TRABAJO DE FIN DE GRADO

Ingeniería informática

VNiVERSiDAD D SALAMANCA

CAMPUS DE EXCELENCIA INTERNACIONAL

"*Anne Connery*", una aventura gráfica basada en inteligencias artificiales

---

Autor:

Pérez Martínez, Daniel

Tutores:

De Paz Santana, Juan Francisco

Villarrubia González, Gabriel

Junio de 2021

Autorización:

D. Juan Francisco De Paz Santana, profesor del Departamento de Informática y Automática y D. Gabriel Villarrubia González profesor del Departamento de Informática y Automática.

Hacen constar:

Que el trabajo titulado "'Anne Connery', una aventura gráfica basada en inteligencias artificiales", que se presenta, ha sido realizado por D. Daniel Pérez Martínez para la superación de la asignatura Proyecto de Fin de Carrera de la Titulación Ingeniería Informática de esta Universidad.

En Salamanca, a 02 de Julio de 2021

DE PAZ SANTANA JUAN FRANCISCO - 76125754D

Firmado digitalmente por DE PAZ SANTANA JUAN FRANCISCO - 76125754D
Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-76125754D, givenName=JUAN FRANCISCO, sn=DE PAZ SANTANA, cn=DE PAZ SANTANA JUAN FRANCISCO - 76125754D
Fecha: 2021.07.02 21:44:56 +02'00'

D. Juan Francisco De Paz Santana

VILLARRUBIA GONZALEZ GABRIEL - 71932845T

Firmado digitalmente por VILLARRUBIA GONZALEZ GABRIEL - 71932845T
Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-71932845T, givenName=GABRIEL, sn=VILLARRUBIA GONZALEZ, cn=VILLARRUBIA GONZALEZ GABRIEL - 71932845T
Fecha: 2021.07.02 23:02:58 +02'00'

D. Gabriel Villarrubia González

RESUMEN

El propósito del trabajo será el desarrollo completo/demostración jugable de un videojuego Píxel Art 2D en forma de aventura gráfica. En este se encarnará a Nathan Keighley, un abogado que tendrá que resolver varios casos. Durante la aventura el jugador deberá recolectar información, investigar con dicha información, realizar negociaciones y enfrentarse a un jurado popular en los juzgados, así como enfrentarse a decisiones morales que cambiarán el transcurso de la historia. Se desarrollará una inteligencia artificial completa para algunos de los personajes más relevantes de la historia (fiscales, jueces), de manera que estos puedan reaccionar a las decisiones del jugador de manera inteligente y a su vez aprender cómo actúa para, en futuras conversaciones, prever como va a actuar el jugador y ajustar la dificultad en función de sus conocimientos. El objetivo fundamental de estas IA será tratar de ganar negociaciones/juicios/disputas frente al abogado principal (el jugador).

PALABRAS CLAVE

Inteligencia artificial, Videojuego, Unreal Engine, Photoshop, Aventura gráfica, PixelArt, Blueprint.
Palabras clave

ABSTRACT

The purpose of the work will be the complete development / playable demonstration of a 2D Pixel Art video game in the form of a graphic adventure. In this he will play Nathan Keighley, a lawyer who will have to solve several cases. During the adventure the player must collect information, investigate with said information, conduct negotiations and face a popular jury in court, as well as face moral decisions that will change the course of history. A complete artificial intelligence will be developed for some of the most relevant characters in history (prosecutors, judges), so that they can react to the player's decisions in an intelligent way and in turn learn how he acts to, in future conversations, foresee how the player will act and adjust the difficulty based on his knowledge. The fundamental objective of these AIs will be to try to win negotiations / trials / disputes against the main lawyer (the player).

KEYWORDS

Artificial Intelligence, Video Game, Unreal Engine, Photoshop, Graphics Adventure, PixelArt, Blueprint.

CONTENIDO

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

1 INTRODUCCIÓN Y ANTECEDENTES

El objetivo del presente documento radica en recoger la memoria para el Trabajo de Fin de Grado titulado “Anne Connery, una aventura gráfica basada en inteligencias artificiales”, realizado por el alumno Daniel Pérez Martínez del Grado de Ingeniería Informática, desde junio de 2020 hasta junio de 2021.

El documento se ha dividido en los siguientes apartados:

- Objetivos del trabajo de fin de grado
- Conceptos teóricos
- Técnicas y herramientas
- Aspectos relevantes del desarrollo
- Conclusiones y líneas de trabajo futuras
- Bibliografía

1.1 EL PROYECTO

El proyecto al cual se refiere este documento consiste en la creación de una demo jugable de un videojuego en forma de aventura grafica con un diseño de tipo PixelArt. Para la realización del proyecto se usará el motor de videojuegos Unreal Engine 4. En la historia del videojuego se encarnará a un abogado que deberá resolver casos, encontrar evidencias, testigos y negociar los mejores tratos para sus clientes. El videojuego se basará principalmente en dos pilares fundamentales, el desarrollo de un sistema de diálogos completo que permita al jugador elegir opciones y guiar la historia a con sus elecciones, el segundo pilar es la creación de una inteligencia artificial que actúe como “enemigo” del jugador, esta deberá ser capaz de negociar propuestas del usuario reaccionando a las acciones del jugador y aprendiendo como actúa para adaptarse a sus acciones.

1.2 HISTORIA DE LOS VIDEOJUEGOS

[1] La historia de los videojuegos tiene su origen en la década de 1950 cuando, tras el fin de la Segunda Guerra Mundial, las potencias vencedoras de la guerra construyeron los primeros ordenadores programables. Los primeros intentos por implementar programas de carácter lúdico no tardaron en aparecer, y se fueron repitiendo durante las siguientes décadas. Los primeros videojuegos modernos aparecieron en la década de los 60, y desde entonces el mundo de los videojuegos no ha dejado de crecer y desarrollarse con el único límite que le ha impuesto la creatividad de los desarrolladores y la evolución tecnológica. En los últimos años, se asiste a una era de progreso tecnológico dominada por una industria que promueve un modelo de consumo rápido donde las nuevas superproducciones quedan obsoletas en pocos meses, pero donde a la

vez un grupo de personas e instituciones -conscientes del papel que los programas pioneros, las compañías que definieron el mercado y los grandes visionarios tuvieron en el desarrollo de dicha industria- han iniciado el estudio formal de la historia de los videojuegos.

El más inmediato reflejo de la popularidad que ha alcanzado el mundo de los videojuegos en las sociedades contemporáneas lo constituye una industria que genera unos beneficios multimillonarios que se incrementan año tras año. Al igual que ocurriera con el cine y la televisión, el videojuego ha logrado alcanzar en apenas medio siglo de historia el estatus de medio artístico, y semejante logro no ha tenido lugar sin una transformación y evolución constante del concepto mismo de videojuego y de su aceptación. Nacido como un experimento en el ámbito académico, logró establecerse como un producto de consumo de masas en tan solo diez años, ejerciendo un formidable impacto en las nuevas generaciones que veían los videojuegos con un novedoso medio audiovisual que les permitiría protagonizar en adelante sus propias historias.

1.3 AVENTURAS GRÁFICAS Y PÍXEL ART

1.3.1 Aventuras graficas

[2] La aventura gráfica es el género del mundo de los videojuegos interactivos cuyo antecedente más inmediato son las aventuras conversacionales de los años 1980 y cuya edad puede establecerse en la primera mitad de la década de 1990. La dinámica de este tipo de experiencia consiste en ir avanzando por el mismo a través de la resolución de diversos rompecabezas, planteados como situaciones que se suceden en la historia, interactuando con personajes y objetos a través de un menú de acciones o interfaz similar, utilizando un cursor para mover al personaje y realizar las distintas acciones. El concepto clásico de aventura gráfica incluía la visión de los personajes en tercera persona la mayor parte del juego. Con la aparición del ordenador personal de 8 bits en la década de 1980 se instauraba en los hogares un nuevo abanico de entretenimiento y ocio. Entre los nuevos juegos que salían para dichos ordenadores aparecieron unos que distaban mucho de la acción pura característica de los comecocos y las plataformas, las aventuras conversacionales.

[2] En dichos juegos la acción se desarrollaba en primera persona y bajo la pantalla gráfica había un cuadro de texto, utilizado para moverse, usar objetos e interactuar con el entorno y los personajes del videojuego mediante sencillas frases del tipo "hablar con el anciano", "usar fuego" o indicando puntos cardinales para ir de un sitio a otro.

1.3.2 Píxel art

[3] El píxel art es una forma de arte digital, creada digitalmente mediante el uso de programas de edición de gráficos rasterizados, donde las imágenes son editadas al nivel del píxel. Las imágenes de la mayor parte de los antiguos videojuegos para PC, videoconsolas y muchos juegos para teléfonos móviles son consideradas obras de *píxel art*. Posee similitudes con el puntillismo, difiriendo principalmente en las herramientas para la creación de las imágenes: computadoras y programas en lugar de pinceles y lienzos. El arte de píxel debe su origen principalmente a los videojuegos, sobre todo a juegos arcade clásicos como por ejemplo *Space Invaders* (1978) o *Pac-Man* (1980) y consolas de 8 bits.

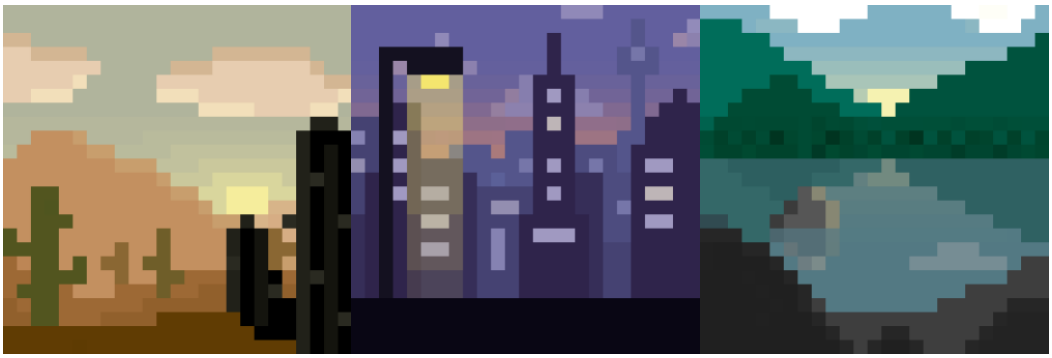


Ilustración 1: Ejemplos PíxelArt

1.4 INTELIGENCIA ARTIFICIAL EN VIDEOJUEGOS

[4] Inteligencia artificial, abreviado IA, en un videojuego, se refiere a las técnicas utilizadas en computadoras y videojuegos para producir la ilusión de inteligencia en el comportamiento de los personajes no jugadores (PNJ). Es un agente electrónico que puede pensar, evaluar y actuar en ciertos principios de la optimización y la coherencia para cumplir con una meta o propósito. Como la IA se centra en el aspecto de la inteligencia y una buena jugabilidad, su enfoque es muy diferente a la de la IA tradicional, soluciones alternativas y trucos son aceptables y, en muchos casos, las habilidades de las IA bajan el tono para dar a los jugadores humanos un sentido de justicia. Esto, por ejemplo, sucede en los videojuegos de acción en primera persona, donde la puntería perfecta estaría más allá de la habilidad humana.

[4] La IA se utilizan en una amplia variedad de campos dentro de un juego. La más evidente es en el control de los PNJ en el juego. La búsqueda de ruta es otro de uso común para la IA, ampliamente visto en los juegos de estrategia en tiempo real. Buscando camino es el método para determinar cómo obtener un PNJ de un punto en un mapa a otro, teniendo en cuenta el terreno, los obstáculos y, posiblemente, "niebla de guerra". Más allá de búsqueda de caminos, la navegación es un subcampo de la IA del juego que se centra en dar a los PNJ la capacidad de navegar en su entorno, la búsqueda de un camino hacia un objetivo, evitando colisiones con otras entidades o colaborar con ellos. La IA también está involucrada con el equilibrio de la dificultad del juego, que consiste en el ajuste de la dificultad de un juego de videojuego en tiempo real basado en la habilidad del jugador.

[4] El concepto de IA emergente ha sido recientemente explorado en juegos como *Creatures*, *Black & White* y *Nintendogs*. Las "mascotas" en estos juegos son capaces de "aprender" de las medidas adoptadas por el jugador y su comportamiento se modifica en consecuencia. En el caso de este proyecto, será el fiscal, el juez o el jurado el que se adaptará a las acciones del jugador y modificará su comportamiento en consecuencia.

2 OBJETIVOS

En este apartado se tiene como objetivo definir con claridad los objetivos del proyecto y definir las prioridades para su desarrollo. A continuación, se deben determinar los requisitos y definir las necesidades del futuro cliente/jugador para conocer con exactitud qué se tiene que desarrollar.

La especificación de requisitos se utiliza como medio de comunicación entre las distintas partes del proyecto (cliente, usuarios, ingenieros y desarrolladores). En él, se recogen las necesidades de los clientes y usuarios, las soluciones aportadas por el equipo de desarrollo para satisfacer las necesidades, la especificación formal de los actores que intervendrán en el sistema, de los casos de uso que lo componen y de la interacción entre los actores con el sistema.

***NOTA: para más información sobre la especificación de objetivos ver el Anexo II – Especificación de requisitos*.**

2.1 OBJETIVOS FUNCIONALES

Se describirán aquellos objetivos que derivan directamente de las necesidades de los usuarios del sistema, además se detalla cómo podrían satisfacerse estas necesidades.

2.1.1 Gestión de diálogos y selección de opciones

El sistema debe ser capaz de gestionar los diálogos de los distintos personajes, poniendo a disposición del usuario una interfaz que el permita leer fácilmente, diferencia cada uno de los integrantes de la conversación. También ha de permitir participar de la conversación mediante un sistema de selección de opciones que permita variar la conversación.

2.1.2 Gestión de casos y documentos independientes

El sistema deberá gestionar los casos que se hayan ido desbloqueando a lo largo de la historia, así como los documentos independientes a los casos para que estos sean accesibles en todo momento por el usuario.

2.1.3 Gestión de documentos por caso

Cada caso ha de tener asignado una serie de documentos, estos han de ser gestionados de forma que se encuentren disponibles cuando se esté investigando ese caso en concreto.

2.1.4 Gestión de guardado de datos

El sistema debe de gestionar los datos de forma que cuando se cierre la aplicación o se llegue a ciertos puntos de la historia se guarden los datos para poder continuar donde se dejó.

2.1.5 Gestión de cargado de datos

El sistema de gestionar la manera de cargar los datos guardados (si existen) cada vez que se inicie la aplicación.

2.1.6 Gestión de paso de niveles

El sistema ha de gestionar el paso de un nivel a otro con las funciones que le acometen en ese momento, paso de variables, instancias, etc. Así como la realización de guardados automáticos en cada inicio de nivel.

2.1.7 Gestión de negociaciones

El sistema ha de ser capaza de responder a las propuestas introducidas por el usuario de forma "inteligente", sabiendo reaccionar en función de varios parámetros iniciales, así como una serie de parámetros variables introducidos por el usuario. Durante la negociación será necesario poner a disposición del usuario la opción de lanzar propuestas, mostrar pruebas encontradas o charlar con fin de convencer a la inteligencia artificial.

2.1.8 Gestión de mapa y movimiento

El sistema debe gestionar un sistema que permita al usuario moverse a través de los distintos escenarios con el fin de desbloquear nuevas pistas, pruebas o nuevas conversaciones que faciliten la resolución del caso favorablemente.

2.1.9 Gestión de búsqueda de información en BD

Se ha de implementar una base de datos ficticia que sirva para almacenar información en la que se debe indagar en busca de pistas e indicaciones para continuar la aventura.

2.1.10 Gestión de desbloqueo de pistas y conversaciones

El sistema debe gestionar la manera en la que se van a desbloquear las pistas a lo largo de la historia, de manera que cada vez que una de estas se desbloque se pueda posteriormente usar en las negociaciones, así como ser visible para que el usuario la pueda consultar en todo momento.

2.2 OBJETIVOS NO FUNCIONALES

En este apartado se especifican los requisitos que permitirán establecer la funcionalidad del proyecto, es decir, las condiciones y restricciones que tiene que cumplir este.

- Utilización del motor grafico Unreal Engine
- Utilización del motor de IA incluido en UE4
- Mantenibilidad
- Usabilidad

2.3 OBJETIVOS PERSONALES

Se propone el desarrollo de una aplicación de ocio, abarcando todos los campos del desarrollo por completo. Se abarcará desde el diseño del videojuego (mediante GDD, Game Desing Documents), el arte visual, la composición musical, la escritura de guion, la gestión, la organización y la programación del proyecto. Se creará una aplicación destinada a jugadores que estén interesados por las aventuras gráficas y el arte píxel art. El objetivo a largo plazo es la finalización completa del videojuego y la puesta a disposición del público en plataformas digitales.

3 TÉCNICAS Y HERRAMIENTAS

3.1 UNREAL ENGINE 4

Unreal Engine es un motor de juego creado por la compañía Epic Games, mostrado inicialmente en el shooter en primera persona *Unreal* en 1998. Aunque se desarrolló principalmente para los shooters en primera persona, se ha utilizado con éxito en una variedad de otros géneros. Con su código escrito en C++, el Unreal Engine presenta un alto grado de portabilidad y es una herramienta utilizada actualmente por muchos desarrolladores de juegos.

La versión más estable es Unreal Engine 4, el cual fue lanzado en 2014 bajo un modelo de subscripción. Desde 2015, puede descargarse gratuitamente, con su código fuente disponible en GitHub.

