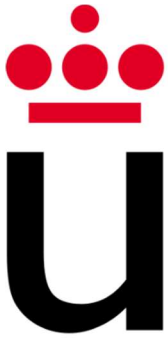


Universidad Rey Juan Carlos

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN DISEÑO Y DESARROLLO DE VIDEOJUEGOS

Curso Académico 2023/2024

Trabajo Fin de Grado

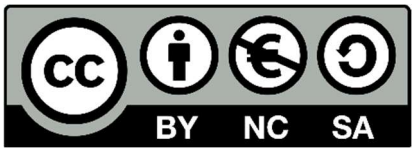
DESARROLLO DE UN JUEGO ARENA DE BATALLA EN UNREAL ENGINE

Autor:

Luis Torres Valera

Tutor:

Aarón Sújar Garrido



Agradecimientos a mi padre, mi madre y mi hermana,

que me han apoyado en este proceso

y lo seguirán haciendo en el futuro.

A todos los compañeros y amigos,

que me han acompañado en el proceso y han aprendido conmigo,

en especial a Roberto, que ha afrontado este reto conmigo.

A nuestro tutor Aarón por guiarnos en este proyecto,

ayudarnos y aconsejarnos durante todo el proceso.

Resumen

Este trabajo de fin de grado se centra en el desarrollo de una demo técnica, junto con mi compañero Roberto Herencias Muñoz, de un videojuego del tipo arena de batalla, utilizando el motor de videojuegos Unreal Engine. Este videojuego muestra una jugabilidad similar en el conocido MOBA League of Legends (Riot Games, 2009) en el que nos hemos basado durante el desarrollo. El juego está implementado como un juego multijugador local, donde los jugadores se medirán en batalla en los diferentes modos de juego: 1 vs 1, 2 vs 2 y todos contra todos de 4 jugadores.

El objetivo de este proyecto no es centrarse en el diseño gráfico y el aspecto visual, sino la implementación y la programación de la lógica del juego, prestando un mayor foco en el sistema de combate. Para el desarrollo de esta faceta se ha hecho uso del plugin "Gameplay Abilities" de Unreal Engine, que permite crear habilidades y mecánicas de combate bajo un marco robusto y flexible. Con el uso de este plugin se han gestionado de manera eficiente las habilidades activas, los efectos pasivos, los efectos de estado, los cooldowns y otros elementos que componen el sistema de combate.

Palabras Clave

Unreal Engine

Gameplay Ability System

Videojuego

Arena de batalla

Multijugador local

Demo técnica

Mecánicas de combate

Inteligencia Artificial

Abstract

This bachelor's thesis focuses on the development of a technical demo, in collaboration with my colleague Roberto Herencias Muñoz, of a battle arena type video game using Unreal Engine. This video game features gameplay like the well-known MOBA "League of Legends" (Riot Games, 2009), which we used as a reference during development. The game is conceived as a local multiplayer game, where players will clash in different game modes: 1 vs 1, 2 vs 2, and free-for-all with 4 players.

The purpose of this project is to focus on the implementation and programming of the game's logic rather than graphic design and visual aspects. The main focus is the combat system, for the development of this aspect, we used Unreal Engine's "Gameplay Abilities" plugin, which allows us to create abilities and combat mechanics under a robust and flexible framework. Thanks to this plugin, we have efficiently managed active abilities, passive effects, status effects, cooldowns, and other elements that build up the combat system.

Keywords

Unreal Engine

Gameplay Ability System

Video game

Battle arena

Local multiplayer

Technical demo

Combat mechanics

Artificial Intelligence

Índice de contenidos

1. Introducción

1.1 Descripción

El proyecto se centra en el desarrollo de una demo técnica realizada de forma conjunta entre un equipo de dos personas. El marco de desarrollo se compone principalmente del motor de videojuegos Unreal Engine 5.3.2 [1], la versión estable más reciente al comienzo del desarrollo del proyecto. Esta demo técnica se centra en conseguir un videojuego funcional con diferentes modos de juego, donde dos o más jugadores pueden elegir entre un conjunto variado de personajes con características únicas y diferenciadas para enfrentarse en batalla. Además, tras cada ronda de juego, cada jugador tendrá acceso a un conjunto de objetos que podrá comprar para modificar los atributos de su personaje en la siguiente ronda. Estos objetos podrán ser obtenidos a través de un sistema de economía donde cada jugador obtendrá una cantidad diferente de recursos en base a sus acciones durante las rondas de juego.

Para implementar todo el sistema de combate se ha hecho uso del *plugin Gameplay Ability System* [2], en adelante lo nombraremos GAS. Este *plugin*, implementado por Unreal Engine, es un *framework* flexible y modular que facilita la creación y gestión de habilidades y efectos, enfocados principalmente a los videojuegos de combate. De esta forma, GAS permite a los desarrolladores implementar una amplia variedad de comportamientos de una forma eficiente y escalable. A través del uso de este *plugin*, durante el proyecto se han desarrollado las habilidades de los personajes, los efectos de estado y pasivos, y la gestión de atributos como pueda ser el maná, la vida, el escudo, entre otros. Además, GAS se encarga de gestionar otros aspectos claves como el costo o el *cooldown* de las habilidades y es una base robusta de cara a incluir un multijugador en línea ya que gestiona la replicación y sincronización entre los diferentes clientes.

Al ser un proyecto orientado al desarrollo de la programación, se centra en menor medida en el diseño del aspecto visual. Por tanto, aunque el material visual presente en las interfaces y los menús si es elaboración propia del proyecto, todo el material gráfico referido al diseño de los personajes y *assets* del juego se ha obtenido de diferentes bibliotecas de contenido como pueda ser *Marketplace* de Unreal Engine [3]. Por otra parte, algunos de los iconos mostrados en el juego, como los iconos de los objetos y habilidades, se han creado a partir de herramientas generativas basadas en la inteligencia artificial, como la herramienta *Adobe Firefly* [4].

Cabe destacar que el proyecto se ha desarrollado en colaboración con mi compañero Roberto Herencias Muñoz. Esto nos permite abordar de forma conjunta la dificultad de aprender el uso de un motor desde cero, pudiendo llegar a un resultado con mayor desarrollo que si se hubiese realizado de forma individual.

1.2 Motivación

Unreal Engine [1] es una herramienta que no se ha empleado durante el transcurso de las asignaturas impartidas en el grado, y sin embargo es un motor que se encuentra en auge en la industria del videojuego tanto a nivel global, como a nivel nacional, como confirma el "Libro Blanco del Desarrollo Español de Videojuegos" [5]. A parte de su creciente auge, Unreal Engine es una herramienta que presenta una gran variedad de funcionalidades que le hace destacar frente a otros motores de desarrollo. Así mismo, destaca por sus gráficos de alta calidad, ya que su motor gráfico permite el uso de iluminaciones realistas y materiales con gran nivel de detalle [6].

Parte de la motivación del proyecto es incluir el *plugin* GAS, que necesita para su implementación en el proyecto el uso de C++ [7], ya que Unreal Engine está desarrollado en

este lenguaje de programación y el desarrollo del proyecto mejoraría el conocimiento del uso de C++ en el desarrollo de videojuegos. Este es uno de los lenguajes más populares en la industria del videojuego, muy usado por diferentes estudios en sus motores propios, ya que es muy eficiente, tiene portabilidad a todas las plataformas, y las principales bibliotecas de desarrollo están implementadas en este lenguaje [8].

El hecho de afrontar este proyecto como una colaboración junto con mi compañero Roberto Herencias Muñoz, parte del interés común de ambos alumnos por aprender a emplear Unreal Engine, asumiendo un reto mayor, al intentar aprender acerca de la mayoría de las áreas de desarrollo que incluye el motor, en vez de centrarse solo en un área de desarrollo como ocurriría con el alcance de un proyecto individual. Por otra parte, el trabajo en equipo permite mejorar habilidades blandas, como la comunicación y la colaboración ente los integrantes, que en un proyecto individual no se podrían mejorar [9].

1.3 Objetivos

El objetivo principal del proyecto sería:

- Realizar una demo técnica de un videojuego funcional empleando el motor Unreal Engine en su versión estable más reciente al comienzo del proyecto (5.3.2).

Este objetivo general se puede desglosar en los siguientes objetivos secundarios:

- Aprender y demostrar un dominio avanzado del motor Unreal Engine y sus principales herramientas.
- Aprender a desarrollar y programar bajo el lenguaje de *scripting* visual de Unreal Engine (*Blueprints*).
- Mejorar en el uso del lenguaje de programación C++ y aprender a utilizar e implementar *plugins* externos con el ejemplo de GAS.

- Desarrollar y probar modos de juego e interfaces multijugador local que permita a varios jugadores enfrentarse en la misma pantalla.
- Explorar e implementar las herramientas de inteligencia artificial del motor Unreal Engine.
- Mejorar el trabajo en equipo y aprender a emplear herramientas colaborativas en un marco de un desarrollo real de un videojuego.
- Lanzar el resultado del videojuego en una plataforma oficial.

1.4 Planificación inicial

Se plantea inicialmente una separación de tareas generales entre los dos integrantes para poder definir de forma clara que parte del desarrollo ha llevado a cabo cada integrante.

Al ser ambos perfiles de programación y estar ambos interesados en la implementación del *plugin Gameplay Ability System* (GAS), la tarea inicial de implementar las bases del plugin y definir la estructura de clases del proyecto se realiza de forma conjunta entre ambos alumnos. El resto de las tareas que se estiman desarrollar en una planificación inicial del proyecto se reparten siguiendo esta división, como se muestra en la **ilustración 1**:

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 iniclal y reparto

En cuanto al planteamiento inicial de la metodología empleada en el desarrollo, se propone el uso de un formato iterativo, con *sprints* de dos semanas, donde al final de cada iteración cada alumno se encarga de revisar y probar las tareas realizadas por su compañero para que ambos tengan noción de todos los campos en los que se han trabajado.

1.5 Estructura de la memoria

La memoria se organiza en tres bloques principales:

- Introducción: En este bloque se presentan los principales objetivos del proyecto junto con una descripción general del mismo y las motivaciones para llevarlo a cabo. Esta parte también incluye una descripción sobre el contexto general del desarrollo de videojuegos, y más específicamente del desarrollo con Unreal Engine. Además, incluye una descripción general de la metodología empleada y de las herramientas utilizadas para realizar el proyecto.
- Diseño y desarrollo del juego: En este bloque se detalla el proceso de diseño y desarrollo de la demo técnica, incluyendo una descripción de los procesos llevados a cabo. Así, se exponen todas las decisiones tomadas y dificultades encontradas durante el desarrollo del proyecto, desde las relativas al diseño previas a la implementación del juego, hasta las implementaciones realizadas en todos los ámbitos del videojuego.
- Conclusiones: En este bloque se recogen los resultados obtenidos durante el transcurso del proyecto y se evalúan según los objetivos planteados en un principio. De esta forma, se valoran las competencias adquiridas durante el desarrollo del proyecto y se planteas posibles líneas de futuro para mejorar los resultados obtenidos.

2. Metodología de desarrollo

2.1 Metodología incremental en espiral

El enfoque que se ha seguido durante el desarrollo del proyecto es el uso de una metodología incremental en espiral [10]. Esta metodología se caracteriza por dividir el desarrollo a través de iteraciones (*sprints*), donde cada iteración mejora y expande las funcionalidades de la anterior. Es especialmente útil para trabajos que requieren un alto grado de flexibilidad y que se basan en una evolución continua que valora el *feedback* recibido en cada iteración. Para complementar esta metodología se ha empleado Kanban para el seguimiento de las tareas y su progreso.

La metodología en espiral se centra en la repetición de procesos clave como son: la planificación de los objetivos para cada iteración, el desarrollo evolutivo del producto y la evaluación y retroalimentación para el siguiente sprint. Esto permite una mayor flexibilidad y adaptabilidad en el proceso de desarrollo, ya que los cambios que surgen del *feedback* pueden ser añadidos con facilidad y, además, reduce los riesgos durante el desarrollo, ya que estos se identifican y se evalúan de forma constante, lo que permite crear estrategias para mitigarlos antes de que supongan un problema mayor para el proyecto. Frente a otras metodologías basadas en el modelo de cascada, la metodología incremental en espiral es más flexible, ya que el resultado puede sufrir cambios a lo largo del desarrollo y, además, al final de cada iteración se obtiene un resultado tangible como producto final, a diferencia de una metodología en cascada, donde el primer resultado se obtiene después de la fase de desarrollo. En la **ilustración 2** se muestra un diagrama del desarrollo en espiral.

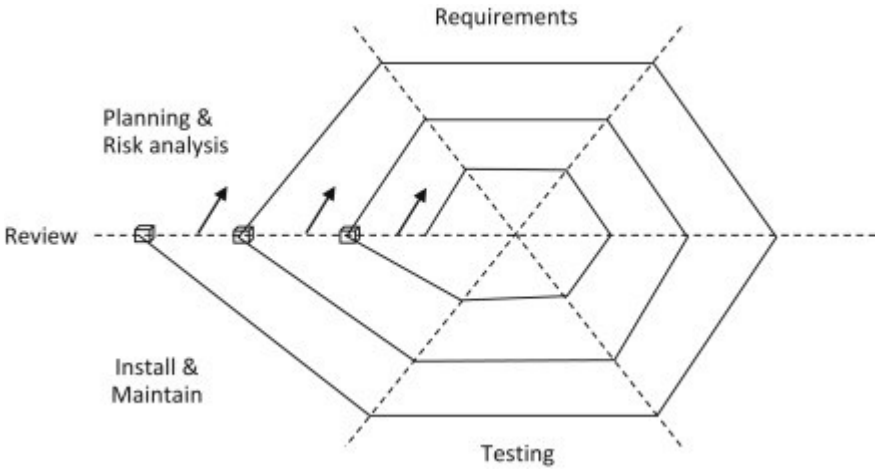


Ilustración 2 - Desarrollo en espiral

Kanban se emplea para visualizar el flujo de trabajo y así, mejorar la eficiencia del proceso [11]. Al planificar cada iteración se crea una lista de tareas a cumplir para cada sprint y se reparte entre las diferentes columnas del tablero en función de su estado. Cuando se progresa en el desarrollo de una tarea, esta se mueve de columna en el tablero para visualizar de forma directa el estado de cada tarea. Esto permite identificar problemas como cuellos de botella o limitar el trabajo en proceso para mejorar la eficiencia.

Si bien no se pueden enmarcar estas metodologías bajo el nombre de algunos desarrollos ágiles como Scrum, ya que no se centran en la colaboración constante del grupo de trabajo y en entregar valor al cliente de forma constante, el uso complementario de iteraciones en espiral y Kanban crean un marco que se centra en los mismos objetivos que un desarrollo ágil, ofreciendo un enfoque estructurado y flexible para el desarrollo software.

2.2 Herramientas

En el siguiente apartado se muestran las principales herramientas empleadas para el desarrollo del proyecto, tanto a nivel técnico, como a nivel organizativo.

2.2.1 Trello

Trello [12] es una herramienta colaborativa que permite crear tableros compartidos, compuestos por columnas y tarjetas. Cada tarjeta se puede crear con una personalización

única permitiendo cambiar categorías, etiquetas, colores descripciones o añadir archivos adjuntos como fotos, documentos, enlaces web entre otros. Además, cada colaborador del tablero podrá asignarse a cada tarjeta para gestionar quien es el encargado de realizar cada tarea, y podrá mover entre las columnas todas las tarjetas del tablero.

En el tablero particular para nuestro proyecto hemos añadido una columna en la que se incluye toda la información relevante del proyecto junto con los enlaces de interés, otra columna *Dashboard* donde se mantienen las tareas generales sin desglosar en tareas más pequeñas y, por último, las columnas características de Kanban (*To do, Doing, Done*) donde se encuentras las tareas individuales según su estado de progreso. Además, el tablero cuenta con etiquetas personalizadas para categorizar cada una de las tareas.

2.2.2 Notion

Notion [13] es una herramienta de productividad que permite crear espacios colaborativos donde se combinan diferentes herramientas como puedan ser: notas, tareas, bases de datos y herramientas de gestión para los proyectos.

Con esta herramienta se pueden crear bibliotecas de contenido, que es muy útil en el proyecto a la hora de crear un documento de diseño del juego de forma conjunta para ambos integrantes.

La estructura de Notion permite crear páginas, que a su vez pueden contener otras páginas o hacer referencia a estas, similar a como funcionaría en una página web. Además, sus herramientas de exportación permiten disponer del contenido del proyecto en un formato html, que posee una navegación más fluida que un documento al uso.

2.2.3 Github Desktop

Github [14] es una plataforma de control de versiones que emplea la tecnología de Git. Esta plataforma ofrece herramientas para la colaboración y gestión de proyectos de

desarrollo software permitiendo a los diferentes desarrolladores trabajar de forma conjunta y eficiente. Su estructura de historial de versiones almacena una copia de cada versión del proyecto permitiendo así la reversión de cambios no deseados y la resolución eficiente de conflictos.

Github Desktop es una aplicación gratuita desarrollada por Github que proporciona una interfaz gráfica para trabajar con Git sobre los repositorios de Github. Esta herramienta facilita la sincronización de los cambios en los repositorios locales y permite la creación de ramas, *commits* y *pull request*, a la vez que facilita la resolución de conflictos al incluir herramientas visuales.

Limitaciones

Github, al igual que otras herramientas de historial de versiones, puede generar conflictos al trabajar en Unreal Engine con los elementos del tipo *uasset* que conforman los *Blueprints*. Estos archivos no pueden ser mezclados entre dos versiones diferentes del mismo, si no que se tienen que sobrescribir. Esto limita el trabajo en equipo en las etapas iniciales del desarrollo ya que varios integrantes no pueden realizar cambios simultáneos sobre un mismo archivo *Blueprint*. De igual manera se podrían ver afectados todos los actores en un nivel concreto, pero Unreal Engine solventa estos problemas con el sistema de *World Partition* incluyendo lo que se conoce como *One Flie Per Actor* que crea un fichero diferente para cada uno de los actores del mapa. De esta forma, si dos personas se encuentran trabajando sobre el mismo mapa, pero no sobre el mismo actor, no tendrían conflictos al subir cambios diferentes.

Por otra parte, cabe destacar que Github no es capaz de procesar subidas de archivos de gran tamaño, por lo que si se quieren incluir en el proyecto un conjunto de archivos extenso se deberá hacer de manera escalonada y evitar archivos con un tamaño superior a 100 MB.

2.2.4 Discord

Discord [15] es una aplicación que permite la comunicación a través de canales de texto, voz y video. Durante el desarrollo del proyecto se ha creado un servidor con diferentes canales de textos separados por categorías, para incluir información relevante para el proyecto como puedan ser: tutoriales, documentación, ejemplos o líneas de discusión sobre las diferentes ramas del proyecto. Además, se han incluido salas de voz para facilitar el trabajo colaborativo, permitiendo realizar llamadas diarias o compartir videos en directo del avance del trabajo.

2.2.5 Hojas de cálculo

Las hojas de cálculo es una herramienta gratuita proporcionada por Google que permite crear tablas con diferentes gráficos y funciones. En el desarrollo del proyecto se ha empleado para realizar una estimación del balance de los personajes, incluyendo datos como los atributos de los personajes, el *cooldown* de sus habilidades o la duración de los efectos para equilibrar las capacidades de todos los personajes. Se pueden consultar estas tablas de balance en el **Anexo III**.

2.2.6 Unreal Engine

Unreal Engine [1] es un motor de juegos desarrollado por Epic Games [1], ampliamente reconocido por su capacidad de renderizado gráfico de alta calidad y su robusta arquitectura. Desde su lanzamiento inicial en 1998, Unreal Engine ha evolucionado para convertirse en una herramienta completa y flexible, adecuada para el desarrollo de videojuegos de diversos géneros y para aplicaciones en campos como la simulación, la arquitectura y el cine.

Algunas de las principales características para valorar el uso de este motor de desarrollo son las siguientes:

- Motor gráfico y de renderizado: Unreal posee un motor de renderizado capaz de producir gráficos de alta calidad, con sistemas de iluminación dinámica y efectos visuales avanzados permitiendo crear entornos visualmente impresionantes. Además, permite soporte integrado para el trazado de rayos en tiempo real, consiguiendo efectos de iluminación más realistas.
- *Scripting* visual mediante *Blueprints*: este sistema permite crear lógica de juego sin necesidad de escribir código. Esto facilita la rápida iteración y el prototipado.
- Motor físico: el motor *Chaos Physics and Destruction* permite simulaciones realistas de destrucción y físicas.
- Desarrollo multiplataforma: Unreal soporta el desarrollo en una gran variedad de plataformas, incluidas la realidad virtual y aumentada.

Comparativa de Unreal Engine y Unity

En comparación con el motor de desarrollo más empleado en el sector, Unity [16], se ha valorado emplear Unreal en el proyecto porque es una herramienta que se encuentra en auge en el sector del desarrollo de videojuegos y le está recortando terreno en el porcentaje de empresas que deciden usar este motor para el desarrollo de sus juegos. Como se expone en el Libro Blanco del Desarrollo Español de Videojuegos [5], este auge se puede deber en parte a la reciente polémica que rodea a Unity por el cambio de sus términos y condiciones sobre el precio del motor para las empresas desarrolladoras. Este cambio favorece a Unreal, ya que una de las principales diferencias por las que algunas empresas independientes se decantaban por Unity eran sus menores costes frente a Unreal, que era una opción más valorada en empresas más grandes.

