

Futuro Imperfecto

Grado en Ingeniería Multimedia



Trabajo Fin de Grado

Autor:

David Segarra Rodríguez

Tutor/es:

Miguel Ángel Lozano

Enero 2014



Contenido

[1. Justificación y Objetivos 3](#_Toc513630668)

[2. Agradecimientos 4](#_Toc513630669)

[3. Dedicatoria 5](#_Toc513630670)

[4. Citas 6](#_Toc513630671)

[5. Índice de figuras 7](#_Toc513630672)

[6. Cuerpo del documento 7](#_Toc513630673)

[6.1 Introducción 7](#_Toc513630674)

[6.2 Estado del arte 8](#_Toc513630675)

[6.3 Estudio de mercado 9](#_Toc513630676)

[6.4 Objetivos 17](#_Toc513630677)

[6.5 Diseño e implementación: GDD (Documento de desarrollo del videojuego) 18](#_Toc513630678)

[6.5.1 Presentación del documento 18](#_Toc513630679)

[6.5.2 Introducción del juego 18](#_Toc513630680)

[6.5.3 Mecánicas de juego 20](#_Toc513630681)

[6.5.4 Niveles 23](#_Toc513630682)

[6.5.5. Interfaz 24](#_Toc513630683)

[6.5.6. IA 25](#_Toc513630684)

[6.6 Metodología y planificación del proyecto 27](#_Toc513630685)

[6.6.1 Resumen de la Fase 1 28](#_Toc513630686)

[6.6.2 Resumen de la fase 2 28](#_Toc513630687)

[6.6.3 Resumen de la fase 3 29](#_Toc513630688)

[6.6.4 Resumen de la fase 4 30](#_Toc513630689)

[6.7 Implementación 30](#_Toc513630690)

[6.7.1 Arquitectura del juego 37](#_Toc513630691)

[6.7.2 IA 37](#_Toc513630692)

[6.7.3 Animaciones 41](#_Toc513630693)

[7. Conclusiones 41](#_Toc513630694)

[8. Bibliografía y referencias 41](#_Toc513630695)

[9. Anexos 41](#_Toc513630696)

# Justificación y Objetivos

El por qué hacer un videojuego como trabajo de fin de grado es un motivo simple, me gustan los juegos. Decidí que era la mejor opción ya que, me daba libertad a la hora de enfocar este proyecto y decidir cómo iba a ser el producto final. Por otra parte, también pondría a prueba mi capacidad de tomar decisiones para añadir, modificar o eliminar características de dicho proyecto, así como, poner a prueba todo lo aprendido en el grado. En cuanto al uso del motor gráfico de Unreal Engine fue porque vi que es una herramienta potente que te permite diseñar y crear juegos mucho más completos que en otros motores, tener un resultado vistoso y también se adaptaba mejor al tipo de juego que voy a hacer, un **Hack and slash**

Los objetivos que me quiero lograr son:

* Crear un producto final completo y robusto
* Que el producto final sea disfrutable como divertido para el jugador
* Mejorar todo lo aprendido durante el grado
* Aprender una nueva tecnología usada en el desarrollo de videojuegos (Unreal Engine 4)
* Desarrollar un proyecto de ingeniería completo

# Agradecimientos

# Dedicatoria

# Citas

«Si algo puede salir mal, probablemente saldrá mal»

Edward A. Murphy Jr.

# Índice de figuras

[Figura 1: Ejemplo de QTE en God of War 9](#_Toc513815275)

[Figura 2: Ejemplo de tienda de mejora del Devil May Cry 3 10](#_Toc513815276)

[Figura 3: Ejemplo de mejora del arma del Devil May Cry 3 11](#_Toc513815277)

[Figura 4: Ejemplo del Modo Katana del Metal Gear Rising 12](#_Toc513815278)

[Figura 5: Ejemplo del Modo Katana del Metal Gear Rising 12](#_Toc513815279)

[Figura 6: Ejemplo del tiempo bruja del Bayonetta 13](#_Toc513815280)

[Figura 7: Ejemplo de diseño de nivel del Mirror’s Edge 14](#_Toc513815281)

[Figura 8: Ejemplo de diseño de nivel del Mirror’s Edge 15](#_Toc513815282)

[Figura 9: Ejemplo de arte visual del anime Psycho Pass 16](#_Toc513815283)

[Figura 10: Ejemplo de arte visual del anime Psycho Pass 16](#_Toc513815284)

[Figura 11Controles de teclado 22](#_Toc513815285)

[Figura 12: Controles de mando 23](#_Toc513815286)

[Figura 13: Menú de juego 25](#_Toc513815287)

[Figura 14: Ejemplo de BP 32](#_Toc513815288)

[Figura 15: Ejemplo de tipos de variables 33](#_Toc513815289)

[Figura 16: Ejemplo de Custom Event 34](#_Toc513815290)

[Figura 17: Ejemplo de función 35](#_Toc513815291)

[Figura 18: Ejemplo de función internamente 35](#_Toc513815292)

[Figura 19: Ejemplo de programación en C++ 37](#_Toc513815293)

# Cuerpo del documento

## Introducción

Durante todo el documento veremos como se desarrolla un videojuego completo, en este caso del género *Hack ‘n’ Slash* que se enfoca su jugabilidad en el combate cuerpo a cuerpo. Para este juego se utilizará lo último en tecnología como, por ejemplo, Unreal Engine. Este motor gráfico nos permitirá contar con grandes avances para facilitarnos la creación de juegos.

A continuación, veremos como se organiza este documento. Veremos los diferentes tipos de motor que hay en el mercado, qué juegos de este género han triunfado más o han aportado grandes cambios a este género, y para acabar, profundizaremos sobre cómo está creado este videojuego, desde lo que nos proporciona el motor hasta lo que se ha implementado; se irá indagando en cómo se ha ido diseñando este trabajo de ingeniería para crear un videojuego desde cero y cómo se estructura un proyecto desde su gestión hasta su producción y finalización pasando por distintas etapas de desarrollo.

## Estado del arte

En el mundo del desarrollo de videojuegos cada vez se está usando más motores gráficos existentes más que, utilizar tus recursos en crear tu propio motor. Pocas empresas son las que hacen esto último ya que, se está compitiendo con, por ejemplo, Unreal Engine de *Epic Games* o Unity de *Unity Technologies* que tienen un equipo de desarrollo dedicado a mejorar el motor e implementar mejoras.

Muchas empresas ya desarrollaron su propio motor gráfico en su momento para su uso privado o comercial como *Frostbite* de EA, pero, al contrario que los motores anteriores, la licencia de este es pago. Es cierto que, Epic Games y Unity reciben royalties si publicas un juego con su motor y ganas más de cierta cantidad establecida.

Hoy en día, muchas empresas usan más estos motores gratuitos para crear sus juegos ya sean por su versatilidad a la hora de crear un videojuego o por ser empresas que no tiene presupuesto para pagar licencias de otros motores privados. Al final, la tecnología más adecuada para desarrollar tu videojuego ya sea como empresa o como individuo, es la que más se adapta a lo que quieres crear. Es cierto que, puedes hacer cualquier juego con cualquier motor, pero, cada motor tiene unas especificaciones más enfocadas para crear un tipo de videojuego u otro. Hay que saber que motor nos puede beneficiar más para gestionar bien los recursos.

Sobre los motores anteriores, hay algunos que están enfocados, o adaptados, a otras plataformas. Unity, por ejemplo, esta muy bien adaptado para crear juegos en móviles que Unreal, aunque se pueden crear juegos para esa plataforma. En cambio, Unreal está muy enfocado para juegos en PC o consola y que son en primera persona, pero, ya lleva muchos años siendo adaptado para poder hacer cualquier tipo de juego con este motor para poder ganar versatilidad y más usuarios.

## Estudio de mercado

Ver en que destacan los distintos juegos del género al cuál voy a enfocarme es importante para diseñar e implementar mejores mecánicas para atraer al jugador, es por eso, que hay que analizar bien a los distintos juegos para poder diferenciar al tuyo del resto.

Después de un estudio de los distintos juegos más icónicos que tiene este género he ido analizando lo que les diferenciaba de los demás. Hay muchos juegos de este género, como de los demás, y me he querido centrar en los más importantes para mi o que creo que más han influenciado más en el género o han conseguido destacar y perdurar más en el tiempo.

Empezamos por la saga de *God of War.* Esta saga de videojuegos nació en 2005 con su primer juego para la *Play Station 2* desarrollado por el SCE Santa Monica Studio. Lo que hizo destacar a este juego sobre sus competidores es un control del personaje muy gratificante y divertido como también la introducción de un nuevo concepto en los videjojuegos: *los quick time event* o eventos donde el jugador tiene que pulsar una serie de botones para conseguir su objetivo. Santa Monica usó esto como una mecánica para ejecutar ataques definitivos sobre los jefes finales, enemigos duros y también los comunes para recompensar al jugador por más puntos.



Figura : Ejemplo de QTE en God of War

Continuamos con otra saga de videojuegos llamada *Devil May Cry* creada por Capcom en 2001 siendo una de las pioneras en sentar las nuevas bases de este género para la generación de esa época. En lo que destacaba era en un sistema de combos distinto al de los demás basado en el tiempo de pulsado entre botones, dependiendo del lapso ejecutabas distintos ataques. También tenía un sistema de puntuaciones por niveles y un sistema de mejora del personaje enfocado a las distintas armas del juego, creando así distintas maneras de personalizar tu estilo de juego.



Figura : Ejemplo de tienda de mejora del Devil May Cry 3



Figura : Ejemplo de mejora del arma del Devil May Cry 3

En las anteriores imágenes podemos ver cómo funciona el sistema de mejora del *Devil May Cry* donde podemos seleccionar el arma que queramos y mejorarle cualquier parámetro que tenga.

Seguimos con la desarrolladora de *Platinum Games*, quienes han hecho muchos juegos de este género que han sido muy importantes en el tiempo por tener unas mecánicas muy pulidas que responden muy bien a lo que hace el jugador. Entre estos juegos podemos destacar la saga de *Bayonetta* creada en 2009 que, como todos los juegos de este género hechos por ellos presentan un gran control del personaje, se diferencian del resto por tener una mecánica de que cuando se esquiva un ataque del enemigo, el tiempo se ralentiza para los enemigos permitiéndonos atacar más fácilmente. También tienen el *Metal Gear Rising* creado en 2013 que destaca en un gran control del personaje que se nos permite bloquear casi cualquier tipo de ataque permitiéndonos contraatacar y presenta una de las mecánicas más vistosas: “el modo katana” llamado así por los desarrolladores que consiste en tener total libertad a la hora de realizar cortes con tu katana permitiéndonos cortar por donde queramos y realizar ataques.



