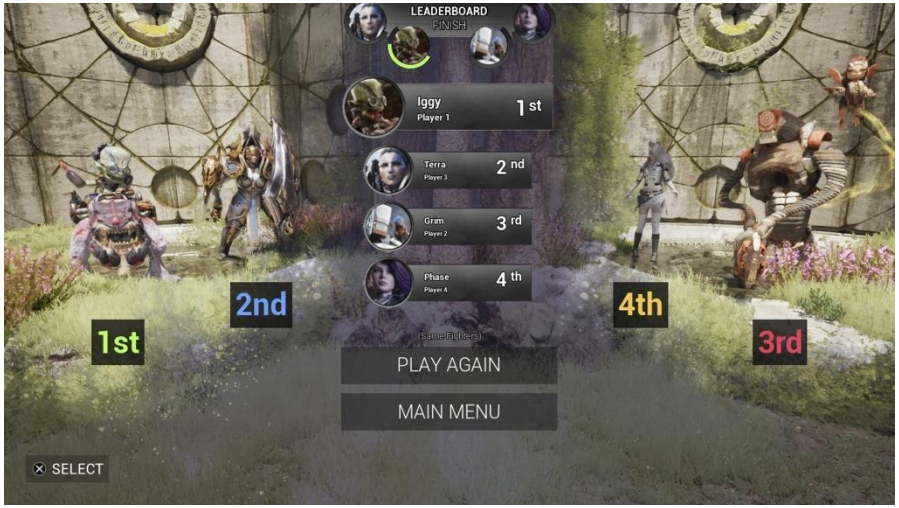



*Ilustración 84 – Menú de resultados de la partida (4 jugadores).*

## Anexo III

# Tablas de balance

---

En este apartado del documento se incluyen tablas realizadas en hojas de cálculo donde se han asignado las estadísticas base de los personajes y el daño de sus habilidades en función de sus atributos. Estás capturas muestran un balance de los personajes que puede verse afectado en un futuro según los datos recopilados por la inteligencia artificial en los combates. También contiene los datos de las habilidades de movimiento comunes a todos los personajes, con sus especificaciones.

## Rehabilitación

| | Estadísticas |
|---|---|
| Cooldowns | Un solo uso (No Stackeable entre Rondas) |
| | |
| Duración | 10 |
| | |
| Escudo | 285 |
| Curación | 220 |

## Extenuación

| | Estadísticas |
|---|---|
| Cooldowns | Un solo uso (No Stackeable entre Rondas) |
| | |
| Duración | 8 |
| Rango | 650 |
| | |
| Velocidad de Movimiento | (-40%) |
| | |
| If (Rehabilitación) Escudo | Elimina el Escudo por completo |

## Hiperactividad

| | Estadísticas |
|---|---|
| Cooldowns | 60 |
| | |
| Duración | 15 |
| | |
| Velocidad de Movimiento | (+48,12%) |
| | |
| If (Extenuación) Velocidad de Movimiento | Elimina la Extenuación y (+8,12%) |

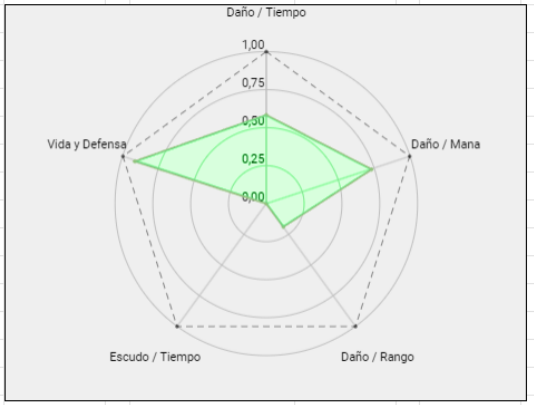
## Super Salto

| | Estadísticas |
|---|---|
| Cooldowns | 30 |
| | |
| Rango | 400 |

*Ilustración 85 – Estadísticas de las habilidades de movimiento.*

| | Estadísticas |
|---|---|
| Perforación Mágica | (Objetos) |
| Perforación Física | (Objetos) |
| Robo de vida | (Objetos) |
| Probabilidad de Crítico | (Objetos) |
| Daño Crítico | 175% |
| | |
| Vida (+25% / Ronda) | 1.034,75 |
| Regeneración de Vida (5s) | 10,46 |
| Armadura | 48,17 |
| Resistencia Mágica | 40,1 |
| | |
| Velocidad de Movimiento | 345 |
| Rango de Ataque | 140,00 |
| | |
| Fragmentos de Onyx | |
| | |
| Mana | 546,15 |
| Regeneración de Mana (5s) | 10,76 |
| | |
| Reducción de enfriamiento | (Objetos) |