Por otra parte, a la hora de elegir un motor entre estas opciones, se ha de tener en cuenta las diferentes características en las que destaca cada uno [17]. Unreal Engine destaca por su calidad gráfica y su alto nivel de detalle que permite obtener muy buenos resultados visuales sin suponer un problema para el rendimiento y la optimización. En su lado, Unity presenta una interfaz muy intuitiva y es un programa muy abierto a los nuevos

desarrolladores ya que es un programa muy versátil. Además, es un programa que cuenta con una gran cantidad de contenido formativo en internet, en gran parte apoyado por la comunidad de desarrolladores de Unity, aunque Unreal está evolucionando positivamente en la cantidad de información y documentación disponible en la web debido a su reciente auge.

2.2.7 Visual Studio

Visual Studio [18] proporciona un entorno de desarrollo integrado (IDE) desarrollado por Microsoft que permite crear aplicaciones software diferentes lenguajes, como pueda ser C++ que es el lenguaje que emplea Unreal Engine.

Se ha decidido emplear Visual Studio frente a diferentes IDE ya que Unreal Engine tiene soporte integrado para esta aplicación, lo que facilita la configuración y el inicio rápido del desarrollo. Debido a esta integración, la depuración es más eficiente dentro del propio IDE pudiendo añadir directamente puntos de interrupción o rastrear el flujo de forma simple.

Por otra parte, Visual Studio incluye también herramientas de ayuda y guiado para facilitar la programación, como pueda ser IntelliSense, que proporciona autocompletado y sugerencias en tiempo real para mejorar la productividad y reducir el número de errores durante la programación.

2.2.8 Marketplace de Unreal Engine

Marketplace de Unreal Engine [3] es una plataforma donde cualquier desarrollador o diseñador puede comprar, vender o compartir contenido digital para ser empleado en proyectos realizados con Unreal. Epic Games, creadores del motor Unreal, también comparten contenido de sus diferentes juegos para el uso libre de los desarrolladores de Unreal.

En el caso particular de este proyecto, se han empleado *assets* de Epic Games de su juego Paragon, para incluirlos como personajes del videojuego. Estos *assets* incluyes mayas 3D, texturas, animaciones y efectos, que se han empleado en el proyecto ya que no se centra en el diseño del aspecto visual.

2.2.9 Quixel Bridge

Quixel Bridge [19] es una herramienta integrada en Unreal Engine que proporciona una biblioteca extensa de recursos digitales, que pueden ser importados directamente desde los proyectos de Unreal. Esta biblioteca cuenta con un amplio catálogo de modelos 3D realistas obtenidos mediante fotogrametría (*Megascans*), además de incluir texturas y materiales de alta calidad.

En el proyecto se han obtenido recursos como rocas, materiales de superficies o diferentes recursos visuales para mejorar los escenarios y mapas del videojuego.

3. Estado del arte

En este capítulo se desarrolla un contexto sobre los videojuegos, más concretamente sobre videojuegos de la categoría arena de batalla y Multijugador local, y juegos desarrollados con el motor Unreal Engine y en particular con el plugin GAS.

3.1 Videojuegos de arena de batalla

Los videojuegos de arena de batalla son un subgénero de los juegos de lucha y acción, que se caracterizan por contar con un entorno cerrado donde varios jugadores se enfrentan y compiten entre sí. Estos juegos se caracterizan por contar con personajes con un conjunto de habilidades y movimientos únicos, que permite una gran variedad de estilos de juego y estrategias.

Algunos ejemplos de este estilo de juegos son: Super Smash Bros [20], desarrollado por Nintendo, Brawlhalla [21], desarrollado por Blue Mammoth Games y publicado por Ubisoft, y está presente en algunos modos de juego de títulos MOBA como League of Legends [22], desarrollado por Riot Games, en su modo Arena.

3.2 Videojuegos multijugador local

Los juegos multijugador local permiten que varios jugadores jueguen en la misma consola u ordenador compartiendo pantalla, ya sea con una cámara general cenital o lateral que muestra todos los personajes, o a través de la pantalla dividida. Este tipo de juegos ha sido fundamental en la creación de experiencias de juego social y competitivo [23].

Estos juegos presentan ejemplos de diversos géneros, desde juegos de lucha como la saga Street Fighter [24], juegos de carreras como Mario Kart [25], o juegos de arena de batalla como los ya mencionados anteriormente: Super Smash Bros [20], Brawlhalla [21], o TowerFall Ascension, desarrollado por Matt Makes Games [26].

Este estilo de juegos se ha visto desplazado en gran medida por el auge de los videojuegos multijugador en línea [27], aunque sigue presente en nuevos formatos que combinan ambas tecnologías de forma híbrida permitiendo en la misma partida jugadores compitiendo en local, a la vez que compiten con otros jugadores conectados en línea.

3.3 Unreal Engine

Unreal Engine [1] es uno de los motores de juego más populares y potentes en la industria del desarrollo de videojuegos. Este motor ofrece una amplia gama de herramientas y características que facilitan la creación de juegos de una calidad alta. Este motor destaca frente a otros por aspectos como sus gráficos de alta calidad, su flexibilidad a la hora de crear juegos de diferentes tipos de géneros y para diversas plataformas, y sus herramientas robustas con un sistema avanzado de partículas, herramientas de animación, físicas avanzadas y un editor visual de niveles [28].

El ejemplo más conocido de juego que emplea el motor de Unreal es Fortnite [29], desarrollado por la propia Epic Games, que ganó una gran repercusión a partir del año 2020 debido a su experiencia de juego fluida, su aspecto visual y el soporte para un gran número de plataformas.

Otro ejemplo menos destacado es Risk of Rain 2 [30], desarrollado por Hopoo Games. Este juego es del género roguelike de acción, que emplea el motor Unreal para gestionar la complejidad de un entorno multijugador cooperativo, con gráficos 3D y una jugabilidad dinámica.

3.4 Gameplay Ability System

El *plugin* Gameplay Abilities [2] destaca por su capacidad para manejar mecánicas de combate y habilidades. Este sistema permite a los desarrolladores definir, gestionar y ejecutar habilidades de manera modular y escalable.

Debido a su soporte para la replicación y sincronización de datos en entornos multijugador, es un plugin empleado en diversos juegos en línea, ya que permite que los efectos y atributos de los diferentes clientes se sincronicen correctamente [31].

Algunos ejemplos de videojuegos que han empleado GAS en su desarrollo son: Dauntless [32], desarrollado por Phoenix Labs; Paragon [33], de Epic Games, que es el juego del que se han obtenido los principales recursos visuales y assets 3D; o Spellbreak [34], desarrollado por Proletariat Inc., un battle royale que se centra en combates mágicos.

3.5 Tendencias actuales y futuras

En el campo de los videojuegos en general, y en el caso de juegos de arena de batalla multijugador se están presentando diferentes tendencias emergentes tales como: la integración y desarrollo de las inteligencias artificiales (IA), para mejorar la experiencia del jugador contra oponentes controlados por IA, creando rivales más desafiantes y realistas; la exploración de diferentes formas de interacción mediante tecnologías de realidad aumentada o realidad virtual; o la implementación de juego cruzado entre diferentes plataformas, lo que mejora el alcance y la accesibilidad al juego al permitir a los jugadores competir independientemente de su dispositivo [35].

3.6 Referencias para el proyecto

3.6.1 League of Legends

League of Legends [22] es un juego desarrollado por Riot Games lanzado en 2009. League of Legends es un juego de arena de batalla multijugador en línea, disponible de manera gratuita para PC. Particularmente, su modo de juego Arena es la principal referencia en la que nos hemos basado para el desarrollo del juego.

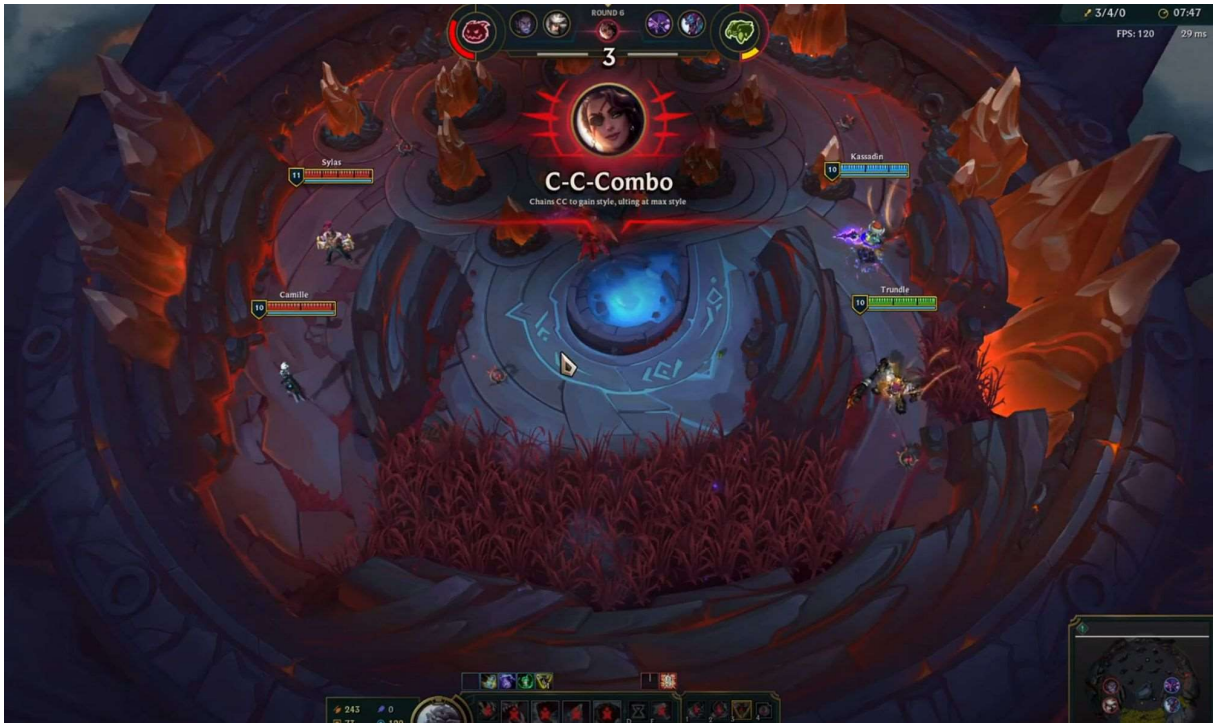


Ilustración 3 - League of Legends

3.6.2 Brawl Stars

Brawl Stars [36] es un juego desarrollado por Supercell y lanzado globalmente en 2018. El objetivo principal es conseguir el mayor número de trofeos mediante combates entre los brawlers, cada uno con habilidades únicas. El juego fue lanzado para las plataformas móviles: iOS y Android.

Ilustración 4 - Brawl Stars

3.6.3 Godstrike

Godstrike [37] es un juego desarrollado por OverPowered Team y lanzado en 2021 en las principales plataformas de PC. Es un juego de combate contra jefes basado en el tiempo, donde el tiempo es a la vez tu vida y tu moneda de cambio para mejorar tus habilidades. El juego presenta un *bullet hell* en cada combate en tercera persona con una vista cenital.

Ilustración 5 - Godstrike

3.6.4 Archero

Archero [38] es un juego desarrollado por Habby y lanzado para plataformas móviles en 2019. El objetivo principal del juego es la supervivencia frente a un entorno hostil donde todos te quieren eliminar.

Ilustración 6 - Archero

3.6.5 Pokemon Unite

Pokemon Unite [39] es un juego de arena de batalla multijugador en línea basado en la exitosa saga de juegos de Pokemon. Este juego es desarrollado por TiMi Studios y distribuido por The Pokemon Company y Nintendo, lanzado inicialmente en Nintendo Switch en 2021, y posteriormente lanzado también para dispositivos móviles.

Ilustración 7 - Pokemon Unite

4. Diseño del Juego

En este capítulo se expone el diseño inicial del videojuego, extraído del documento de diseño del videojuego (GDD), en el que se basa el desarrollo futuro del videojuego.

4.1 Información general

4.1.1 Descripción

Project Onyx es un juego de arena de combate 1 vs 1, 2 vs 2 o un modo de todos contra todos para 4 jugadores. En cada enfrentamiento, los jugadores competirán en varias rondas, y el ganador será el que logre reducir primero la vida general de su oponente.

Los luchadores en el juego utilizarán sus habilidades especiales para vencer en cada ronda, obteniendo mejoras a través de objetos que podrán usar en los combates siguientes. Este es un juego multijugador local que se desarrolla en una arena vista desde una perspectiva cenital o elevada, con una cámara que sigue la acción principal.

4.1.2 Público objetivo y plataformas

Enfocado en un público joven de entre 16 y 30 años, familiarizado con otros juegos de lucha y géneros como RPG y MOBA, que buscan una experiencia de juego más casual.

El desarrollo del juego se centra únicamente en la plataforma de PC, aunque es totalmente compatible con su adaptación a consolas como PlayStation, Xbox y Switch.

4.1.3 Género

Lucha, Arena de batalla, 1vs1, por equipos y batalla campal (*free-for-all*).

4.1.4 Características clave

- Partidas rápidas
- Gameplay casual
- Mapas reducidos
- Juego competitivo
- Efectos visuales
- Soporte para controladores
- Multijugador local

4.2 Jugabilidad

El juego se centra en la experiencia de combate multijugador a través de los diferentes modos de juego.

4.2.1 Modo 1 contra 1

Al iniciar la partida los jugadores seleccionan un luchador y deberán combatir a muerte haciendo uso de sus habilidades y hechizos. Cada partida se divide en rondas, donde cada ronda es un combate a muerte diferente y al acabar el combate, el jugador que queda en pie se declara ganador de la ronda y obtiene un bonus de monedas *onyx* frente al perdedor, que además de recibir menos monedas, recibirá una penalización sobre su vida de juego. Esta vida de juego es diferente a la salud de los luchadores, que al inicio de cada ronda partirán con la salud al completo, mientras que la vida del juego sería la vida que decide si ganas o pierdes la partida.

El sistema de vida total o vida del juego garantiza que cada partida se desarrolla en varias rondas de combate, lo que permite a los jugadores que han perdido en las primeras rondas remontar el marcador de vida total. Así, cada ronda es más importante que la anterior, ya que en cada ronda se resta una mayor cantidad de vida total al perdedor,

pudiendo recuperar en una cuarta ronda, por ejemplo, una cantidad de vida similar a la perdida en varias rondas anteriores.

Entre cada ronda de combates, los jugadores podrán comprar objetos para mejorar los atributos de combate en la siguiente ronda con las monedas obtenidas.

La partida termina cuando la vida total de uno de los jugadores llega a cero.

4.2.2 Modo 2 contra 2

Este modo comparte la misma estructura que el modo anterior en cuanto a sistema de vida general y rondas.

La diferencia principal es que en este modo combaten dos equipos de dos jugadores cada uno y la ronda acaba cuando mueren los dos jugadores de un mismo equipo. Al acabar las rondas se entregan el mismo número de monedas a cada jugador del equipo, entregando un bonus a los jugadores del equipo que ha ganado.

El sistema de vida general funciona como una vida compartida para ambos jugadores del equipo.

4.2.3 Modo todos contra todos de 4 jugadores

Al igual que los dos modos anteriores, la estructura de rondas y vida general se comparte, con la diferencia de que existen cuatro equipos de un jugador cada uno en la partida. El reparto de monedas es particular ya que el primero recibe una bonificación sobre los demás y el segundo y el tercero reciben un bonus sobre el último.

4.2.4 IA contra IA

De manera adicional, se añade este modo no jugable que permite probar los personajes controlados por una IA. Este modo se desarrolla con intención de poder crear y probar la IA que controla los personajes, y con la idea de emplearlo como un modo de ajuste para balancear los personajes en función de los resultados obtenidos.

4.3 Economía de juego

Como se ha comentado de forma general en los distintos modos de juego, al terminar cada ronda cada jugador recibe una cantidad de monedas en función de los resultados de la ronda. Estas monedas se llaman Onyx, en relación con el nombre del juego.

El primer jugador en morir es el que menos *onyx* recibe, 1500 al finalizar la ronda, el ganador recibe 2000 *onyx* más un bonus de la vida con la que termina la ronda, y si hay cuatro jugadores, los jugadores segundo y tercero obtienen 1750 *onyx*. El bonus para el ganador es la proporción de vida final sobre el premio total. Por ejemplo, si acabase la ronda con un 25% de su vida recibiría el premio de 2000 *onyx* más el 25% de 2000 *onyx* que sumaría un total de 2500 *onyx*.

Bonus de derrota

Por la acumulación de varias rondas perdidas el perdedor puede recibir un bonus de *onyx* para compensar la diferencia entre el ganador y el perdedor y evitar asi el efecto de bola de nieve de ganar varias rondas seguidas.

Para recibir el bonus el jugador debe acumular 3 rondas perdidas. En el modo de cuatro jugadores por quedar en segundo o tercer lugar se contará como 0.5 rondas perdidas.

La cantidad de bonus recibida es de 6000 *onyx* (4500 *onyx* extra). En el caso de entrar en bonus de derrota quedando segundo o tercero, se recibirán 4500 *onyx* (2750 *onyx* extra).

Sistema de vida general

La vida general de cada equipo es la que determina cuando finaliza cada partida. Por norma general la vida general se sustrae en mayor medida a los jugadores con peores resultados durante la ronda, y la cantidad de vida sustraída crece cada ronda que pase.

La fórmula para calcular la vida que se resta es la siguiente: el jugador que primero muere recibe la penalización del 100% del valor base, y los segundos y terceros jugadores reciben la penalización del 50% sobre el valor base. Este valor base se calcula a partir de las rondas disputadas. Entre la primera y la tercera ronda el valor base es de -10, hasta la

quinta ronda el valor base aumenta a -20, y a partir de la sexta ronda el valor base de la vida que se resta es de -30.

4.4 Personajes

Los personajes se caracterizan por tener habilidades y atributos únicos. Cada personaje cuenta con un set de cuatro habilidades únicas y cuatro habilidades de movimiento comunes a todos los luchadores. Todos los personajes cuentan con los mismos atributos, pero cada uno posee unos valores distintos para cada atributo en función de su clase o categoría.

Los luchadores se pueden clasificar en base al tipo de daño que aplican sus habilidades, pudiendo realizar daño físico, daño mágico o daño mixto; por su rango en función de si sus habilidades son mayoritariamente a distancia o por el contrario es un personaje melee; o según su rol, pueden incluirse en las categorías de: asesino, luchador, mago, tirador o tanque. Algunos personajes poseen habilidades de potenciación o escudo también.

Todos los *assets* empleados para el desarrollo del juego, al no centrarse el proyecto en el apartado visual, sino en el apartado técnico, han sido obtenidos en el Marketplace de Unreal con licencia de uso libre para proyectos realizados con el motor Unreal Engine. Estos personajes pertenecen al juego Paragon [33] (Epic Games,2016), y los paquetes de contenido incluyen modelos 3D, animaciones, efectos y texturas. Para poder hacer uso de este contenido se ha tenido que adaptar la parte técnica del mismo, desarrollando la lógica de las diferentes animaciones y efectos que se han incluido, así como la lógica de las habilidades en sí, que no se incluyen en el paquete de *assets*.

Los personajes incluidos en el juego [40] y su clasificación es la siguiente:

- **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.

Los atributos presentes en todos los personajes, donde cada personaje poseerá valores únicos, son los siguientes:

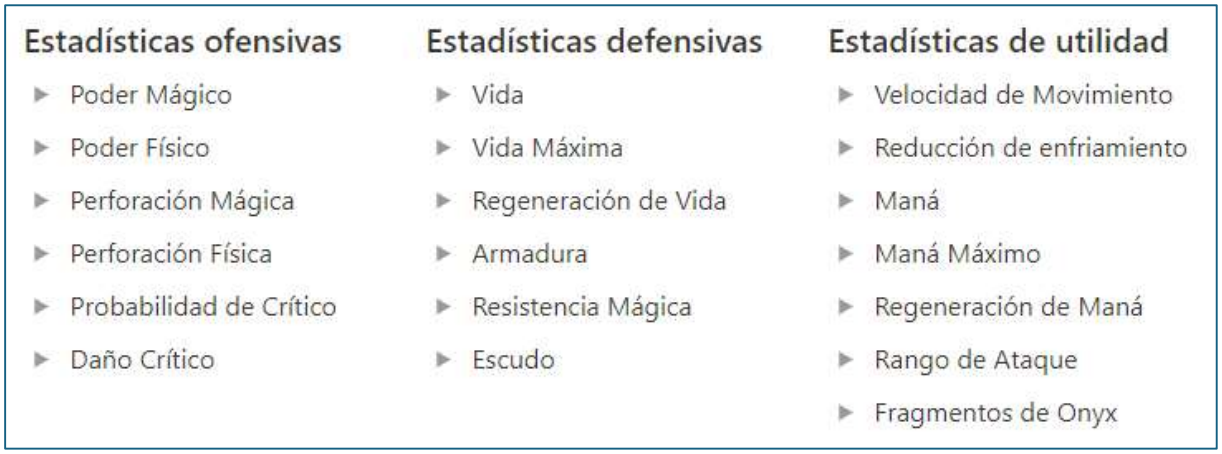


Ilustración 8 - Lista de atributos de los personajes

4.5 Habilidades de movimiento

Cada personaje posee un set de habilidades únicas que los diferencia de los demás, pero todos comparten un set de habilidades de movimiento común que les permite curarse durante la partida, mejorar su velocidad de movimiento, ralentizar a los rivales o huir y esquivar ataques rivales con un gran salto.