Figura : Ejemplo del Modo Katana del Metal Gear Rising



Figura : Ejemplo del Modo Katana del Metal Gear Rising

En estas imágenes podemos ver cómo funciona el modo katana del Metal Gear Rising. Puedes cortar en cualquier dirección y si cortas en el punto débil del enemigo puedes recuperar toda la vida con lo que le extraes.



Figura : Ejemplo del tiempo bruja del Bayonetta

En esta última imagen es del juego *Bayonetta* dónde se ve como se activa el *tiempo bruja* y el tiempo va más lento lo cual, esto te permite ejecutar más combos sobre un enemigo.

Una vez visto todos estos juegos que son los más icónicos de este género podemos ir viendo cuál de ellos ha sido los que más han funcionado o inspirado a otros juegos. A grandes rasgos, todos han aportado algo al género, unos más que otros, que les han hecho destacar y marcar tendencias que otros han seguido cómo: el sistema de mejora del personaje, *quick time events*, sistemas de combos avanzados, etc.…

Otras de las demás mecánicas expuestas son más típicas y acotadas a esos juegos por cómo es el personaje principal y forman parte de sus características y de cómo es él que no tendría mucho sentido imitarlas en otro tipo de juegos, sino más bien, adaptarlas al juego en cuestión.

En resumen, todos los juegos nombrados anteriormente han aportado bastante a la industria que se han podido permitir sacar secuelas de ellos e incluso sacar juegos del mismo género.

En cuanto al arte del juego, se ha pensado adoptar una temática futurista de estilo plano y simple. Esto significa que, comparado con otros estilos futuristas más cargados de detalles, el objetivo de este es hacer un estilo simple en cuanto a los componentes de los escenarios usando el blanco o colores fríos en los objetos.

Hay otros juegos dónde se usa este tipo de arte porque quieren dar un mensaje visual de limpieza y bienestar, aunque luego no sea así. En otros juegos se usan un estilo mucho más detallado y cargado para que se vea más espectacular.

Para este juego se han ido analizando y visto distintos estilos de arte de otros juegos hasta encontrar algo parecido a lo que se quería, uno de esos juegos es *Mirror’s Edge*, juego de ambientación futurista que usa este tipo de diseño:



Figura : Ejemplo de diseño de nivel del Mirror’s Edge



Figura : Ejemplo de diseño de nivel del Mirror’s Edge

Como se muestran en estas dos imágenes, este es uno de los tipos de arte que se quiere logar para los escenarios del juego y los distintos complementos de éste.

Otro del tipo de arte analizado es del anime *Psycho Pass*, ambientado en un futuro próximo, que hace poco le han sacado un juego estilo *visual novel,* pero, su arte y los edificios también es parte del estilo que se quiere captar en el videojuego, estilo liso y limpio con figuras rectas e industriales acompañados algunos de los edificios con bordes redondeados perfectos.



Figura : Ejemplo de arte visual del anime Psycho Pass



Figura : Ejemplo de arte visual del anime Psycho Pass

Como podemos ver en estas dos imágenes, este estilo también se adapta a lo que se pide del juego o cómo se quiere logras que se vea el estilo visual del juego.

## Objetivos

*.*

Los objetivos que cumplir con este proyecto son:

* Crear un producto de calidad y que funcione.
* Crear un proyecto de ingeniería.
* Aprender a usar nuevas tecnologías que se usan en el mercado
* Mejorar las capacidades de programación y organización de un proyecto
* Mejorar las capacidades de programación de videojuegos

Objetivos del videojuego:

* Crear una arquitectura para el videojuego en Unreal.
* Implementar una IA acorde al estilo del videojuego.
* Implementar las mecánicas base del género.
* Integrar las animaciones correspondientes para cada actor del juego.
* Implementar una interfaz gráfica para gestionar el juego.
* Agregar los modelos correspondientes para el juego.

## Diseño e implementación: GDD (Documento de desarrollo del videojuego)

### 6.5.1 Presentación del documento

En esta parte del documento veremos como se estructura un videojuego siguiendo la estructura del GDD (documento de desarrollo de un videojuego) que sigue una serie de pautas dónde hay que definir qué características presenta el videojuego, como se va a estructurar las mecánicas del jugador, que elementos va a tener la IA, cómo se va a mover e interactuar el jugador con el escenario y los distintos elementos que lo componen: objetos, paredes, enemigos, etc.

### 6.5.2 Introducción del juego

El videojuego Futuro Imperfecto se desarrollará usando el motor Unreal Engine y será para la plataforma PC. Aquí se irán especificando los distintos apartados que se implementarán en el juego y que sirva como guía para el desarrollo.

Futuro Imperfecto es un juego que está basado en distintos tipos de juego del mercado como, por ejemplo: Bayonetta, donde tendrá distintos elementos típicos de la saga; Nier Automata, dónde cogerá distintos elementos hack ‘n’ slash: el uso de combos cuerpo a cuerpo, el movimiento del personaje será en 3D donde podrá moverse libremente en un escenario, con límites establecidos, donde tendrá que ir avanzando para acabar el nivel. En algunos momentos del nivel, no todos, la cámara podrá cambiar de perspectiva para dar distintos enfoques. (Ej: <https://youtu.be/d15dPiC9V8Q?t=16m49s> ).

#### 6.5.2.1 Concepto del juego

Futuro Imperfecto es un videojuego donde controlamos a nuestro pobre protagonista que vive en una sociedad distópica dónde el gobierno ha creado una organización de asesinos que eliminan a los criminales. Un día, nuestro protagonista es fijado como un criminal sin haber realizado ningún acto de ese estilo, ¿será un fallo del sistema o habrá algo más?

#### 6.5.2.2 Características principales

El juego se basa en estos apartados:

* Mecánicas fluidas: las mecánicas básicas del juego tienen que ser agradables para el jugador y que se sienta cómodo al saltar, disparar y realizar ataques cuerpo a cuerpo.
* Frenético: cuando el jugador ponga en práctica las mecánicas de combate, tiene que sentir que se sientan satisfactorias a la hora de luchar.

#### 6.5.2.3 Género

Futuro Imperfecto es una mezcla de varios tipos de géneros:

* Acción: se pondrán a prueba la velocidad, el tiempo de reacción, la destreza y capacidad de leer los patrones de los enemigos. El jugador tendrá que usar los distintos elementos a su disposición para poder superar las distintas fases del juego.
* RPG: tendremos distintos poderes que conseguiremos al derrotar al jefe de cada zona que pasemos. Este poder está relacionado con el poder del jefe en cuestión.
* Hack ‘n’ slash: nuestro personaje dispone de un ataque cuerpo a cuerpo que podrá combinar para realizar distintos combos. Combo de ataques débiles o combos entre ataques débiles y fuertes.

#### 6.5.2.4 Propósito y público objetivo

Este juego está enfocado a un público que le gusten los juegos de acción y aventuras con una jugabilidad fluida y entretenida.

#### 6.5.2.5 Estilo visual

La atmosfera del juego se ambientará en un futuro próximo con elementos redondeados, lisos y limpios, edificios con ventanas grandes, con fachadas limpias y de estructuras poco orgánicas, siendo más perfectas y rectas o con pequeños detalles redondeados de curvatura perfecta; de colores claros y/o fríos para dar la sensación de un futuro distópico. En algunas partes de la ciudad dónde ocurren los hechos del juego están algo más sucias, con toques de suciedad en las paredes o dibujos, y otras estarán más limpias y relucientes dando a lugar distintos sitios con características únicas.

Como un ejemplo más claro, podemos decir que el estilo visual que tendrá este juego será algo parecido al del juego Mirros’s Edge principalmente y entre otros (tengo que poner más ejemplos)

#### 6.5.2.6 Alcance

Este videojuego está enfocado a gente que quiera eliminar muchos enemigos y pasar un buen rato.

### 6.5.3 Mecánicas de juego

Aquí se irán especificando las distintas mecánicas que tendrá el juego y demás elementos relacionados a lo largo de los distintos apartados.

#### 6.5.3.1 Jugabilidad

Mecánicas en 3D donde el jugador podrá moverse libremente por el espacio asignado, saltar, interactuar con los enemigos y objetos del escenario. La base de estas mecánicas será del tipo hack ‘n’ slash, donde el jugador tendrá una serie de ataques para derrotar a los enemigos.

El jugador también podrá cambiar de modo de ataque, posee un arma distancia que cambia el estilo de juego, de atacar a corta distancia a mayor. Este ataque tendrá sus propios combos distintos a los de cuerpo a cuerpo y el jugador podrá cambiar de estilo cuando más lo necesite. Ambos modos de juego poseen una barra que se llena, una vez esté completa, el jugador podrá relazar un ataque más potente de ese estilo que causará más daño a los enemigos. Una vez usa ese poder, tendrá que esperar unos segundos para volver a llenar la barra. La barra se va cargando con el uso de los combos, si cambias de estilo, perderás el progreso de esta barra. Si estás durante unos segundos sin hacer combos esta barra irá decreciendo, también si recibes daño de los enemigos.

#### 6.5.3.2 Flujo del juego

Esquema que hay que poner un diagrama

Menú principal 🡪 Nivel tutorial 🡪 Jefe del tutorial 🡪 Nivel X 🡪 Jefe del nivel X

El modo de juego principal es de un jugador, donde se nos irá contando una historia y tendremos que recorrer una serie de niveles semi-lineales (estos niveles no serán completamente una línea recta en lo relacionado al diseño de nivel, sino que, el jugador tendrá momentos donde habrá más caminos o lugares grandes donde hacer varias cosas cómo resolver algún pequeño puzzle)

El jugador dispondrá de todos los combos que puede hacer y no tendrá que desbloquear ninguno (es posible que se cambie según el desarrollo del juego) y las mejoras que podrá comprar o desbloquear serán de salud, energía del guante de magnetismo y/o mejoras de daño de ataque. Estas mejoras se podrán conseguir buscando por el escenario o comprándolas antes de elegir la misión que quieres hacer, este sistema nos permitirá ir mejorando a nuestro personaje para que sea más fuerte y nos sea más fácil el juego.

#### 6.5.3.4 Movimiento y físicas

El movimiento del personaje no está regido por ninguna fuerza física sino por, velocidades; aunque si colisiona con un objeto al cual se le ha aplicado una fuerza, éste se verá afectado por ella.