| | Ataque Standard | Habilidad 1 | Habilidad 2 | Habilidad 3 |
|---|---|---|---|---|
| Poder Mágico | - | - | - | - |
| Poder Físico | 67,90 | 60 (+105% AD) | 100 (+160% AD) | 252 (+231% AD) |
| Daño Total | 67,90 | 131,295 | 208,64 | 408,849 |
| | | | | |
| Escudo | - | - | - | - |
| Escudo Total | - | - | - | - |
| | | | | |
| Coste de Mana | - | 50 | 60 | 100 |
| Cooldowns | 0,85 | 16 | 26 | 85 |
| Duración | 0,85 | 1,05 | 1,3 | 3 |
| Rango | 140 | 600 | 450 | 175 |
| | | | | |
| **Balance** | | | | |
| | | | | |
| Daño / Tiempo | 79,88 | 8,21 | 8,02 | 4,81 |
| Daño / Mana | 67,90 | 2,63 | 3,48 | 4,09 |
| Daño / Rango | 95,06 | 787,77 | 938,88 | 715,49 |
| Escudo / Tiempo | - | - | - | - |
| Vida y Defensa | - | - | - | - |

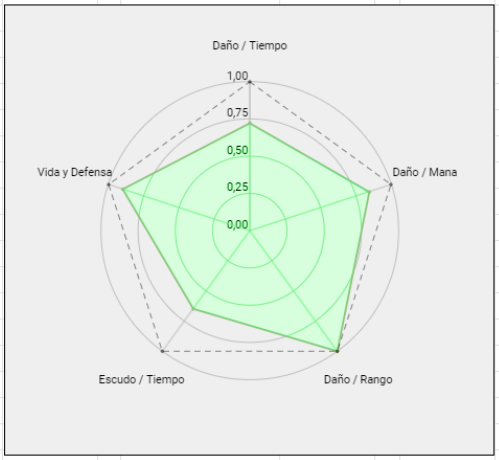



| | | |
|---|---|---|
| 100,92 | Mayor Daño / Tiempo | 172,30 | 0,59 |
| 78,09 | Mayor Daño / Mana | 106,31 | 0,73 |
| 2.537,20 | Mayor Daño / Rango | 13.588,72 | 0,19 |
| 0,00 | Mayor Escudo / Tiempo | 12,19 | 0,00 |
| 1.667,32 | Mayor Vida y Defensa | 1.829,18 | 0,91 |

*Ilustración 86 – Estadísticas de Countess.*

| | Estadísticas |
|---|---|
| Perforación Mágica | (Objetos) |
| Perforación Física | (Objetos) |
| Robo de vida | (Objetos) |
| Probabilidad de Crítico | (Objetos) |
| Daño Crítico | 175% |
| | |
| Vida (+25% / Ronda) | 1044,75 |
| Regeneración de Vida (5s) | 5,72 |
| Armadura | 44,56 |
| Resistencia Mágica | 35,13 |
| | |
| Velocidad de Movimiento | 325 |
| Rango de Ataque | 1000 |
| | |
| Fragmentos de Onyx | |
| | |
| Mana | 457,5 |
| Regeneración de Mana (5s) | 10,65 |
| | |
| Reducción de enfriamiento | (Objetos) |