Este set se compone de las siguientes habilidades:

- *HiperDash*: permite al jugador teletransportarse en un rango de 400 unidades en la dirección en el la que apunta. Esta habilidad tiene un tiempo de enfriamiento de 30 segundos después de su activación.
- *HiperActivity*: aumenta la velocidad de movimiento del jugador durante 15 segundos. La velocidad aumenta en un 48,12% sobre la velocidad de movimiento habitual. Tras su activación el tiempo de enfriamiento es de 60 segundos.
- *Healing*: al activarse, el jugador se cura 220 unidades de vida de forma progresiva durante 10 segundos de duración. Además, se aplica un escudo de 285 unidades de vida sobre el jugador en el momento de la activación. Esta habilidad solo se puede activar una vez durante cada ronda.
- *Exhaust*: esta habilidad reduce la velocidad de movimiento de todos los enemigos que se encuentren en el rango de 650 unidades, disminuyendo la velocidad de movimiento en un 40%. De forma adicional, esta habilidad romperá cualquier escudo aplicado sobre los enemigos que se encuentran en el rango de la habilidad. Esta habilidad solo se puede activar una vez por ronda.

Estas habilidades permiten a los jugadores una amplia variedad estratégica.

4.6 Objetos

Tras cada ronda de combate, con los fragmentos de *onyx* obtenidos al finalizar la ronda, los jugadores podrán comprar objetos para mejorar sus atributos. Estos objetos se pueden comprar en repetidas ocasiones hasta un máximo de 7 objetos por personaje. Existen 8 objetos diferentes, que se adaptan a las diferentes clases y categorías de luchadores.

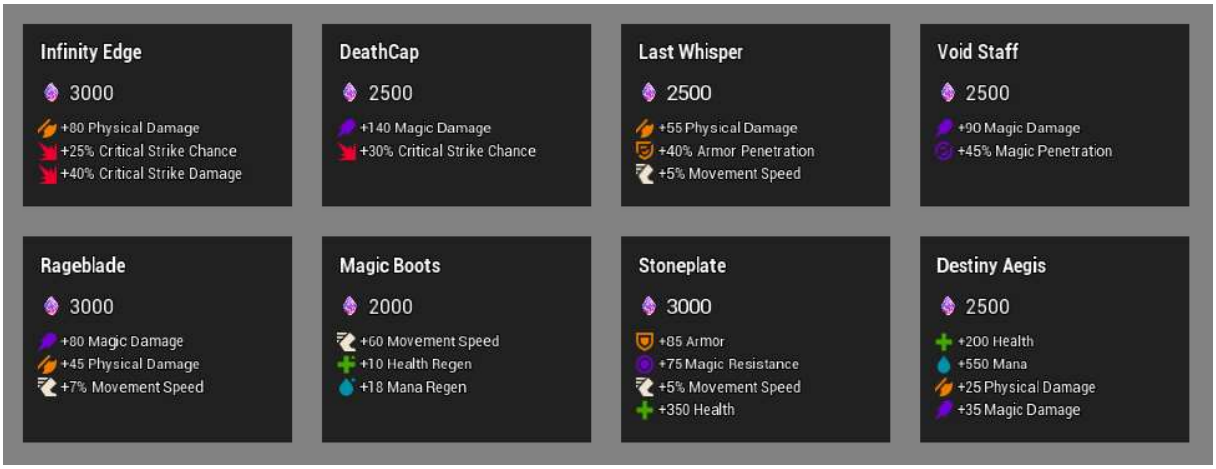


Ilustración 9 - Objetos de la tienda

4.7 Inteligencia Artificial

En un principio se planteó la inteligencia artificial en el proyecto como dos tipos distintos de inteligencia artificial simples donde cada alumno se encargaría de desarrollar uno. Un alumno se encargaría de desarrollar una IA capaz de controlar enemigos neutrales simples, al estilo de súbditos en un juego de oleadas, que otorgasen una recompensa al jugador que los eliminase. Por otra parte, el otro alumno desarrollaría una IA que ayudase a un personaje concreto, siendo su guardián y apoyándole con escudos o curaciones.

Durante el desarrollo del proyecto, se valoró cambiar la idea inicial de incluir IA de forma simplificada, con la idea de crear una única IA capaz de controlar de manera autónoma los propios campeones del juego.

Por una parte, se rechazaron las ideas previas porque no aportaban valor a los modos de juego que queríamos crear. Los súbditos neutrales no encajaban con la idea de mapas y rondas que habíamos desarrollado, y el personaje con IA incorporada se distanciaba mucho del resto de personajes, cambiando por completo su jugabilidad y restándole importancia a la parte del combate programada con GAS, que es el foco principal del proyecto.

Por otra parte, pensamos que una IA capaz de controlar los personajes podría ser de gran utilidad a la hora de crear peleas entre diferentes IA para valorar de una forma más práctica que personajes eran consistentemente mejores en el juego y, por lo tanto, que personajes habría que balancear. En caso de que en el desarrollo futuro se consiguieran buenos resultados con la IA obtenida, esta podría ser la base para crear un modo de un jugador contra la IA para hacer posible jugar al videojuego de forma individual.

En el desarrollo de esta IA más compleja, ambos alumnos se han encargado de colaborar en la implementación de esta. Uno ha creado la base del comportamiento general desarrollando el árbol de comportamientos, mientras que el otro se ha dedicado a explorar el sistema de *Eviroment Query System* (EQS) para crear comportamientos más realistas a la hora de definir el movimiento de los personajes y adaptarlo a la maya de navegación.

4.8 Controles

El juego se encuentra adaptado para jugar con dos o más mandos. Al ser un juego multijugador local con una extensa lista de controles no sería posible adaptar la jugabilidad a teclado y ratón. En el **Anexo I** se puede consultar el esquema de controles para los diferentes mandos.

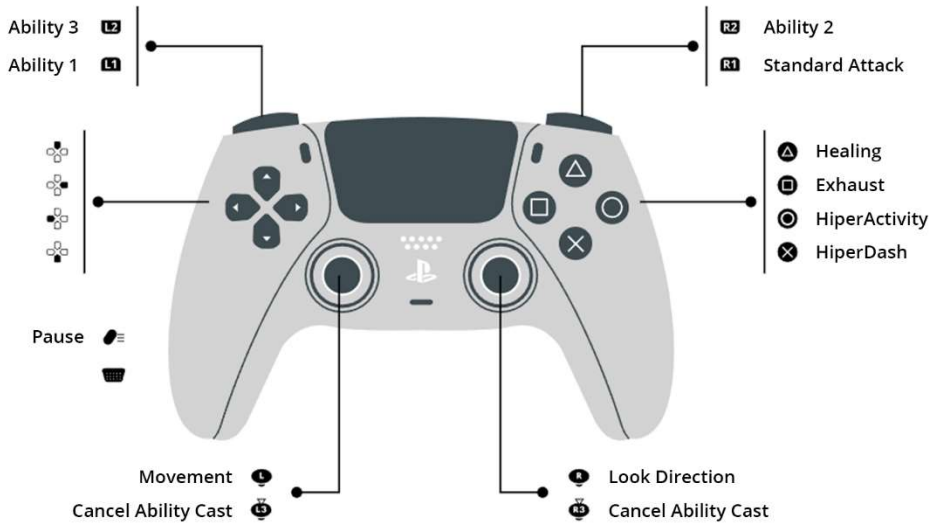


Ilustración 10 - Controles sobre un mando DualSense

5. Desarrollo del juego

En este capítulo se detalla la implementación de los principales apartados del proyecto, haciendo énfasis en los apartados mayoritariamente desarrollados por mí.

5.1 Arquitectura general de Unreal

Para crear la estructura del proyecto se parte de la base de la estructura de clases principales de Unreal Engine [41], mostrada en la ilustración 11. A partir de esas clases, se ha implementado nuestra propia estructura para el proyecto, creando clases hijas que heredan el comportamiento general de estas clases y lo expanden para adaptarlo a nuestro juego. Para nombrar las clases del proyecto se han seguido las convenciones de nomenclatura de Unreal [42].

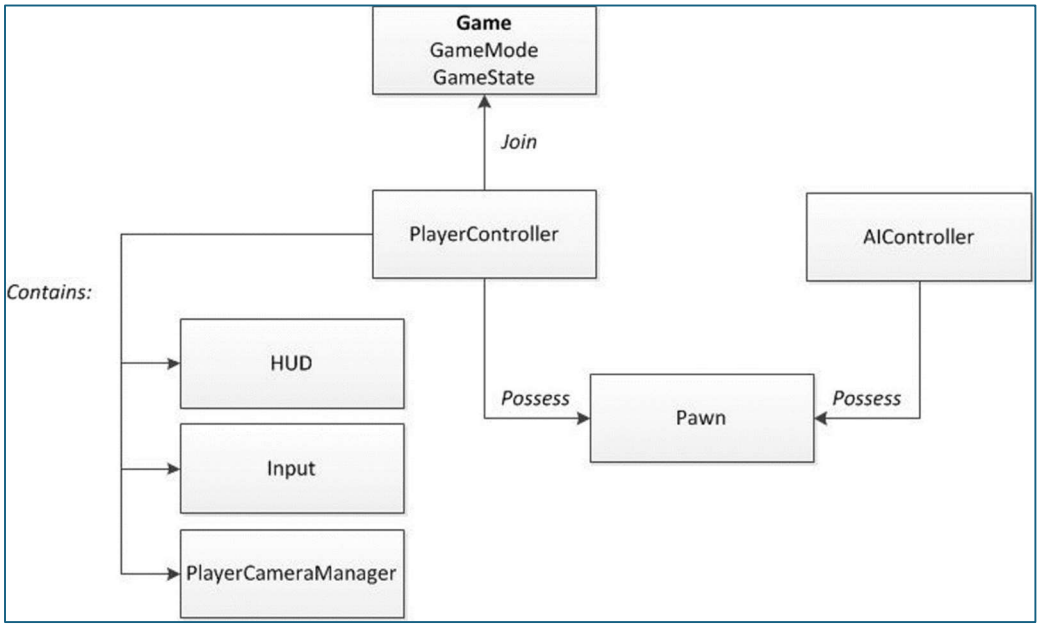


Ilustración 11 - Estructura de clases de Unreal

5.1.1 Game Instance

Game Instance actúa como un *singleton* global del juego, y es persistente a través de los niveles. Se emplea para almacenar datos que tienen que persistir a lo largo de los niveles, como estadísticas de los jugadores, datos de inicialización del juego e información relevante para distintos niveles del juego.

En el caso particular de nuestras partidas, se encarga de guardar los atributos de monedas, vida general y la lista objetos comprados por los jugadores a lo largo de las rondas, a parte de hacer de puente entre el menú de selección de modo, el menú de selección de personajes y las rondas de combate.

A parte, cuenta con métodos que se encargan de modificar estas variables como, por ejemplo, un método que reinicia todos los valores a los valores por defecto cuando acaba una partida para poder empezar una partida nueva.

5.1.2 Game Mode

Game Mode se encarga de definir las reglas del juego y gestionar la interacción de los personajes con el nivel. A efectos prácticos, se podría definir como un mánager de juego. Al principio de las rondas, es el encargado de *spawnear* los personajes y asignar los controladores. Al ser un juego multijugador local, la interfaz tambien es compartida para todos los personajes y será el *Game Mode* el encargado de inicializarla y actualizarla a partir de eventos que envían los personajes.

Al ser encargado de gestionar las reglas del juego, dirige la lógica de otorgar monedas a los jugadores de la partida cuando acaba una ronda y gestiona también la vida general de todos los equipos de la partida.

Cada nivel posee un *Game Mode* único que se crea al cargar el nivel, por lo que en el juego existe un *Game Mode* para cada uno de los niveles. En los niveles que no son de combate como, por ejemplo, en la selección de luchadores o en la tienda, el *Game Mode* se encarga de gestionar las interfaces que se muestran por pantalla.

5.1.3 Player Controller

Las clases *controller* representan lo que seria la mente de un *Pawn*. En el caso de los *Player Controller*, sirven de puente entre las acciones del jugador y las acciones del personaje que controlan. Se encargan de gestionar la entrada del jugador para realizar acciones sobre los personajes. Tambien contienen referencias a las clases que aportan una salida al jugador, como la cámara o la interfaz.

5.1.4 Pawn / Character

Pawn es una clase que deriva de Actor que representa a las entidades que pueden ser poseídas o controladas por un controlador, ya sea un jugador o una IA. En el caso de la clase *Character*, es una derivación de *Pawn* que se usa para los personajes humanos o humanoides, que se caracteriza por contar con *Character Movement Component*.

5.1.5 AI Controller

Al igual que el *Player Controller* representa la mente de un *Pawn*. En el caso del AI *Controller* se encarga de comunicar las decisiones que toma la inteligencia artificial al personaje que controlan.

En el caso concreto del proyecto se encarga de crear la *BlackBoard* que usará el *Behaviour Tree* e inicializar sus variables, y se encargará de ejecutar el *Behaviour Tree*.

5.1.6 Actor

La clase Actor representa cualquier entidad física dentro del juego, es decir, cualquier objeto cuyas instancias podrían formar parte de un nivel. Esta clase comparte en común la transformación del actor, que representa la posición, rotación y escala del objeto en el nivel.

Su funcionalidad puede ser complementada por los diferentes componentes de comportamiento, y es capaz de ejecutar lógica del juego en cada *frame* a través de la función *Tick*.

5.1.7 UObject / Component

La clase *UObject* es la clase base de Unreal Engine, que genera un soporte para la serialización, replicación y gestión de memoria para todas las clases que derivadas. *Actor Component* es una clase que hereda de *UObject* que se encarga de extender la funcionalidad de los actores sin tener que heredar en clases complejas. Esto permite un desarrollo más flexible y modular, y que a cada tipo de actor se le añaden los componentes que necesita.

Algunos ejemplos de los componentes más usados pueden ser:

- *UStaticMeshComponent*: añade una maya estática al actor
- *UCameraComponent*: añade una cámara al actor
- *USphereComponent*: añade un volumen de colisión con forma esférica

5.2 Estructura de clases de GAS

El *plugin* Gameplay Abilities [2], también conocido como Gameplay Ability System (GAS) añade al proyecto una serie de clases que sirven de base para el desarrollo de un sistema de combate escalable, flexible y modular. A partir de estas clases, se han creado clases propias que heredan y expanden la lógica para adaptar la implementación de este *plugin* a los objetivos de nuestro videojuego.

5.2.1 Ability System Component

Es la clase principal del *plugin* GAS. Cualquier actor que quiera implementar alguna funcionalidad del *plugin* GAS, ya sea lanzar habilidades, tener atributos o recibir efectos, deberá contar con un componente Ability System. Es el encargado de manejar cualquier interacción del actor con el sistema de combate.

En el caso particular del proyecto se ha creado una subclase, de cara a mantener la escalabilidad del proyecto, por si en un futuro fuese necesario que el componente realizase lógica específica para el contexto de nuestro juego [43].

5.2.2 Gameplay Ability

Las *Gameplay Abilities* representan cualquier acción o habilidad que es capaz de ejecutar el actor. Algunos ejemplos de habilidad que podría realizar un jugador para un juego serían: saltar, correr, disparar, abrir una puerta, construir o consumir una poción.

Las *Gameplay Abilities* incorporan funcionalidades por defecto, como el nivel de la habilidad, el efecto de *cooldown* o el efecto de coste de habilidad. Todas las habilidades cuentan con funciones como *ActivateAbility*, donde se puede implementar la lógica de la habilidad o *EndAbility* que se ejecuta cuando la habilidad es cancelada o termina de forma natural. Todas las habilidades pueden interactuar con otras habilidades bloqueando su activación, cancelándolas o activándolas en cadena.

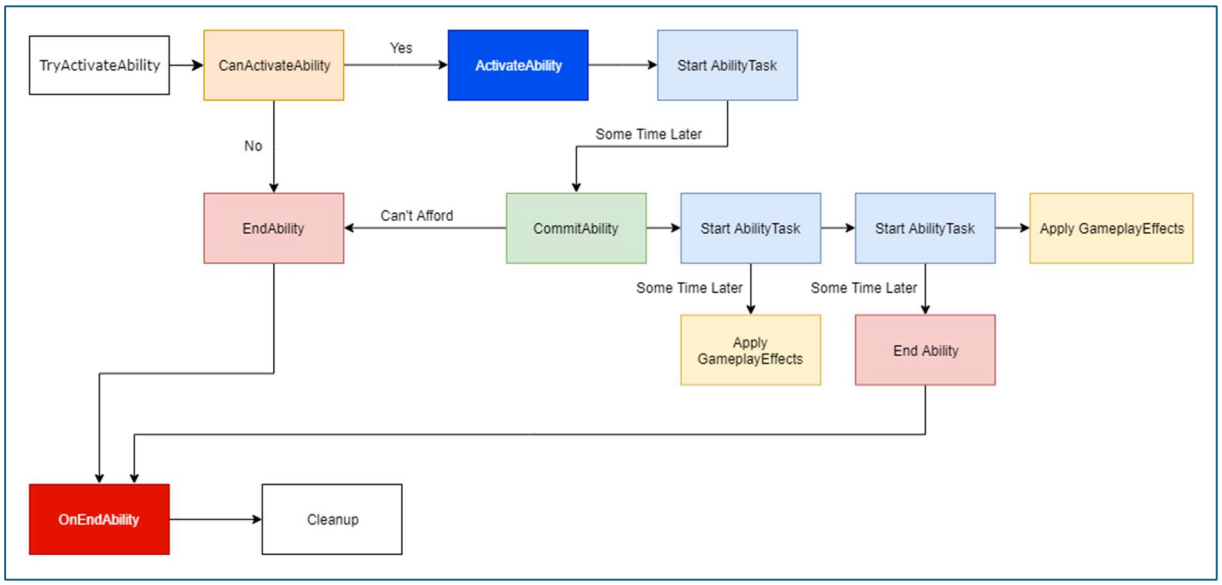


Ilustración 12 - Diagrama de flujo de un ejemplo de habilidad

5.2.3 Attribute Set

Los atributos son un valor *float* que representan cualquier valor numérico que pueda ser relevante dentro del juego. Estos atributos pueden representar la vida del personaje, la velocidad de movimiento o el número de activaciones de una habilidad.

Los atributos están definidos por una estructura que se compone de dos valores. El valor de base y el valor actual. El valor base es el valor real del atributo sin tener en cuenta modificaciones temporales, mientras que el valor actual es el valor que alcanza el atributo cuando se le aplican modificaciones temporales. Si un efecto modifica un valor de forma instantánea, este cambio afecta al valor base, pero si el efecto modifica un valor de forma temporal, se modificará el valor actual para que cuando se acabe el efecto recupere el valor almacenado en el valor base. Esto puede ser útil para potenciaciones de atributos como la velocidad de movimiento, que permite aumentar la velocidad durante unos segundos y luego vuelve a su valor anterior.

Los atributos no pueden existir fuera del contexto del *Attribute Set* o set de atributos. Esta clase se encarga de definir y manejar los cambios en los atributos. Los atributos en los personajes se pueden gestionar en diferentes sets de atributos, en caso de que se quiera modularizar los atributos de cada personaje. En el caso de nuestro proyecto, se ha implementado un único set de atributos para todos los actores que usan GAS porque crear diferentes sets no aportaba ningún valor añadido al proyecto.

5.2.4 Gameplay Effect

Los *Gameplay Effect* son los encargados de realizar cambios en los actores, cambiando el valor de los atributos, asignando diferentes tags al que realiza el efecto o al que lo recibe, aplicando efectos de estado o aplicando potenciaciones o debilitaciones. Los efectos no tienen lógica añadida, solamente se encargan de cambiar atributos con modificadores y aplicar etiquetas. Cada efecto puede ser, según su duración, instantáneo, temporal o infinito. Además, los efectos no instantáneos pueden ejecutarse de forma periódica. Los temporales,

aplicaran los periodos disponibles hasta que se acabe el tiempo de ejecución, y los infinitos ejecutarán constantemente estos periodos a no ser que el efecto sea eliminado del actor.