Los enemigos son afectados por la fuerza magnética que crea el jugador para ser lanzados por los aires.

##### 6.5.3.4.1 Interacción entre elementos

El jugador podrá interactuar con los distintos objetos que habrá en el escenario que se especificarán en este apartado:

* Podrá colisionar con distintos elementos del escenario ya sean paredes, los propios enemigos, cajas, elementos decorativos del escenario, puertas.
* Habrá objetos los cuales el jugador podrá interactuar de ellos de formas distintas. Hay puertas que podrá abrirlas para pasar a otras zonas, paneles que permiten acceso a zonas más importantes u objetos que podrá romper que tienen objetos consumibles o importantes para la historia.
* Otra mecánica de juego que habrá es un rayo magnético que atrae a los enemigos (que son robot de acero). El jugador podrá atraer a uno (o varios) enemigos para poder eliminarle o lanzarlos por los aires. Este rayo tiene una barra de energía que se ira cargando cuando se cojan baterías o mediante ataques a los enemigos. A los jefes de nivel o enemigos más fuertes, será más difícil poder atraerles o repelerles ya que, necesitan estar debilitados para poder usar el rayo sobre ellos.

##### 6.5.3.4.2 Controles

Los controles están más pensados para jugar con mando, pero se adaptarán lo mejor posible al teclado y ratón.



Figura Controles de teclado

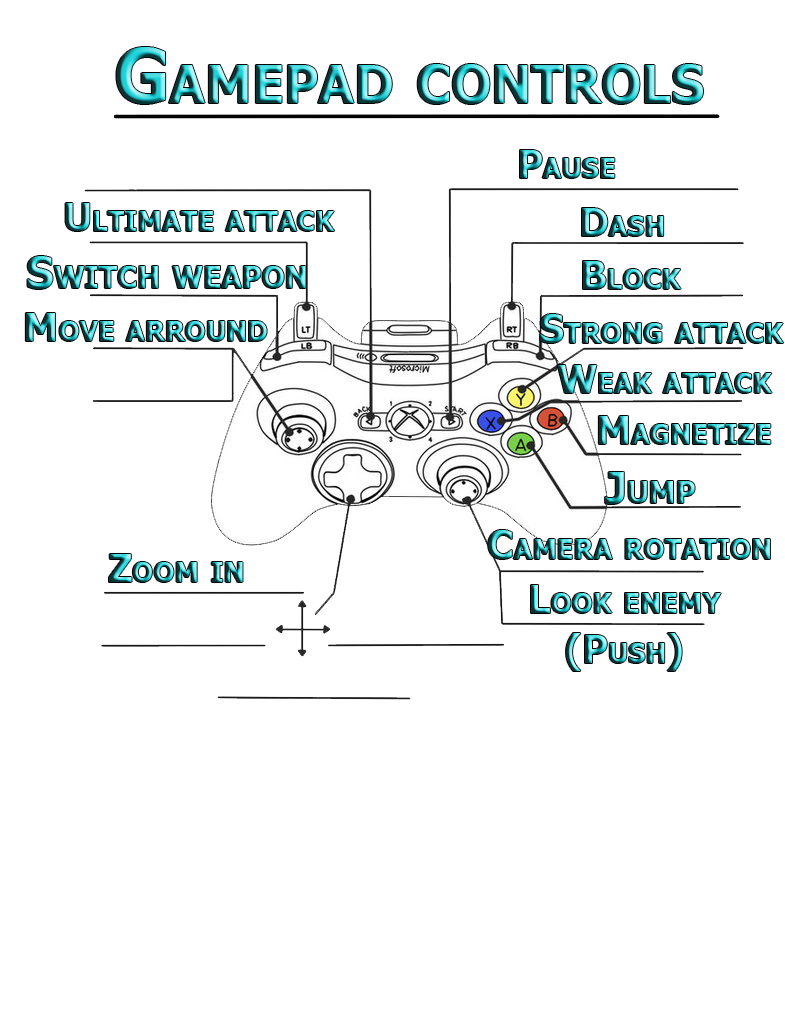


Figura : Controles de mando

#### 6.5.3.5 Historia

Los eventos del juego ocurren en el futuro donde la ciudad donde transcurren los hechos está regida por una organización que eliminan a la gente suponen un peligro para ellos. Nuestro protagonista, sin saber por qué, es objetivo de esta organización que envía robots para matarle. Conforme avancemos en la historia veremos porqué ocurre esto y la razón por la que el protagonista va viendo que ha perdido fragmentos de sus recuerdos en momentos concretos de su vida donde ve que no cuadran bien.

La historia empieza con nuestro protagonista haciendo una misión para un equipo perteneciente a esa organización encargados de recopilar información sobre el objetivo a asesinar, aunque no todo iba a salir bien. Justo antes de acabar, es traicionado por su organización y no sabemos por qué, por eso, tendremos que escapar y buscar la razón por la cual somos objetivo de ellos.

### 6.5.4 Niveles

#### 6.5.4.1 Diseño general del nivel

El esquema base que seguirán los niveles es: una zona principal, dónde el jugador empieza; distintas fases donde hay distintos tipos de enemigos en las que tendrá que ir avanzando; y, por último, la zona del jefe final dónde el jugador tendrá que enfrentarse a un enemigo más fuerte y ganarle.

#### 6.5.4.2 Elementos del nivel

Los niveles tendrán distintos elementos, tanto visuales como objetos interactuables, para que cada uno se siento algo distinto al anterior. También aparecerán los enemigos de maneras distinta a la de los demás niveles.

#### 6.5.4.3 Diferentes diseños del nivel

(Añadir bocetos conceptuales de distintos niveles diseñados, sea en papel o montados en Unreal)