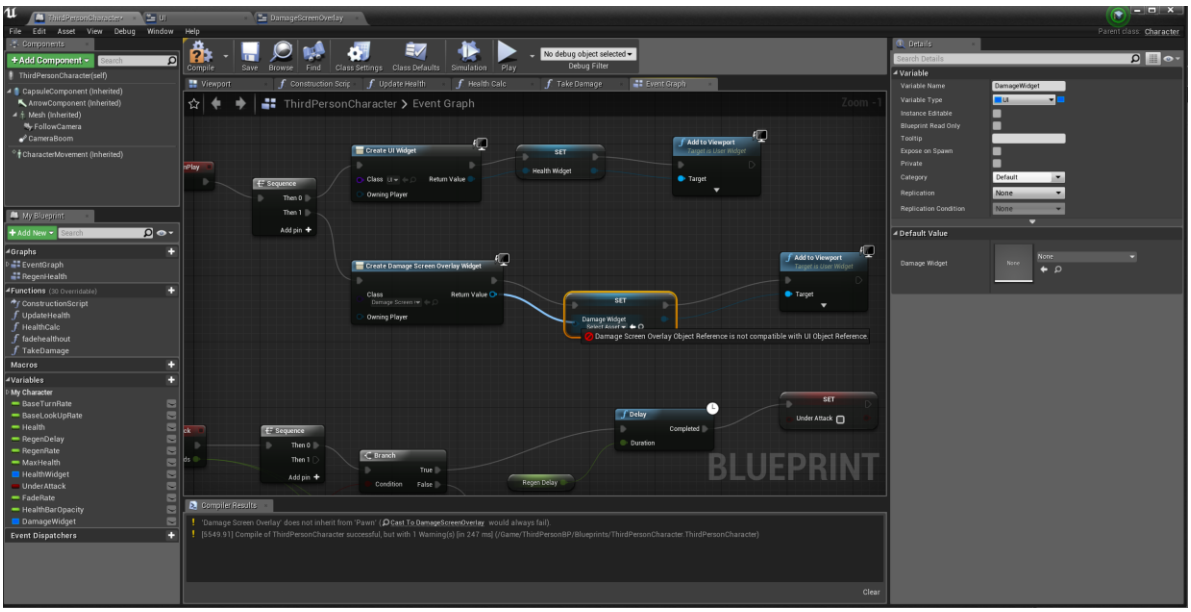


Ilustración 2: Unreal Engine 4

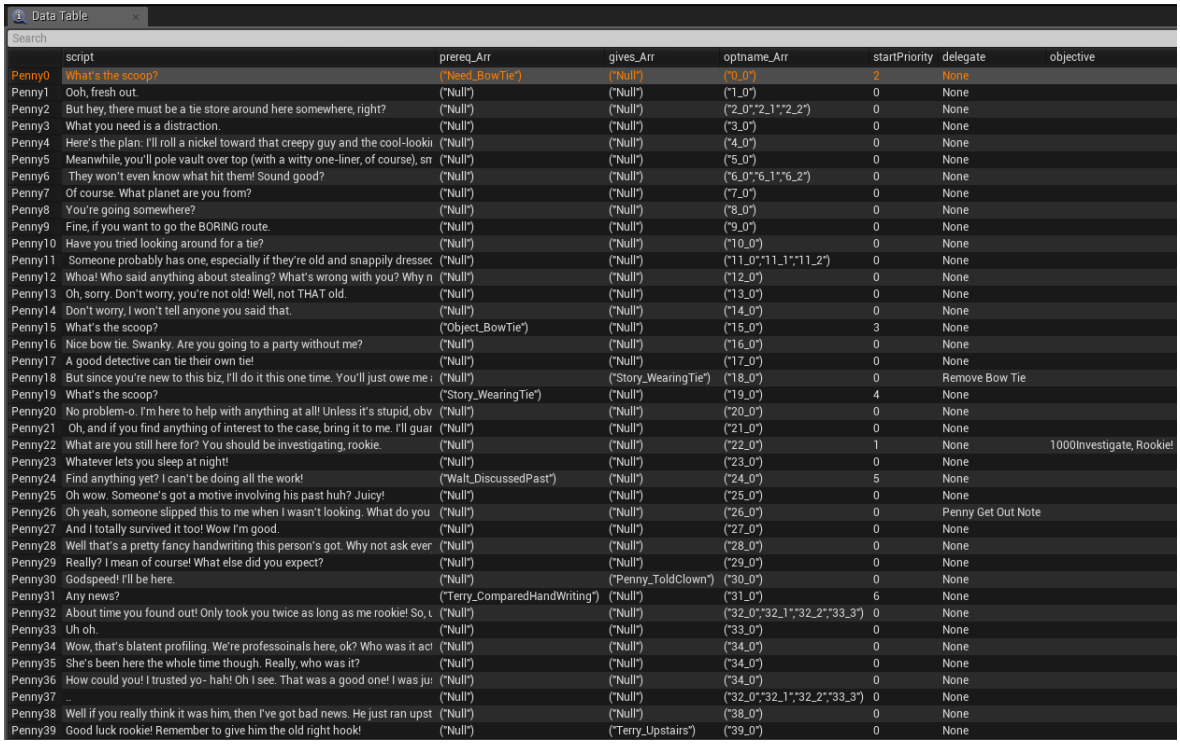
3.1.1 Blueprints en Unreal Engine 4

El sistema de Blueprint Visual Scripting en Unreal Engine es un completo sistema de scripting de juego basado en el concepto de utilizar una interfaz a base de nodos para crear elementos de juego desde el Editor de Unreal. Al igual que muchos lenguajes de scripting comunes, se utiliza para definir clases u objetos orientados a objetos (OO) en el motor.

Este sistema es extremadamente flexible y potente, ya que ofrece a los diseñadores la posibilidad de utilizar prácticamente toda la gama de conceptos y herramientas que generalmente sólo están disponibles para los programadores. Además, los Blueprint disponen de una implementación base de C++ que permite a los programadores crear sistemas que pueden ser ampliados.

3.1.2 DataTable Blueprints

Las DataTable Blueprints de UE4, son un tipo especial de Blueprint capaz de almacenar y tratar datos en forma de tabla, estas serán usadas para la creación de las bases de datos ficticias. Este tipo de clase tiene asociados a ella una serie de métodos en forma de nodos que facilitan en gran medida el uso y tratado de los datos contenidos en la tabla.



Data Table

Search

	script	prereq_Arr	gives_Arr	optname_Arr	startPriority	delegate	objective
Penny0	What's the scoop?	("Need_BowTie")	("Null")	("0_0")	2	None	
Penny1	Ooh, fresh out.	("Null")	("Null")	("1_0")	0	None	
Penny2	But hey, there must be a tie store around here somewhere, right?	("Null")	("Null")	("2_0","2_1","2_2")	0	None	
Penny3	What you need is a distraction.	("Null")	("Null")	("3_0")	0	None	
Penny4	Here's the plan: I'll roll a nickel toward that creepy guy and the cool-lookii	("Null")	("Null")	("4_0")	0	None	
Penny5	Meanwhile, you'll pole vault over top (with a witty one-liner, of course), sn	("Null")	("Null")	("5_0")	0	None	
Penny6	They won't even know what hit them! Sound good?	("Null")	("Null")	("6_0","6_1","6_2")	0	None	
Penny7	Of course. What planet are you from?	("Null")	("Null")	("7_0")	0	None	
Penny8	You're going somewhere?	("Null")	("Null")	("8_0")	0	None	
Penny9	Fine, if you want to go the BORING route.	("Null")	("Null")	("9_0")	0	None	
Penny10	Have you tried looking around for a tie?	("Null")	("Null")	("10_0")	0	None	
Penny11	Someone probably has one, especially if they're old and snappily dressec	("Null")	("Null")	("11_0","11_1","11_2")	0	None	
Penny12	Whoa! Who said anything about stealing? What's wrong with you? Why n	("Null")	("Null")	("12_0")	0	None	
Penny13	Oh, sorry. Don't worry, you're not old! Well, not THAT old.	("Null")	("Null")	("13_0")	0	None	
Penny14	Don't worry, I won't tell anyone you said that.	("Null")	("Null")	("14_0")	0	None	
Penny15	What's the scoop?	("Object_BowTie")	("Null")	("15_0")	3	None	
Penny16	Nice bow tie. Swanky. Are you going to a party without me?	("Null")	("Null")	("16_0")	0	None	
Penny17	A good detective can tie their own tie!	("Null")	("Null")	("17_0")	0	None	
Penny18	But since you're new to this biz, I'll do it this one time. You'll just owe me	("Null")	("Story_WearingTie")	("18_0")	0	Remove Bow Tie	
Penny19	What's the scoop?	("Story_WearingTie")	("Null")	("19_0")	4	None	
Penny20	No problem-o. I'm here to help with anything at all! Unless it's stupid, obv	("Null")	("Null")	("20_0")	0	None	
Penny21	Oh, and if you find anything of interest to the case, bring it to me. I'll guar	("Null")	("Null")	("21_0")	0	None	
Penny22	What are you still here for? You should be investigating, rookie.	("Null")	("Null")	("22_0")	1	None	1000Investigate, Rookie!
Penny23	Whatever lets you sleep at night!	("Null")	("Null")	("23_0")	0	None	
Penny24	Find anything yet? I can't be doing all the work!	("Walt_DiscussedPast")	("Null")	("24_0")	5	None	
Penny25	Oh wow. Someone's got a motive involving his past huh? Juicy!	("Null")	("Null")	("25_0")	0	None	
Penny26	Oh yeah, someone slipped this to me when I wasn't looking. What do you	("Null")	("Null")	("26_0")	0	Penny Get Out Note	
Penny27	And I totally survived it too! Wow I'm good.	("Null")	("Null")	("27_0")	0	None	
Penny28	Well that's a pretty fancy handwriting this person's got. Why not ask ever	("Null")	("Null")	("28_0")	0	None	
Penny29	Really? I mean of course! What else did you expect?	("Null")	("Null")	("29_0")	0	None	
Penny30	Godspeed! I'll be here.	("Null")	("Penny_ToldClown")	("30_0")	0	None	
Penny31	Any news?	("Terry_ComparedHandWriting")	("Null")	("31_0")	6	None	
Penny32	About time you found out! Only took you twice as long as me rookie! So, t	("Null")	("Null")	("32_0","32_1","32_2","33_3")	0	None	
Penny33	Uh oh.	("Null")	("Null")	("33_0")	0	None	
Penny34	Wow, that's blatent profiling. We're professoinals here, ok? Who was it act	("Null")	("Null")	("34_0")	0	None	
Penny35	She's been here the whole time though. Really, who was it?	("Null")	("Null")	("34_0")	0	None	
Penny36	How could you! I trusted yo- hah! Oh I see. That was a good one! I was ju	("Null")	("Null")	("34_0")	0	None	
Penny37	..	("Null")	("Null")	("32_0","32_1","32_2","33_3")	0	None	
Penny38	Well if you really think it was him, then I've got bad news. He just ran upst	("Null")	("Null")	("38_0")	0	None	
Penny39	Good luck rookie! Remember to give him the old right hook!	("Null")	("Terry_Upstairs")	("39_0")	0	None	

Ilustración 3: DataTable de UE4

3.1.2.1 Estructura JSON

JSON (JavaScript Object Notation) es un formato de texto sencillo para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de JavaScript, aunque, debido a su amplia adopción como alternativa a XML, se considera un formato independiente del lenguaje. Este es el formato que se usará para los datos de las bases de datos ficticias. Para la generación de los datos se hace uso de la web https://www.mockaroo.com/ , una web que permite la generación de una estructura de datos con los campos deseados y con el formato adecuado para el caso concreto, en mi caso JSON.

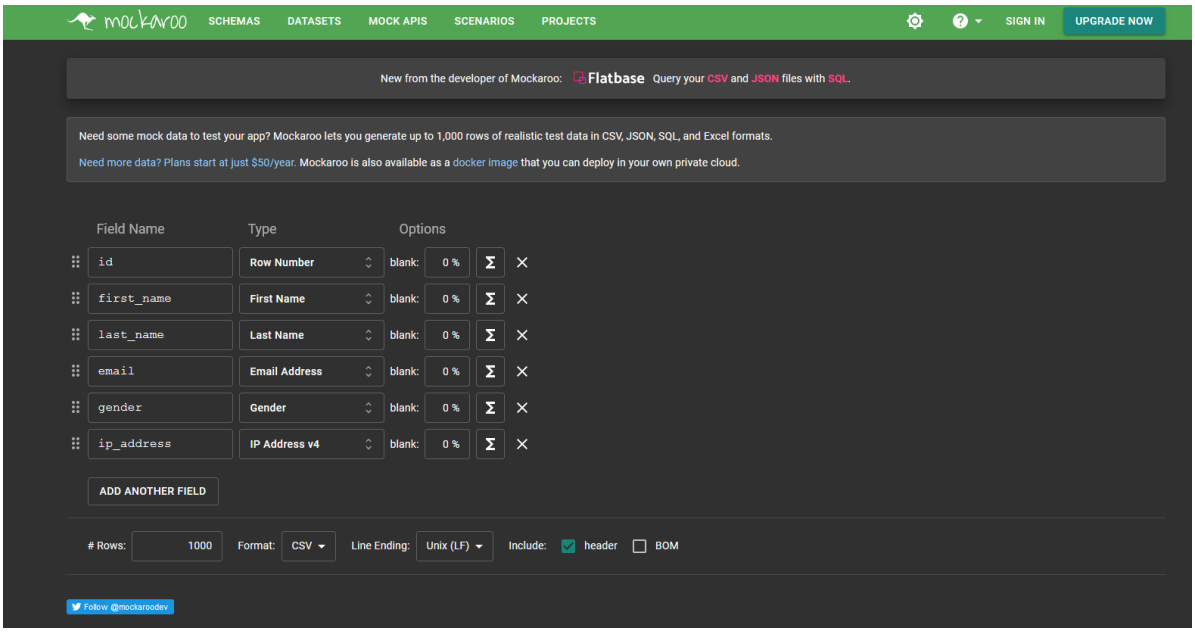


Ilustración 4: Mockaroo.com

3.1.3 Inteligencia artificial en UE4

La creación de Inteligencia Artificial (IA) para los personajes u otras entidades en sus proyectos en Unreal Engine 4 (UE4) se logra a través de múltiples sistemas que trabajan juntos. Desde un árbol de comportamiento (Behavior Trees) que se ramifica entre diferentes decisiones o acciones, pasando por la ejecución de una consulta para obtener información sobre el entorno a través del sistema de consulta del entorno (Environment Query System, EQS), hasta el uso del sistema de percepción de la IA para recuperar información sensorial como la vista, el sonido o la información sobre el daño; todos estos sistemas desempeñan un papel clave en la creación de una IA creíble en sus proyectos. Además, todas estas herramientas pueden depurarse con las herramientas de depuración de la IA, lo que permite saber qué está pensando o haciendo la IA en cada momento. Cuando se crea una IA en UE4 y se utiliza cada uno de estos sistemas, una buena forma de pensar en la construcción de la IA es que el proceso de toma de decisiones lo gestionan los árboles de comportamiento, los estímulos del entorno (como la información sensorial) se envían a los árboles de comportamiento desde el sistema de percepción de la IA, y las consultas sobre el propio entorno se gestionan a través de EQS.

3.1.3.1 Behavior Trees

Los assets de Árboles de Comportamiento en Unreal Engine 4 se pueden utilizar para crear inteligencia artificial para personajes no jugadores en tus proyectos. Mientras que el asset Árbol de Comportamiento se utiliza para ejecutar ramas que contienen lógica, para determinar qué ramas deben ser ejecutadas, el Árbol de Comportamiento se basa en otro asset llamado Blackboard que sirve como el "cerebro" de un Árbol de Comportamiento.

La Pizarra contiene varias Claves definidas por el usuario que contienen información utilizada por el Árbol de Comportamiento para tomar decisiones.

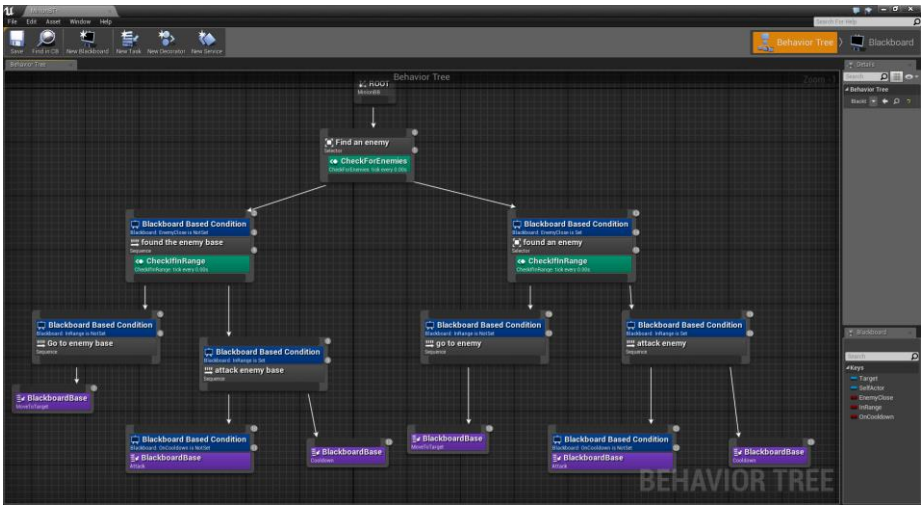


Ilustración 5: Árbol de comportamiento de UnrealEngine

3.2 PLANIFICACIÓN TEMPORAL

Dadas las características del proyecto, para la planificación del proyecto se va a utilizar Scrum.

En Scrum se realizan entregas parciales y regulares del producto final, priorizadas por el beneficio que aportan al receptor del proyecto. Por ello, Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales.

Scrum también se utiliza para resolver situaciones en que no se está entregando al cliente lo que necesita, cuando las entregas se alargan demasiado, los costes se disparan o la calidad no es aceptable, cuando se necesita capacidad de reacción ante la competencia, cuando la moral de los equipos es baja y la rotación alta, cuando es necesario identificar y solucionar ineficiencias sistemáticamente o cuando se quiere trabajar utilizando un proceso especializado en el desarrollo de producto.

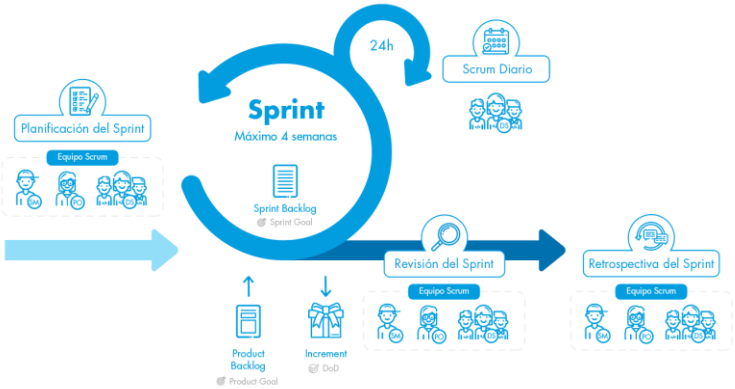


Ilustración 6: Scrum

3.3 DISEÑO DE INTERFACES Y ARTE DEL VIDEOJUEGO

Para la realización de los gráficos del videojuego se ha usado Photoshop junto con las herramientas de UI que proporciona Unreal Engine 4, estas son los WidgetBlueprints.