Siguiendo diferentes ejemplos de nuestro juego, hacer daño a un personaje rival sería un efecto instantáneo, lanzar una habilidad de mejora de movimiento aplica un efecto temporal, lanzar una habilidad de curación aplica un efecto temporal que se ejecuta periódicamente, la recuperación pasiva del maná del personaje sería un efecto infinito que se ejecuta periódicamente y el efecto que aplica la etiqueta para reconocer a que equipo pertenece cada personaje sería un efecto infinito.

5.2.5 Gameplay Tags

Las *Gameplay Tags* son etiquetas específicas del *plugin* GAS. Estas *tags* son muy útiles a la hora de definir efectos de estado, reconocer habilidades, gestionar cooldown, reconocer efectos activos o asignar *Gameplay Cues* a los personajes.

Todas las clases mencionadas anteriormente interactúan con las *Gameplay Tags* de diferentes maneras, siendo así una de las herramientas más útiles que posee el plugin GAS. Algunos ejemplos de uso de las etiquetas en habilidades y efectos son el uso de *tags* para gestionar los *cooldowns* de las habilidades, bloquear otras habilidades si ya hay alguna habilidad con un *tag* específico activa, bloquear efectos si el actor posee una *tag* concreta o encontrar efectos concretos a partir de las *tags* que poseen para desactivarlos.

5.2.6 Gameplay Cues

Los *Gameplay Cues* se encargan de ejecutar contenidos no relacionados con el *gameplay*, como puedan ser efectos visuales, sonidos, partículas, modificaciones de cámara, etcétera.

Son especialmente útiles por su fácil activación dentro del entorno de GAS, al poderse activar directamente desde la aplicación de un efecto. Si el efecto es temporal, al iniciar el efecto se activará el *Gameplay Cue* y al finalizar el efecto se desactivará y eliminará el

Gameplay Cue. Para asociar un efecto a una *Gameplay Cue* solo hace falta el uso de una *Gameplay Tag* que identifique al *Gameplay Cue* que se quiere ejecutar con el efecto.

5.3 Estructura de personajes y actores interactuables

En el proyecto, los personajes y actores con los que se puede interactuar deben tener una estructura base común que incluya las principales clases del *plugin* GAS. Para su desarrollo se han creado las siguientes clases en C++:

- *Onyx Character*: esta clase es la base para todos los personajes que posteriormente heredan de ella.
- *Onyx Actor*: esta clase es la base para todos los actores que posteriormente heredan de ella.

Se puede consultar un diagrama UML inicial en el **Anexo I**.

5.3.1 Estructura general

Todas las clases mencionadas anteriormente que implementan el *plugin* GAS emplean la interfaz *IAbilitySystemInterface* que permite acceder al *Ability System Component* desde cualquier parte del proyecto. Como se menciona anteriormente, cualquier actor que use GAS debe tener un componente *Ability System*, ya que es el que gestiona la lógica principal del *plugin*.

A parte del componente, los personajes y actores poseen un set de atributos de la subclase *OnyxAttributeSet* que implementa los atributos principales del proyecto. Para asignar los valores iniciales de los atributos de cada personaje se emplea un efecto de inicialización y, además, se incluye una lista de efectos que se puede rellenar con efectos que queremos que se ejecuten sobre el actor o personaje desde el momento inicial en el que comienza la partida. Esta lista permite incluir efectos pasivos, como la recuperación de vida o maná base, y aplicar etiquetas perpetuas si fuese necesario. Todas estas variables

poseen especificadores de propiedades [44] para ser visibles y editables desde las clases *Blueprint* en el editor.

De forma particular, la clase definida para los personajes posee una *Gameplay Tag* para indicar a que equipo pertenece, que se asigna desde el *Game Mode* en la creación de los personajes, y un efecto que se aplica al morir para cancelar todos los efectos de estado y pasivos y asignar una etiqueta que identifique al personaje como muerto. Además, incluye una lista con todas las habilidades que puede lanzar el personaje en cuestión.

Además, al ser necesario mantener una comunicación con la interfaz general del juego, la clase de la que heredan todos los personajes posee diferentes eventos para actualizar la interfaz visual del juego. Estos eventos se suscriben a delegados del set de atributos que se emiten cuando un valor cambia.

5.3.2 Inicialización

Para inicializar los actores y personajes, se crea el componente de Ability System y el set de atributos en el constructor de la clase. Desde el método BeginPlay, que se lanza al comenzar la partida, se suscriben los eventos a los delegados del set de atributos, se inicializan los valores de los atributos con el efecto especificado y se aplican los efectos iniciales indicados en la lista de efectos. Posteriormente, se otorgan las habilidades a los personajes, ya que los actores no poseen ninguna habilidad en el juego.

Es importante otorgar las habilidades a los personajes, ya que los personajes no pueden ejecutar ninguna habilidad que no se les haya otorgado con posterioridad.

5.3.3 Movimiento y apuntado

Al ser un proyecto en vista cenital con una única cámara, ya que todos los jugadores comparten la misma pantalla de juego, el movimiento de los jugadores es diferente a un juego en tercera o primera persona, donde cada jugador controla su propia cámara.

Para este caso particular, el personaje se mueve a través del input del joystick izquierdo del mando del jugador, y ese movimiento se transmite en coordenadas del mundo para los

personajes. Es decir, al mover el joystick hacia arriba el personaje se desplaza en el eje x del mundo, desplazándose hacia arriba en la pantalla, no en el eje local del personaje, que se desplazaría hacia la dirección a la que estuviese mirando.

Para el apuntado es similar, ya que se controla con el joystick derecho del mando. La mayor diferencia es que no se desplaza la cámara, por lo que no se trata la rotación como incrementos particulares, si no como una dirección objetivo a la que el jugador pretende llegar. Al mover el joystick derecho, se almacena la última dirección en la que se ha apuntado, y a través del evento *Tick* se actualiza la orientación del personaje con una interpolación de cuaterniones que sigue una velocidad asignada en el desarrollo, para evitar transiciones agresivas.

5.3.4 Previsualización de habilidades

Para mejorar el *feedback* visual de los jugadores al tirar habilidades se ha incluido una previsualización de cada habilidad. De esta forma, el jugador puede saber si su habilidad impactará antes de lanzarla, y cual es su recorrido total.

Para implementarlo, se ha tomado la decisión de lanzar las habilidades cuando se suelta el botón asignado a la habilidad y no en el momento de su pulsación. De esta forma mientras el botón permanece pulsado, se muestra la previsualización de la habilidad, y al soltarse el botón se lanza la habilidad.

Estas previsualizaciones se han desarrollado con un componente *decal* para cada habilidad, que muestra sobre la arena de combate una proyección vertical del área en el que la habilidad tendría efecto.

5.3.5 Herencia y polimorfismo

Para favorecer la escalabilidad del proyecto, la clase *Onyx Character* es la clase base donde se crea la estructura base de GAS y de esta derivan todas las clases de los personajes a partir de herencia.

La principal clase *blueprint* del proyecto es *Fighter* que es la primera clase *blueprint* que hereda de *Onyx Character*. En esta clase se incluye la lógica general de los inputs, la lógica de movimiento y apuntado, y algunas variables necesarias en otros apartados del proyecto. Además, se crean eventos personalizados, que a efectos de C++ serían métodos virtuales, que se implementan en las diferentes clases hijas.

Cada personaje posee una clase particular de *blueprint* que hereda de la clase *Fighter*. En esta clase se cambian todas las propiedades particulares de cada jugador en concreto, y se definen los diferentes eventos personalizados a partir del polimorfismo de clases. Así cada clase gestiona eventos como lanzar un proyectil o aplicar la colisión de un área de daño de manera única.

5.4 Sistema de vida y daño

El sistema de daño es una base principal del combate en el juego. Se ha desarrollado un sistema complejo que tiene en cuenta las características de ataque y defensa de los personajes. Para llevarlo a cabo se ha usado como referencia un *framework* con GAS [45] basado en el juego Risk of Rain 2 [30].

5.4.1 Atributos principales de defensa y ataque

El sistema de daño se sustenta en los diferentes atributos defensivos y ofensivos que poseen los personajes. En el juego existen dos categorías de daño, el daño físico y el daño mágico. Cada personaje aprovecha mejor uno de estos en función de su arquetipo, aunque hay personajes con daño mixto que pueden aprovechar ambos.

Por un lado, los atributos ofensivos son el poder físico, el poder mágico, la penetración de armadura, la penetración de resistencia mágica, la probabilidad de crítico y el daño crítico. El poder físico y mágico es la base de fuerza de cada tipo de daño que posee el personaje. La mayoría de los personajes escalan el valor de daño de sus habilidades con estos atributos.

Los atributos de penetración física y mágica representan el valor porcentual de armadura o resistencia mágica que pueden reducir al aplicar el daño, útil para superar a los tanques con estadísticas defensivas. La probabilidad de crítico es el porcentaje de éxito de que un ataque básico golpee crítico, y el daño crítico es el porcentaje extra de daño que aplica un ataque crítico.

Por otro lado, los atributos defensivos son la vida, la vida máxima, la regeneración de vida, la armadura, la resistencia mágica y el escudo. La vida representa los puntos de vida que posee el personaje y la vida máxima la cantidad de puntos de vida que puede alcanzar como límite. La regeneración de vida es el valor de puntos de vida por segundo que recupera de forma pasiva el personaje. La armadura y resistencia mágica es un valor que representa la cantidad de daño físico o mágico que puede mitigar cada personaje. El valor del daño final aplicado en el ejemplo del daño físico seguiría la siguiente fórmula:

$$Damage * \frac{100}{100 + Armor * (1 - ArmorPenetration)}$$

Donde la armadura se ve reducida cuanto mayor es el valor de la penetración y los puntos de armadura son cada vez menos eficientes a la hora de reducir daño. Poniendo de ejemplo un valor de 100 de armadura y 0 de penetración, el daño mitigado sería del 50%, mientras que si se amplía la armadura hasta 200 el daño mitigado sería de 67% siendo mucho menos eficiente la compra. El escudo es un valor de vida adicional con carácter temporal que se va desgastando de manera progresiva hasta desaparecer. El escudo permite superar la vida máxima ya que es independiente al valor de vida.

5.4.2 Sistema de aplicación de daño

Para aplicar el daño a los personajes se añaden dos meta atributos que se encargan de transferir la información del daño. Estos atributos son el daño físico y el daño mágico. Su

función en el set de atributos es que las habilidades no modifiquen directamente la vida de los personajes, si no que sea el set de atributos quien gestione el daño ocasionado.

De esta forma, las habilidades modifican el atributo daño físico o daño mágico y es el set de atributos el que se encarga de gestionar la lógica. Cuando uno de estos valores cambia se reinicia su valor nuevamente a cero y se procesa el valor recibido, de tal forma que se multiplica por el coeficiente de armadura mostrado anteriormente y después se comprueba el valor del escudo. Si tiene escudo se aplica primero el daño al escudo, y si sobra daño se aplica sobre la vida del personaje.

De esta forma las habilidades no se tienen que preocupar de cuáles son las características del personaje al que afectan, ya que es el mismo personaje el que computa el daño y aplica la resta de vida pertinente.

5.4.3 Gestión de equipos

Para gestionar que los personajes sean capaces de reconocer a sus compañeros de equipo, ya que no existe el fuego amigo dentro del juego, se ha añadido una etiqueta que indica a que equipo pertenece el personaje.

Esta etiqueta se gestiona desde el *Game Mode*, que es el encargado de crear a los jugadores y conoce el modo de juego y los equipos a los que pertenece cada jugador.

A la hora de aplicar daño en una habilidad se procede a comprobar las etiquetas del atacante y el objetivo y si no coinciden se ejecuta el efecto del daño.

5.4.4 Gestión de muerte

Cuando uno de los jugadores muere se tiene que notificar al *Game Mode* para gestionar las reglas de partida, y además se tiene que desactivar el uso del personaje durante la partida.

Para gestionar la muerte de un personaje, se usa el evento vinculado al delegado del atributo de vida. Este identifica cuando el valor de la vida llega a cero y lanza otro evento que reproduce una animación de muerte, sustrae el control del jugador, retira cualquier

efecto de estado o pasivo y añade una etiqueta de muerte al personaje. Cuando termina la animación de muerte se desactivan las colisiones y la visibilidad del personaje.

Por otra parte, el *Game Mode* es el encargado de gestionar cuando acaba la partida en función de las muertes. Si solo queda un jugador vivo o todos los jugadores son del mismo equipo se acaba la ronda se procede a realizar la lógica de puntuación y recompensas.

5.5 Sistema de habilidades

Aunque cada jugador tenga un set único de habilidades, estas se pueden categorizar en función de una estructura que se repite entre diferentes habilidades de distintos personajes. Estas estructuras que agrupan el tipo de habilidad concreta se definen en los siguientes apartados.

5.5.1 Habilidades de área de colisión

Este tipo de habilidad es típica de los personajes que golpean a corto rango, aunque otros personajes como los magos, también pueden poseer habilidades de área.

Se caracterizan por realizar el daño a los enemigos que se encuentran dentro del área de efecto. En caso de ataques a melé, será una caja de dimensiones pequeñas que se encuentra delante del personaje, y en el caso de habilidades en área será una esfera o una caja que ocupe el área en el que tiene efecto la habilidad. Estas primitivas de colisión son componentes de los personajes en su clase *blueprint*.

Cuando se activa la habilidad, se reproduce una animación de ataque, y posteriormente, mientras se reproduce la animación se espera a recibir un *Gameplay Event*. Durante la animación se define en el punto exacto de impacto un evento de animación. Este evento llama al evento personalizado de la clase *Fighter* con la información de cuál ha sido la habilidad activada. Este evento se define en cada una de las subclases de los personajes, donde se comprueban todos los actores interactuables que se encuentran en colisión con el área de la habilidad. Con la lista de actores se genera un *Gameplay Event* que especifica

quién lanza la habilidad, y una lista con todos los actores afectados. Cuando la habilidad recibe el *Gameplay Event*, crea un efecto encargado de aplicar el daño. A este efecto se le aplica la cantidad de daño realizando los cálculos previos pertinentes de cada habilidad, teniendo en cuenta daños base, escalados, estados, etcétera. Una vez creado el efecto, se aplica a cada uno de los actores que recibe en la lista de objetivos de la habilidad. Al acabar la animación, se acaba la habilidad, lo que permite lanzar la siguiente habilidad.

5.5.2 Habilidades de proyectil

En el caso de las habilidades con proyectil, la estructura general es similar, pero cambia en la parte de aplicar los efectos, ya que no los aplica la habilidad, si no el proyectil. Estas habilidades son características de tiradores y magos, que pueden disparar balas, o invocar magias.

Cuando se activa este tipo de habilidad, se reproduce la animación de ataque y se espera a un *Gameplay Event*. Al igual que en otras habilidades, en el momento de invocar el proyectil se crea un evento de animación, que en la declaración de la subclase del personaje crea un *Gameplay Event* con la etiqueta proyectil. La habilidad recoge el evento y se encarga de *spawnear* un actor proyectil, que hereda de la clase *Onyx Projectile*. En este actor se incluye como instigador, el actor que lanza la habilidad, como rango la distancia que recorre el proyectil antes de destruirse, y como efecto de daño, el efecto que aplica el proyectil al colisionar con otro actor. Este efecto se crea en la habilidad teniendo en cuenta los valores de daño y otros atributos del personaje.

Al terminar la animación se acaba la habilidad. De esta forma, aunque un proyectil aún este en el aire el personaje puede lanzar una nueva habilidad. Si la habilidad esperase a que colisionase el proyectil para realizar el daño, esta habilidad se quedaría bloqueando al resto hasta que desapareciese cada proyectil.

5.5.3 Habilidades de spawn de área de colisión

Este tipo de habilidades son muy similares a las habilidades de proyectil, con la diferencia que una vez se produce el *spawn* del área de colisión, esta no tiene físicas de movimiento y permanece inmóvil.

La estructura general de la habilidad es la misma, pero en lugar de hacer *spawn* de un proyectil, lo hace de un actor específico que posee el área de colisión. Al igual que al proyectil, se le otorga al actor un efecto con la información del daño que tiene que aplicar si colisiona con otro actor. Cuando el actor de la habilidad activa la colisión comprueba si hay colisión al igual que en las habilidades de área de colisión, y ha la lista de actores que colisiona les aplica el efecto definido por la habilidad. De forma adicional, este actor posee un *decal* que indica el área en el que tiene impacto de una manera más clara que el efecto en sí, para que los jugadores rivales sepan con certeza cuál es el rango de la habilidad.

5.5.4 Habilidades de potenciación

En estas habilidades se potencian los atributos del instigador, o se mejora el funcionamiento de otras habilidades.

En el caso de mejorar alguna característica del que lanza la habilidad, se aplica un efecto temporal con las mejoras aplicadas. En caso de que la mejora afecte a otras habilidades, como pueda ser una mejora de rango o de velocidad de ataque, se indica aplicando un efecto temporal que otorga una etiqueta al actor. De esta forma, desde el resto de las habilidades, antes de activarlas se comprueba si el personaje posee el efecto de estado que mejora la habilidad, y si es así, cambia los parámetros deseados, como pueda ser el rango o el cooldown.

5.5.5 Habilidades de escudo

Estas habilidades suelen ser habilidades simples, similares a las habilidades de potenciación. Al iniciar la habilidad, se reproduce una animación o se aplica directamente un

efecto. En este efecto se añade la cantidad de escudo que otorga la habilidad, en función de los atributos del personaje, un valor baso un escalado, según esté definido en cada habilidad.

Este efecto aplica una *Gameplay Cue* que muestra en sistema de partículas de un escudo esférico sobre los personajes a los que se aplica. Cuando el escudo vuelve a cero se elimina la *Gameplay Cue* y se muestran unas partículas de escudo roto.

5.5.6 Habilidades de movimiento

Las habilidades de movimiento son comunes para todos los personajes y en gran parte combinan los demás tipos de habilidades en ellas.

El curar aplica una potenciación que mejora la curación base durante unos segundos y, además, aplica un escudo al ser lanzado.

El extenuar aplica una reducción de movilidad a aquellos enemigos que se encuentren en un área de colisión.

La hiper velocidad aplica una potenciación de velocidad de movimiento.

El hiper salto es el único que no emplea una estructura similar al resto de habilidades, ya que calcula el punto transitable más alejado dentro del rango definido, siguiendo la dirección a la que apunta el personaje, y le teletransporta a esa posición.

5.6 Luchadores: Tiradores y Magos

En el reparto inicial de tareas se asignaron los personajes a desarrollar por cada alumno en función de sus arquetipos. En mi caso particular, se me asignaron los personajes magos y tiradores, que comparten características comunes como el amplio rango de alcance. Generalmente, la mayoría de las habilidades de estos arquetipos son de proyectil, aunque también poseen habilidades de área, potenciación, escudo y *spawn* de área de colisión.

5.6.1 Grim.exe

Este personaje pertenece a la categoría de tirador, ya que dispara pulsos de energía con usando su rifle de gran alcance y canalizando la energía a través de su cuerpo. Destaca por tener buenos atributos defensivos, ya que posee escudos, y una gran velocidad de ataque que le permite disparar un gran número de proyectiles en pocos segundos.

En su habilidad básica dispara proyectiles que aplican daño físico en función del poder físico del personaje. En esta habilidad, puede infligir daño crítico.

En su habilidad primaria potencia la velocidad de ataque y el rango de ataque de su habilidad básica durante 5 segundos. Esta habilidad se puede utilizar cada 14 segundos.

En su habilidad secundaria se aplica un escudo a si mismo con el valor de 195 puntos de vida más el 70% de su poder físico. Esta habilidad la puede usar cada 31 segundos.

En su habilidad terciaria Grim.exe lanza una gran bola de plasma que canaliza desde su cuerpo que aplica un gran daño físico, con un daño base de 475 puntos más el 165% de su poder físico.

5.6.2 Phase

Phase pertenece a la categoría de mago y se caracteriza por emplear habilidades que dependen generalmente del poder mágico.

En la habilidad básica invoca un proyectil mágico, que inflige daño en función del poder mágico. Esta habilidad puede realizar un impacto crítico.

En la habilidad primaria crea una gran bola mágica que al golpear aplica un daño de 185 puntos más el 120% del poder mágico. Esta habilidad la puede lanzar cada 20 segundos.

En la habilidad secundaria realiza una explosión eléctrica en un rango de 350 unidades que ralentiza al enemigo y aplica 160 puntos de daño más el 30% del poder mágico. Esta habilidad se puede lanzar cada 14 segundos.

En la habilidad terciaria genera una explosión mágica que afecta a los enemigos en un rango de 500 unidades y aplica un daño de 375 puntos más 115% del poder mágico. Esta habilidad se puede activar cada 100 segundos.

5.6.3 Sparrow

Sparrow es la tiradora por excelencia del juego. Este personaje basa su daño en el poder físico y lo aplica lanzando diferentes flechas con su arco.

En su habilidad básica lanza flechas que aplican daño en función de su poder físico y pueden aplicar daños críticos.

En su habilidad primaria lanza una gran flecha que inflige un daño de 125 puntos más el 125% del poder físico. Esta habilidad se puede lanzar cada 14 segundos.

En su habilidad secundaria lanza un haz de 4 flechas en abanico que pueden impactar a múltiples enemigos. Estas flechas aplican un daño de 50 puntos más 100% del poder físico y se puede activar cada 11 segundos.