### 6.5.5. Interfaz

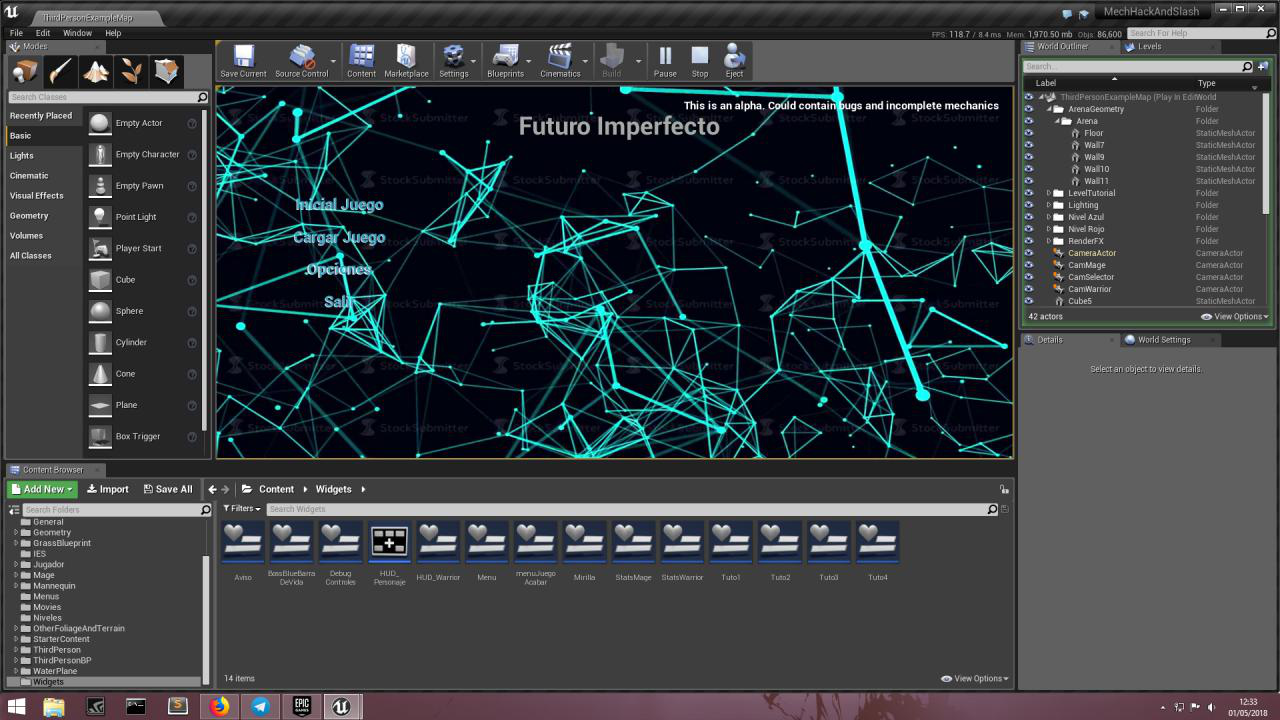


Figura : Menú de juego

#### 6.5.5.1 Diagrama de flujo

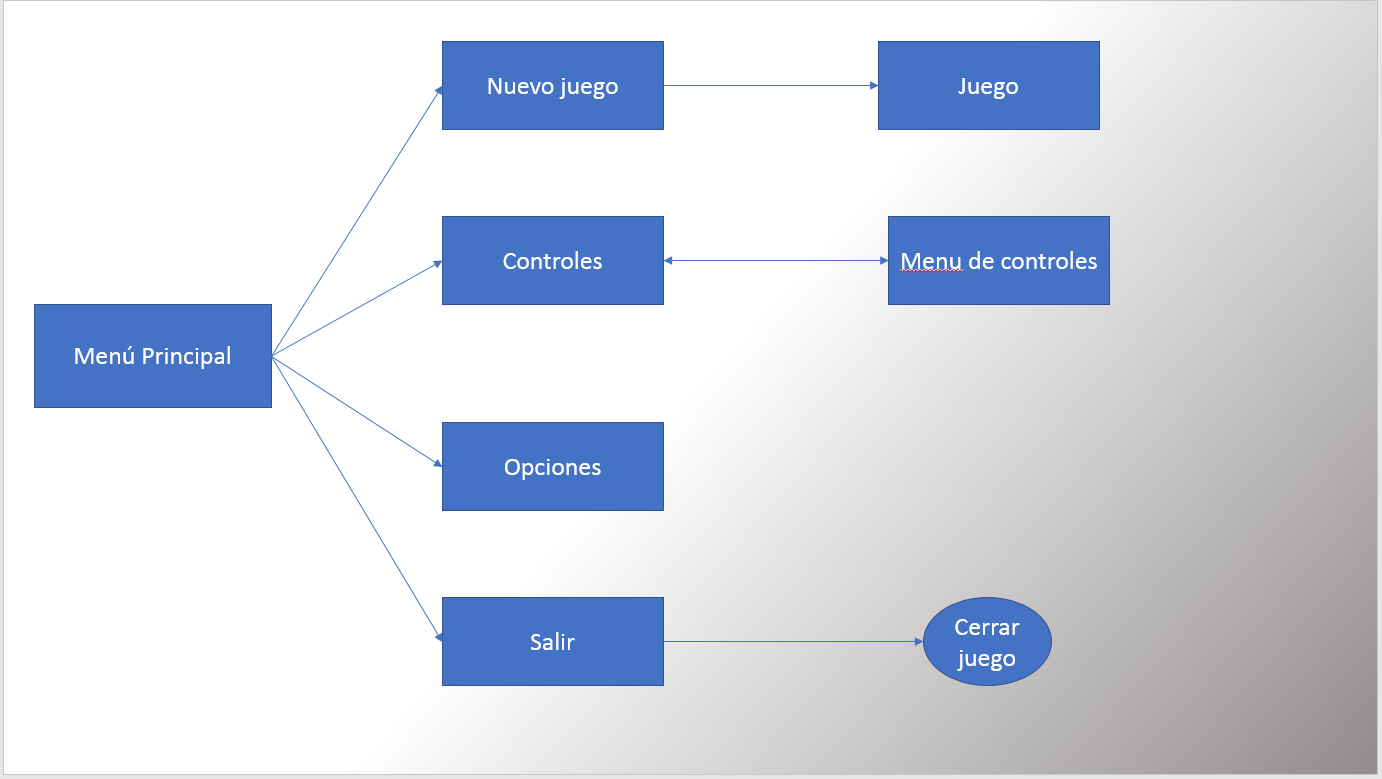


Figura : Diagrama de los menus

Desde el menú principal podemos empezar una partida, ver los controles o cerrar el juego y acabar la ejecución del programa.

#### 6.5.5.2 Cámara

La cámara será en 3º persona donde el jugador podrá rotar sobre el personaje y se irá ajustando, dependiendo de las condiciones del escenario, si está cerca de alguna pared se regula la distancia automáticamente

### 6.5.6. IA

La IA (Inteligencia Artificial) es uno de los puntos más importantes de este tipo de juegos ya que, tiene que ser desafiante pero entretenida a la vez.

Habrá distintos tipos de enemigos. Están divididos en 3 grandes grupos:

* Enemigo básico: Este enemigo será el que más abunde en el juego, en distintas variantes como por ejemplo un enemigo volador, otro normal o con un escudo… Su comportamiento es de patrullar una zona o estar quietos hasta que se encuentran con el jugador. Una vez le ha visto, éste va a por él usando distintos patrones de ataques. Estos ataques estarán sujetados a su visión, distancia del jugador y también a su propia vida.
* Enemigo fuerte: Al contrario del enemigo básico, este enemigo es mucho más fuerte e inteligente que él. Serán menos abundantes que estos y tendrán comportamientos más elaborados para que resulten más desafiantes al jugador. También habrá distintos tipos de enemigos: voladores, con espadas, a distancia…

Su set de movimientos estará mucho más completo que el del enemigo básico, un ejemplo es que si está a una distancia media hará un set de ataques distinto que a corta distancia; y si tiene menos vida podrá ejecutar distintos ataques que antes no podía y usarlos junto a los anteriores. Su movimiento por el escenario es muy parecido al de los enemigos normales, se irá moviendo por una ruta o zona hasta que vea al jugador o sea atacado por éste.

* Enemigo jefe: Cuando el jugador llegue al final del nivel, tendrá que enfrentarse a un jefe final donde pondrá a prueba todo lo que ha aprendido. Este jefe es mucho más resistente y fuerte que los demás enemigos. Sus patrones serán más complejos que los otros enemigos e irán cambiando conforme a menos vida tenga. También puede que el enemigo sea más agresivo que en un principio o más precavido.

Sobre como los enemigos decidirán la distancia adecuada para realizar sus ataques y categorizar dicha distancia se usará la lógica difusa o algún algoritmo que permita realizar la estimación de la distancia más humana o real. Todas las IAs conocerán las limitaciones del escenario y por donde se pueden mover gracias a los algoritmos de pathfinding y el navmesh correspondiente que calcula las zonas transitables por donde se puede navegar.

Añadir tabla resumen para ver las distintas funciones de la IA

|  |  |  |  |
| --- | --- | --- | --- |
|  | Enemigo Básico | Enemigo Fuerte | Jefe final |
| Sistema de movimiento básico  (pathfinding/navmesh) | x | x | x |
| Sistema de ataque básico (basado en distancias) | x | x | x |
| Sistema de combos avanzado (basado en la vida restante) | - | x | x |
| Comportamiento básico | x | x |  |
| Comportamiento avanzado (Behavior tree) |  |  | x |

## 6.6 Metodología y planificación del proyecto

La planificación del desarrollo del proyecto seguirá una metodología ágil basa en iteraciones, las cuales, durarán entre 2 o 3 semanas, que forman las distintas fases repartidas a lo largo de los meses. Estas iteraciones tendrán objetivos primarios y secundarios dónde cada uno tiene más prioridad que otros. Si por algún motivo no se ha podido completar alguno de los objetivos, se deberá explicar el por qué no se ha conseguido y estudiar si se puede aplazar para otra iteración o ver si se puede eliminar del proyecto o simplificarlo si es posible.

Esta metodología es una de las mejores que se puede aplicar a un proyecto siendo sólo una persona ya que, plantearse objetivos a corto plazo para ver sus resultados es una buena manera de automotivación para seguir adelante con el proyecto y no abandonarlo.

En cuanto al software utilizado para realizar el juego se usará el motor gráfico de [Unreal Engine en su versión 4.16](https://www.unrealengine.com/en-US/what-is-unreal-engine-4). Se ha elegido este motor por su gran potencia y versatilidad a la hora de desarrollar todo tipo de juegos cómo también su gran impacto en el mercado, cada vez más y más juegos se crean con este motor debido a la calidad y fiabilidad del producto desarrollado.

Los modelos del juego serán creados usando el programa de modelado Maya. Es un programa muy usado en el mundo empresarial por su gran dinamismo y versatilidad a la hora de modelar cualquier tipo de objeto del mundo. Crea modelos de gran calidad y permite la exportación a cualquier tipo de motor gráfico de hoy en día.

Para controlar las distintas versiones de los documentos, versiones del juego y demás archivos del proyecto se usará la aplicación de GitHub, un software de control de versiones que permite sincronizar código de manera más optima como también controlar que tipo de archivos se suben como también poder volver a versiones anteriores si ocurre algún error o se pierde información importante.

### 6.6.1 Resumen de la Fase 1

En la primera fase se propuso como objetivo final conseguir una demo jugable de las mecánicas básicas que iba a tener el juego que son:

* Programar las mecánicas básicas de movimiento
* Programar las mecánicas básicas de ataque
* Diseño de nivel básico
* Diseño básico de menús

Hay objetivos que se marcaron para esta fase que no se han podido cumplir y se han descartado crearlos para sustituirlos por otros gratuitos:

* Texturas de objetos
* Modelados
* Diseño de armas

### 6.6.2 Resumen de la fase 2

En esta segunda fase se propuso como objetivo principal completar una IA funcional cumpliendo ciertos requisitos que son:

* Movimiento de la IA
* Buscar y localizar al jugador
* Atacar al jugador
* Muerte
* Patrullaje
* Crear una Alpha del juego (una versión jugable con principio y fin)
* Diseñar el nivel tutorial y nivel final

Se retrasaron algunos de los distintos objetivos por falta de tiempo:

* Depurar mecánicas del jugador
* Depurar mecánicas del enemigo
* Modificar los niveles
* Mejorar los tutoriales

Otros objetivos se descartaron por falta de tiempo y/o no estaban relacionados con el objetivo principal del proyecto y se han sustituidos por elementos gratuitos:

* Animaciones de los personajes y enemigos
* Modelado de los personajes y enemigos
* Modelado de armas
* Modelado de elementos del escenario
* Texturas de los diferentes modelos

### 6.6.3 Resumen de la fase 3

En esta iteración el objetivo principal es completar las tareas retrasadas a lo largo del proyecto que fueran más críticas:

* Depurar mecánicas del jugador
* Depurar mecánicas del enemigo
* Cambiar el diseño de los niveles
* Buscar modelos de los elementos del escenario

### 6.6.4 Resumen de la fase 4

En la última iteración el objetivo principal es tener el juego ya completo y jugable, a falta de ultimar detalles del juego:

* Mejorar varios aspectos del juego
* Implementación de los menús definitivos
* Adaptar los menús al gamepad
* Mejorar el estado de “final de juego”
* Implementar botiquines

## 6.7 Implementación

*La sección de implementación supongo que todavía le queda bastante, pero sería conveniente empezar situando al lector en cómo se realiza el desarrollo de UE4, cómo se organiza el código, y dando en primer lugar una visión general de la arquitectura de tu juego. Una vez se ha visto de forma general, en cada subvención puedes ir detallando cómo se ha implementado cada parte del proyecto (IA, animaciones, menús y transiciones, gestión de datos, etc).*

*Ya he confirmado la solicitud del proyecto. Sobre la memoria, en la parte donde introduces los blueprints sería conveniente estructurar mejor el texto para que el lector entienda bien en qué consisten. Se puede empezar hablando de qué se puede representar con un nodo de un blueprint, viendo una figura donde el nodo se vea bien, y luego hablar de las conexiones que puede haber entre ellos y otros elementos, para ver lo que se puede hacer con ellos. Luego, antes de pasar a la IA, es importante dar un vistazo general a la arquitectura del videojuego, y luego ya pasar a detallar cada módulo, incluyendo la IA.*

Unreal tiene dos menaras diferenciadas de programar un juego. Una de ellas son los *BluePrints (nombrados BP para abreviar)* y la otra es C++ adaptado a su motor gráfico, de ambas formas puedes crear un videojuego y se hablará sobre sus características luego.

Los *BP* es una manera de programar gráfica donde se colocan nodos, que contienen líneas de programación en C++, que ejecutan una serie de acciones. Los BP son una herramienta muy potente que te permite implementar muchas mecánicas de forma rápida y ver su funcionamiento, también son muy usadas para prototipado rápido y de aprendizaje en el motor. Por otro lado, C++, permite crear funciones o características específicas que no existen en los BP y es una herramienta muy versátil; C++ es mucho más rápido a la hora de ejecutarse, por lo que, C++ es mucho más eficiente a la hora de que un juego sea sólido y estable. A la hora de la verdad, los BP y C++ son usados a la vez, hay cosas que son más rápidas de implementar en un BP que en C++ y viceversa. La fusión de ambos permite tener un juego eficiente y estable para su disfrute.