| | Ataque Standard | Habilidad 1 | Habilidad 2 | Habilidad 3 |
|---|---|---|---|---|
| Poder Mágico | - | - | - | - |
| Poder Físico | 71,44 | Ataque Standard | - | 475 (+165% AD) |
| Poder Total | 71,44 | 71,44 | - | 592,876 |
| | | | | |
| Escudo | - | - | 195 (+70% AD) | - |
| Escudo Total | - | - | 245,008 | - |
| | | | | |
| Rango de Ataque | - | (+40%) | - | - |
| Velocidad de Ataque | - | (+80%) | - | - |
| | | | | |
| Coste de Mana | - | 40 | 20 | 100 |
| Cooldowns | 0,9 | 14 | 31 | 65 |
| Duración | 0,9 | 5 | -40 shield / sec | 3,8 |
| Rango | 1000 | 1400 | - | 1000 |
| | | | | |
| **Balance** | | | | |
| | | | | |
| Daño / Tiempo | 79,38 | 35,44 | - | 9,12 |
| Daño / Mana | 71,44 | 12,40 | 12,25 | 5,93 |
| Daño / Rango | 714,40 | 6.945,56 | - | 5.928,76 |
| Escudo / Tiempo | - | - | 7,90 | - |
| Vida y Defensa | - | - | - | - |

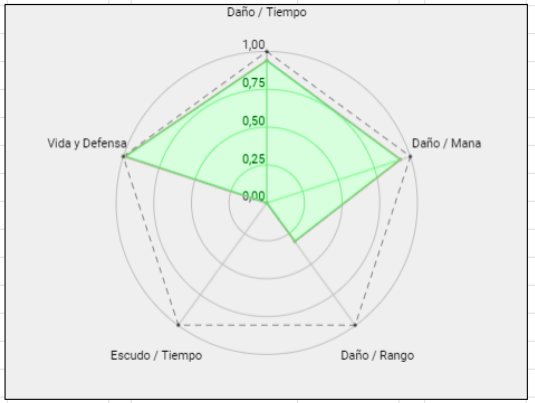



| | | |
|---|---|---|
| 123,94 | Mayor Daño / Tiempo | 172,30 | 0,72 |
| 89,77 | Mayor Daño / Mana | 106,31 | 0,84 |
| 13.588,72 | Mayor Daño / Rango | 13.588,72 | 1,00 |
| 7,90 | Mayor Escudo / Tiempo | 12,19 | 0,65 |
| 1.638,40 | Mayor Vida y Defensa | 1.829,18 | 0,90 |

*Ilustración 87 – Estadísticas de Grim.*

| | Estadísticas |
|---|---|
| Perforación Mágica | (Objetos) |
| Perforación Física | (Objetos) |
| Robo de vida | (Objetos) |
| Probabilidad de Crítico | (Objetos) |
| Daño Crítico | 175% |
| | |
| Vida (+25% / Ronda) | 1115,05 |
| Regeneración de Vida (5s) | 8,87 |
| Armadura | 51,59 |
| Resistencia Mágica | 40,1 |
| | |
| Velocidad de Movimiento | 350 |
| Rango de Ataque | 240 |
| | |
| Fragmentos de Onyx | |
| | |
| Mana | 513,5 |
| Regeneración de Mana (5s) | 9,87 |
| | |
| Reducción de enfriamiento | (Objetos) |

| | Ataque Standard | Habilidad 1 | Habilidad 2 | Habilidad 3 |
|---|---|---|---|---|
| Poder Mágico | - | - | - | (+150% AP) |
| Poder Físico | 86,56 | 112 (+38% AD) | 120 (+50% AD) | 450 (+75% AD) |
| Poder Total | 86,56 | 154,8928 | 163,28 | 514,92 |
| | | | | |
| Escudo | - | - | - | - |
| Escudo Total | - | - | - | - |
| | | | | |
| Coste de Mana | - | 40 | 60 | 85 |
| Cooldowns | 0,75 | 9 | 8 | 60 |
| Duración | 0,75 | 0,75 | 1,2 | 1,4 |
| Rango | 240 | 300 | 300 | 600 |
| | | | | |
| **Balance** | | | | |
| Daño / Tiempo | 115,41 | 17,21 | 20,41 | 8,58 |
| Daño / Mana | 86,56 | 3,87 | 2,72 | 6,06 |
| Daño / Rango | 207,74 | 464,68 | 489,84 | 3.089,52 |
| Escudo / Tiempo | - | - | - | - |
| Vida y Defensa | - | - | - | - |