3.3.1 Adobe Photoshop

Para la parte de los gráficos se ha utilizado el programa de edición de imágenes de Adobe, Photoshop. Photoshop puede editar y componer imágenes rasterizadas y soporta varios modelos de colores: RGB, CMYK, CIELAB, colores sólidos y semitonos. Photoshop usa sus propios formatos de archivo PSD y PSB para soportar estas características.

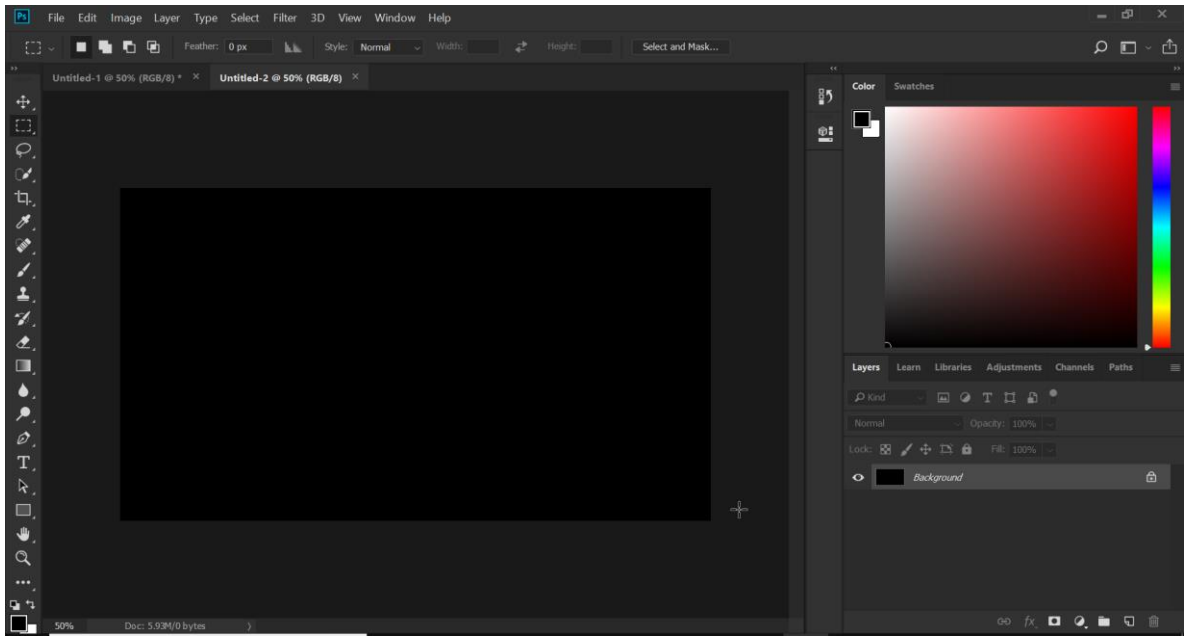


Ilustración 7: Photoshop

3.3.2 Widget Blueprints

Los WidgetBlueprints son un caso especial de los Blueprints dentro del motor, estos están compuestos de dos partes, la parte de código (Graph), que es similar a un Blueprint clásico y la parte de interfaz (Designer), que permite añadir todos los elementos visuales a la pantalla. Varios nodos permiten configurar la interacción entre ambas partes. Ejemplos de estos nodos que actúan de nexo son “OnClick”, “OnChange”, etc. que permiten asociar funcionalidad a los botones, sliders y demás elementos de la interfaz.

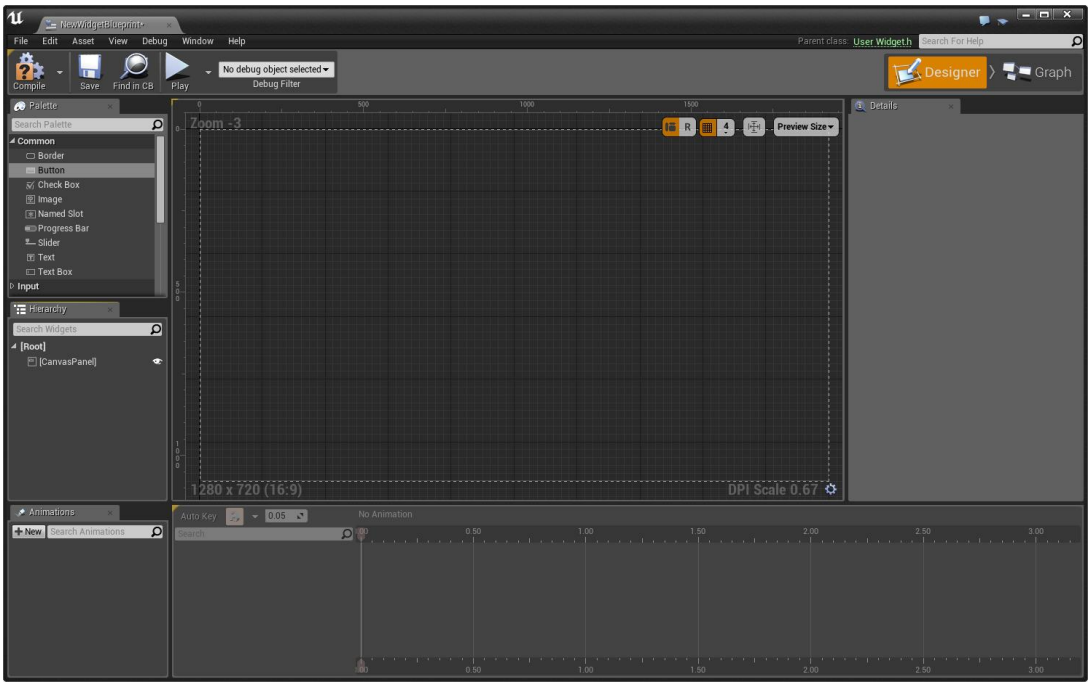


Ilustración 8: WidgetBlueprint

3.4 DIAGRAMAS Y DISEÑO

Para la generación de los diagramas de paquetes, secuencia, etc. visibles en los anexos I, II, III, así como en esta memoria se han usado la herramienta online que proporciona Visual Paradigm en su versión de navegador.

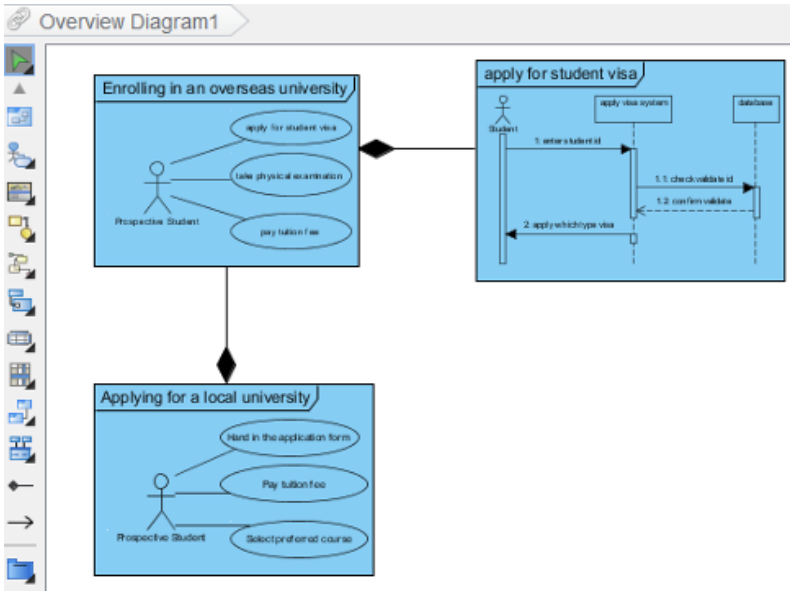


Ilustración 9: Visual Paradigm

Visual Paradigm es una herramienta UML CASE que admite UML 2, SysML y notación de modelado de procesos empresariales del grupo de gestión de objetos. Además del soporte de modelado, proporciona capacidades de generación de informes e ingeniería de código, incluida la generación de código.

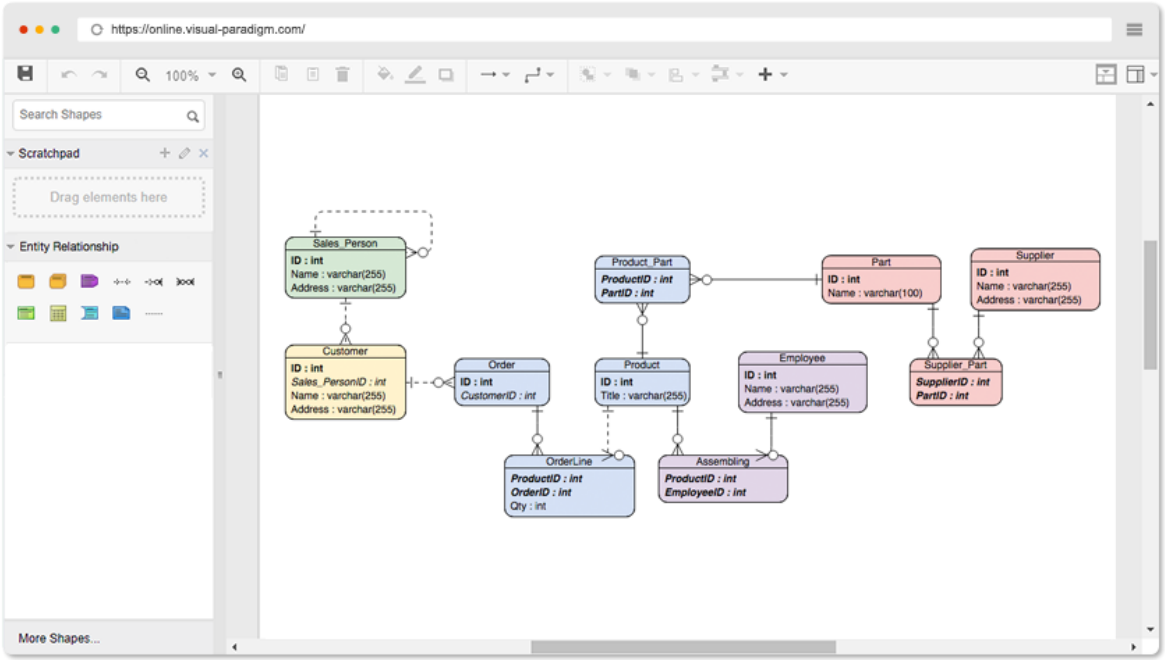


Ilustración 10: Visaul Paradigm Online

4 ASPECTOS RELEVANTES DEL DESARROLLO

En este apartado se expondrá un desarrollo resumido de cada uno de los niveles que han sido desarrollados. Así como los componentes comunes a todos los niveles.

4.1 ESTIMACIÓN DE ESFUERZO

Para la planificación temporal del proyecto se decidió usar SCRUM, y dado que este no impone ninguna técnica especifica de estimación se escogió la técnica de "T-Shirt Sizes", técnica que consiste en asignar tallas de camiseta en vez de utilizar números, para el desarrollo de esta parte se establecieron los sprints principales teniendo en cuenta el tiempo en el que se debía realizar el proyecto. Una vez se tenían claros los 6 sprints se empezaron a deliberar las tareas que se debía incluir en cada uno de estos. Y por último se asignó la estimación usando tallas de camisetas, siempre teniendo en cuenta la carga que cada una de las tareas supondría.

ID	Nombre	Estimación
1	Sprint 1 – Fase de iniciación	XL
1.1	Definir visón del proyecto	XS
1.2	Definir equipo de trabajo	XS
1.3	Desarrollo de Product Backlog	S
1.4	Definición de prioridades	XS
1.5	Entregable – Anexo I	XL
2	Sprint 2 - Fase de planificación	XL
2.1	Definición de objetivos	S
2.2	Identificación de casos de uso	XS
2.3	Identificación de funcionalidades	S
2.4	Investigación sobre IA en Unreal Engine 4	L
2.5	Entregable – Anexo II	XL
3	Sprint 3 – Fase de implementación	XL
3.1	Descomposición del dominio del problema	S
3.2	Módulos de persistencia y recogida de inputs	M
3.3	Diagrama de módulos	S
3.4	Modelo de persistencia de datos	M
3.5	Implementación de persistencia e inputs	L
3.6	Entregable – Anexo III	XL
4	Sprint 4 – Fase de implementación	XL

4.1	Modelo de diseño	M
4.2	Vista de arquitectura	S
4.3	Implementación de desbloqueo y análisis de datos	XL
4.4	Implementación de la capa de visualización	XL
4.5	Entregable – Anexo IV	XL
5	Sprint 5 – Fase de revisión	XL
5.1	Documentación técnica	S
5.2	Estructuración	S
5.3	Testeo por niveles	M
5.4	Entregable – Anexo V	XL
6	Sprint 6 – Fase de lanzamiento	XL
6.1	Documentación practica	XS
6.2	Pruebas del sistema completo con usuarios	L
6.3	Modificaciones de UI e IPO	M
6.4	Entrega de la versión final	XS
6.5	Retrospectiva y conclusiones	XS
6.6	Entregable Anexo VI	XL

Tabla 1: Estimación de esfuerzo

4.2 PLANIFICACIÓN TEMPORAL

Para la realización de la planificación temporal, la primera tarea a realizar es identificación todas las tareas que se van a ver involucradas en el proyecto, ya que estas definirán el tiempo y el éxito del proyecto.

NOTA: para más información sobre la planificación temporal ver el Anexo I – Planificación temporal.

Como ejemplo de la planificación, este es el listado de tareas que engloba todo un proyecto. Cualquier tarea que se deba hacer debe estar en el *product backlog* y con un tiempo estimado por el equipo de desarrollo y la importancia de dicha tarea. Posteriormente se añadirá el tiempo real utilizado para finalizar la tarea.

ID	Nombre	Tiempo estimado	Tiempo real	Importancia	Estado
1	Sprint 1 – Fase de iniciación	15	15	10	Finalizado
1.1	Definir visón del proyecto	2	2	10	Finalizado
1.2	Definir equipo de trabajo	1	1	10	Finalizado

1.3	Desarrollo de Product Backlog	6	6	9	Finalizado
1.4	Definición de prioridades	2	3	8	Finalizado
1.5	Entregable – Anexo I	15	15	10	Finalizado
2	Sprint 2 - Fase de planificación	15	15	10	Finalizado
2.1	Definición de objetivos	3	4	10	Finalizado
2.2	Identificación de casos de uso	2	2	10	Finalizado
2.3	Identificación de funcionalidades	4	4	10	Finalizado
2.4	Investigación sobre IA en Unreal Engine 4	9	8	8	Finalizado
2.5	Entregable – Anexo II	15	15	10	Finalizado
3	Sprint 3 – Fase de implementación	15	15	10	Finalizado
3.1	Descomposición del dominio del problema	3	3	9	Finalizado
3.2	Módulos de persistencia y recogida de inputs	6	6	10	Finalizado
3.3	Diagrama de módulos	3	3	10	Finalizado
3.4	Modelo de persistencia de datos	6	5	10	Finalizado
3.5	Implementación de persistencia e inputs	10	12	10	Finalizado
3.6	Entregable – Anexo III	15	15	10	Finalizado
4	Sprint 4 – Fase de implementación	15	15	10	Finalizado
4.1	Modelo de diseño	6	7	9	Finalizado
4.2	Vista de arquitectura	5	4	9	Finalizado
4.3	Implementación de desbloqueo y análisis de datos	15	15	10	Finalizado
4.4	Implementación de la capa de visualización	15	15	10	Finalizado
4.5	Entregable – Anexo IV	15	15	10	Finalizado
5	Sprint 5 – Fase de revisión	15	15	10	Finalizado
5.1	Documentación técnica	4	5	9	Finalizado
5.2	Estructuración	5	51	10	Finalizado
5.3	Testeo por niveles	8	9	10	Finalizado
5.4	Entregable – Anexo V	15	15	10	Finalizado
6	Sprint 6 – Fase de lanzamiento	15	15	10	Finalizado
6.1	Documentación practica	2	3	10	Finalizado
6.2	Pruebas del sistema completo con usuarios	10	10	10	Finalizado

6.3	Modificaciones de UI e IPO	5	6	8	Finalizado
6.4	Entrega de la versión final	2	2	10	Finalizado
6.5	Retrospectiva y conclusiones	2	2	9	Finalizado
6.6	Entregable Anexo VI	15	15	10	Finalizado

Tabla 2: Product Backlog

4.3 ESPECIFICACIÓN DE REQUISITOS

La fase de iniciación del proyecto tiene como principal objetivo definir los objetivos que se pretenden cumplir con claridad, así como definir las prioridades del desarrollo.

***NOTA: para más información sobre la especificación de requisitos ver el Anexo II – Especificación de requisitos*.**

Este anexo contiene información sobre los siguientes apartados:

- Participantes del proyecto
- Presentación del proyecto
- Objetivos del sistema
- Requisitos del sistema
- Actores
- Diagrama de paquetes generales
- Vista de interacción de casos de uso

En este apartado se hará un resumen de estos puntos para obtener una información más detallada visitar el anexo en concreto.

4.3.1 Presentación del proyecto

El propósito del trabajo será el desarrollo completo/demostración jugable de un videojuego Píxel Art 2D en forma de aventura gráfica. En este se encarnará a Nathan Keighley, un abogado que tendrá que resolver varios casos. Durante la aventura el jugador deberá recolectar información, investigar con dicha información, realizar negociaciones y enfrentarse a un jurado popular en los juzgados, así como enfrentarse a decisiones morales que cambiarán el transcurso de la historia. Se desarrollará una inteligencia artificial completa para algunos de los personajes más relevantes de la historia (fiscales, jueces), de manera que estos puedan reaccionar a las decisiones del jugador de manera inteligente y a su vez aprender cómo actúa para, en futuras conversaciones, prever como va a actuar el jugador y ajustar la dificultad en función de sus conocimientos. El objetivo fundamental de estas IA será tratar de ganar negociaciones/juicios/disputas frente al abogado principal (el jugador).