En su habilidad terciaria invoca una lluvia de flechas, que es un *spawn* de área de colisión, que golpea a los enemigos que se encuentran dentro con un daño de 260 puntos más el 120% del poder físico. Se puede lanzar la habilidad cada 80 segundos.

5.6.4 Iggy&Scorch

Este personaje, compuesto por el duende Iggy y su mascota monstruosa Scorch, es un personaje mago, que posee la particularidad de tener un gran daño mixto, ya que sus habilidades aplican daño mágico y daño físico.

En su habilidad básica Scorch dispara bolas de fuego que aplican daño en función del poder físico y el poder de habilidad. Esta habilidad puede hacer impacto crítico.

En su habilidad primaria se aplica a si mismo una potenciación que mejora la velocidad de movimiento, y mejora la velocidad de ataque de la habilidad básica durante 5 segundos. Esta habilidad se puede activar cada 14 segundos.

En su habilidad secundaria se aplica un escudo de 195 puntos de base más el 50% de la resistencia mágica y el 50% de la armadura. Esta habilidad se puede activar cada 20 segundos.

En su habilidad terciaria se invoca una lluvia de meteoritos, que es un *spawn* de área de colisión, que aplica un daño mixto de 160 puntos de daño mágico más el 60% del poder mágico y 140 puntos de daño físico más el 60% del poder físico. Esta habilidad se puede activar cada 85 segundos.

5.7 Tienda y objetos

Entre cada ronda de combate los jugadores tienen acceso a la tienda, que le permite comprar objetos para mejorar sus atributos de combate.

5.7.1 Tienda

La tienda forma parte de un nivel aparte al nivel de combate donde se incluye un mapa de fondo con una interfaz que ocupa la mayor parte de la pantalla. Esta interfaz se genera de forma automática siguiendo la tabla de datos de objetos que posee el juego. En esta interfaz se crea un botón para cada uno de los objetos existentes en el juego, y en cada botón se muestra el nombre, un icono del objeto, sus características y el precio de compra.

El menú permite a cada jugador comprar el objeto que tienen seleccionado, deshacer la última compra, acceder a los objetos de su inventario y vender estos objetos. Deshacer una compra no conlleva una penalización, pero vender un objeto recibe una penalización sobre el valor de dinero que se obtiene, consiguiendo recuperar el 70% del valor total del objeto. Esto permite modificar los objetos equipados si tu inventario ya está lleno, pero recibe una penalización para que no se pueda abusar del uso en un corto periodo de tiempo de los objetos.

5.7.2 Objetos

Los objetos dentro del juego se gestionan a partir de una estructura propia que almacena un icono, un precio, un nombre, una descripción de las mejoras que aplica y un *Gameplay Effect*. Estos datos salvo el efecto se encargan de asignar la información a la interfaz de la tienda y el efecto es el encargado de aplicar la parte lógica del objeto.

El *Game Instance* se encarga de guardar una copia de todos los inventarios de los jugadores entre cada ronda, y al comenzar un nuevo combate, aplica los efectos de los objetos a cada jugador. Estos efectos incluyen la información de los atributos que mejora cada uno y esta información se añade de manera aditiva a las características que ya poseía cada jugador.

5.8 Mapas y escenarios

En Project Onyx se han creado diferentes mapas para cada uno de los niveles del juego. Algunos cumplen una labor decorativa, como el mapa de la pantalla de inicio, y otros cumplen reglas de diseño para adaptarse al combate del juego.

5.8.1 Diseño de nivel

Para adaptarse a las características del combate de nuestro juego, se crearon previamente unas directrices de diseño para crear el mapa principal de combate, con la idea de favorecer la escalabilidad del proyecto si se decidían crear nuevos mapas de combate.

La idea principal era crear un mapa de tamaño limitado para favorecer una batalla frenética que no permitiese a los jugadores huir constantemente. Por otra parte, este tamaño reducido favorecería a la cámara general del juego, ya que la cámara tiene que asegurar que se muestran siempre ambos personajes en combate, y si se alejasen mucho se perdería mucho la escala de los luchadores.

Además, se añadieron obstáculos que limitaban el libre movimiento de los personajes por el área de combate. Esto permite a los personajes con rango jugar con los obstáculos del

terreno para poder sortear a enemigos de corto rango con mayor movilidad que ellos. Por otra parte, limita las rutas de huida lo que permite a los personajes que se encuentran persiguiendo a su rival anticipar las rutas que puede tomar.

Por último, se definió la forma de la arena como un espacio que sigue la forma de una hiperelipse. Con esta forma se pretende que el espacio de combate siempre quede en un punto céntrico de la pantalla, evitando la existencia de esquinas, para que las interfaces de los jugadores no causen interferencias con la posición de los personajes en la pantalla del juego.

5.8.2 Blocking

Para llevar a cabo el desarrollo de la arena de combate se partió de la creación de una estructura base con poca resolución conocida como *blocking*. El objetivo del *blocking* es delimitar el área de juego. Para realizarlo se suele emplear bloques de gran tamaño, de ahí su nombre. En un mapa complejo se pueden usar esquemas de color en el *blocking* para diferenciar las diferentes áreas: un color para las zonas transitables, otro para las intransitables, un color para las zonas escalables o interactuables, o marcadores para los puntos de anclaje o elementos de guía. En nuestro caso particular, solo se quería delimitar las zonas transitables de la arena por lo que no se crearon colores adicionales.

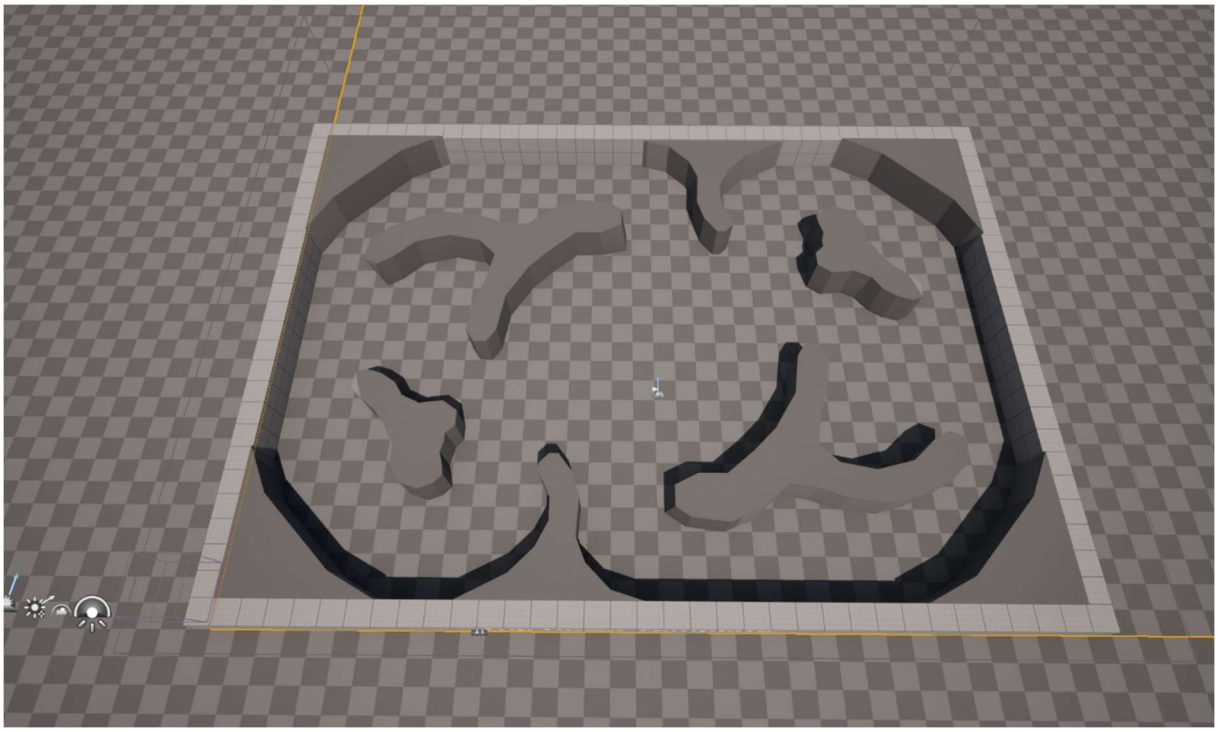


Ilustración 13 - Blocking de la arena de combate

5.8.3 Landscape

Para añadir profundidad a la superficie del nivel y evitar usar un suelo plano, se creó un terreno con la herramienta *Landscape* de Unreal. A través de esta herramienta se pueden crear y editar terrenos complejos que añaden realismo a los niveles desarrollados.

Modelado

Con el uso de la herramienta de modelado de *Landscape* se puede modificar la forma de la superficie añadiendo depresiones y elevaciones del terreno. Además, permite emplear herramientas que suavizan o allanan el terreno y herramientas que simulan la erosión del aire o el agua.

Materiales

Para poder añadir un material a tu superficie creada con *Landscape* este material debe cumplir unas características específicas. Los materiales de *Landscape* permiten crear una estructura de capas de mezclado. En cada capa de mezclado se puede añadir una textura

diferente, y el resultado de material depende de los pesos de mezcla de cada una de las texturas.

Pintado

La herramienta de pintado permite editar el material del *Landscape* en cada uno de los puntos de la superficie como si de un pincel se tratase. Con estos pinceles puedes editar los pesos de las capas de mezclado para elegir que material predomina en cada punto de la superficie.

5.8.4 Disposición de assets

La idea principal de añadir *assets* al nivel es ajustarse al *blocking* sin comprometer las limitaciones de espacio que este define, y mejorar el aspecto visual del escenario con el uso de diferentes rocas, árboles y estructuras.

Nanite

Para mejorar el rendimiento del juego en el nivel de combate se ha añadido el uso de *Nanite* para los *asset* del nivel.

Nanite [46] es una tecnología de renderizado que se encarga de manejar grandes cantidades de geometría en tiempo real. De esta forma Nanite mejora el resultado visual del juego sin comprometer el rendimiento.

Para llevar a cabo esto, Unreal virtualiza la geometría y ajusta automáticamente el nivel de detalle de cada malla.

5.8.5 Vegetación

Para añadir vegetación al nivel se ha empleado la herramienta de Unreal *Folliage*. Esta herramienta permite pintar múltiples tipos de vegetación, como césped, arbustos o árboles, directamente sobre el terreno, pudiendo modificar su densidad.

La herramienta organiza y agrupa automáticamente la vegetación en *clusthers* para reducir el número de *draw calls* al renderizar grandes cantidades. Además, oculta

automáticamente toda la vegetación no visible o a una distancia significativa para mejorar el rendimiento.

5.9 Cámara

En Project Onyx la cámara es global para todos los jugadores y tiene que situarse centrada a los luchadores y a una distancia que siempre sean visibles. Por ello, la cámara posee una lógica propia que gestiona estos aspectos.

Para realizar la lógica de la cámara se ha creado una clase *Main Camera*. En esta clase se define un actor con un componente cámara y un componente *Spring Arm*, que es un brazo que sostiene la cámara.

En la inicialización de esta clase, la cámara recoge una referencia a todos los jugadores de la partida. A partir de esa referencia calcula la media aritmética de las posiciones de los jugadores y guarda el resultado como posición objetivo. Durante el *Tick* la cámara se mueve hacia la posición objetivo a una velocidad fijada, y cuando la alcanza, vuelve a calcular la posición objetivo.

De manera similar, con la intención de que los personajes siempre permanezcan en pantalla a pesar de estar alejados, en base a la distancia de los jugadores al punto céntrico, se calcula la longitud del brazo objetivo. Durante el *Tick* se estira o se encoje el brazo hasta llegar al largo objetivo y se vuelve a calcular esta distancia. Este cálculo tiene una peculiaridad ya que la relación de aspecto de la cámara no es cuadrada, por lo que al alejarse los personajes en el eje Y (de izquierda a derecha) no es necesario alejar tanta distancia la cámara como cuando se alejan los personajes en el eje X (de arriba a abajo). Por esa razón, al calcular la distancia de los personajes al centro se aplica un coeficiente a la distancia de un de los ejes, por lo que la distancia no sigue una forma circular, si no una forma elíptica.

5.10 Sistema de input

Unreal Engine 5 incluye el sistema *Enhanced Input* que permite manejar de manera avanzada y flexible la entrada de datos de los jugadores. Este sistema permite definir diferentes acciones, para posteriormente asignar a cada acción una o múltiples entradas físicas.

El sistema de inputs permite crear diferentes contextos y cambiar entre ellos de forma dinámica. Por ejemplo, se podría crear un contexto para un personaje a pie y otro para cuando el personaje está nadando o manejando un vehículo. De esta forma el cambio dinámico entre esquemas de entrada facilita la gestión para diferentes modos de juego.

En el caso particular del proyecto, se ha empleado únicamente un contexto, con la particularidad de añadir modificadores a las entradas físicas. Esto permite, por ejemplo, limitar la entrada en un eje de un joystick a una única dirección si se quiere usar el joystick para moverse hacia una única dirección en un menú.

5.11 Sistema de animaciones

Unreal Engine Incluye su propia clase para gestionar las animaciones. Esta clase es *AnimBlueprint* de la que heredan la lógica de las animaciones de cada uno de los personajes. Esta clase permite crear grafos de animaciones incluyendo máquinas de estado con transiciones para modificar la pose del personaje, mezclar diferentes poses a partir de los pesos en cada hueso de cada pose, y añadir slots que reproducen montajes de animación.

Para el desarrollo del proyecto se ha adaptado una máquina de estados que determina la animación cuando el personaje se encuentra moviéndose. Además, se ha añadido un *slot* para reproducir los montajes de animación de los personajes cuando se realiza una habilidad.

5.12 Inteligencia artificial

Durante el desarrollo del proyecto se ha creado una inteligencia artificial capaz de controlar los personajes del juego en combate. Esta IA se emplea en el modo IA vs IA donde un personaje se enfrenta a otro, ambos controlados por la inteligencia artificial.

5.12.1 Controlador y Blackboard

El controlador de IA cumple la función de cerebro que se encarga de ejecutar el árbol de comportamiento y sirve de nexo entre la lógica de comportamiento y el personaje.

Para realizar la comunicación entre las diferentes clases que intervienen en la lógica de comportamiento y el árbol de comportamientos se emplea el patrón *Blackboard*. La *Blackboard* sirve como almacén de datos comunes y sigue una estructura de clave valor. Este sistema admite diferentes clases de datos como valor, como puedan ser: referencias a objetos, vectores, valores *booleanos*, valores *float* o cualquier otro tipo de valor común.

Al inicializar el controlador, este guarda en la *Blackboard* el *Ability System Component* del actor que controla, calcula su rango de combate en base a los datos de sus habilidades y lo recoge en la *Blackboard*, y busca su enemigo en el nivel para guardar también una referencia a su enemigo y otra referencia al *Ability System Component* del enemigo.

Posteriormente ejecuta el árbol de comportamientos, lo que permite comenzar la toma de decisiones.

5.12.2 Árbol de comportamiento, tareas y decoradores

Unreal Engine apuesta como método nativo de inteligencia artificial por los árboles de comportamiento. A partir de su herramienta para crear y modificar árboles de comportamiento permite a los desarrolladores crear comportamientos complejos mediate una interfaz simple y accesible.

Los árboles de comportamiento de Unreal se componen de los siguientes nodos:

- Nodo raíz: es la puerta de entrada al árbol de comportamientos. Cada vez que se ejecuta el árbol se empieza en este nodo.
- Selectores: al igual que en cualquier árbol de comportamiento, este nodo detiene su ejecución, y devuelve un valor satisfactorio a su nodo padre, cuando uno de sus hijos devuelve un valor satisfactorio.
- Secuencia: Este nodo ejecuta cada uno de sus nodos hijos a no ser que un de ellos no devuelva un valor satisfactorio, lo que detiene la ejecución.
- Tareas: son nodos que realizan una lógica concreta. Devuelven o no un valor satisfactorio en función de los resultados de la ejecución de la lógica.
- Decoradores: son anexos a otros nodos que permiten filtrar la ejecución del nodo en base a diferentes condicionantes. Además, incluyen la opción de abortar la ejecución de nodos con menor prioridad si la condición cambia de valor durante la ejecución.

Para el caso concreto de nuestro proyecto se han tenido que implementar deferentes decoradores y tareas.

Como decoradores se ha implementado un decorador que comprueba que el enemigo se encuentra a rango para lanzar una habilidad en específico. Este decorador comprueba la distancia entre el personaje y su enemigo y si es menor al rango especificado o, en caso de especificarlo por el identificador de habilidad, del rango de la habilidad se devuelve un valor satisfactorio. En caso contrario se devuelve el valor no satisfactorio. También, se implementa un decorador que comprueba el porcentaje de vida en función de su vida máxima. Si el valor obtenido es inferior al umbral que se indica se devuelve un resultado satisfactorio. Esto permite realizar tareas específicas solo cuando la vida está por debajo de un umbral indicado.

En cuanto a tareas, se han implementado diferentes tareas que se encargan de: intentar activar una habilidad, intentar aplicase un escudo, mirar en una dirección o encontrar una posición a la que escapar.

Se pueden consultar los diagramas de comportamiento y el árbol de comportamiento en el **Anexo I**.

5.12.3 Evironment Query System (EQS)

El *Enviroment Query System* de Unreal, conocido como EQS, es un sistema de utilidad que evalúa el entorno mediante un generador de posiciones probables, y asigna un valor de viabilidad en función a una serie de filtros y funciones de utilidad, para decidir un mejor destino posible.

En el proyecto se emplea para mejorar el comportamiento de movimiento de los personajes. En lugar de moverse en línea recta hacia el otro personaje, evalúa en conjunto de puntos que se generan alrededor del enemigo, eligiendo un destino que se encuentre lo más cercano a tu posición actual, dentro de tu rango de combate con el enemigo.

También lo empleamos para decidir dónde y cómo huir, creando una nube de puntos alrededor del personaje, eligiendo aquellos que se encuentran más alejados del enemigo. Para dotar al comportamiento de una mayor variedad se eligen los destinos de forma aleatoria entre el 25% de los mejores resultados.

6. Conclusiones

6.1 Resultado final

El resultado final del proyecto es un videojuego funcional que incluye cuatro modos diferentes, tres de ellos jugables y un modo de juego de simulación de combates entre las inteligencias artificiales.

Project Onyx incluye en sus modos nueve personajes diferentes con distintas habilidades únicas cada uno. Estos personajes cuentan también con diferentes aspectos estéticos y con atributos con valores personalizados cada uno. Además de las habilidades únicas de los personajes, todos tienen en común cuatro habilidades de movimiento

Durante el desarrollo de los combates los jugadores pueden comprar objetos que mejoran las estadísticas de combate. Cada uno de los personajes puede equipar siete objetos. Los objetos no son únicos y se pueden comprar en más de una unidad, pero cada compra ocupa un espacio de inventario. Se han implementado un total de ocho objetos diferentes, donde cada uno aporta diferentes estadísticas de combate.

A parte de los personajes, se han incluido actores interactuables que son cajas con vida que al romperlas otorgan una recompensa de dinero, lo que puede cambiar el curso de los combates.

Para complementar el proyecto se ha desarrollado una inteligencia artificial capaz de realizar combates con los personajes del juego. Esta IA se emplea en el modo de simulación de combates de IA vs IA. Este modo se emplea para realizar un balance de personajes que complemente las herramientas de balanceo que se estudian los datos de combate. En base a los combates simulados se obtienen resultados empíricos de las situaciones de combate que en los datos no se pueden obtener. Por ejemplo, permite observar que habilidades son más difíciles de impactar, como los proyectiles, y que personajes encuentras más dificultades para poder aplicar su daño sin ver comprometido su rango de combate.

Como añadido adicional al proyecto, se ha empaquetado y se ha subido la versión final del juego a la plataforma itch.io [47], en formato descargable junto a una descripción con capturas del juego y los créditos a los propietarios de los materiales de terceros empleados en el proyecto.

Se pueden observar los resultados obtenidos en el juego a través de las capturas obtenidas del juego final en el **Anexo II**.

6.2 Objetivos cumplidos

Durante el desarrollo de Project Onyx se han cumplido de forma satisfactoria los objetivos propuestos al principio del proyecto, destacando el objetivo principal de desarrollar una demo técnica con el motor de desarrollo de videojuegos Unreal Engine en su versión 5.3. Este objetivo se muestra cumplido, ya que se ha obtenido un juego funcional que puede constatar la consecución de otros objetivos como: aprender y demostrar un conocimiento amplio sobre Unreal Engine y su lenguaje de programación visual (Blueprints), un conocimiento del plugin GAS y de la programación de clases en C++, el desarrollo de interfaces y menús capaces de gestionar el juego multijugador local y la implementación de inteligencias artificiales a través de las herramientas de Unreal.

Además, se ha demostrado la capacidad de trabajar en equipo y colaborar para llevar a cabo el desarrollo de un proyecto común. Estos resultados se pueden constatar a través de la publicación del juego en una plataforma oficial como es itch.io.