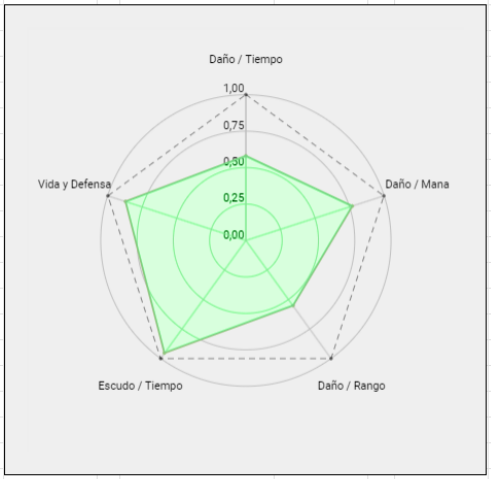



| | | |
|---|---|---|
| 161,62 | Mayor Daño / Tiempo | 172,30 | 0,94 |
| 99,21 | Mayor Daño / Mana | 106,31 | 0,93 |
| 4.251,78 | Mayor Daño / Rango | 13.588,72 | 0,31 |
| 0,00 | Mayor Escudo / Tiempo | 12,19 | 0,00 |
| 1.813,68 | Mayor Vida y Defensa | 1.829,18 | 0,99 |

*Ilustración 88 – Estadísticas de Grux.*

| | Estadísticas |
|---|---|
| Perforación Mágica | (Objetos) |
| Perforación Física | (Objetos) |
| Robo de vida | (Objetos) |
| Probabilidad de Crítico | (Objetos) |
| Daño Crítico | 175% |
| | |
| Vida (+25% / Ronda) | 1026,05 |
| Regeneración de Vida (5s) | 5,92 |
| Armadura | 41,58 |
| Resistencia Mágica | 35,13 |
| | |
| Velocidad de Movimiento | 330 |
| Rango de Ataque | 1000 |
| | |
| Fragmentos de Onyx | |
| | |
| Mana | 483 |
| Regeneración de Mana (5s) | 11,52 |
| | |
| Reducción de enfriamiento | (Objetos) |

| | Ataque Standard | Habilidad 1 | Habilidad 2 | Habilidad 3 |
|---|---|---|---|---|
| Poder Mágico | 10,12 | Ataque Standard | - | 165 (+60% AP) |
| Poder Físico | 63,25 | Ataque Standard | - | 140 (+60% AD) |
| Poder Total | 73,37 | 73,37 | - | 349,022 |
| | | | | |
| Escudo | - | - | 195 (+50% resistencia +50% armadura) | - |
| Escudo Total | - | - | 233,355 | - |
| | | | | |
| Velocidad de Movimiento | - | (+30%) | - | - |
| Velocidad de Ataque | - | (+60%) | - | - |
| Rango de Ataque | - | (+150%) Ancho | - | - |
| | | | | |
| Coste de Mana | - | 80 | 20 | 100 |
| Cooldowns | 1,1 | 14 | 20 | 85 |
| Duración | 1,1 | 5 | -40 shield / sec | 8 |
| Rango | 1000 | 1000 | - | 750 |
| | | | | |
| **Balance** | | | | |
| | | | | |
| Daño / Tiempo | 66,70 | 29,78 | - | 4,11 |
| Daño / Mana | 73,37 | 5,21 | 11,67 | 3,49 |
| Daño / Rango | 733,70 | 4.168,75 | - | 2.617,67 |
| Escudo / Tiempo | - | - | 11,67 | - |
| Vida y Defensa | - | - | - | - |