4.3.2 Objetivos del sistema

Una parte fundamental al inicio del proyecto es la definición de los objetivos del sistema, estos son finalidades, logros, misiones o visiones que se pretenden conseguir al finalizar el proyecto. Los objetivos determinan el funcionamiento del sistema, para lograrlos deben tenerse en cuenta tanto los elementos y relaciones. Para cada uno de los objetivos existe una tabla (Disponibles en el Anexo II – Especificación de requisitos). Para cada uno de los objetivos se establece un autor, una descripción, la estabilidad necesaria, su estado (validado o no) y un nivel de urgencia.

***NOTA: para más información sobre la especificación de requisitos ver el Anexo II – Especificación de requisitos*.**

4.3.3 Requisitos de información

Los requisitos de información describen la información que debe almacenar y gestionar el sistema para dar soporte a los procesos de negocio. Los requisitos funcionales hacen referencia a la funcionalidad que se espera del proyecto. Cada uno de los requisitos de información está incluido en el Anexo II – Especificación de requisitos. Para cada uno de los requisitos de información se genera una tabla con datos sobre sus dependencias, los datos específicos que se deben almacenar, el tiempo de vida, la importancia, la urgencia, el estado y la estabilidad.

***NOTA: para más información sobre la especificación de requisitos ver el Anexo II – Especificación de requisitos*.**

4.3.4 Requisitos funcionales

Los requisitos funcionales hacen referencia a la funcionalidad que se espera del proyecto. Estos definen una función del sistema de software o sus componentes. Los requisitos funcionales pueden ser: cálculos, detalles técnicos, manipulación de datos y otras funcionalidades específicas que se supone, un sistema debe cumplir. Cada uno de ellos esta representado por una tabla en el Anexo II – Especificación de requisitos.

***NOTA: para más información sobre la especificación de requisitos ver el Anexo II – Especificación de requisitos*.**

4.3.5 Requisitos no funcionales

En este caso estos requisitos forman parte del sistema, pero no están relacionados con la funcionalidad de este. Se refieren a todos los requisitos que no describen información a guardar, ni funciones a realizar, sino características de funcionamiento, por eso suelen denominarse atributos de calida.

***NOTA: para más información sobre la especificación de requisitos ver el Anexo II – Especificación de requisitos*.**

4.3.6 Actores

Un actor es cualquier elemento que intercambia datos con el sistema. Un actor puede ser un usuario, hardware externo, u otro sistema. La diferencia entre un actor y un usuario de sistema individual es que un actor representa una clase concreta de usuario en lugar de un usuario real.

Debido a las características del proyecto a desarrollar sólo intervienen dos actores, uno humano y el otro una máquina. Las tablas de estos están disponibles en el Anexo II – Especificación de requisitos.

***NOTA: para más información sobre la especificación de requisitos ver el Anexo II – Especificación de requisitos*.**

4.3.7 Diagrama de paquetes del sistema general

En el siguiente diagrama se muestra la organización en paquetes del sistema y seguidamente un ejemplo de uno de los paquetes en específico.

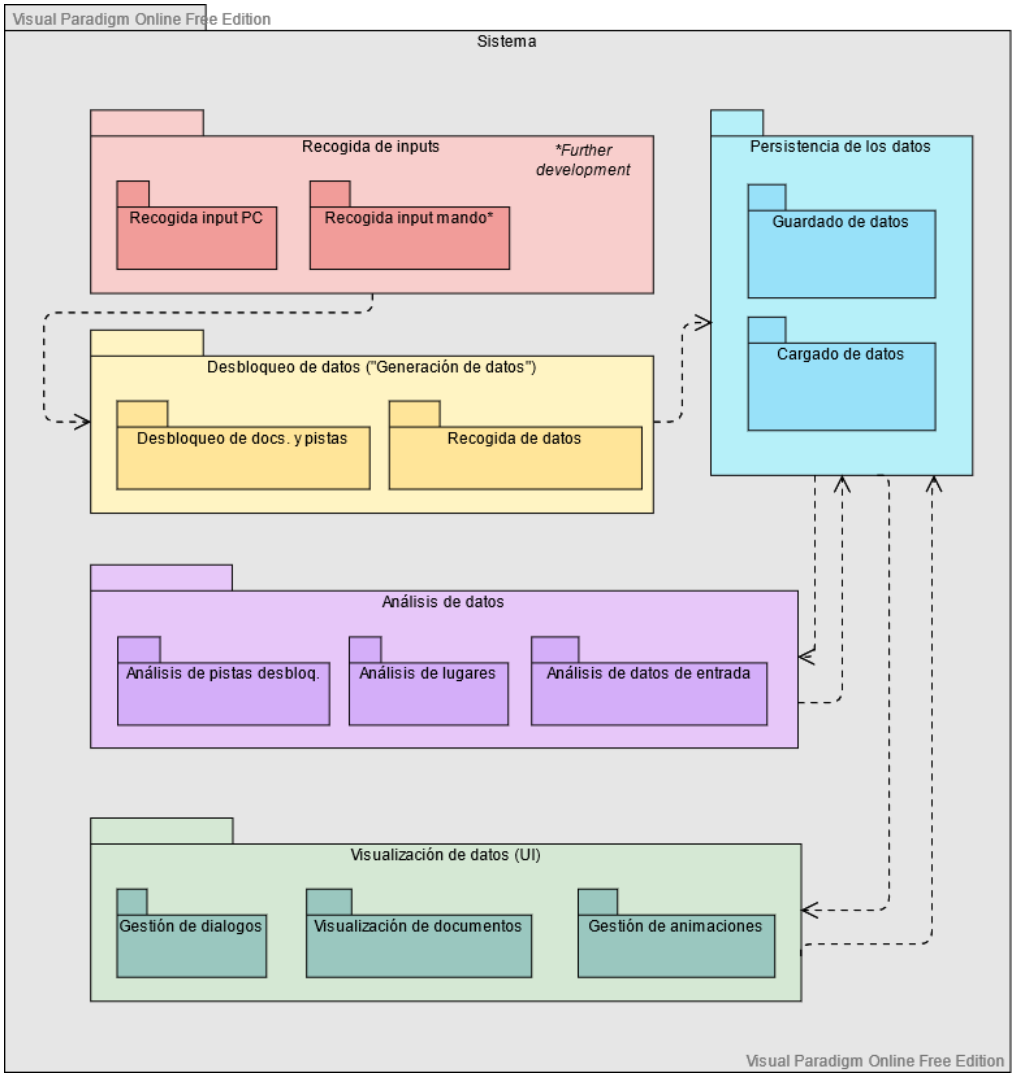


Ilustración 11: Diagrama de paquetes de análisis y servicio

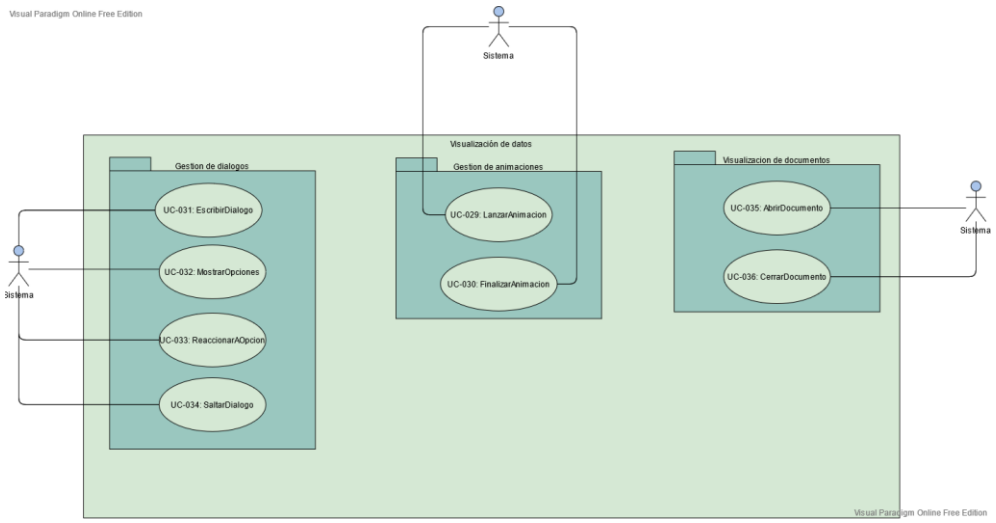


Ilustración 12: Paquete visualización de datos

En el diagrama de la ilustración anterior se pueden ver los casos de uso del paquete de visualización de los datos, para la realización de este tipo de diagramas se ha usado Visual Paradigm Online. Creando cada uno de los paquetes y añadiendo los casos de uso a estos se obtiene un diagrama de casos de uso que nos permite con un rápido vistazo ver toda la funcionalidad que se debe desarrollar. Posteriormente se creará una tabla para cada caso de uso donde se podrá ver en detalle la funcionalidad concreta de cada caso de uso. En estas tablas se incluyen las dependencias de cada caso de uso, sus precondiciones, una descripción detallada del funcionamiento y un listado de pasos con las acciones que se deben tomar para cumplir con el funcionamiento de ese caso de uso.

A continuación, un ejemplo de la tabla de un caso de uso.

UC-032	**MostrarOpciones**	
Versión	1.0	
Autores	Daniel Pérez Martínez	
Fuentes	Equipo de desarrollo	
Dependencias	**OBJ-001 Gestión de diálogos y selección de opciones** **FR-001 Gestión de diálogos** **FR-002 Selección de opciones de dialogo**	
Descripción	El sistema debe ser capaz de ofrecer varias opciones de dialogo para que el usuario pueda elegir por donde dirigir la conversación.	
Precondición	Se debe estar en un dialogo	
Secuencia normal	**Paso**	**Acción**
	1	El sistema muestra las opciones
	2	El usuario elige una de las opciones
	3	El sistema carga el texto concreto a la opción elegida
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**

	-	-
Rendimiento	**Paso**	**Acción**
	-	-
Frecuencia esperada	-	
Importancia	Importante	
Urgencia	Media	
Estado	Validado	
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 3: Ejemplo de caso de uso

4.4 ANÁLISIS DE REQUISITOS

En este anexo se tiene como objetivo detallar los puntos mencionados en los anexos "Anexo I - Planificación temporal" y "Anexo II - Especificación de requisitos". Se tratará de encontrar una solución óptima para construir el sistema. Para alcanzar este objetivo será necesario realizar un análisis de los requisitos que debe cumplir el sistema informando de los problemas y facilidades que supondrá el desarrollo.

NOTA: para más información sobre la especificación de requisitos ver el Anexo III – Análisis de requisitos.

Se ha de desarrollar una aplicación de ocio, concretamente un videojuego limitado a la plataforma del PC desarrollado en el motor grafico de videojuegos Unreal Engine 4. El videojuego debe ser capaz de implementar una inteligencia artificial con la cual se pueda negociar un pacto entre un abogado defensor (el jugador) y el fiscal y/o juez (IA). Así como todos los requisitos necesarios para que se pueda avanzar a través de la historia, incluyendo sistemas de investigación, de conversación, de decisión, etc.

La principal dificultad del proyecto radica en el desarrollo de la inteligencia artificial, ya que se pretende que sea lo suficiente "inteligente" como para poder simular en mayor medida el comportamiento de un fiscal y/o juez en una conversación o una negociación.

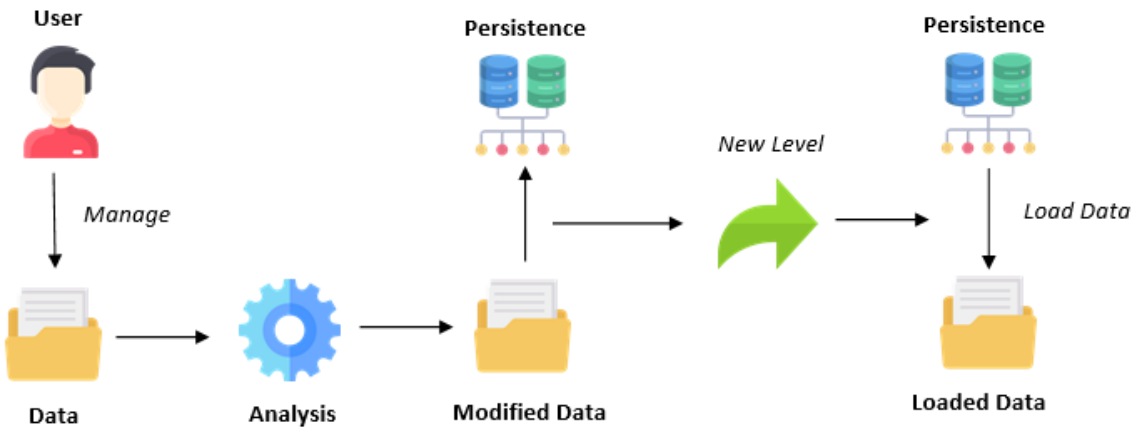


Ilustración 13: Esquema del control de datos

4.4.1 Modelo de dominio

En ingeniería de software, un modelo de dominio en Lenguaje Unificado de Modelado (UML) es un tipo de diagrama de estructura estática que describe la estructura de un sistema mostrando las clases del sistema, sus atributos y las relaciones entre los objetos.

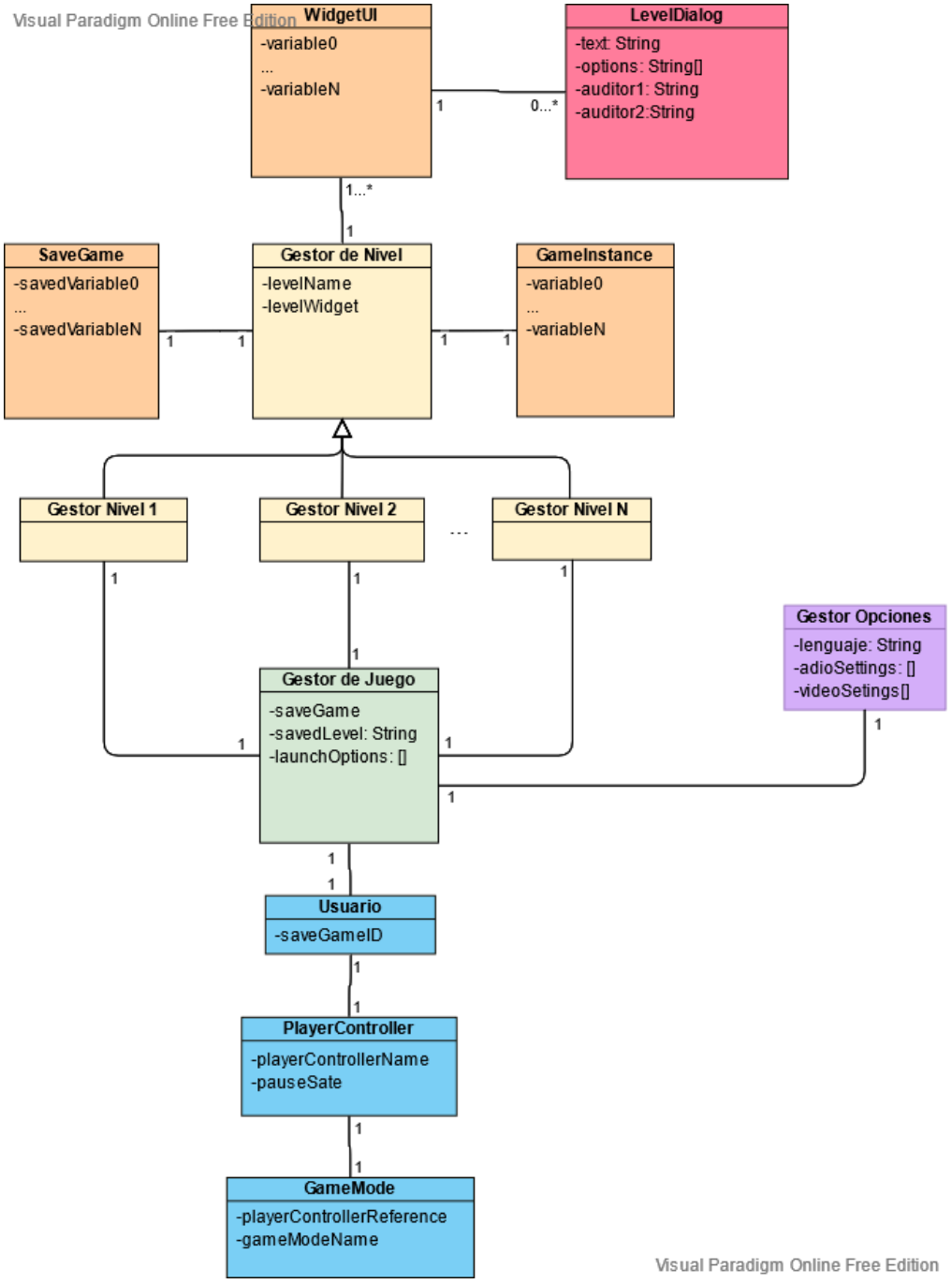


Ilustración 14: Modelo de dominio

4.5 DISEÑO DEL SISTEMA

Esta parte de la documentación corresponde al Anexo IV – Diseño del sistema. En este anexo se continua con el diseño del proyecto; Diseñando la solución propuesta, describiendo la arquitectura y los patrones utilizados. Así mismo se describirá la interfaz creada y los cambios y mejoras más relevantes que se han llevado a cabo tras pruebas con usuarios.

NOTA: para más información sobre la especificación de requisitos ver el Anexo IV – Diseño del sistema.

4.5.1 Patrones de diseño

Para este proyecto se utiliza un patrón muy similar al MVC, modelo- vista-controlador. Este es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

El **Modelo** que contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia.

La **Vista**, o interfaz de usuario y por lo tanto los gráficos del videojuego, que compone la información que se envía al cliente y los mecanismos interacción con éste.