6.3 Impacto del proyecto

La implementación de este proyecto conforma un marco de desarrollo de juego, tanto en el ámbito académico, como en el profesional. Este documento sirve como apoyo para todo aquel que quiera realizar un videojuego con el motor Unreal Engine, y más específicamente

con el plugin Gameplay Ability System, ya que se describen los problemas y resultados obtenidos durante el desarrollo del proyecto. Algunos de los problemas que solventa este proyecto son:

- La falta de ejemplos y documentación del *plugin* GAS para crear un sistema de combate complejo.
- La adaptación de todo el juego al funcionamiento en multijugador local, incluyendo menús e interfaces de juego.
- La adaptación de recursos externos de diversas fuentes para el motor Unreal Engine, concretamente en la versión 5.3.

Estos problemas se recogen a lo largo del documento y se muestran las soluciones tomadas, de cara a servir de apoyo para todo aquel que quiera hacer un proyecto similar.

6.4 Líneas futuras

A lo largo del desarrollo del proyecto y tras su finalización han surgido diferentes ideas de futuro que podrían añadir valor al videojuego.

El hecho de que todos los modos de juego sean multijugador local limita que jugadores individuales quieran probar el juego. Incluir modos de juego que sean de un solo jugador, `para que los jugadores no tengan que depender de tener un compañero para poder probar el juego podría ser una solución para este problema. Esto se puede dividir en diferentes modos que solventarían este punto. A continuación, se muestran algunos ejemplos:

- Un modo de juego de 1 vs IA. Actualmente, la IA de combate no es tan compleja como para suponer un buen reto para un jugador experimentado. La mejora de esta IA y la implementación de un modo de combate contra la IA permitiría a los jugadores probar el modo de combate sin un compañero.
- Un modo de juego multijugador online. Debido a la complejidad del desarrollo de un juego en red con la herramienta Unreal Engine, que a principio del desarrollo

no conocíamos en detalle, por lo que decidimos no incluirlo como objetivo inicial del proyecto, aunque se platea añadirlo más adelante. Este modo permite jugar con otros jugadores sin la necesidad de compartir un entorno físico.

- Un modo de juego de supervivencia. De cara a futuro se ha planteado también crear otros modos de juego que no se centren en el combate en arena. Este modo permitiría probar los personajes del juego enfrentándolo a hordas de enemigos con el objetivo de sobrevivir el máximo tiempo posible.

Por otra parte, se plantea mejorar los combates ya existentes añadiendo diferentes eventos interactuables, como enemigos neutrales que otorguen recompensas, la obtención de mejoras aleatorias tras completar un número de rondas y otros añadidos que mejoren la variedad y versatilidad del combate.

También se plantea añadir nuevos personajes, nuevas arenas de combate y nuevos objetos para ampliar las opciones que brinda el juego.

Y, por último, se plantea implementar un menú de opciones funcional que permita cambiar valores de sonido, gráficos y controles dentro del juego.

Ludografía

En este apartado se recogen todos los juegos que se han mencionado y se han usado como referencia durante la realización del proyecto.

- HAL Laboratory, *Super Smash Bros.,* Nintendo, 1999
- Blue Mammoth Games, *Brawlhalla,* Ubisoft, 2017
- Riot Games, *League of Legends,* 2009
- Capcom, *Street Fighter,* 1987
- Nintendo, *Mario Kart,* 1992
- Matt Makes Games , *Towerfall Ascension,* 2013
- Epic Games, *Fortnite,* 2017
- Hopoo Games, *Risk of Rain 2,* 2019
- Phoenix Labs, *Dauntless,* Epic Games, 2019
- Epic Games, *Paragon,* 2016.
- Proletariat Inc., *Spellbreak,* 2020
- Supercell, *Brawl Stars,* 2018
- OverPowered Team, *Godstrike,* 2021
- Habby, *Archero,* 2019
- TiMi Studios, *Pokemon Unite,* The Pokemon Company & Nintendo, 2021

Bibliografía

[1] Epic Games, «Unreal Engine,» 2004 - 2024. [En línea]. Available: https://www.unrealengine.com/.

[2] Epic Games, «Gameplay Ability System,» Unreal Engine, 2004 - 2024. [En línea]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/gameplay-ability-system-for-unreal-engine.

[3] Unreal Engine, «Marketplace,» 2004 - 2024. [En línea]. Available: https://www.unrealengine.com/marketplace/en-US/store.

[4] Adobe, «Adobe Firefly,» [En línea]. Available: https://firefly.adobe.com/.

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

[6] Strive Mindz, «How Unreal Engine is Transforming Game Development Industry,» 2024. [En línea]. Available: https://www.strivemindz.com/blog/how-unreal-engine-is-transforming-game-development-industry/.

[7] Á. Jover-Álvarez, «Unreal Engine 5 - The truth of the Gameplay Ability System,» [En línea]. Available: https://vorixo.github.io/devtricks/gas/.

[8] J. Bhatia, «C++ Game Development: How Is C++ Used in Game Development?,» pwskills, 2024. [En línea]. Available: https://pwskills.com/blog/c-game-development/.

[9] Silicon Valley University, «The importance of collaboration and teamwork in the creative industry,» 2017. [En línea]. Available: https://usv.edu/blog/importance-collaboration-teamwork-creative-industry/.

[10] A. Taktak, P. S. Ganney, D. Long y R. G. Axell, Clinical Engineering, 2020.

[11] O. Al-Baik y J. Miller, «The kanban approach, between agility and leanness: a systematic review,» de *Empirical Software Engineering*, 2015, p. 1861–1897.

[12] Atlassian, «Trello,» [En línea]. Available: https://trello.com/.

[13] «Notion,» [En línea]. Available: https://www.notion.so/es-es.

[14] «Github,» [En línea]. Available: https://github.com/.

[15] «Discord,» [En línea]. Available: https://discord.com/.

[16] «Unity,» [En línea]. Available: https://unity.com/es.

[17] hackr.io, «Unity vs Unreal: Which Game Engine Should You Choose?,» 2024. [En línea]. Available: https://hackr.io/blog/unity-vs-unreal-engine.

[18] Microsoft, «Visual Studio,» [En línea]. Available: https://visualstudio.microsoft.com/es/.

[19] Epic Games, «Quixel,» [En línea]. Available: https://quixel.com/bridge.

[20] HAL Laboratory, *Super Smash Bros.,* Nintendo, 1999.

[21] Blue Mammoth Games, *Brawlhalla,* Ubisoft, 2017.

[22] Riot Games, *League of Legends,* 2009.

[23] E. Neuhaus, «Socializing with Games: Why Local Multiplayer is still important,» Game Developer, 2014. [En línea]. Available: https://www.gamedeveloper.com/game-platforms/socializing-with-games-why-local-multiplayer-is-still-important.

[24] Capcom, *Street Fighter,* 1987.

[25] Nintendo, *Mario Kart,* 1992.

[26] Matt Makes Games , *Towerfall Ascension,* 2013.

[27] Eye of Unity, «The Rise of Online Gaming: How the Internet Has Revolutionized the Gaming Industry,» Medium, 2024. [En línea]. Available:

https://medium.com/operations-research-gig/the-rise-of-online-gaming-how-the-internet-has-revolutionized-the-gaming-industry-f5126848e2fa.

[28] RBARRIER, «Lo que Unreal Engine 5 puede significar para la industria de los videojuegos,» IGN, 2022. [En línea]. Available: https://es.ign.com/unreal-engine-5/181427/feature/lo-que-unreal-engine-5-puede-significar-para-la-industria-de-los-videojuegos.

[29] Epic Games, *Fortnite,* 2017.

[30] Hopoo Games, *Risk of Rain 2,* 2019.

[31] U. Kustra, «Why you should use the Gameplay Ability System in basically every Unreal Engine game,» Medium, 2024. [En línea]. Available: https://ukustra.medium.com/why-you-should-use-the-gameplay-ability-system-in-basically-every-unreal-engine-game-bc148b64498a.

[32] Phoenix Labs, *Dauntless,* Epic Games, 2019.

[33] Epic Games, *Paragon,* 2016.

[34] Proletariat Inc., *Spellbreak,* 2020.

[35] H. Koss, «What Does the Future of Gaming Look Like?,» built in, 2024. [En línea]. Available: https://builtin.com/articles/future-of-gaming.

[36] Supercell, *Brawl Stars,* 2018.

[37] OverPowered Team, *Godstrike,* 2021.

[38] Habby, *Archero,* 2019.

[39] TiMi Studios, *Pokemon Unite,* The Pokemon Company & Nintendo, 2021.

[40] «Predecessor: Heroes,» [En línea]. Available: https://www.predecessorgame.com/heroes.

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

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

[43] Tranek, «GASDocumentation,» Github Repository, [En línea]. Available: https://github.com/tranek/GASDocumentation.

[44] «Unreal's Property Specifiers,» benui, [En línea]. Available: https://benui.ca/unreal/uproperty/.

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

[46] B. Karis, R. Stubbe y G. Wihlidal, «Nanite: A Deep Dive into Next-Generation Virtualized Geometry,» de *SIGGRAPH 2021*, 2021.

[47] «itch.io,» [En línea]. Available: https://itch.io/.

[48] The Game Dev Cave, «Unreal Gameplay Ability System - #1 Ability System Component,» Youtube video, 2023. [En línea]. Available: https://www.youtube.com/watch?v=suygHt7OhrM&list=PLoReGgpfex3woa35rnoXRyF9N3_p7QVQ2.

ANEXO I

Diagramas

En el siguiente anexo se recogen todos los diagramas usados durante el diseño y el desarrollo del proyecto. Esto incluye los diagramas para los controles, disponible para tres mandos diferentes: *Dualsense*, *Xbox* y *Switch.* También incluye los diagramas de clases UML del proyecto, los diagramas de flujo del videojuego y los esquemas empleados para desarrollar la inteligencia artificial, junto al árbol de comportamientos.