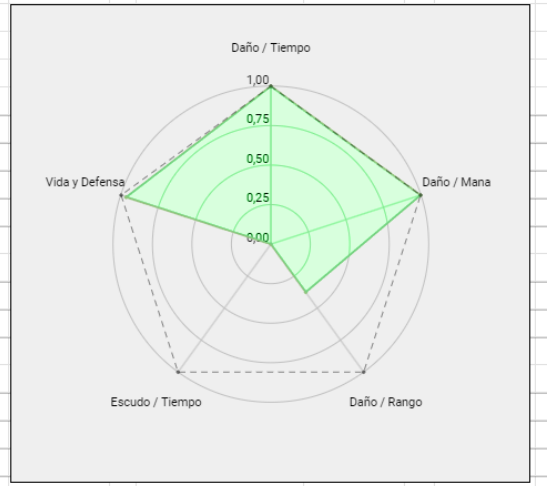



| | | |
|---|---|---|
| 100,58 | Mayor Daño / Tiempo | 172,30 | 0,58 |
| 82,07 | Mayor Daño / Mana | 106,31 | 0,77 |
| 7.520,12 | Mayor Daño / Rango | 13.588,72 | 0,55 |
| 11,67 | Mayor Escudo / Tiempo | 12,19 | 0,96 |
| 1.594,13 | Mayor Vida y Defensa | 1.829,18 | 0,87 |

*Ilustración 89 – Estadísticas de Iggy&Scorch.*

| | Estadísticas |
|---|---|
| Perforación Mágica | (Objetos) |
| Perforación Física | (Objetos) |
| Robo de vida | (Objetos) |
| Probabilidad de Crítico | (Objetos) |
| Daño Crítico | 175% |
| | |
| Vida (+25% / Ronda) | 1084,5 |
| Regeneración de Vida (5s) | 8,96 |
| Armadura | 52,41 |
| Resistencia Mágica | 40,1 |
| | |
| Velocidad de Movimiento | 350 |
| Rango de Ataque | 160 |
| | |
| Fragmentos de Onyx | |
| | |
| Mana | 517,75 |
| Regeneración de Mana (5s) | 9,87 |
| | |
| Reducción de enfriamiento | (Objetos) |

| | Ataque Standard | Habilidad 1 | Habilidad 2 | Habilidad 3 |
|---|---|---|---|---|
| Poder Mágico | - | - | - | 110 (+60% AP) |
| Poder Físico | 83,8 | Ataque Standard | 145 (+100% AD) | 250 (+100% AD) |
| Poder Total | 83,8 | 83,8 | 228,8 | 443,8 |
| | | | | |
| Escudo | - | - | - | - |
| Escudo Total | - | - | - | - |
| | | | | |
| Velocidad de Movimiento | - | (+30%) | - | - |
| Velocidad de Ataque | - | (+80%) | - | - |
| | | | | |
| Coste de Mana | - | 40 | 65 | 100 |
| Cooldowns | 0,9 | 14 | 7 | 90 |
| Duración | 0,9 | 5 | 2,4 | 1,75 |
| Rango | 160 | 160 | 200 | 800 |
| | | | | |
| **Balance** | | | | |
| Daño / Tiempo | 93,11 | 41,57 | 32,69 | 4,93 |
| Daño / Mana | 83,80 | 14,55 | 3,52 | 4,44 |
| Daño / Rango | 134,08 | 931,11 | 457,60 | 3.550,40 |
| Escudo / Tiempo | - | - | - | - |
| Vida y Defensa | - | - | - | - |



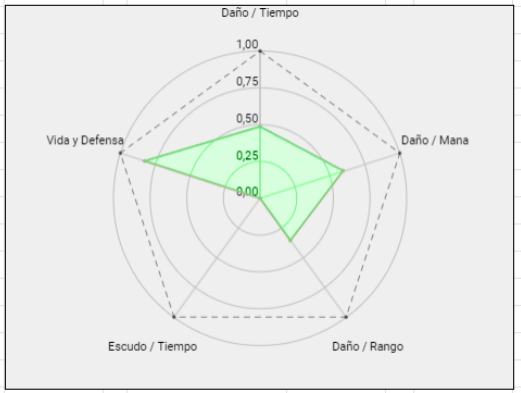

| | |
|---|---|
| 172,30 | |
| 106,31 | |
| 5.073,19 | |
| 0,00 | |
| 1.767,84 | |

| | | |
|---|---|---|
| Mayor Daño / Tiempo | 172,30 | 1,00 |
| Mayor Daño / Mana | 106,31 | 1,00 |
| Mayor Daño / Rango | 13.588,72 | 0,37 |
| Mayor Escudo / Tiempo | 12,19 | 0,00 |
| Mayor Vida y Defensa | 1.829,18 | 0,97 |