El **Controlador**, que actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

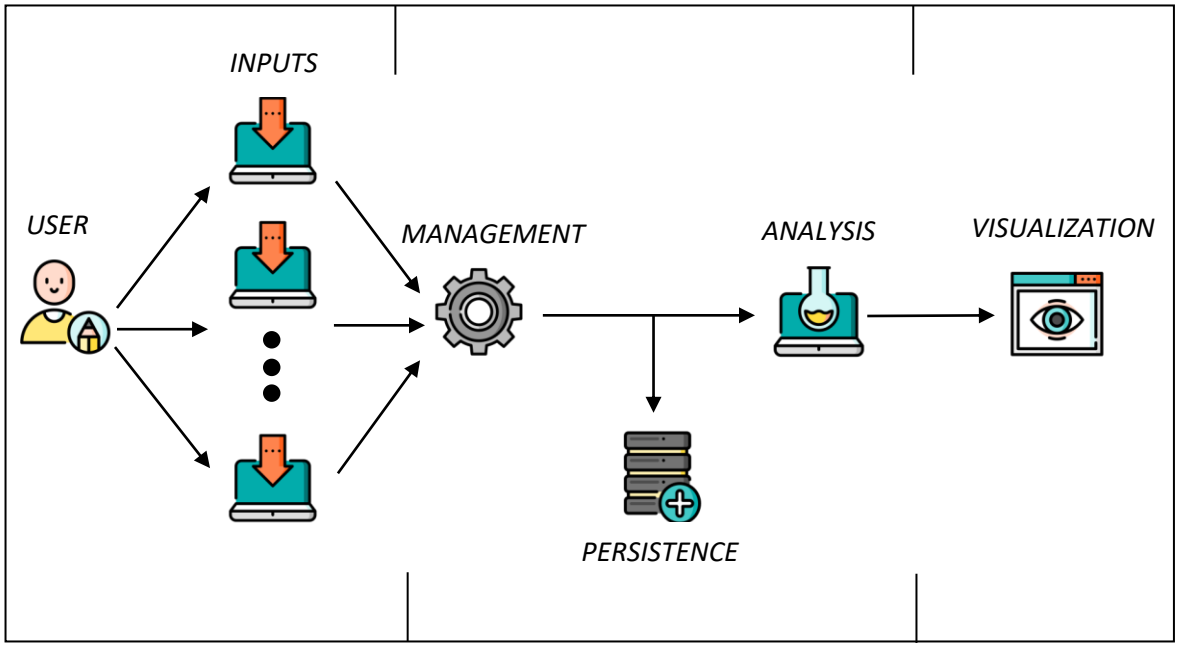


Ilustración 15: Esquema básico de la arquitectura de la plataforma

4.5.2 Game Design Document

Las siguientes partes contenidas en este anexo han sido dirigidas más hacia el diseño de un videojuego orientado a la industria y no tanto como se diseñaría un producto software tradicional. Por lo que se ha decidido realizar un Game Design Document (GDD).

Titulo	**“Anne Connery” (título provisional)**
Género	Aventura grafica
Audiencia	Jugadores casuales, adultos
Plataformas	PC (Windows 10, Linux, Mac)
Modos de juego	Campaña para un jugador
Temática	Mundo legal, abogacía…
Estética	PixelArt 2D

Tabla 4: Ficha del juego

NOTA: para ver el GDD completo visitar el apartado con el mismo nombre en el Anexo IV – Diseño del sistema.

4.5.3 Flujo de partida

Durante el diseño también fue importante definir el flujo de la partida mediante diagramas de actividad de forma que con un vistazo rápido se pudieran ver las acciones que deben tomar para llegar a ciertos objetivos. Estos diagramas se realizaron de los niveles más complejos y de los que podrían suponer más complicaciones a la hora de programarlos. Un buen ejemplo de esto es el nivel de negociaciones.

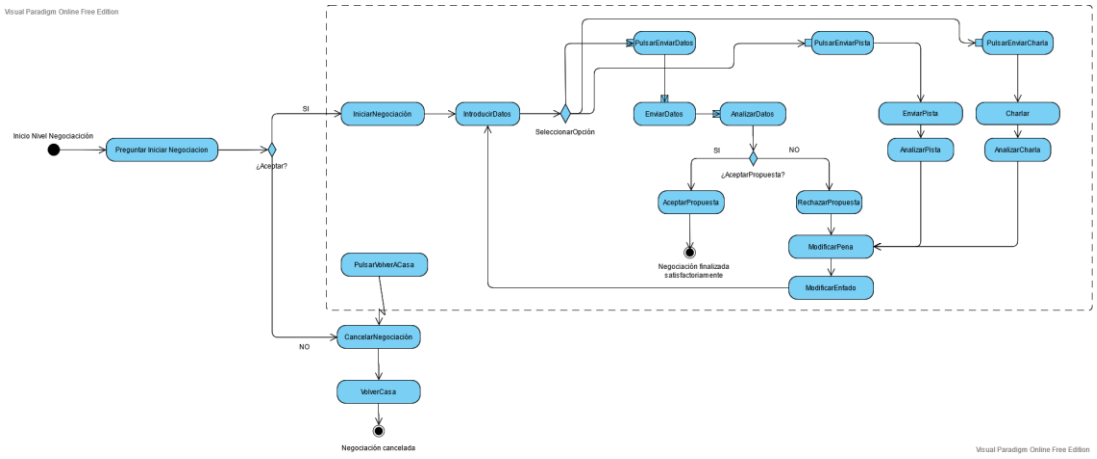


Ilustración 16: Diagrama de actividad de las negociaciones

4.5.4 Diseño de interfaz

El diseño de la interfaz es un trabajo fundamental a la hora de hacer un videojuego, dado que no solo es importante que el usuario comprenda rápidamente como se juega y que opciones tiene,

también es una parte fundamental tener un aspecto visual llamativo para atraer a los jugadores y guiarlo a través de la historia.

Para el diseño de los escenarios se tuvieron en cuenta algunas inspiraciones como el videojuego de la imagen que se muestra a continuación.

Ilustración 17: The Red String Club

Después del trabajo de exploración realizado se llegaron a obtener los escenarios deseados con un acabado como el que se muestra en la ilustración 16.

Ilustración 18: Pantalla de título del juego final

Una vez se tenían ciertas interfaces más o menos completadas se hicieron una serie de pruebas con usuarios para ver el nivel de comprensibilidad que tenían las interfaces, algunas de las mejoras que se obtuvieron de estas pruebas se muestran en el anexo ya mencionado. Así como el diseño de personajes, diálogos, escenarios, etc.

4.6 IMPLEMENTACIÓN

4.6.1 Componentes generales

4.6.1.1 GameMode

En este se encuentran las características más relevantes sobre las clases que se van a utilizar a lo largo del desarrollo, indicando, por ejemplo, clases como la del GameState, GameSesion o la más importante, el PlayerController. Este último, es la única clase que se ha modificado, dejando el resto por defecto.

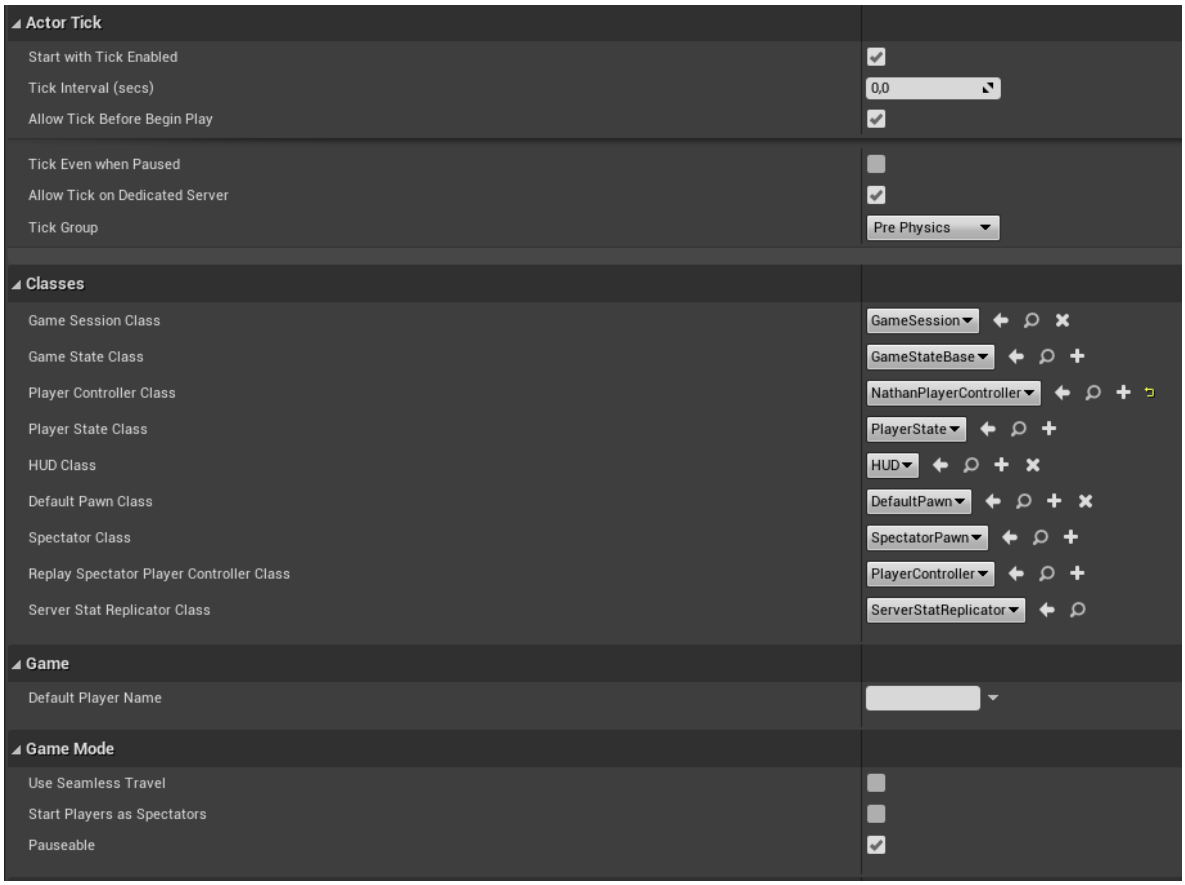


Ilustración 19: Configuración del GameMode

Establecer la clase PlayerController a una personalizada nos permitirá tener un control más detallado de los inputs del usuario, así como mantener una serie de variables globales a todo el proyecto, permitiendo así que estas sean accesibles desde cualquier punto de la ejecución.

4.6.1.2 NathanPlayerController

Esta es la clase principal del proyecto, podría definirse como la clase que acompaña al usuario a lo largo de todo el juego, con esta clase se permite modificar ciertos inputs para poder realizar

acciones específicas a lo largo de la historia. Los dos inputs más destacables se muestran en los siguientes subapartados.

4.6.1.2.1 Menú de Pausa

Se ha establecido un input para la pausa, que como su nombre indica se utiliza para pausar el juego y permitir una serie de opciones al jugador, como cambiar el idioma, salir, continuar… Para el funcionamiento de este se ha creado un widget que nos permite navegar entre estas opciones.

El Blueprint que controla la pausa, cuando recibe el input, comprueba que el juego no esté pausado, si no lo está crea el widget de pausa la añade a la pantalla y pone el juego en modo pausa.

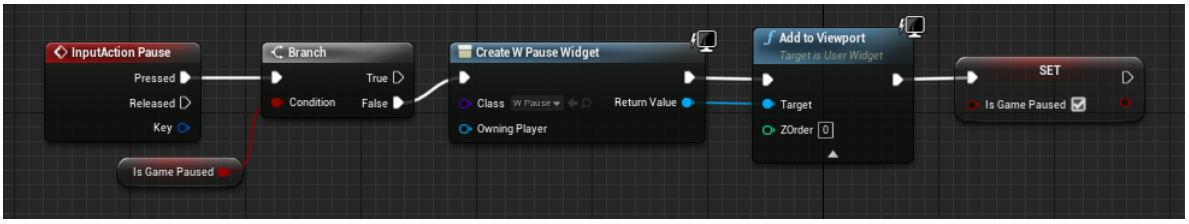


Ilustración 20: Blueprint pausa

Ilustración 21: Menú Pausa

4.6.1.2.2 Teclas numéricas (teléfono)

Otro input que se establece en la clase PlayerController es el de las teclas numéricas, que permite al usuario marcar números en el teléfono de forma más sencilla. Cada vez que se realiza una pulsación de un número del teclado, se recoge y se comprueba que se esté "utilizando el teléfono" (para ello existe una variable "InPhoneWidget?" que permite conocer si el widget W_Phone está

en pantalla o no), si es así se añade el numero a la pantalla del teléfono y se reproduce un sonido de pulsación para dar realimentación al usuario.

Ilustración 22: Widget teléfono

4.6.1.2.3 Otros

Dentro de la clase NathanPlayerController también podemos encontrar varias variables que nos permiten controlar un minijuego de tocar la guitarra (*en desarrollo), así como una serie de variables que son, necesariamente accesibles desde otros niveles.

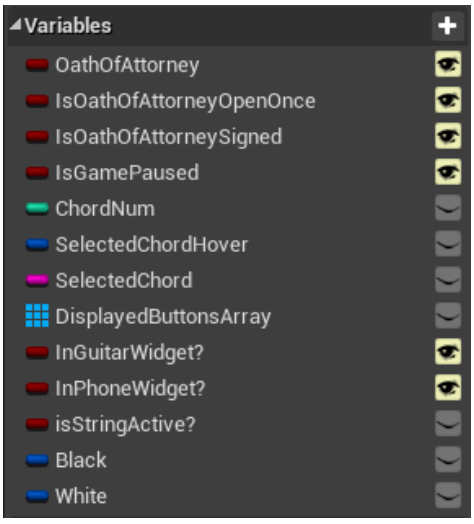


Ilustración 23: Variables del PlayerController

Tambien se encuentran en esta clase varias funciones que permiten el funcionamiento del minijuego de tocar la guitarra.

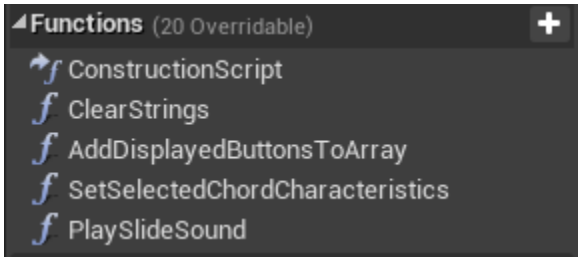


Ilustración 24: Funciones del PlayerController

4.6.1.3 MySaveGame

Se ha implementado un sistema de autoguardado que guarda automáticamente en ciertos momentos de la aventura, para poder cerrar el juego y continuar jugando por el mismo sitio al volver a iniciarlo.

Se trata de una clase en la que se van a encontrar todas las variables que han de guardarse entre partida y partida. Cada vez que se llame a la función de guardado (*SaveGame*, incluida en MyGameInstance), se recogerán todos los datos importantes en ese momento y se guardaran en una instancia de la clase MySaveGame.

De la misma manera cada vez que se inicia el juego se realiza una llamada a la función de cargado (*LoadGame*, incluida en MyGameInstance) para cargar los datos anteriormente guardados.

Para realizar este tipo de cargas y guardados se hace uso de variables “duplicadas”, existe una variable en MySaveGame, donde se guardan los datos y una en MyGameInstance, de la cual se leen los datos durante la partida. En el momento de guardar se toman los datos de las variables de MyGameInstance y se sobre escriben en las variables de MySaveGame.

Algunas de las variables más importantes son:

- Lenguage (guardael idioma activo cuando se cerró el juego)
- Contactcs (guarda la lista de contactos del móvil)
- MapOfDocs (guarda todos los documentos desbloqueados y su estado)
- ArrayOfDocs (guarda todos los documentos desbloqueados)
- LevelName (guarda el nombre del ultimo nivel iniciado para volver a él)
- ProsecutorState (guarda el estado de “animo” y las propuestas del fiscal)
- Variables para casos, existen dos variables para cada uno de los casos:
    - Array de documentos del caso
    - Array de pruebas del caso

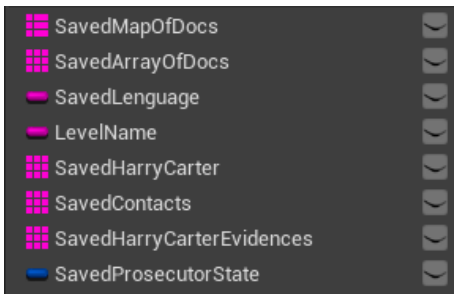


Ilustración 25: Variables de MySaveGame

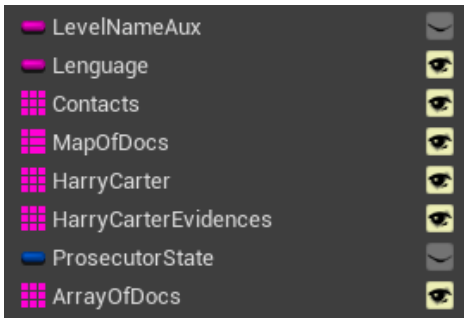


Ilustración 26 - Variables de MyGameInstance

La función de guardado recoge la información de MyGameInstance y la guarda en MySaveGame. Para ello lo primero que hace es cargar el slot de guardado donde se están guardando los datos, dado que solo se permite una sola partida guardada se indica el nombre de este slot en el nodo LoadGameFromSlot, con el valor devuelto realizamos un casting a la clase MySaveGame y con la instancia a dicha clase asignamos nuestra LoadSaveInstance. Una vez realizado esto, podemos asignar a todas las variables que queramos guardar, Savedxxxxx, los valores concretos obtenidos de las variables de MyGameInstance. En el caso especial del LevelName, tenemos que obtener su valor gracias al nodo GetCurrentLevelName. Una vez todos los valores han sido asignados los guardamos haciendo uso de la instancia LoadSaveInstance que hemos obtenido anteriormente y el nodo SaveGameToSlot.