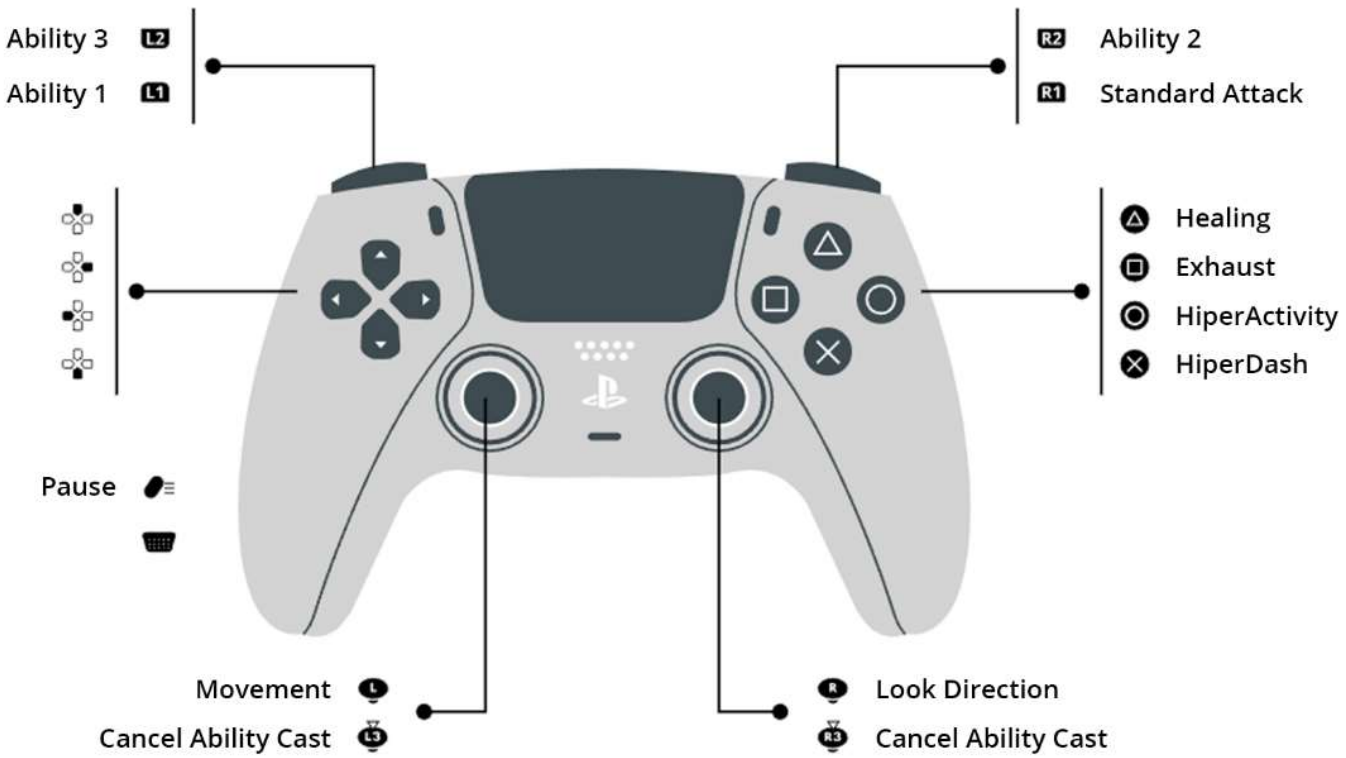


Ilustración 14 - Controles en mando Dualsense

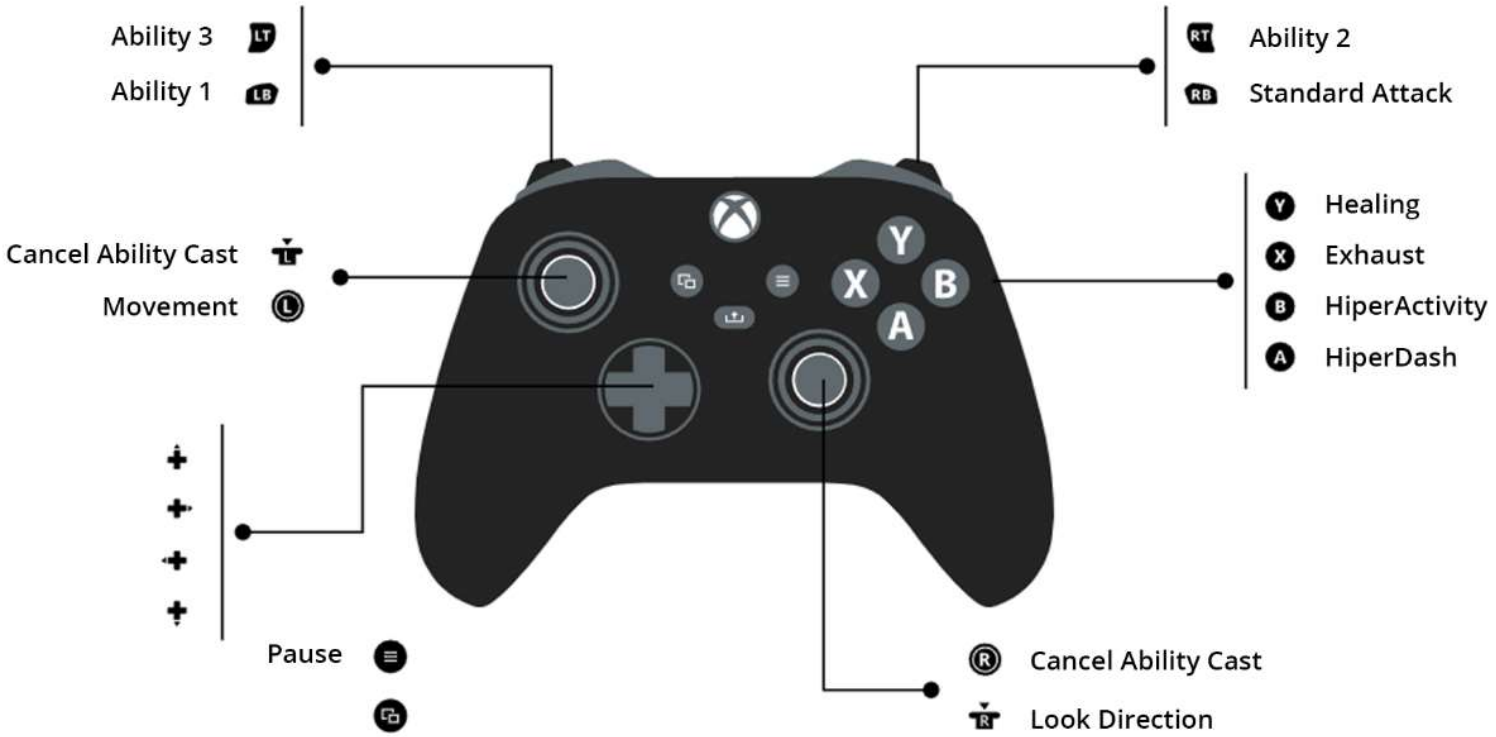


Ilustración 15 - Controles en mando Xbox

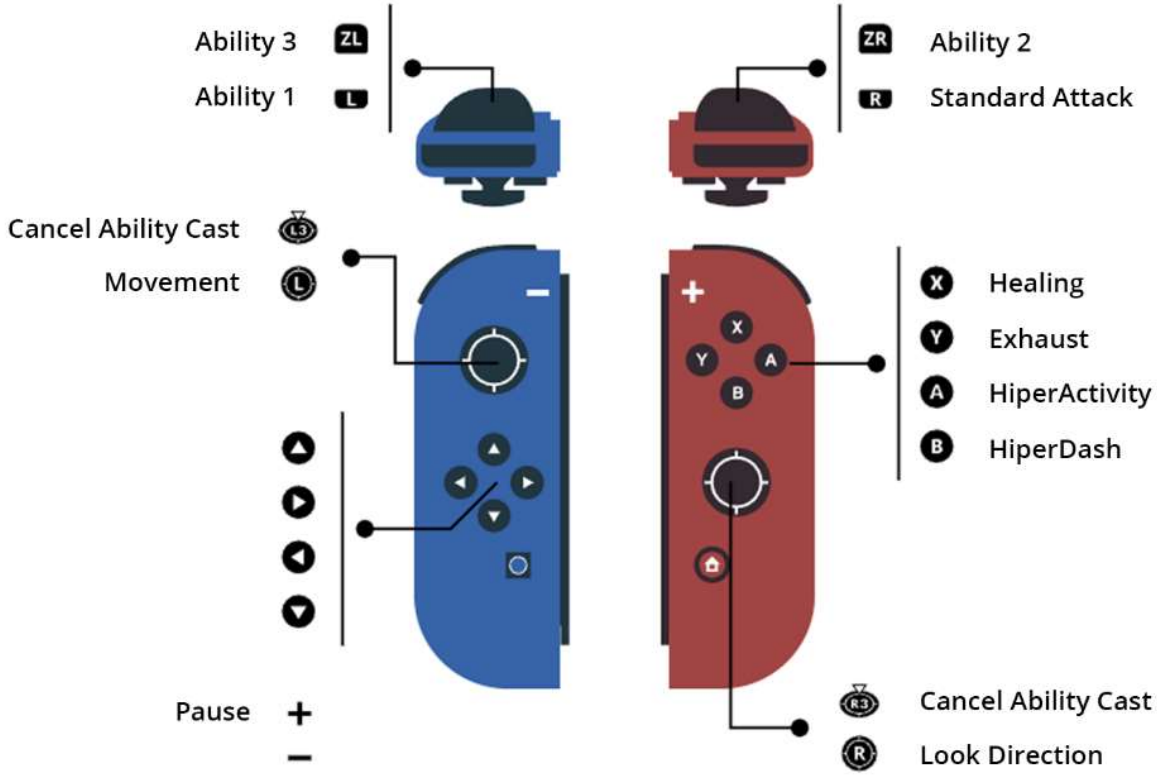


Ilustración 16 - Controles en mando Switch

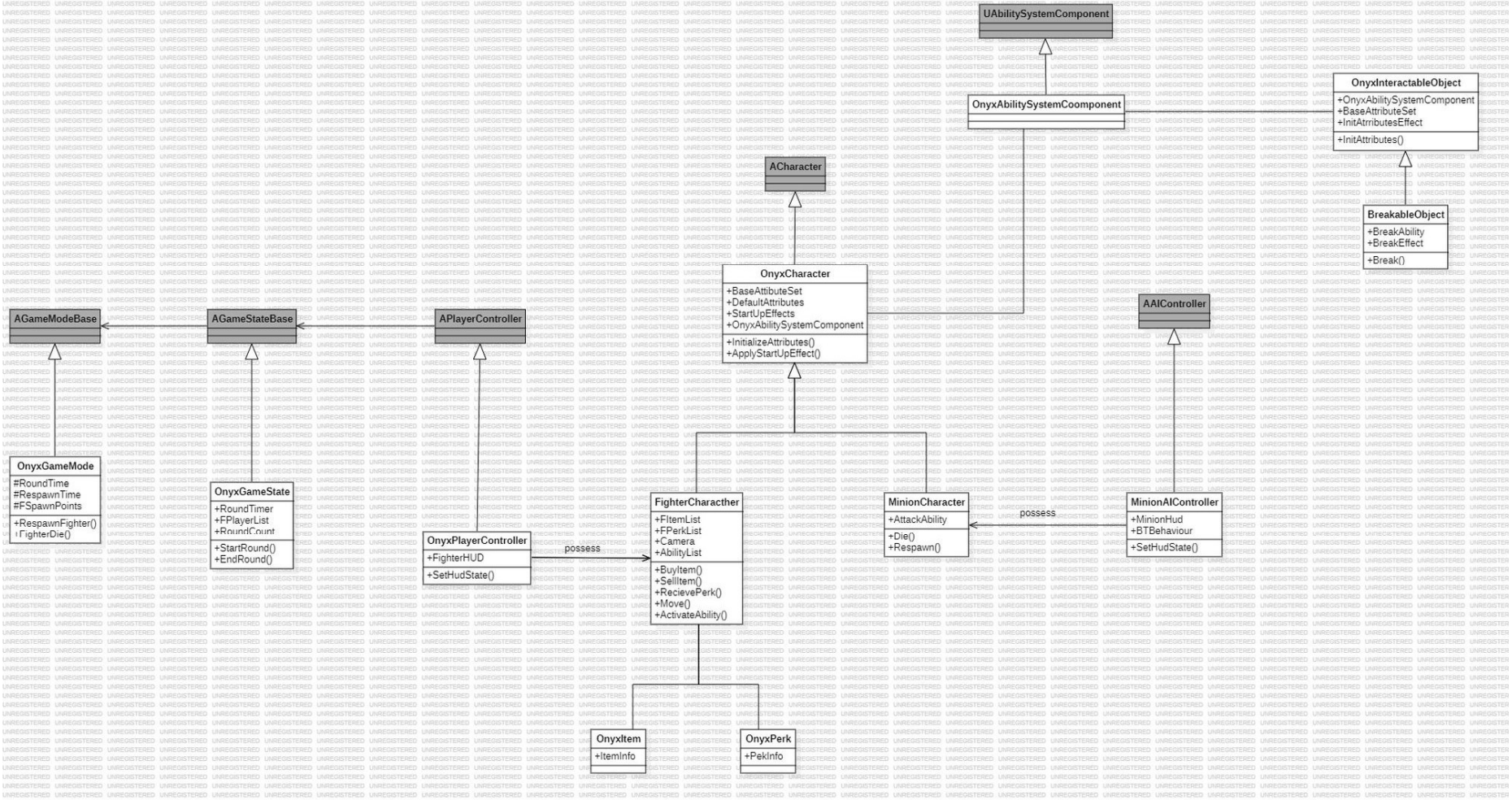


Ilustración 17 - Diagrama UML del proyecto

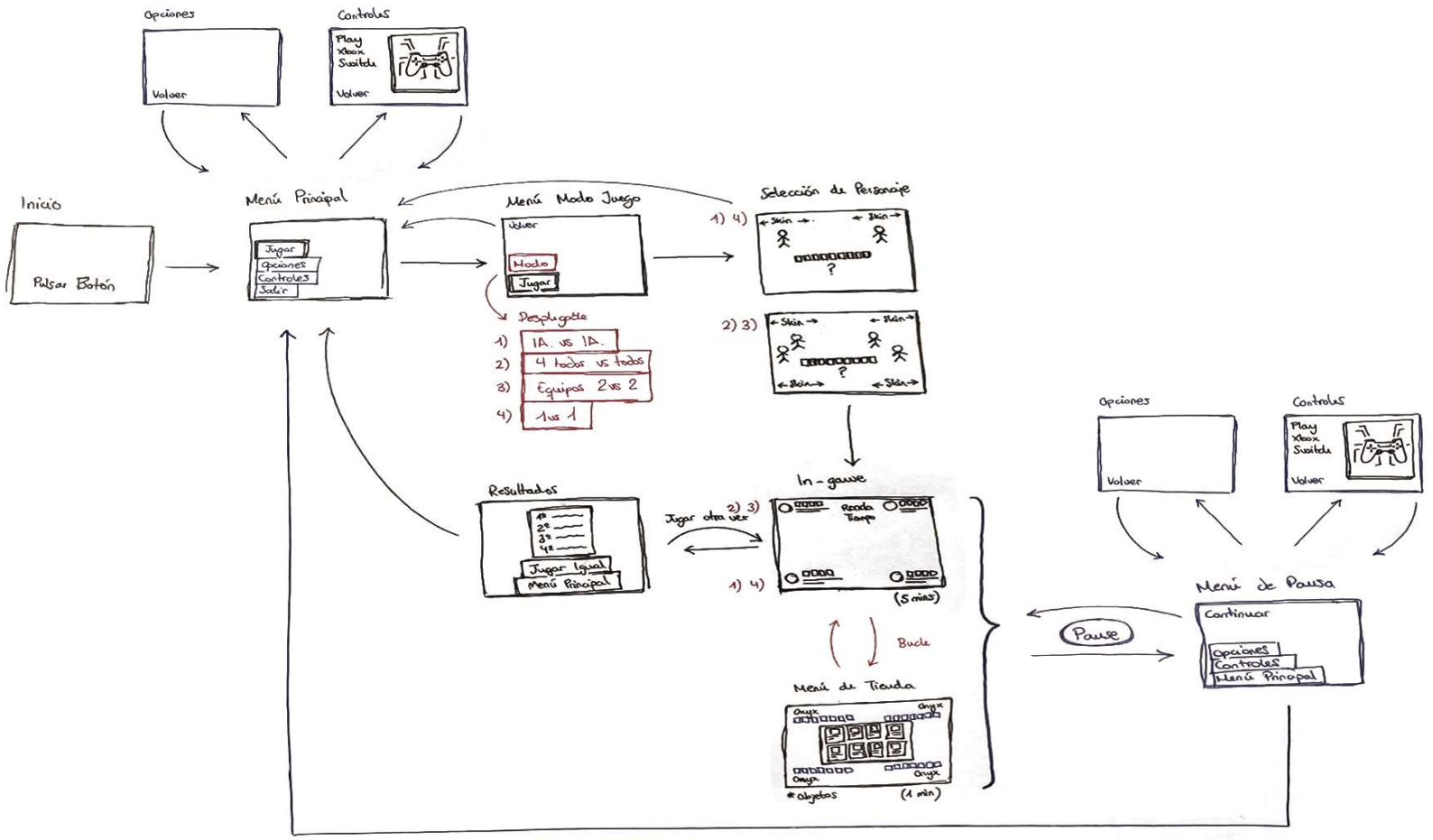


Ilustración 18 - Boceto del diagrama de flujo

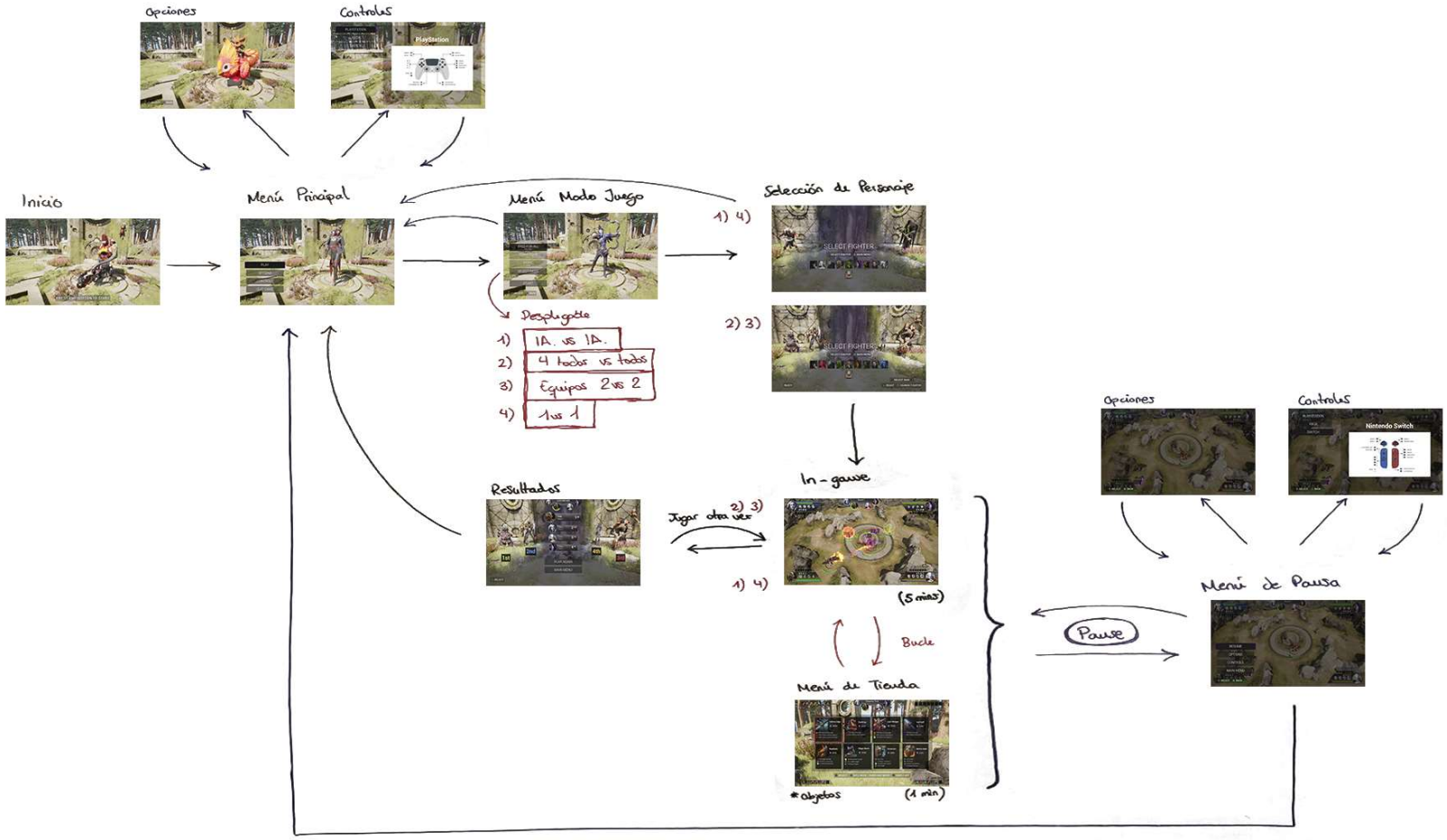


Ilustración 19 - Resultado del diagrama de flujo

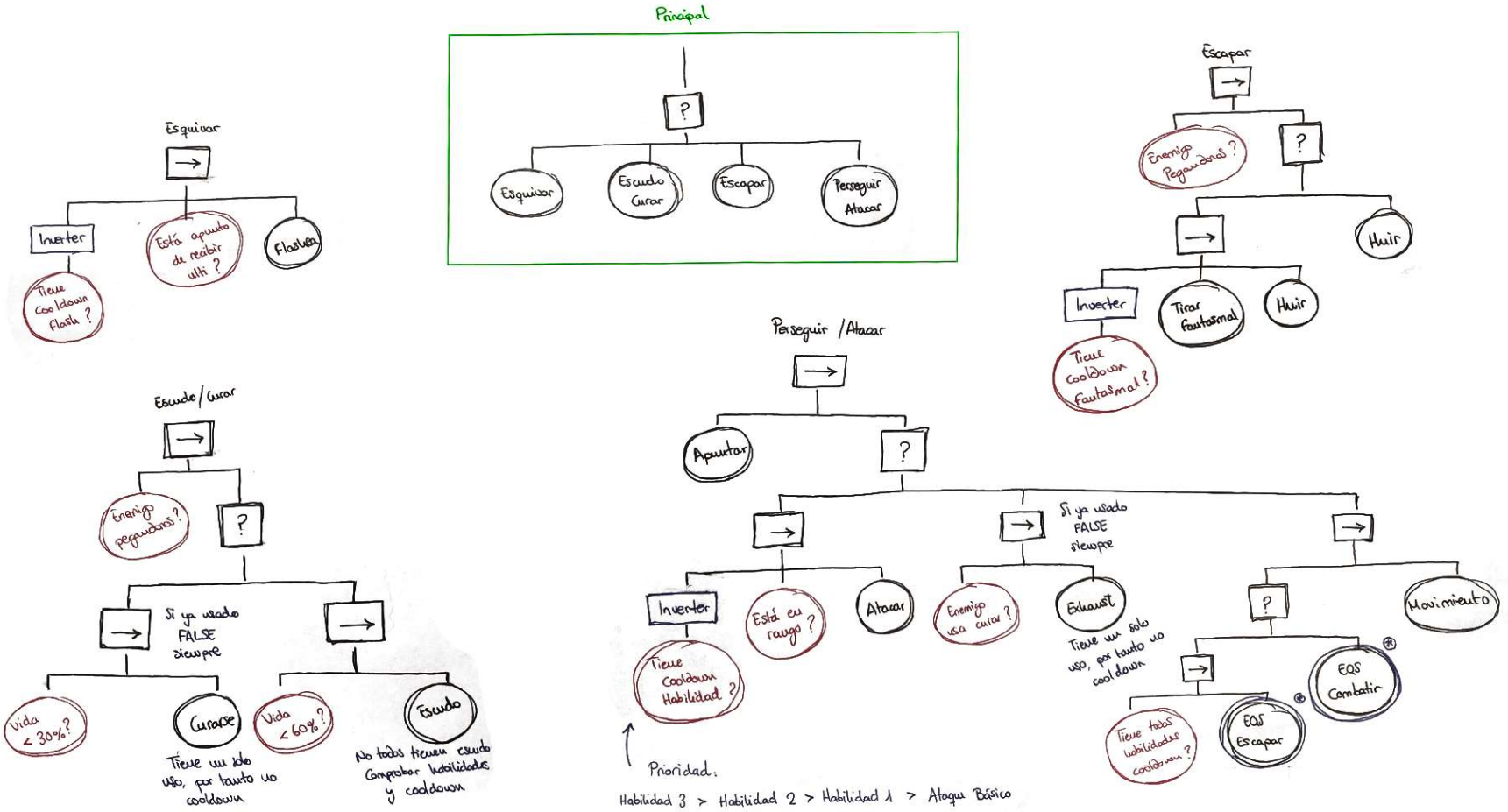


Ilustración 20 - Esquema de comportamiento de IA

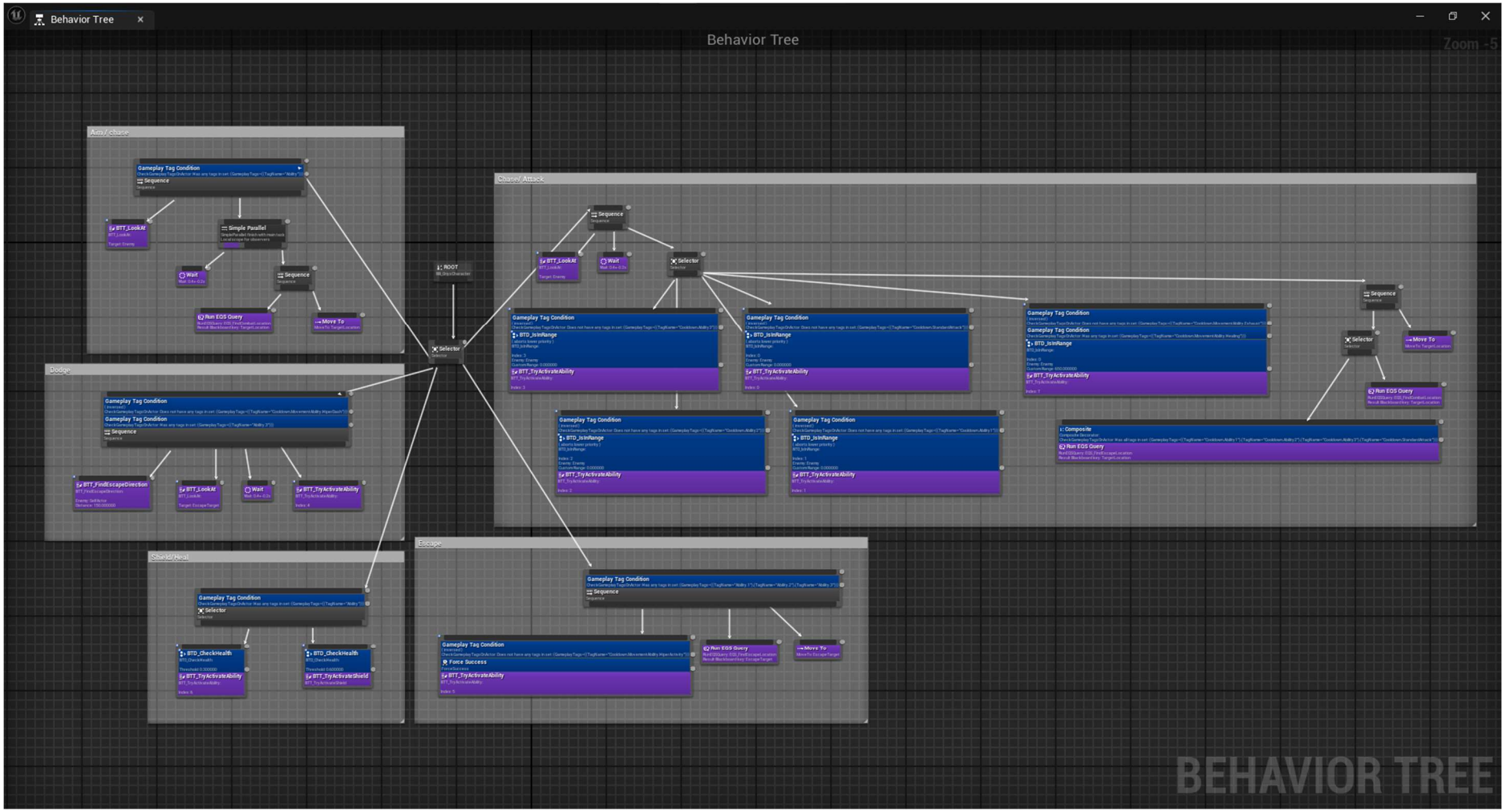


Ilustración 21 - Árbol de comportamientos en Unreal Engine

ANEXO II

Galería de capturas

En este anexo se recogen diferentes capturas obtenidas del juego. En estas capturas se muestran todos los menús y modos de juego disponibles durante el juego.