*Ilustración 90 – Estadísticas de Khaimera.*

| | Estadísticas |
|---|---|
| Perforación Mágica | (Objetos) |
| Perforación Física | (Objetos) |
| Robo de vida | (Objetos) |
| Probabilidad de Crítico | (Objetos) |
| Daño Crítico | 175% |
| | |
| Vida (+25% / Ronda) | 971,05 |
| Regeneración de Vida (5s) | 7,67 |
| Armadura | 41,54 |
| Resistencia Mágica | 35,13 |
| | |
| Velocidad de Movimiento | 330 |
| Rango de Ataque | 1000 |
| | |
| Fragmentos de Onyx | |
| | |
| Mana | 572,83 |
| Regeneración de Mana (5s) | 10,16 |
| | |
| Reducción de enfriamiento | (Objetos) |

| | Ataque Standard | Habilidad 1 | Habilidad 2 | Habilidad 3 |
|---|---|---|---|---|
| Poder Mágico | 54,03 | 12,259 (+8,028% AP) * 15 | 160 (+30% AP) + STUN | 375 (+115% AP) |
| Poder Físico | - | - | - | - |
| Poder Total | 54,03 | 248,947926 | 176,209 | 437,1345 |
| | | | | |
| Escudo | - | - | - | - |
| Escudo Total | - | - | - | - |
| | | | | |
| Coste de Mana | - | 85 | 75 | 100 |
| Cooldowns | 1 | 20 | 14 | 100 |
| Duración | 1 | 3,25 | 1,2 | 1,2 |
| Rango | 1000 | 600 | 350 | 500 |

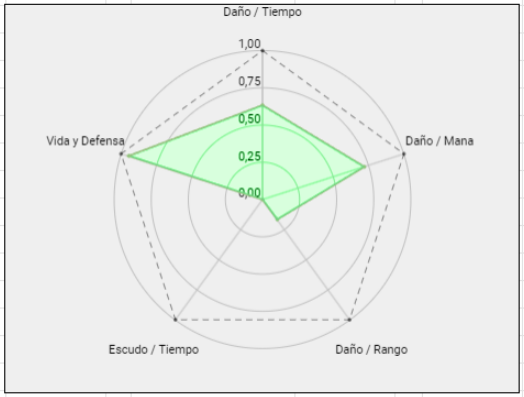



| Balance | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Daño / Tiempo | 54,03 | 12,45 | 12,59 | 4,37 | 83,44 | Mayor Daño / Tiempo | 172,30 | 0,48 |
| Daño / Mana | 54,03 | 2,93 | 2,35 | 4,37 | 63,68 | Mayor Daño / Mana | 106,31 | 0,60 |
| Daño / Rango | 540,30 | 1.493,69 | 616,73 | 2.185,67 | 4.836,39 | Mayor Daño / Rango | 13.588,72 | 0,36 |
| Escudo / Tiempo | - | - | - | - | 0,00 | Mayor Escudo / Tiempo | 12,19 | 0,00 |
| Vida y Defensa | - | - | - | - | 1.508,49 | Mayor Vida y Defensa | 1.829,18 | 0,82 |

*Ilustración 91 – Estadísticas de Phase.*

| | Estadísticas |
|---|---|
| Perforación Mágica | (Objetos) |
| Perforación Física | (Objetos) |
| Robo de vida | (Objetos) |
| Probabilidad de Crítico | (Objetos) |
| Daño Crítico | 175% |
| | |
| Vida (+25% / Ronda) | 1055,8 |
| Regeneración de Vida (5s) | 7,96 |
| Armadura | 53,59 |
| Resistencia Mágica | 40,1 |
| | |
| Velocidad de Movimiento | 335 |
| Rango de Ataque | 200 |
| | |
| Fragmentos de Onyx | |
| | |
| Mana | 537 |
| Regeneración de Mana (5s) | 10,46 |
| | |
| Reducción de enfriamiento | (Objetos) |