Una vez introducido cómo se programa con Unreal, vamos a ver las características de cada uno. Primero veremos los BP y como son a través de imágenes. Veremos que variables tiene, que elementos podemos crear, cómo funciones o métodos; y como fluye la ejecución del programa.

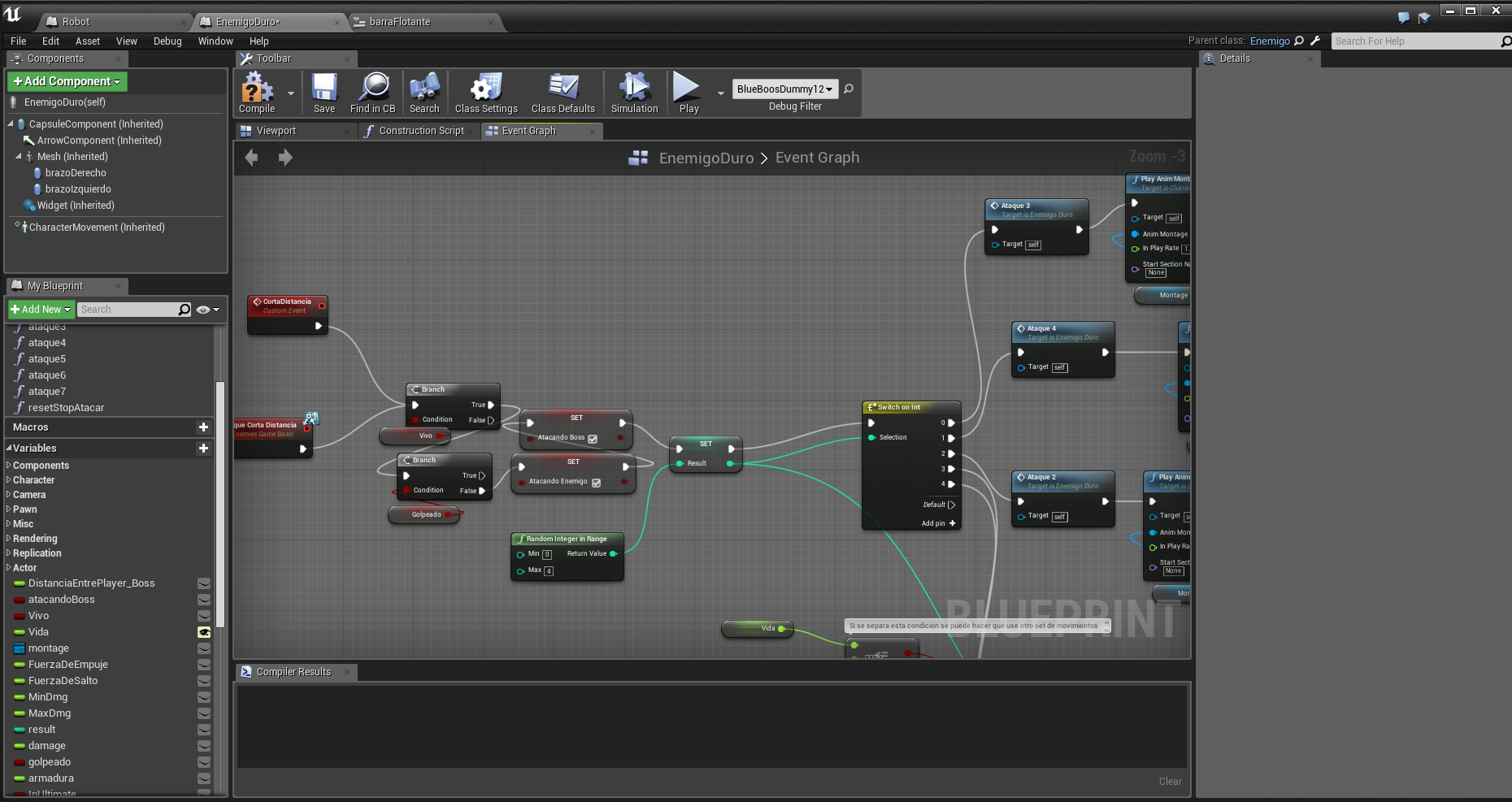


Figura : Ejemplo de BP

En la Figura 15 podemos ver un ejemplo de BP, usado en el juego, dónde iremos analizando las distintas partes que se observan en la imagen.

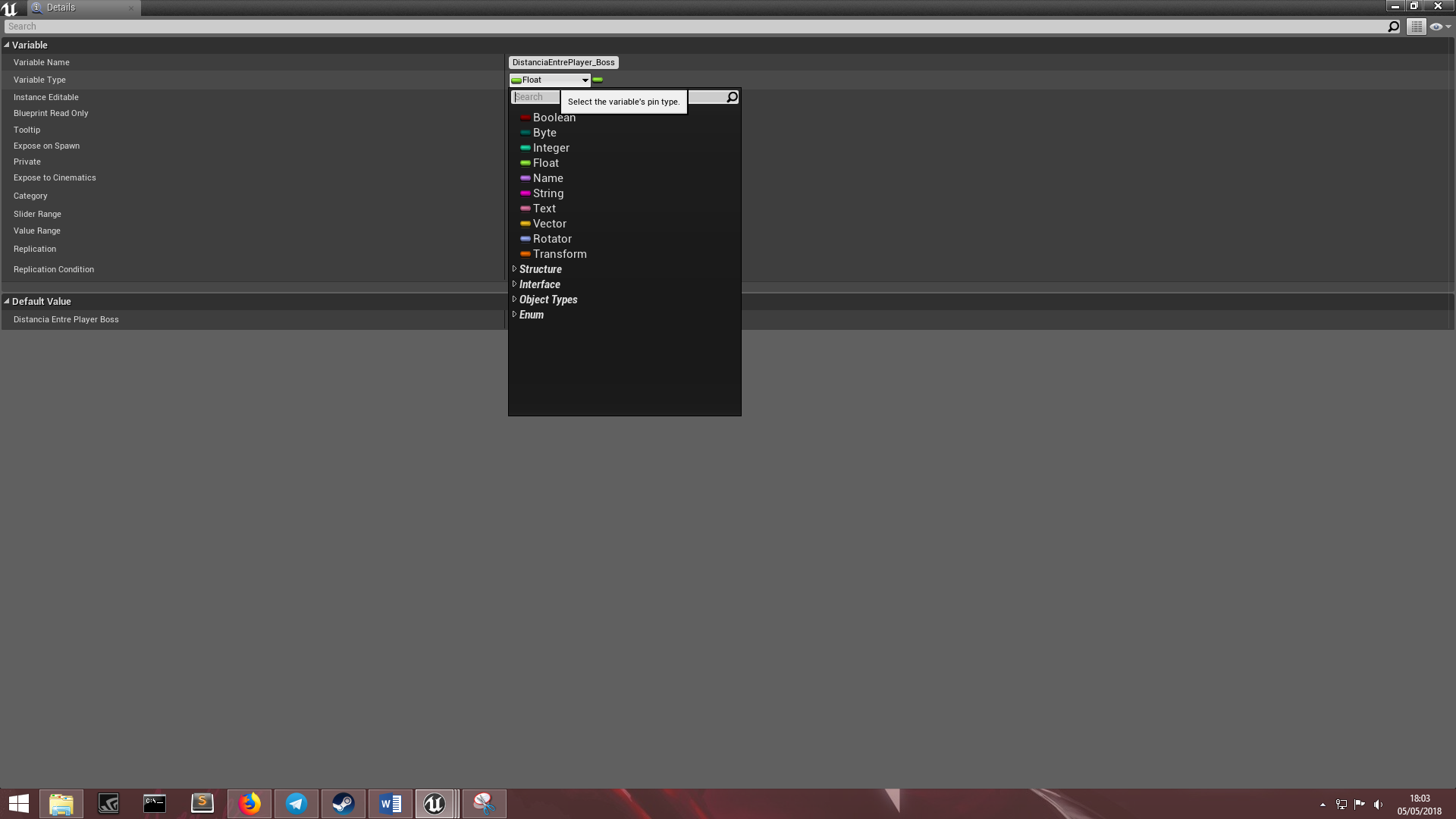


Figura : Ejemplo de tipos de variables

En la figura 16 podemos ver los tipos de variable que se pueden crear en un BP, desde las primitivas, hasta más complejas como: BP propios que tiene Unreal o que has creado tú u objetos que tienen su propia estructura. Cada tipo de variable tiene un color identificativo para que sea más fácil reconocerlas visualmente. En caso de las variables primitivas que se ven en la figura 13 son las más usadas, aunque, si es una variable más compleja cómo los objetos qué están en el grupo de “Object Types”, suelen ser de color azul y dependiendo de que atributo necesitemos de ese objeto, una referencia a éste o que clase de objeto es, comparten el mismo color porque son variables de la misma gama.



Figura : Ejemplo de Custom Event

En la figura 17 podemos ver un *custom event*, que es un elemento que tiene un funcionamiento parecido a una función en programación de objetos. Este elemento enlaza con los cálculos matemáticos o funcionalidades de nuestro BP y pueden ser llamados desde otros BP para que se ejecuten. Estos eventos pueden recibir variables, como se han visto en la Figura 13 dónde tendrá un pin de color correspondiente al tipo de variable que necesite, que pueden modificar otras variables del mismo BP o usarlas para guardar su información dentro de éste.