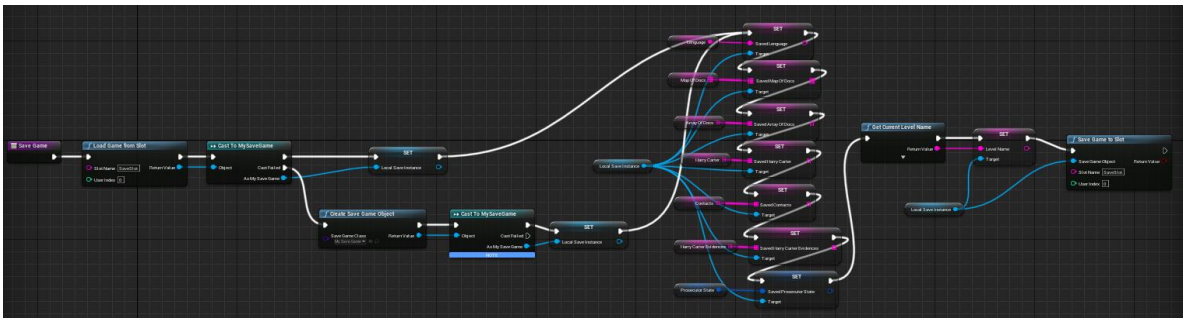


Ilustración 27: Blueprint Función Save

La uncion de cargado recoge la información de MySaveGame y la guarda en MyGameInstance. Para ello se realiza un mecanismo inverso al de la función anterior. Primero cargamos el slot de

guardado donde se están guardando los datos, realizamos el casting a nuestra clase MySaveGame, y al contrario que en el caso anterior, asignamos a las variables de MyGameInstance, los valores que recogemos de las variables del casting de MySaveGame, por último le pasamos al nodo OpenLevel el nombre del ultimo nivel que se jugó.

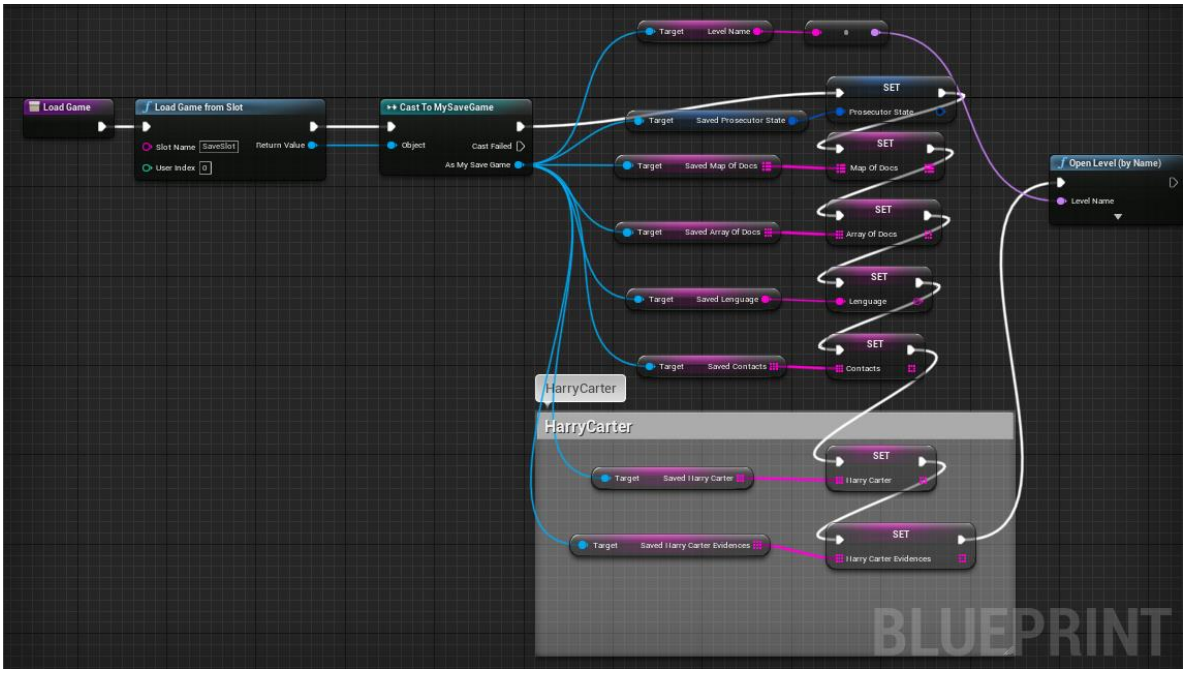


Ilustración 28: Blueprint Función Load

4.6.1.4 MyGameInstance

Se puede decir que esta clase es el objeto más importante del proyecto, se trata de una instancia accesible desde cualquier punto de la ejecución y desde cualquier otra clase. En esta clase se encuentran gran parte de las funciones más generales o de más continuado uso y las variables más importantes (como las variables de guardado mencionadas anteriormente). De esta forma no es necesario repetir código y solamente referenciar la GameInstance y acceder a estas funciones o variables.

Tipos de funciones:

- Ambientación
    - Audio
        - PlayX (reproduce una pista de audio entre niveles)
        - StopX (deja de reproducir una pista de audio)
    - Video
        - FadeIn (inicia la animación de desfundido a negro)
        - FadeOut (inicia la animación de fundido a negro)

- Audio de tecleo (varias funciones para reproducir un sonido cuando se escribe texto letra a letra)

- Generales
  - Funciones de guardado
  - ChangeTalkingBox (cambia la imagen del bocadillo de los personajes de izquierda a derecha)

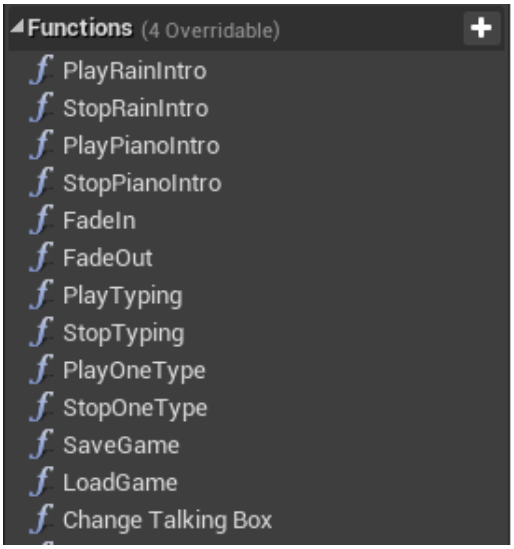


Ilustración 29: Funciones de MyGameInstance

El funcionamiento de Playxxxx es simple, crea un sonido con los parámetros especificados y lo reproduce, en este caso es necesario guardar el sonido en una variable de MyGameInstance para poder pasarla entre niveles y que el audio no se corete al finalizar un nivel e iniciar el siguiente.

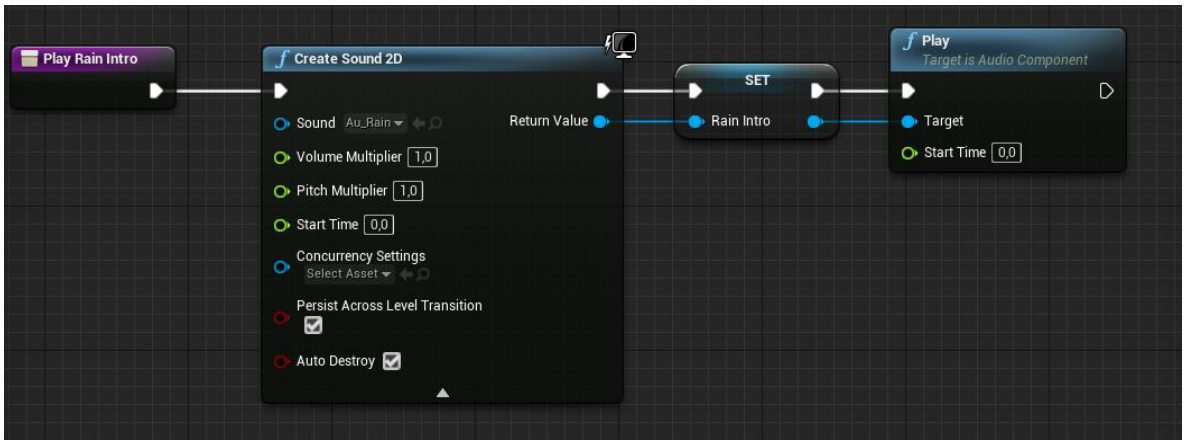


Ilustración 30: Función PlayRainIntro

PlayOneType, es un caso especifico de reproduccion de sonido que se llama cada vez que se escribe una letra de un dialogo para dar la senscion de tecleo.

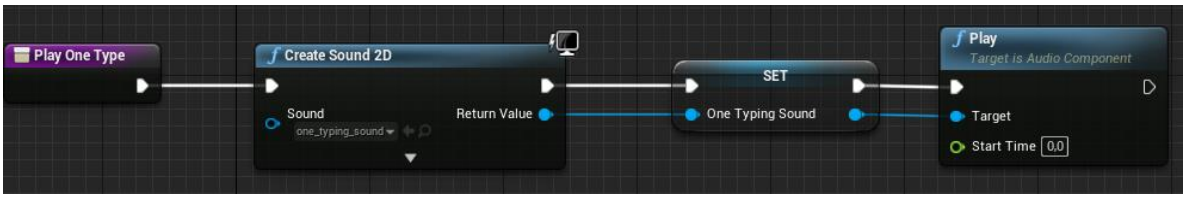


Ilustración 31: Función PlayOneType

FadeIn y FadeOut son dos funciones que se suelen utilizar al salir de un nivel y entrar en el siguiente, se trata de una animacion de fundido a negro, para ello existen dos widgets en los que estan programadas las animaciones para que cuando se creen dichos widgets se reproduzcan. (Ilustración 19: Blueprint del Widget WAnim_FadeIn)

Ilustración 32: función FadeIn

Exsisten dos Widgets en los cuales se encuentras las animaciones de fundido a negro.

En la imagen inferior se puede ver la animacion de fundido a negro, donde solamente se baja o sube la opacidad de un objeto “border” de color negro de 1 a 0 o biceversa en un tiempo de 1.5 segundos.

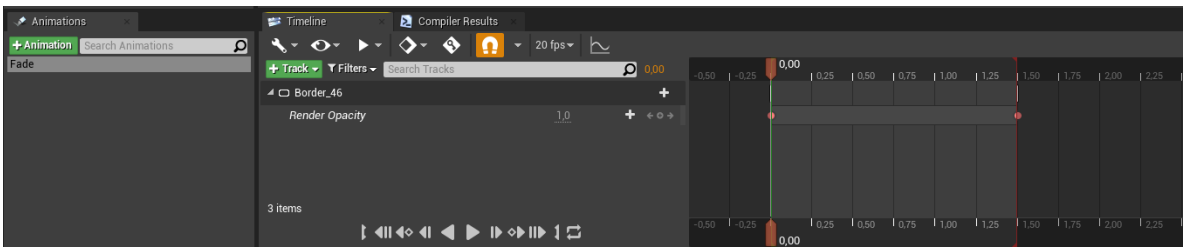


Ilustración 33: Timeline de la animación Fade In

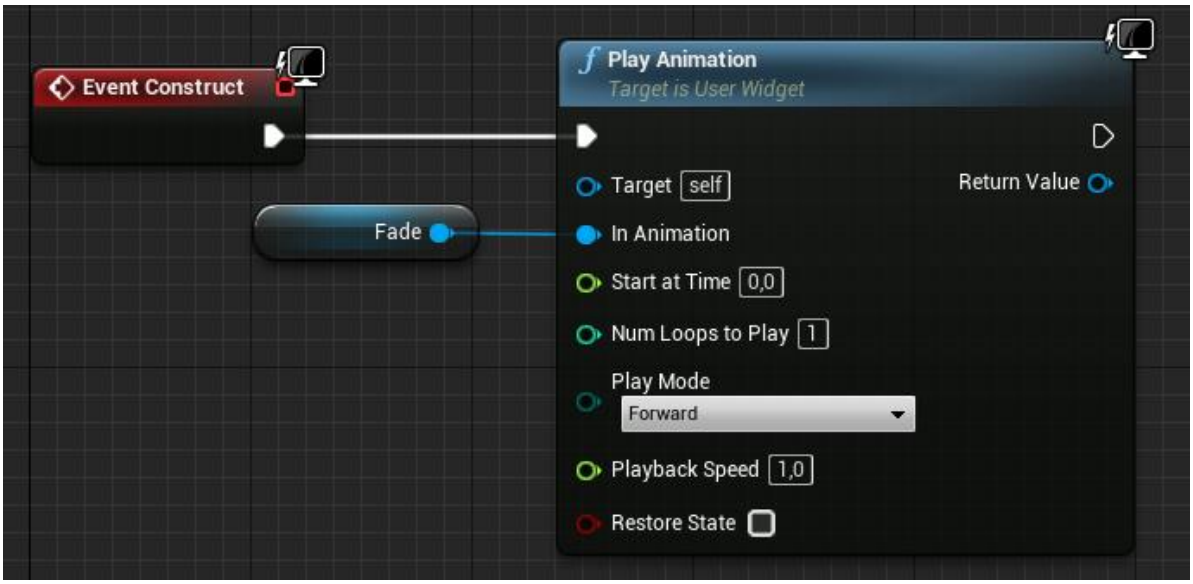


Ilustración 34: Blueprint del Widget WAnim_FadeIn

La función *ChangeTalkingBox* se encarga de cambiar la posición de las imágenes de dialogo cuando cambia el personaje que habla. Esta función es llamada desde los widgets de dialogo cada vez que se requiera dicho cambio. Para ello es necesario pasarles una referencia a los dos bocadillos (imágenes), a la *textBox* donde se colocará el texto y a la imagen que representa que el texto a finalizado y se puede pasar el al siguiente (*skipArrow*). Una se tienen las referencias a estos objetos del widget en concreto se comprueba cuál de las dos posiciones del bocadillo está activa y en función de ello se reposicionan los objetos con *SetRenderTransform* y convenientemente se ocultan o muestran con *SetVisibility*.

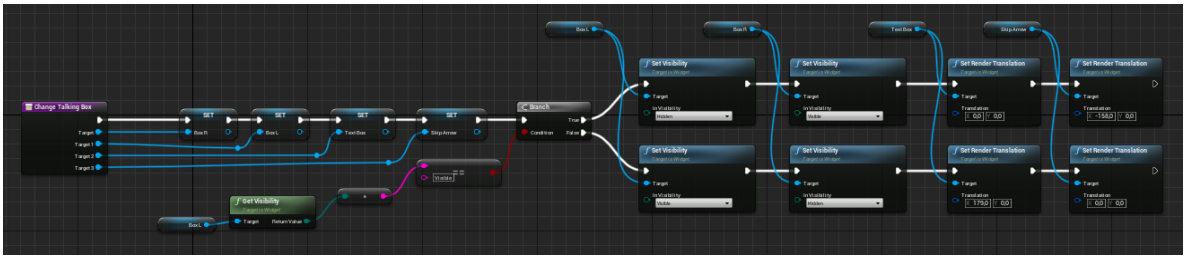


Ilustración 35: Función ChangeTalkingBox

También existen gran cantidad de variables que se irán comentando a lo largo del informe según se utilicen.

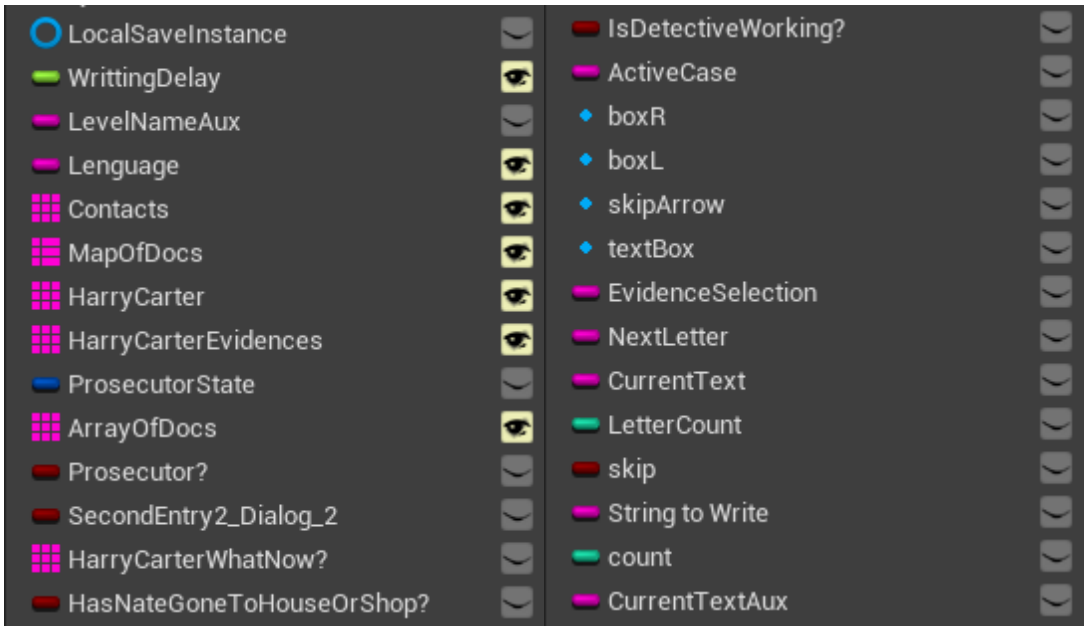


Ilustración 36: Variables de MyGameInstance

4.6.2 Menus e interfaces

4.6.2.1 Tutoriales

Los tutoriales son una parte muy importante dentro del proyecto ya que explican al jugador como ha de usar la interfaz que se le muestra. Todos los tutoriales están programados de la misma forma y saltarán en determinados puntos del juego donde sea necesario explicar una nueva interacción.

Ilustración 37: Tutorial

Los tutoriales están hecho a través de las interfaces graficas de tipo Widget Blueprint y se encuentran en la carpeta 0_General/Tutorials.

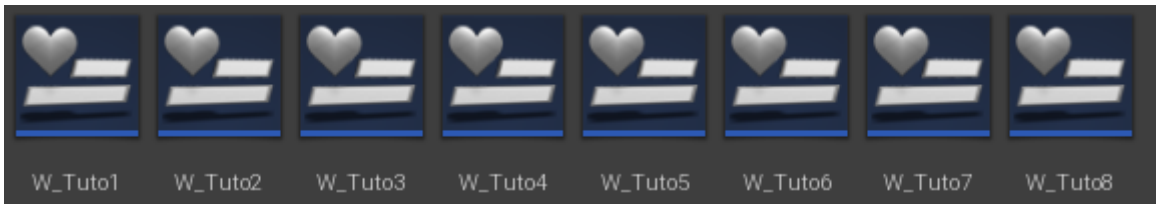


Ilustración 38: Ficheros de tutoriales

Internamente podemos encontrar dos funciones, el *Event PreConstruct* y un *OnClick* para cerrar la ventana con la información. El primero nos sirve para que se seleccione el texto en el idioma que esté configurado antes de que se muestre el tutorial. Para ello es necesario castear a MyGameInstance donde se encuentra guardado el Idioma y dependiendo del que sea *settear* los textos convenientes.