| | Ataque Standard | Habilidad 1 | Habilidad 2 | Habilidad 3 |
|---|---|---|---|---|
| Poder Mágico | 61,8 | 140 (+70% AP) | 110 (+60% AP) + STUN | 210 (+43,75% AP) |
| Poder Físico | - | - | - | - |
| Poder total | 61,8 | 183,26 | 147,08 | 237,0375 |
| | | | | |
| Escudo | - | - | - | - |
| Escudo Total | - | - | - | - |
| | | | | |
| Coste de Mana | - | 20 | 45 | 100 |
| Cooldowns | 1 | 6,5 | 15 | 24 |
| Duración | 1 | 2,5 | 1,8 | 2,7 |
| Rango | 200 | 300 | 600 | 300 |

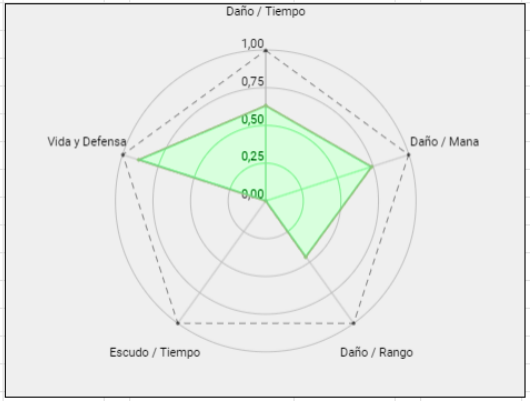



| Balance | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Daño / Tiempo | 61,80 | 28,19 | 9,81 | 9,88 | 109,68 | Mayor Daño / Tiempo | 172,30 | 0,64 |
| Daño / Mana | 61,80 | 9,16 | 3,27 | 2,37 | 76,60 | Mayor Daño / Mana | 106,31 | 0,72 |
| Daño / Rango | 123,60 | 549,78 | 882,48 | 711,11 | 2.266,97 | Mayor Daño / Rango | 13.588,72 | 0,17 |
| Escudo / Tiempo | - | - | - | - | 0,00 | Mayor Escudo / Tiempo | 12,19 | 0,00 |
| Vida y Defensa | - | - | - | - | 1.726,38 | Mayor Vida y Defensa | 1.829,18 | 0,94 |

*Ilustración 92 – Estadísticas de Sevarog.*

| | Estadísticas |
|---|---|
| Perforación Mágica | (Objetos) |
| Perforación Física | (Objetos) |
| Robo de vida | (Objetos) |
| Probabilidad de Crítico | (Objetos) |
| Daño Crítico | 175% |
| | |
| Vida (+25% / Ronda) | 1038,95 |
| Regeneración de Vida (5s) | 5,67 |
| Armadura | 44,17 |
| Resistencia Mágica | 35,13 |
| | |
| Velocidad de Movimiento | 325 |
| Rango de Ataque | 1000 |
| | |
| Fragmentos de Onyx | |
| | |
| Mana | 418,25 |
| Regeneración de Mana (5s) | 9,57 |
| | |
| Reducción de enfriamiento | (Objetos) |

| | Ataque Standard | Habilidad 1 | Habilidad 2 | Habilidad 3 |
|---|---|---|---|---|
| Poder Mágico | - | - | - | - |
| Poder Físico | 70,65 | 125 (+125% AD) | 50 (+100% AD) | 260 (+120% AD) |
| Poder Total | 70,65 | 213,3125 | 120,65 | 344,78 |
| | | | | |
| Escudo | - | - | - | - |
| Escudo Total | - | - | - | - |
| | | | | |
| Coste de Mana | - | 75 | 65 | 100 |
| Cooldowns | 0,9 | 14 | 11 | 80 |
| Duración | 0,9 | 4,2 | 0,9 | 2 |
| Rango | 1000 | 800 | 1000 | 750 |
| | | | | |
| Balance | | | | |
| | | | | |
| Daño / Tiempo | 78,50 | 15,24 | 10,97 | 4,31 |
| Daño / Mana | 70,65 | 2,84 | 1,86 | 3,45 |
| Daño / Rango | 706,50 | 1.706,50 | 1.206,50 | 2.585,85 |
| Escudo / Tiempo | - | - | - | - |
| Vida y Defensa | - | - | - | - |