Ilustración 22 - Menú de inicio

Ilustración 23 - Menú principal

Ilustración 24 - Selección de modo

Ilustración 25 - Menú de controles

Ilustración 26 - Menú de selección (2 jugadores)

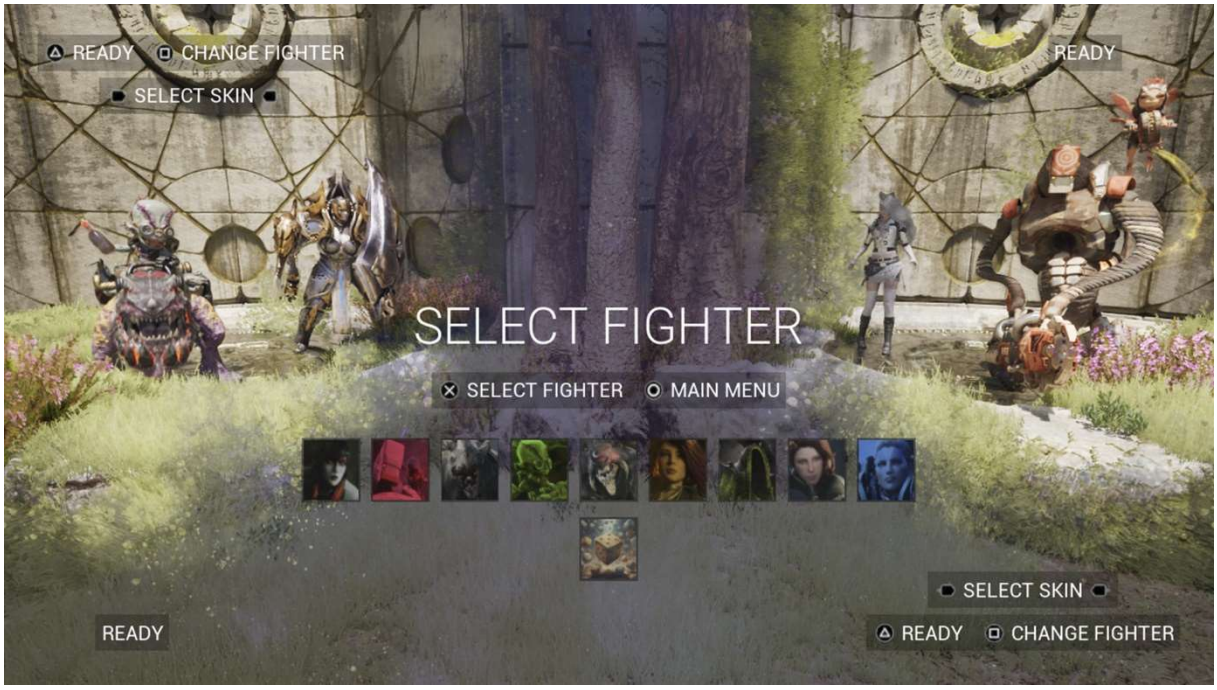


Ilustración 27 - Menú de selección (4 jugadores)

Ilustración 28 - Combate todos contra todos

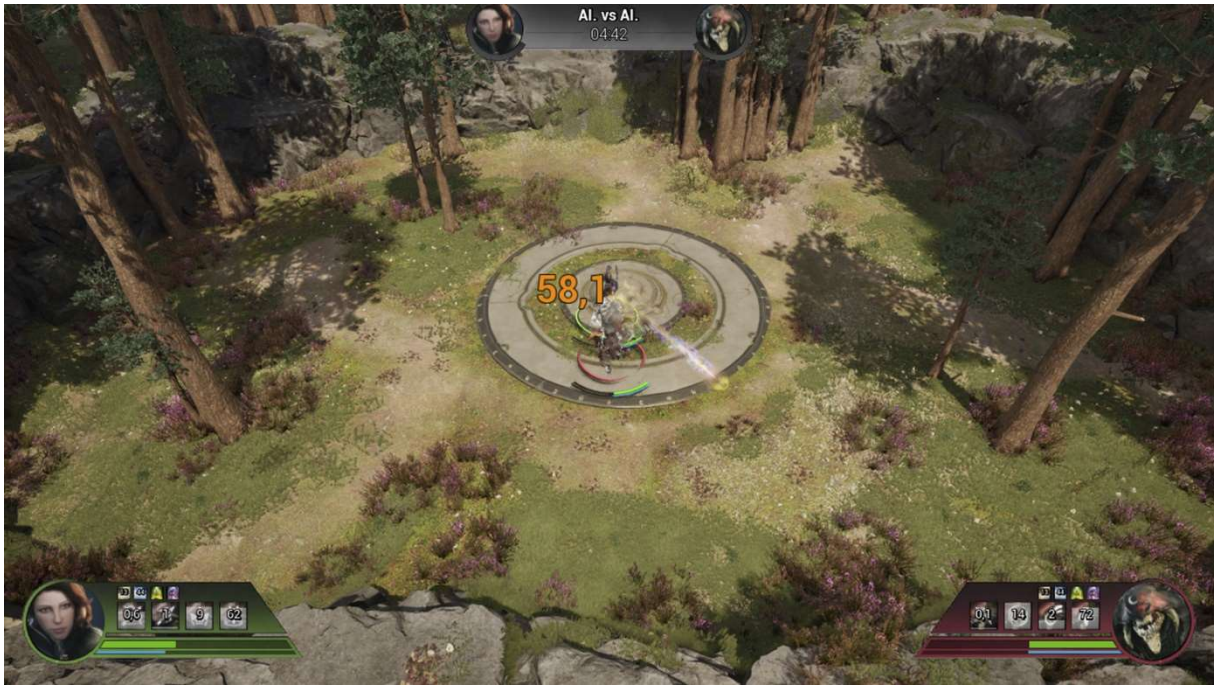


Ilustración 29 - Combate IA

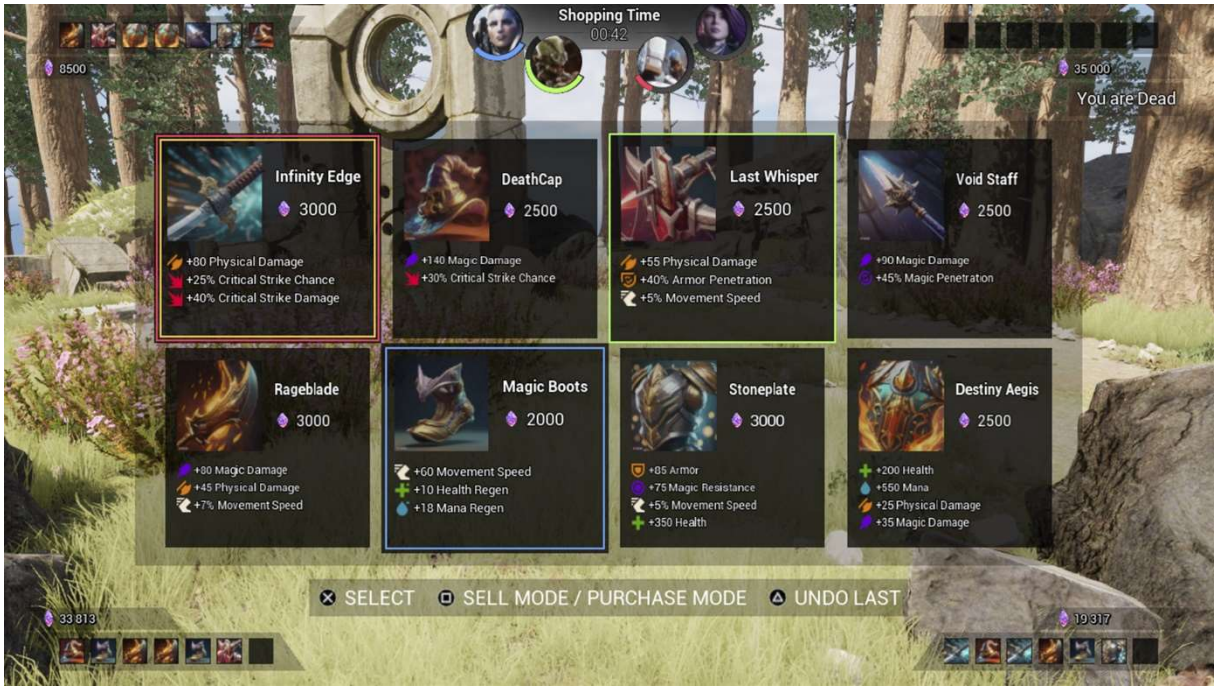


Ilustración 30 – Tienda

Ilustración 31 - Menú de pausa

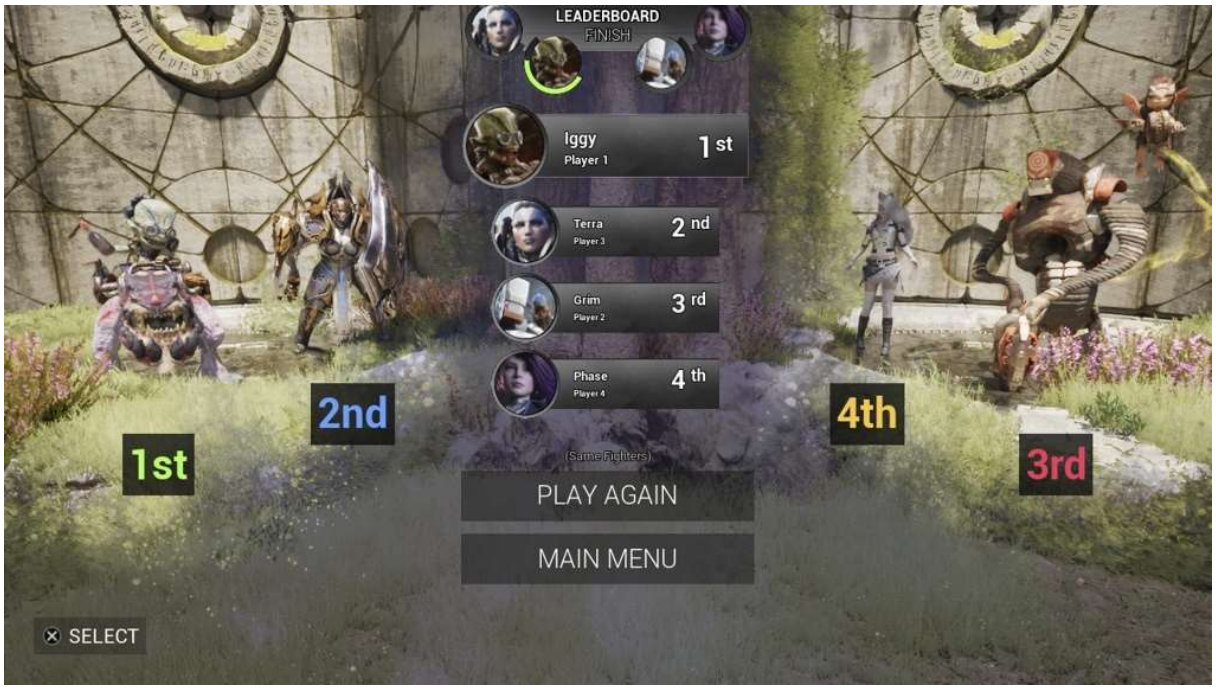


Ilustración 32 - Menú de final de partida

ANEXO III

Tablas de balance

En el siguiente anexo se exponen las tablas de balance de todos los personajes. Estas tablas recogen todas las estadísticas principales de los personajes, tanto los atributos base, como las estadísticas de las habilidades concretas. Estas estadísticas se emplean para calcular datos de combate, lo que permite comparar el potencial de combate entre diferentes personajes. A partir de estas comparaciones se puede detectar si algún personaje no está balanceado con relación a los demás.

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%)

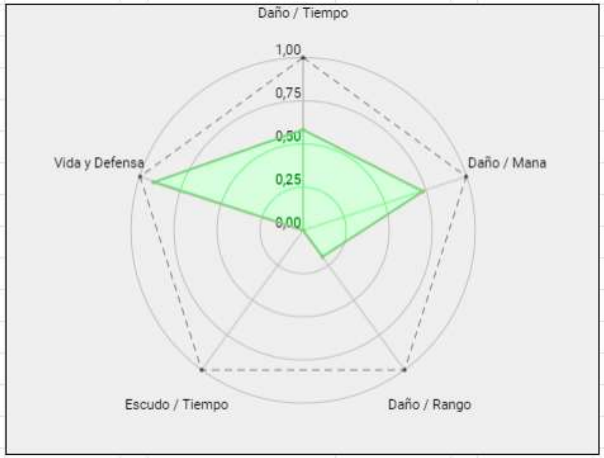
Super Salto

	Estadísticas
Cooldowns	30
Rango	400

Ilustración 33 - 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

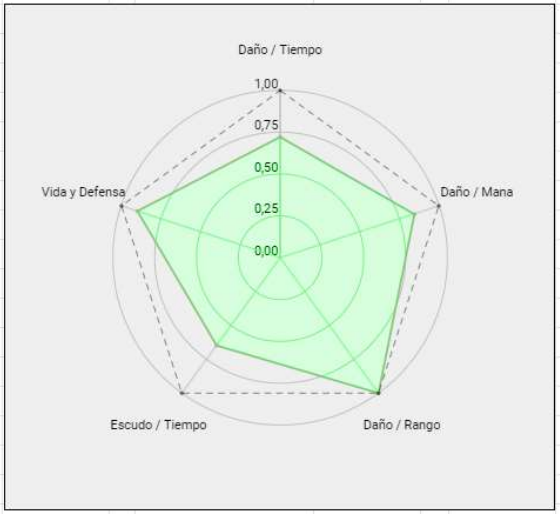


Balance								
Daño / Tiempo	79,88	8,21	8,02	4,81	100,92	Mayor Daño / Tiempo	172,30	0,59
Daño / Mana	67,90	2,63	3,48	4,09	78,09	Mayor Daño / Mana	106,31	0,73
Daño / Rango	95,06	787,77	938,88	715,49	2.537,20	Mayor Daño / Rango	13.588,72	0,19
Escudo / Tiempo	-	-	-	-	0,00	Mayor Escudo / Tiempo	12,19	0,00
Vida y Defensa	-	-	-	-	1.667,32	Mayor Vida y Defensa	1.829,18	0,91

Ilustración 34 - Estadísticas 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	-	-	-	-

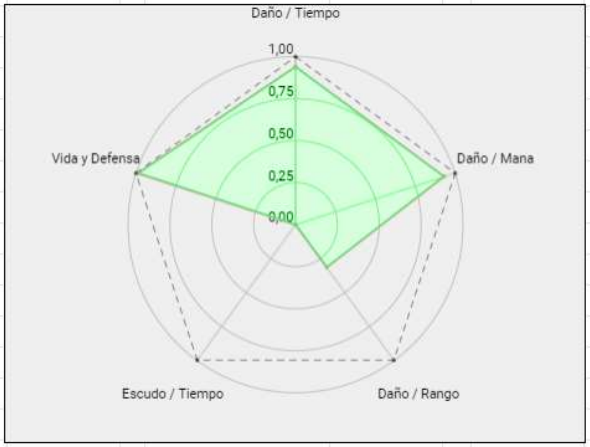


123,94	Mayor Daño / Tiempo	172,30	0,72
89,77	Mayor Daño / Mana	106,31	0,84
13.588,72	Mayor Daño / Rango	13.588,72	1,00
7,90	Mayor Escudo / Tiempo	12,19	0,65
1.638,40	Mayor Vida y Defensa	1.829,18	0,90

Ilustración 35 - Estadísticas Grim.exe

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	-	-	-	-

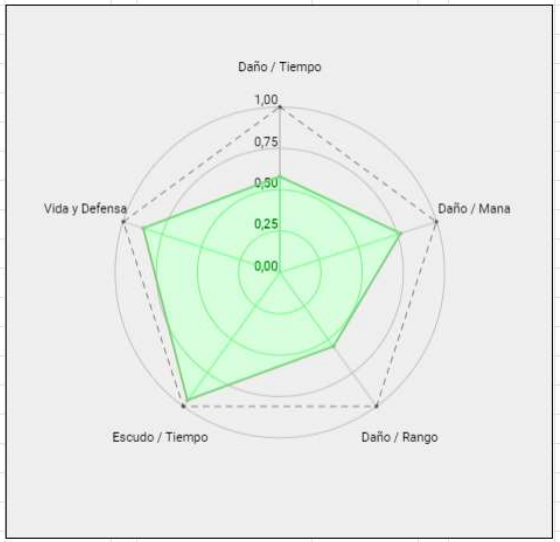


161,62	Mayor Daño / Tiempo	172,30	0,94
99,21	Mayor Daño / Mana	106,31	0,93
4.251,78	Mayor Daño / Rango	13.588,72	0,31
0,00	Mayor Escudo / Tiempo	12,19	0,00
1.813,68	Mayor Vida y Defensa	1.829,18	0,99

Ilustración 36 - Estadísticas 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	-	-	-	-

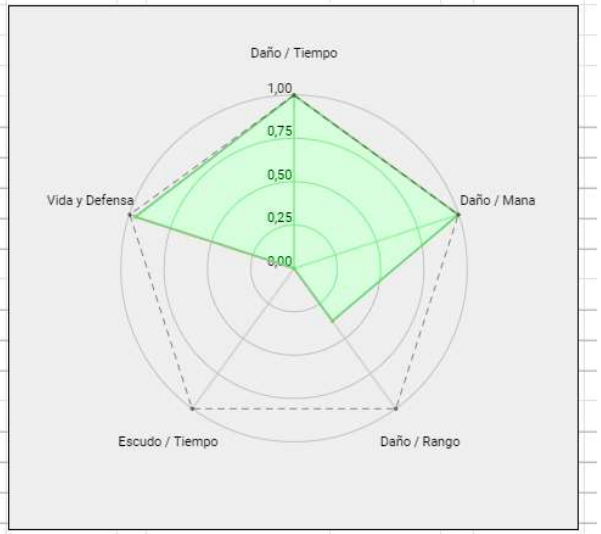


100,58	Mayor Daño / Tiempo	172,30	0,58
82,07	Mayor Daño / Mana	106,31	0,77
7.520,12	Mayor Daño / Rango	13.588,72	0,55
11,67	Mayor Escudo / Tiempo	12,19	0,96
1.594,13	Mayor Vida y Defensa	1.829,18	0,87

Ilustración 37 - Estadísticas 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

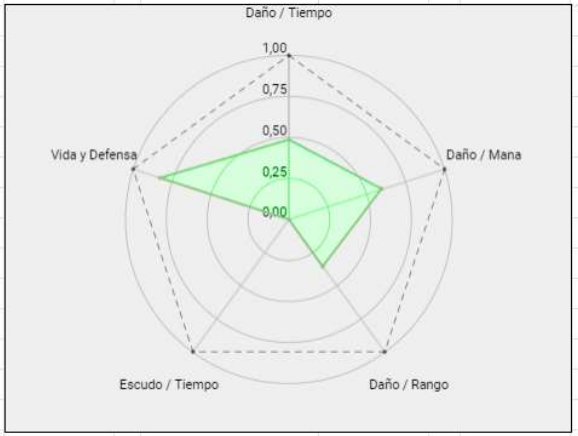


Balance								
Daño / Tiempo	93,11	41,57	32,69	4,93	172,30	Mayor Daño / Tiempo	172,30	1,00
Daño / Mana	83,80	14,55	3,52	4,44	106,31	Mayor Daño / Mana	106,31	1,00
Daño / Rango	134,08	931,11	457,60	3.550,40	5.073,19	Mayor Daño / Rango	13.588,72	0,37
Escudo / Tiempo	-	-	-	-	0,00	Mayor Escudo / Tiempo	12,19	0,00
Vida y Defensa	-	-	-	-	1.767,84	Mayor Vida y Defensa	1.829,18	0,97

Ilustración 38 -Estadísticas 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
Daño / Mana	54,03	2,93	2,35	4,37
Daño / Rango	540,30	1.493,69	616,73	2.185,67
Escudo / Tiempo	-	-	-	-
Vida y Defensa	-	-	-	-

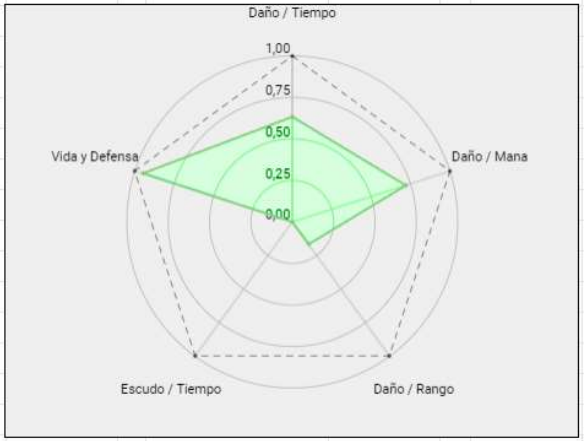


83,44	Mayor Daño / Tiempo	172,30	0,48
63,68	Mayor Daño / Mana	106,31	0,60
4.836,39	Mayor Daño / Rango	13.588,72	0,36
0,00	Mayor Escudo / Tiempo	12,19	0,00
1.508,49	Mayor Vida y Defensa	1.829,18	0,82

Ilustración 39 - Estadísticas 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
Daño / Mana	61,80	9,16	3,27	2,37
Daño / Rango	123,60	549,78	882,48	711,11
Escudo / Tiempo	-	-	-	-
Vida y Defensa	-	-	-	-

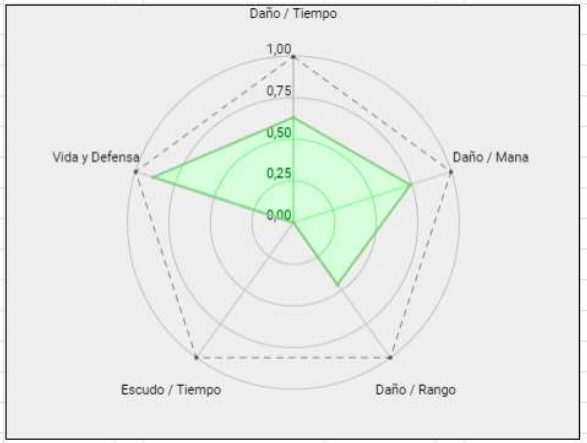


109,68	Mayor Daño / Tiempo	172,30	0,64
76,60	Mayor Daño / Mana	106,31	0,72
2.266,97	Mayor Daño / Rango	13.588,72	0,17
0,00	Mayor Escudo / Tiempo	12,19	0,00
1.726,38	Mayor Vida y Defensa	1.829,18	0,94

Ilustración 40 - Estadísticas 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	-	-	-	-

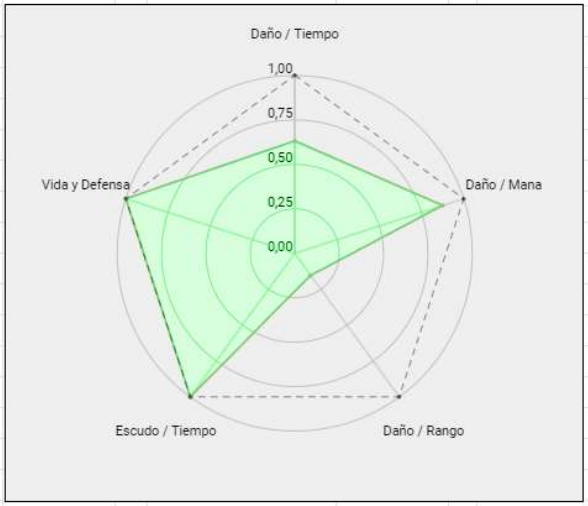


109,01	Mayor Daño / Tiempo	172,30	0,63
78,80	Mayor Daño / Mana	106,31	0,74
6.205,35	Mayor Daño / Rango	13.588,72	0,46
0,00	Mayor Escudo / Tiempo	12,19	0,00
1.627,36	Mayor Vida y Defensa	1.829,18	0,89

Ilustración 41 - Estadísticas 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 42 - Estadísticas Terra