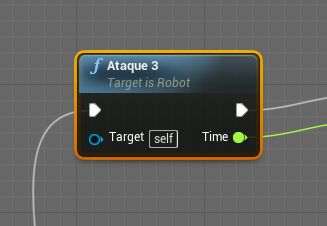


Figura : Ejemplo de función

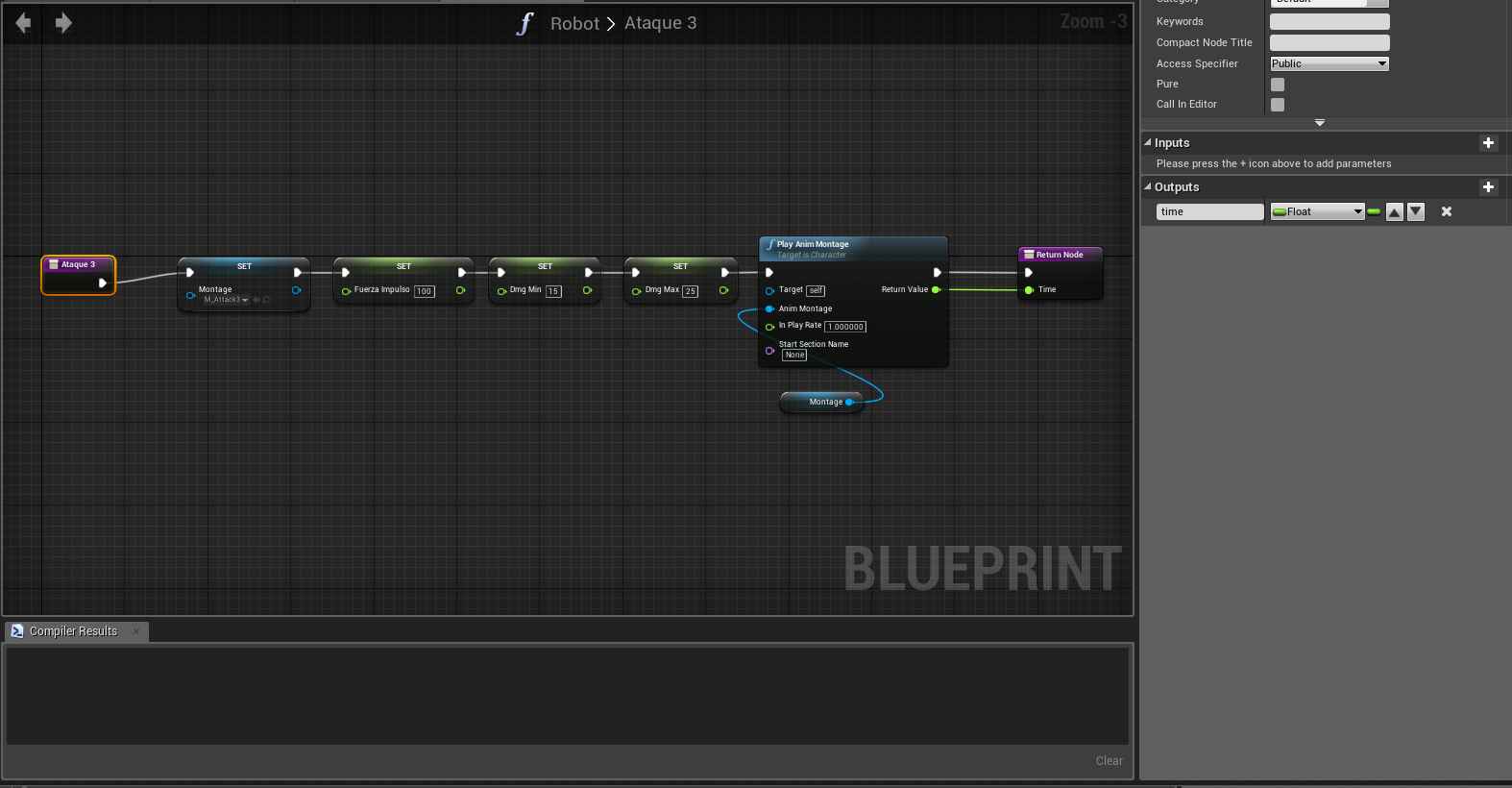


Figura : Ejemplo de función internamente

En la figura 19 podemos ver un ejemplo de función. Las funciones en Unreal se suelen usar para gestionar cálculos internos del propio BP o funciones que ayudan a los custom event (figura 17) para realizar acciones específicas. Al igual que los custom event, las funciones pueden recibir cualquier tipo de variable, y también, pueden retornar variables para su posterior uso que se distinguen por el color del pin que tienen.

Estos son los elementos más importantes de un BP, todos los elementos anteriores comparten en común un output de color blanco que indica el flujo de la programación que se conecta a otro nodo que lo recibe ya que, ese pin especial puede ser tanto de entrada como de salida. Algunos BP tiene sólo de salida o de entrada o tienen ambos. De esta manera visual se puede ver como va a ir el flujo del juego en ese determinado BP.

Una vez hemos descrito que es un BP, que elementos tiene y como funciona, pasamos a ver como es el C++ que ofrece Unreal, aunque en este proyecto no se ha hecho nada con C++ debido a la falta de tiempo.

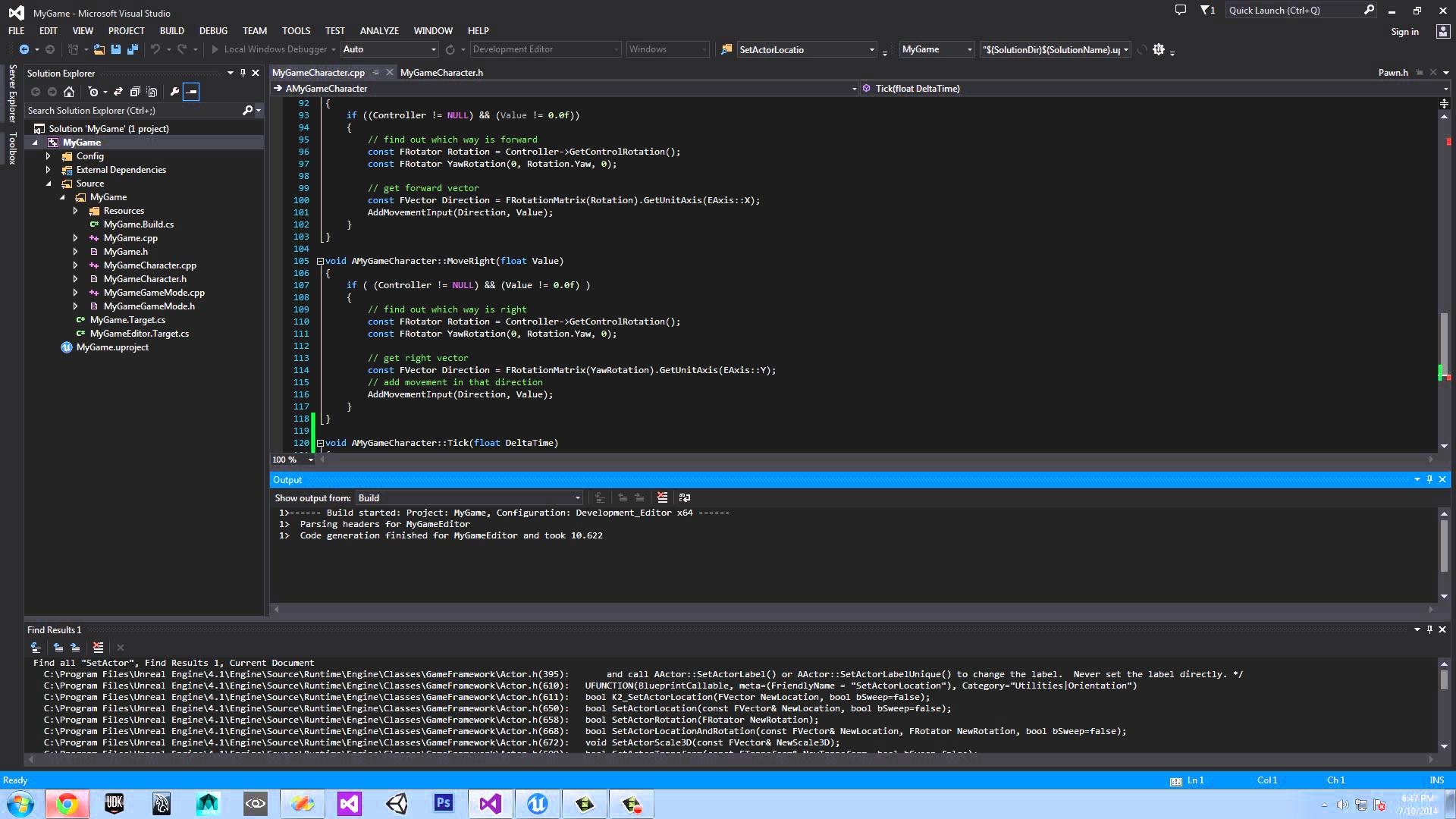


Figura : Ejemplo de programación en C++

En la figura 20 podemos ver cómo está estructurado en la clase de jugador sus métodos de movimiento. Al igual que los BP, en C++ puedes crearte cualquier tipo de variable, ya sea primitiva o de objetos propios de Unreal o tuyos, como también crearte funciones que hagan una serie de cálculos o ejecuten instrucciones para cualquier elemento del juego y retornen datos o estructuras que se usaran en otras partes del juego.

Estas son las dos maneras distintas de programar en Unreal. Se ha visto más en profundidad los BP porque son la estructura principal del proyecto y se han trabajado más con ellos que con C++. En los siguientes apartados se irá viendo, por partes, cómo se ha estructurado el juego en distintas categorías viendo lo que se ha usado del motor como lo que se ha implementado.