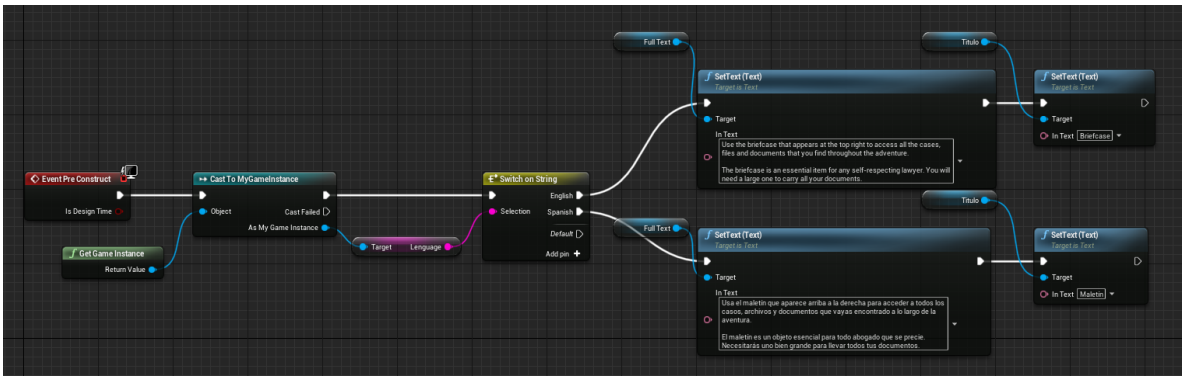


Ilustración 39: Evenet PreConstruct Tutorial

El segundo simplemente elimina el tutorial de la vista del jugador al hacer click sobre el aspa de arriba a la derecha. Dado que tras este tutorial debe aparecer otro, se crea el nuevo tutorial y se elimina el anterior.

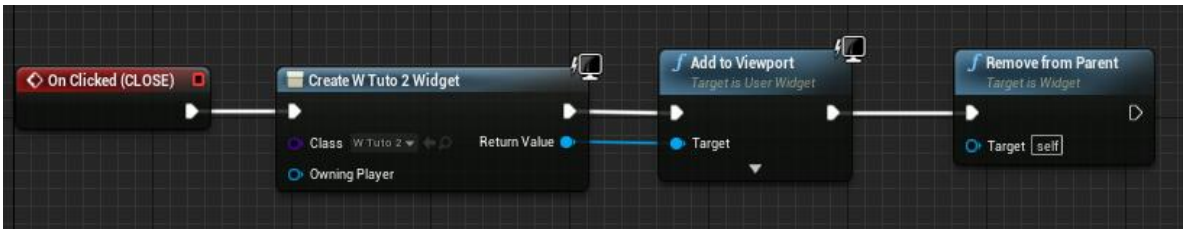


Ilustración 40: Onclick close tutorial

4.6.2.2 Cuaderno de casos

El cuaderno de casos y documentos es una de las interfaces más accedidas durante las partidas. Dado que a través de este se acceden a los documentos de cada caso, incluidos en el maletín (que se explicara posteriormente.). Este cuaderno se trata de una interfaz en forma de libreta de papel que ofrece al usuario tres opciones, casos abiertos, donde se encuentran los casos que actualmente están activos y deben resolverse, casos cerrados, donde se encuentran los casos que ya se han resuelto, otros documentos, donde se encuentran casos de otras personas en los que no ejerces como abogado y otros documentos variados.

Ilustración 41: Cuaderno de casos

Haciendo click en cualquiera de las entradas se mostrará el maletín concreto del caso con sus documentos o en documento si este no fuera un caso.

Internamente, al igual que los tutoriales y las demás interfaces que se presentarán en este informe tienen un *Event PreConstruct* donde se obtiene el idioma y se poden los textos en su debido idioma. Así como una *OnClick* para cerrar el cuaderno.

NOTA: las funcionalidades comentadas anteriormente son idénticas a las mostradas para los tutoriales y futuras interfaces por lo que se omite su explicación.

En el evento de construcción será necesario obtener los documentos desbloqueados de MyGameInstance, donde se encuentra un array donde se van añadiendo los casos y documentos a medida que se desbloquean, también existe un mapa que indica el estado de cada uno de los documentos (*open*, *closed*, *other*), en función de estos se crea un botón (*W_NoteBook_DocButton*) que nos permitirá acceder a ese caso y se añade a la sección del cuaderno concreta.

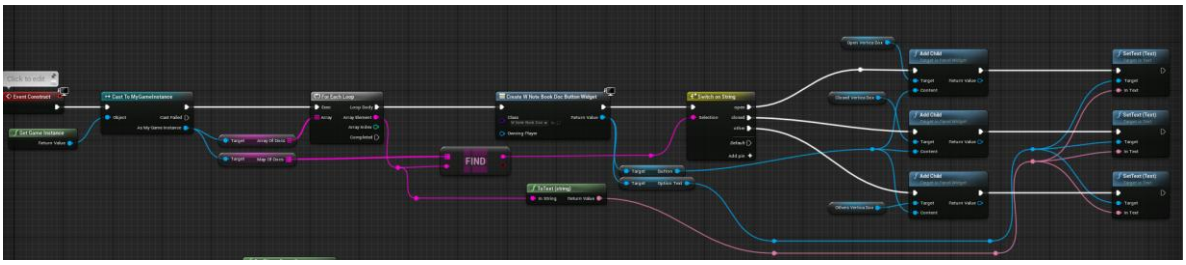


Ilustración 42: Evento de construcción Cuaderno de casos

Cada uno de los botones creados tiene una funcionalidad asociada a si mismo. Esta está programada en *W_NoteBook_DocButton*, donde se obtienen el evento del click de el botón y se lee la propiedad *text* que contiene el nombre del botón. En función de esta se creará la interfaz conveniente, ya sea de un caso (maletín) o un simple documento. Después de crear esta interfaz se elimina el cuaderno y se emite un sonido.

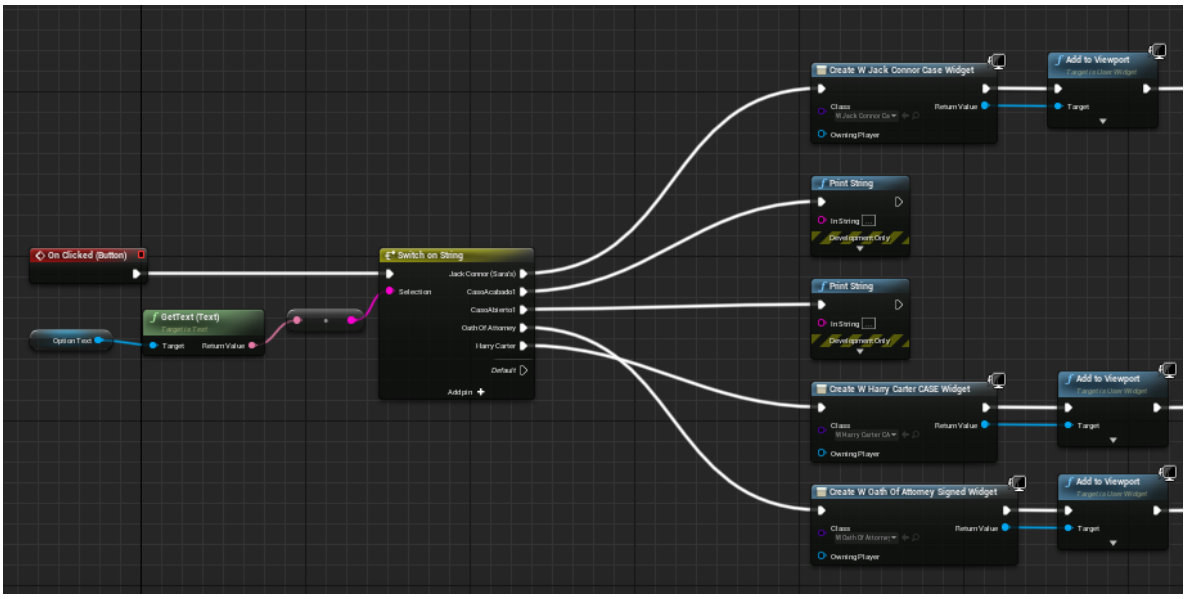


Ilustración 43: Obtención de propiedad "text" y creación de la n. interfaz

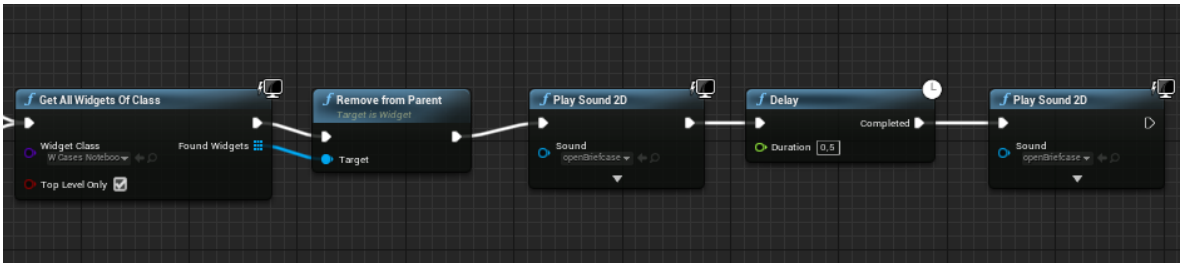


Ilustración 44: Eliminación del Widget anterior y emisión de sonido

4.6.2.3 Maletín de casos

Justo después de abrir un caso a través del cuaderno de documentos nos encontraremos con su maletín. Donde se encontrarán todos los documentos e información asociados a ese caso.

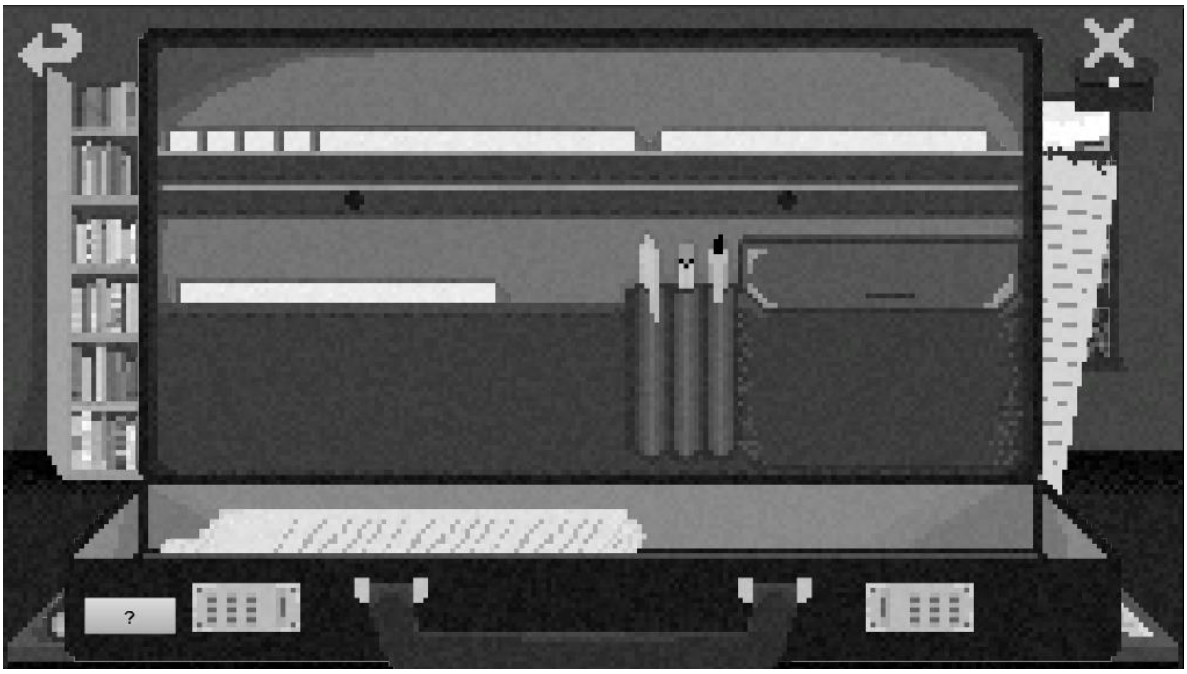


Ilustración 45: Maletín con los documentos del caso

Todos los documentos tienen un hover para ser más visibles al usuario y todos ellos están programados de la misma manera, alternando la imagen que se muestra en el momento en el que el usuario pasa el ratón por encima del documento.

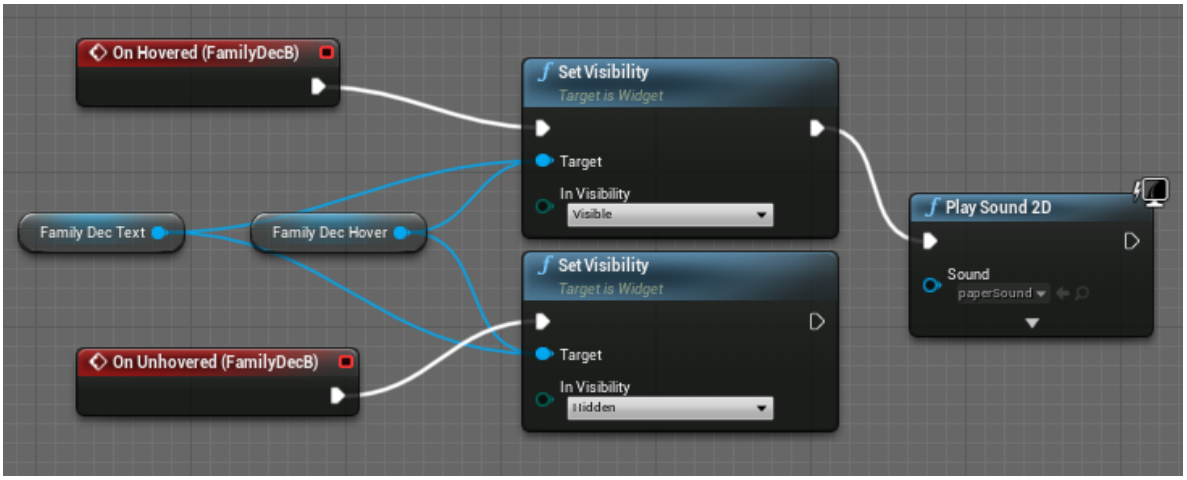


Ilustración 46: Ejemplo general de hover

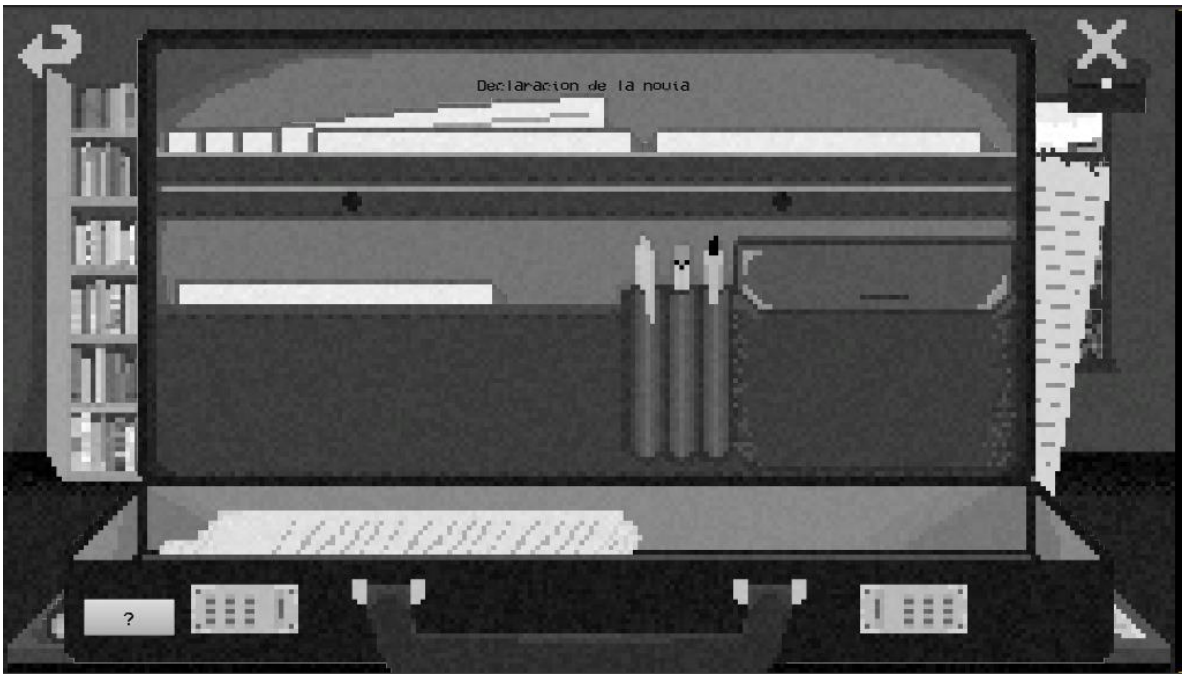


Ilustración 47: Ejemplo de hover

Respecto al funcionamiento interno del maletín existen gran cantidad de funcionalidades, como se muestran en la siguiente captura estas están claramente divididas en varias secciones.

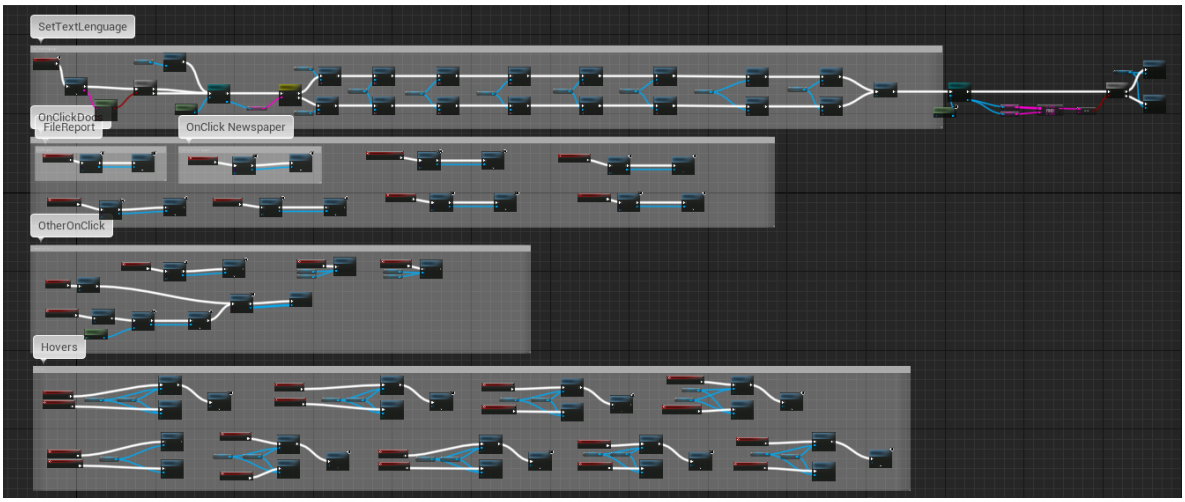


Ilustración 48: Vistazo general al funcionamiento del maletín

- *Hovers*: en esta sección se encuentran todos los *hovers*.
- *OnClicks*: en esta sección se encuentran los *OnClick* de cada documento, que muestran el documento al hacer click sobre el mismo.