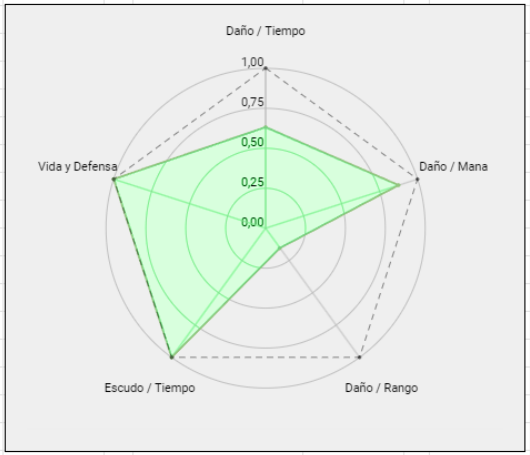



| | | |
|---|---|---|
| 109,01 | Mayor Daño / Tiempo | 172,30 | 0,63 |
| 78,80 | Mayor Daño / Mana | 106,31 | 0,74 |
| 6.205,35 | Mayor Daño / Rango | 13.588,72 | 0,46 |
| 0,00 | Mayor Escudo / Tiempo | 12,19 | 0,00 |
| 1.627,36 | Mayor Vida y Defensa | 1.829,18 | 0,89 |

*Ilustración 93 – Estadísticas de Sparrow.*

| | Estadísticas |
|---|---|
| Perforación Mágica | (Objetos) |
| Perforación Física | (Objetos) |
| Robo de vida | (Objetos) |
| Probabilidad de Crítico | (Objetos) |
| Daño Crítico | 175% |
| | |
| Vida (+25% / Ronda) | 1102,3 |
| Regeneración de Vida (5s) | 13,75 |
| Armadura | 59,54 |
| Resistencia Mágica | 40,1 |
| | |
| Velocidad de Movimiento | 340 |
| Rango de Ataque | 160 |
| | |
| Fragmentos de Onyx | |
| | |
| Mana | 492,1 |
| Regeneración de Mana (5s) | 7,98 |
| | |
| Reducción de enfriamiento | (Objetos) |

| | Ataque Standard | Habilidad 1 | Habilidad 2 | Habilidad 3 |
|---|---|---|---|---|
| Poder Mágico | - | - | - | - |
| Poder Físico | 83,75 | - | 110 (+120% AD) | 250 (+75% AD) |
| Poder Total | 83,75 | - | 210,5 | 312,8125 |
| | | | | |
| Escudo | - | 195 (+40% Vida Rest) | - | - |
| Escudo Total | - | 195 | - | - |
| | | | | |
| Perforación Física | - | - | - | - |
| | | | | |
| Coste de Mana | - | 40 | 35 | 100 |
| Cooldowns | 1,1 | 16 | 7 | 100 |
| Duración | 1,1 | -40 shield / sec | 2,8 | 1,9 |
| Rango | 160 | - | 250 | 450 |
| | | | | |
| **Balance** | | | | |
| | | | | |
| Daño / Tiempo | 76,14 | - | 30,07 | 3,13 |
| Daño / Mana | 83,75 | 4,88 | 6,01 | 3,13 |
| Daño / Rango | 134,00 | - | 526,25 | 1.407,66 |
| Escudo / Tiempo | - | 12,19 | - | - |
| Vida y Defensa | - | - | - | - |




| | | |
|---|---|---|
| 109,34 | Mayor Daño / Tiempo | 172,30 | 0,63 |
| 92,89 | Mayor Daño / Mana | 106,31 | 0,87 |
| 2.067,91 | Mayor Daño / Rango | 13.588,72 | 0,15 |
| 12,19 | Mayor Escudo / Tiempo | 12,19 | 1,00 |
| 1.829,18 | Mayor Vida y Defensa | 1.829,18 | 1,00 |

*Ilustración 94 – Estadísticas de Terra.*