### 6.7.1 Arquitectura del juego general

Diagrama de bloques general de los elementos del juego

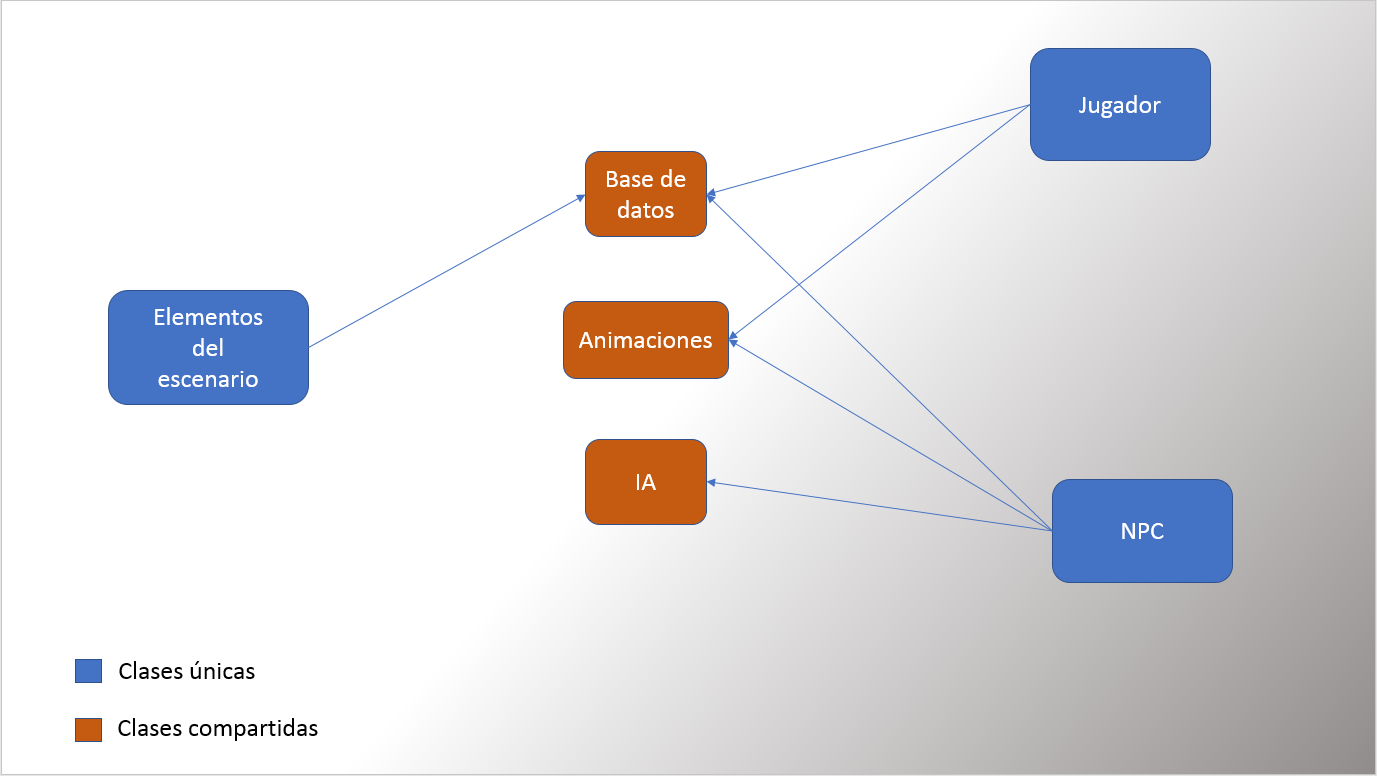


Figura : Diagrama de bloques del juego

En la figura 21 podemos observar un diagrama básico de como está formado el juego. Podemos diferenciar dos tipos de bloques que comparten información básica que luego se usan en el juego, más adelante se irá profundizando en los distintos elementos que forman el juego con su diagrama explicativo.

### 6.7.2 Jugador

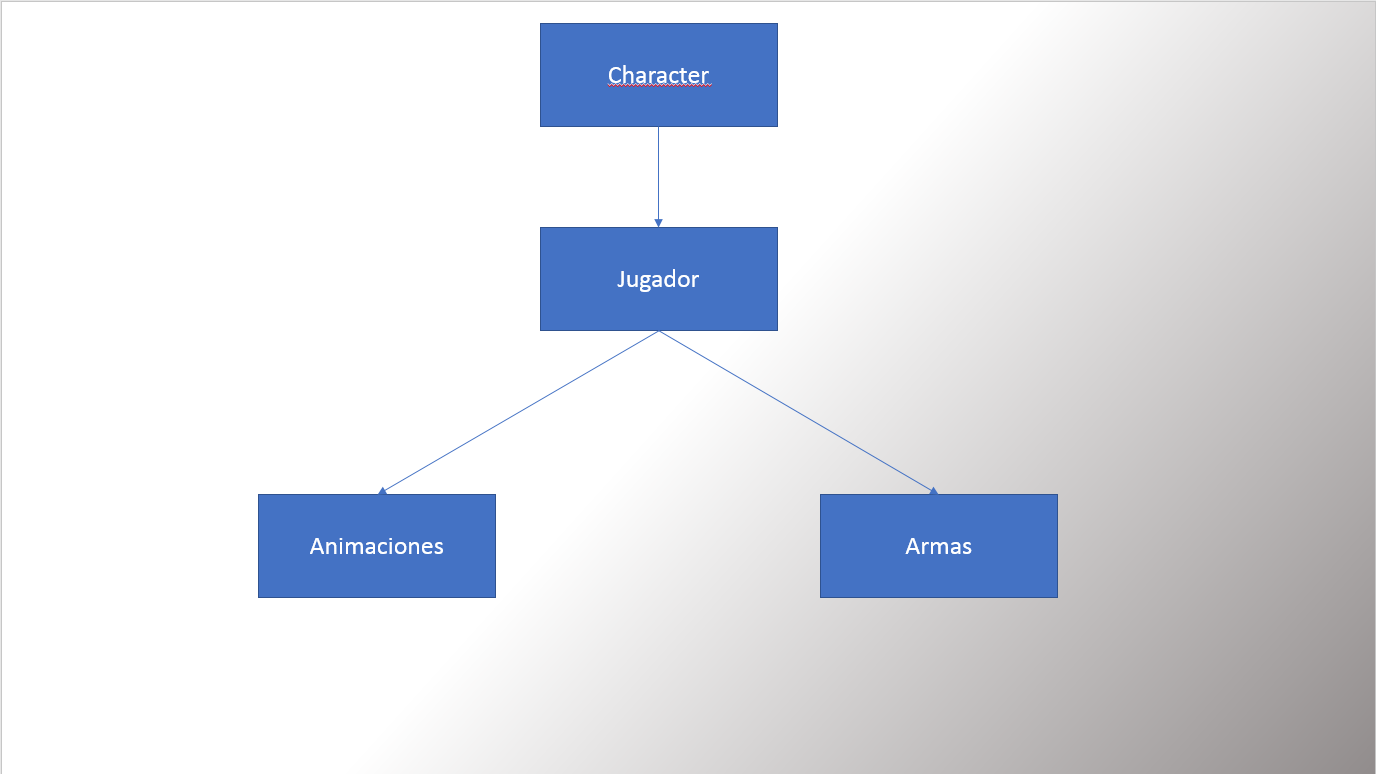


Figura : Diagrama de clases del jugador

En este apartado vamos a centrarnos sobre cómo se ha implementado el jugador y como se ha estructurado su jerarquía de herencia adaptándolo al motor. Se irá hablando en orden de como se representa la figura 22, de arriba abajo y de izquierda derecha.

Al ser nuestro jugador un personaje que se va a mover mediante velocidades y a interactuar con los distintos elementos del escenario, ya sean paredes, enemigos, objetos, etc. Se ha hecho que herede del BP prediseñado por el motor que provee de distintos elementos para movernos por el mapa como una cámara, un control donde podemos especificar la velocidad de nuestro personaje cuando anda, salta o se agacha; y así cómo, un elemento para gestionar las colisiones con el entorno del nivel. Este BP, también los da la oportunidad de añadirle que maya vamos a mover y cómo se va a gestionar sus animaciones, se hablarán de cómo van las animaciones y su gestión más adelante.

Una vez visto como se gestiona el padre de nuestro jugador, vamos a ver cómo se ha organizado y estructurado el BP del jugador.

### 6.7.3 NPC



Figura : Diagrama de clases de los NPC

#### 6.7.3.1 IA

Unreal nos proporciona distintas de estas herramientas para facilitarnos el trabajo, aunque la gran mayoría de los elementos importantes de la IA se han trabajado con la base de estas herramientas usando distintos algoritmos. De los distintos elementos nombrados anteriormente Unreal nos proporciona herramientas para:

* Navmesh: Unreal tiene un elemento que calcula por nosotros las áreas navegables de nuestros escenarios, haciendo que la IA sepa por donde puede moverse calculando la ruta más corta. Este navmesh se puede modificar distintos elementos para modificar el comportamiento de la IA.

Figura : Ejemplo del navmesh

Esta imagen muestra cómo se comporta el navmesh de Unreal siendo el color azul una zona de más prioridad frente a la roja.

* Behavior Tree: Unreal nos proporciona una plantilla especifica donde podemos crear nuestro árbol de comportamiento para nuestra IA, dónde le podemos decir qué y cómo tiene que actuar, dependiendo de distintos parámetros.

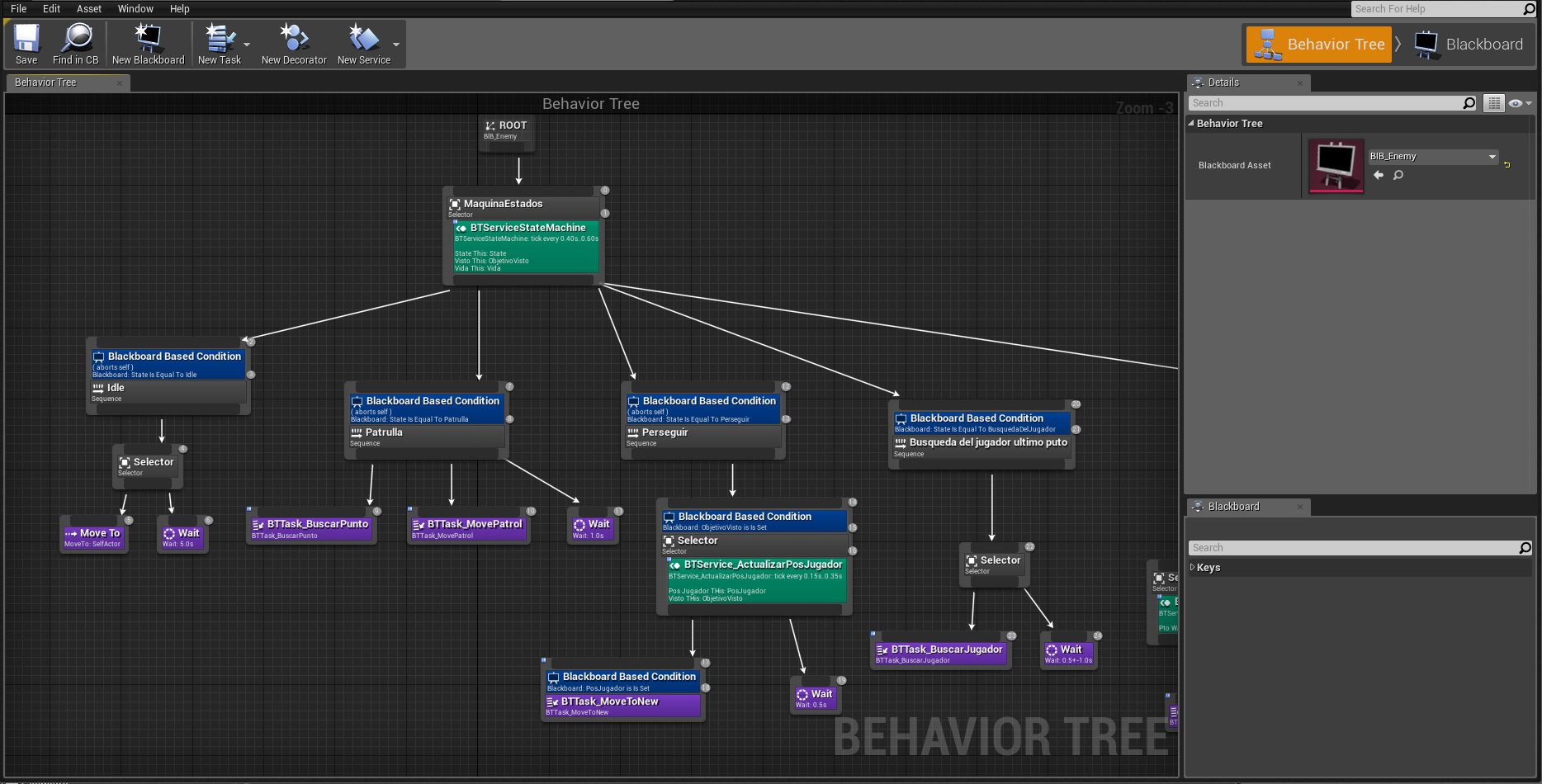


Figura : Ejemplo de behavior tree

En esta imagen podemos ver la toma de decisiones que hará nuestra IA. Unreal nos proporciona las herramientas para facilitarnos la creación de este árbol, pero, que la IA pueda ejecutar distintas acciones se ha tenido que hacer.

* Blackboard (sistema de memoria): Unreal nos facilita un elemento para guardar información que luego usará la IA en el árbol de comportamiento.

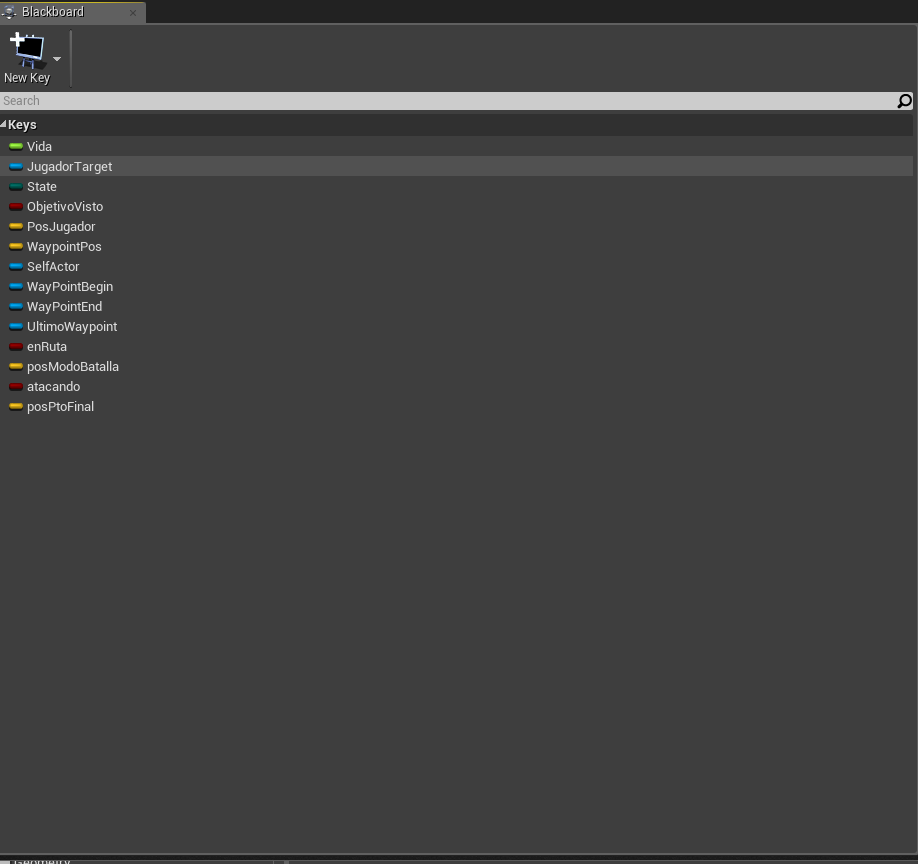


Figura : Ejemplo de blackboard

* AIController (Sistema de control de la IA): Este elemento nos permite conectar nuestra inteligencia (de los distintos elementos nombrados anteriormente) con nuestra lógica del enemigo que está en otro sitio. Este elemento nos permite dotar de sensores de vista, oído, olfato, etc… para hacer nuestra IA mucho más inteligente y poder usarlos en los elementos anteriores.



Figura : Ejemplo de sensor de visión

Con esta imagen se puede ver el sensor de vista aplicado a los enemigos. El área verde indica que estamos en su campo de visión.

Estos son los distintos elementos que ofrece Unreal para nuestra IA, todo lo demás ha sido añadido adaptando distintos algoritmos de IA que se han creado a la lógica de Unreal. Estos son:

* Pathfinding: Unreal no proporciona uno como tal, se ha creado uno usando como base los waypoints (puntos de rutas); que la propia IA elige su destino y, hasta que no llega a ese destino, no vuelve a seleccionar otro. Hay muchas maneras de hacer este algoritmo, pero la que se ha usado en el juego usa los nodos visitados. Una vez que la IA ha pasado por ese nodo, no vuelve a visitarlo hasta que acabe el viaje (llega a su destino)
* Sistema de ataque: El sistema de ataque de cada IA es distinta en cada juego, por eso unreal no ofrece un standard a la hora de implementar esa mecánica en su motor, sino que, te da facilidades para que se implemente uno propio. En este caso, es un sistema de combos que el enemigo te ataca cuando esté a cierta distancia. Estos ataques no continúan si el jugador sale de cierto rango.
* Comportamiento básico: Como se ha nombrado anteriormente, Unreal nos facilita una herramienta para crear nuestro propio árbol de comportamiento. La IA patrulla, usando el pathfinding y los waypoints como apoyo, persigue al jugador cuando le ve y si le pierde de vista va al último punto dónde le ha visto y, ataca cuando está a la distancia adecuada. Todos esos elementos se han implementado.

### 6.7.4 Elementos del escenario

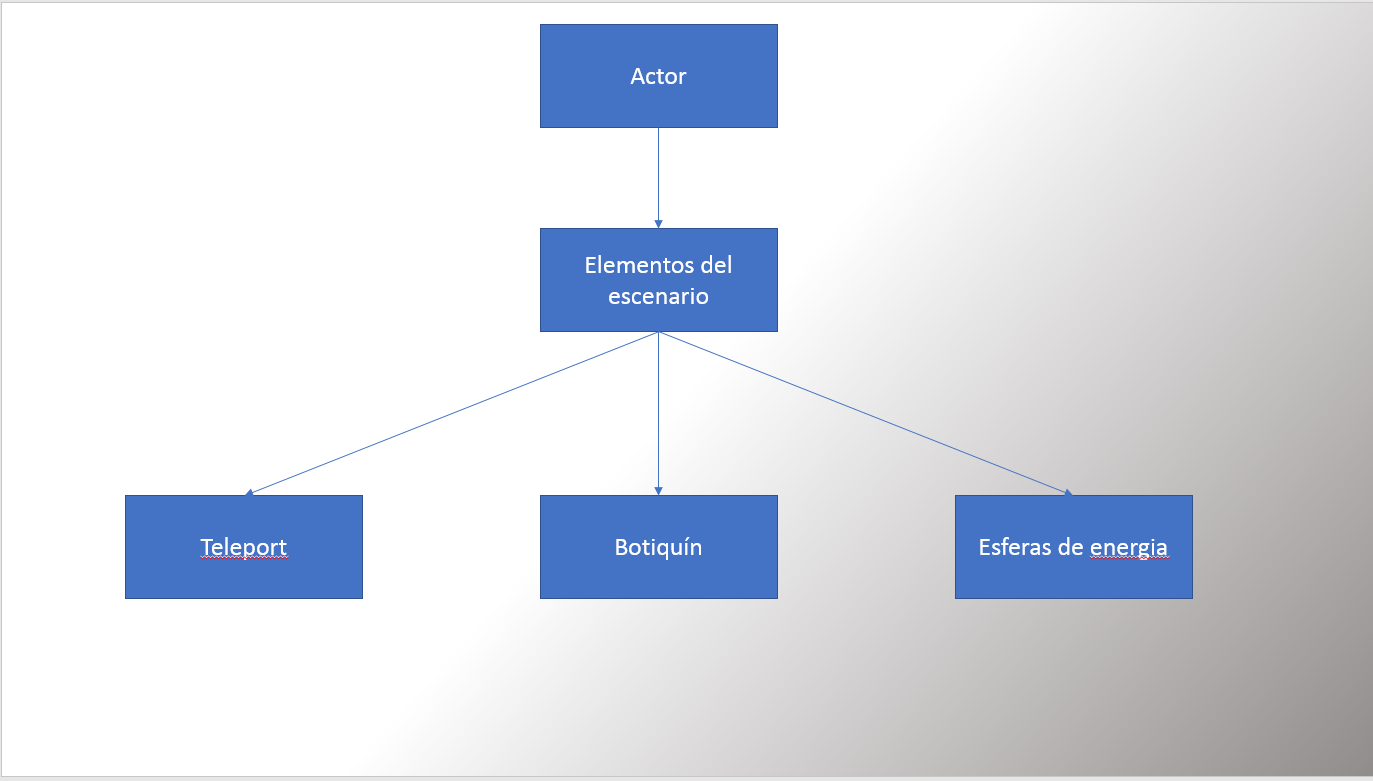


Figura : Diagrama de los elementos del escenario

### 6.7.5 Animaciones

Unreal tiene una herramienta para poder reutilizar animaciones en otros esqueletos distintos. Esta herramienta ha facilitado el uso de las animaciones enfocadas a un esqueleto poder usarlas en otro usando un intérprete que traduce qué hueso es el equivalente en el otro esqueleto dónde se quiere pasar la animación. Gracias a este elemento, se pueden reutilizar las animaciones para usarlas en el máximo número de elementos jugables del juego que se mueven por el escenario para no perder tiempo.

# Conclusiones

# Bibliografía y referencias

# Anexos