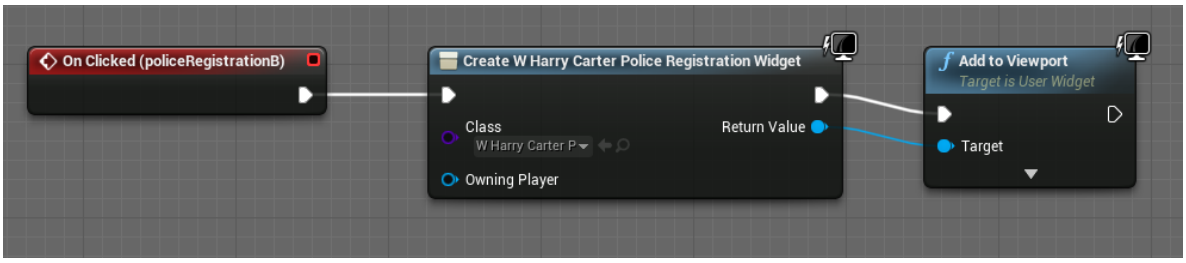


Ilustración 49: Onclick generico

- *Other OnClicks*: se trata de *OnClick* más generales, como cerrar el maletín, volver al cuaderno o acceder a las opciones del caso.
- *PreConstruct*: en este se encuentra la selección de idioma y la comprobación de los documentos que han sido desbloqueados. Para esta comprobación existe una funciona llamada "CheckUnlockedDocs" que se encuentra en la sección final del *PreConstruct*.

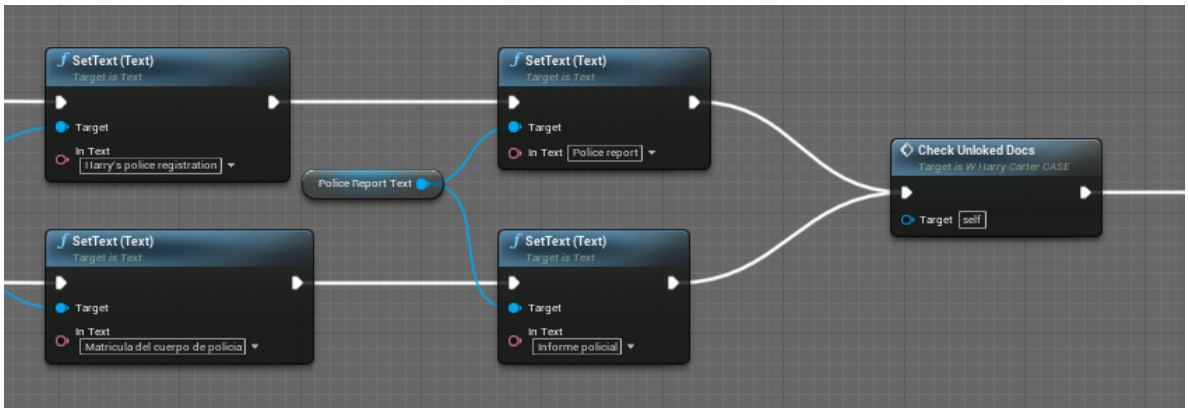


Ilustración 50: Sección final de PreConstruct Maletín

Esta función es la encargada de ver que documentos se han desbloqueado y mostrarlos en pantalla para ser accedidos. El funcionamiento de la función es bastante simple, en primer lugar castea a MyGameInstance para obtener el array *HarryCarter* donde se van añadiendo los documentos a medida que se desbloquean. Una vez lo tiene los recorre y en función de los *string* que estén en este hace visibles unos botones (documentos) u otros.

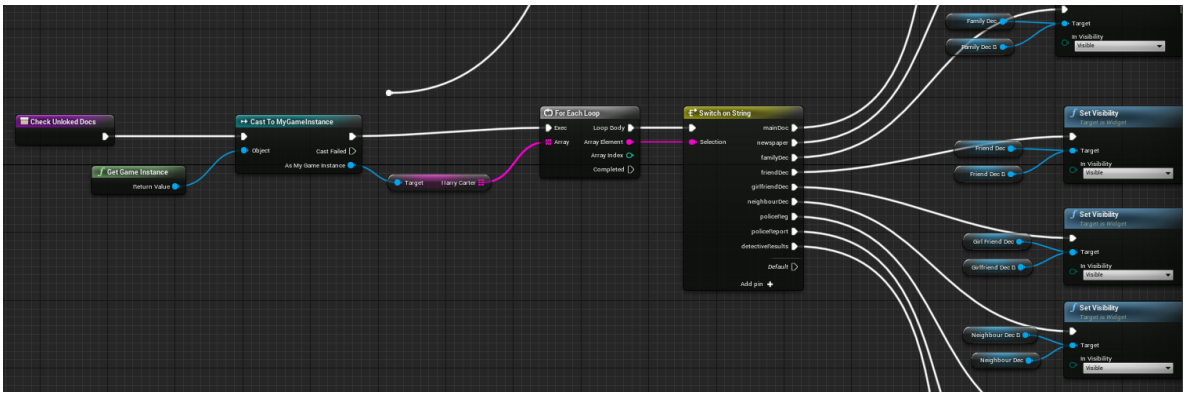


Ilustración 51: CheckUnlockedDocs

4.6.2.4 Menú What now?

Este menú es accesible desde el maletín de los casos abiertos. Se trata del menú que nos permite seleccionar entre investigar, negociar o ir a juicio. Para que solo sea accesible en los casos abiertos al final del PreConstruct de cada caso se comprueba si el caso está abierto o cerrado para mostrar o no el botón que permite el acceso a este menú.

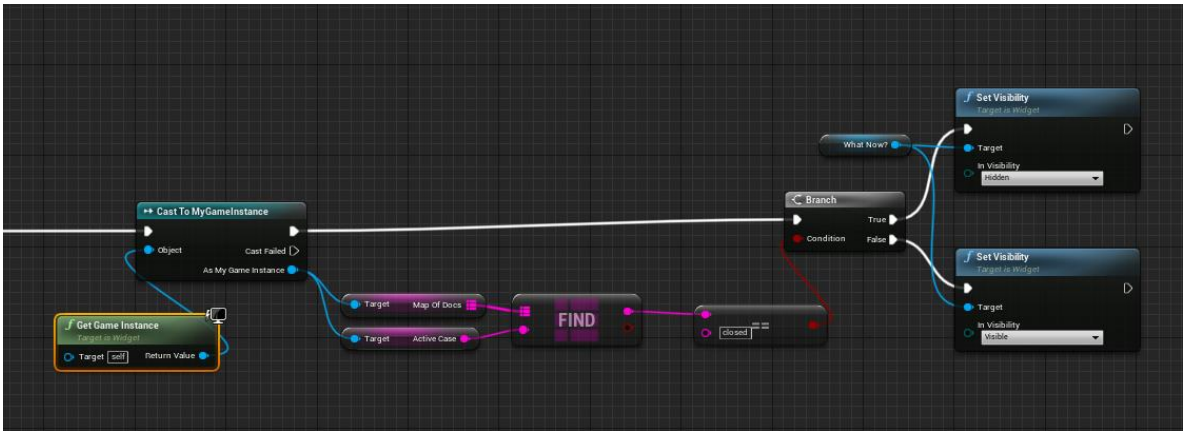


Ilustración 52: Comprobar el acceso a WhatNow?

Este menú consta de tres botones con imágenes que permite acceder a distintas secciones del juego.

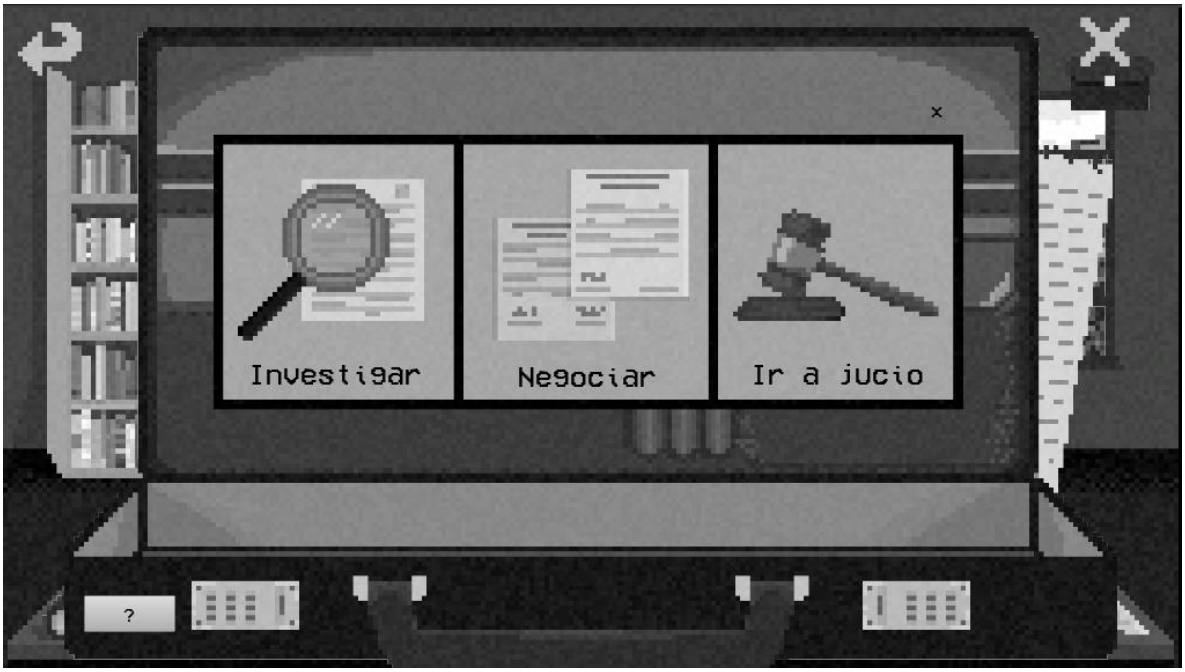


Ilustración 53: Menú WhatNow?

La funcionalidad, igual que anteriormente, inicia por seleccionar el idioma. A demás contiene tres *OnClick* que permiten cargar los niveles correspondientes a investigar, negociar o ir a juicio. Los tres son de la siguiente forma.

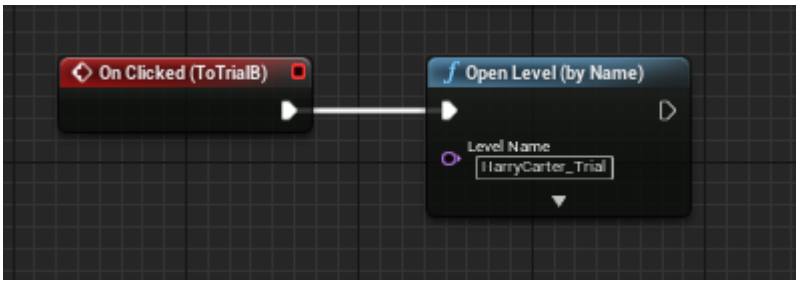


Ilustración 54: OnClick WhatNow?

Si durante la negociación el fiscal decide ir a juicio el botón de negociar quedará bloqueado y no se podrá volver a intentar negociar. Para ello se genera el siguiente código a la hora de construir el menú, después de seleccionar el idioma.

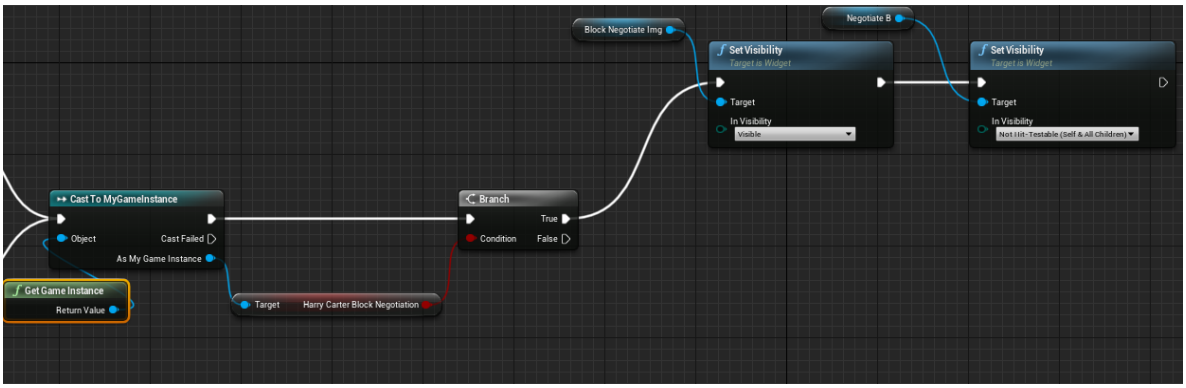


Ilustración 55: Bloquear negociaciones

4.6.2.5 Teléfono móvil

El teléfono móvil nos permite llamar a los distintos personajes, su interfaz esta mejor explicada en el Anexo VI - Manual de usuario, por lo que en este apartado nos centraremos en la funcionalidad. De nuevo esta está separada en grandes apartados.

- *Varios OnClick*: cerrar, colgar, llamar, abrir los contactos, etc.

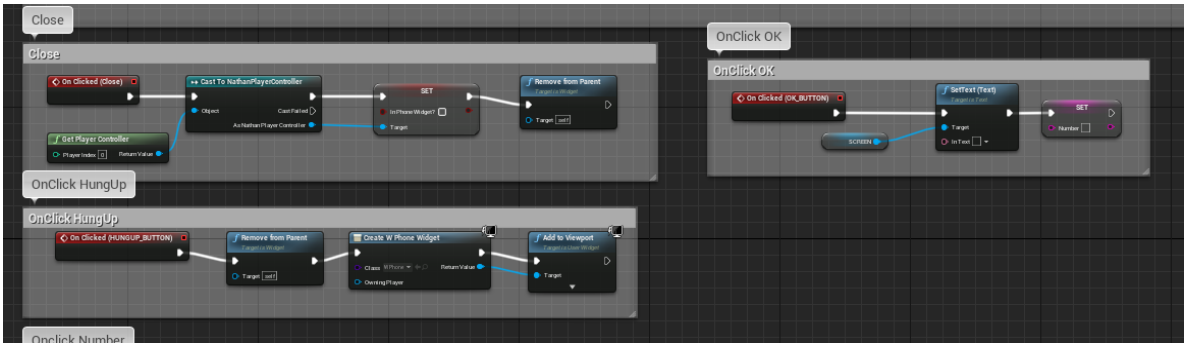


Ilustración 56: Varios Onclick

- *Onclick numbers*: marcan en pantalla el número que se ha decidido marcar (también se puede usar el teclado).

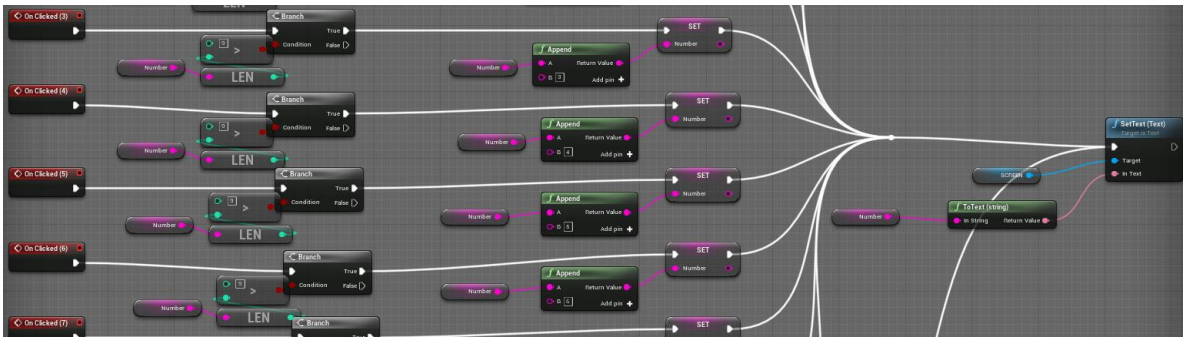


Ilustración 57: Onclcik números

- *Hovers*: *hovers* de las teclas al pasar por encima de ellas.

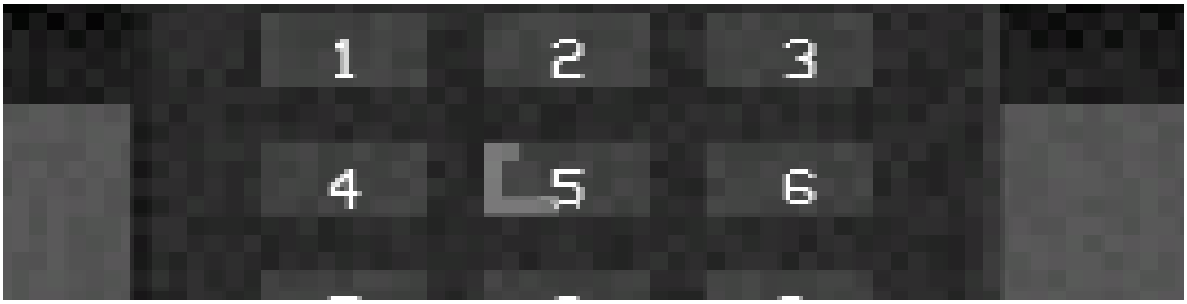


Ilustración 58: Hover tecla 5

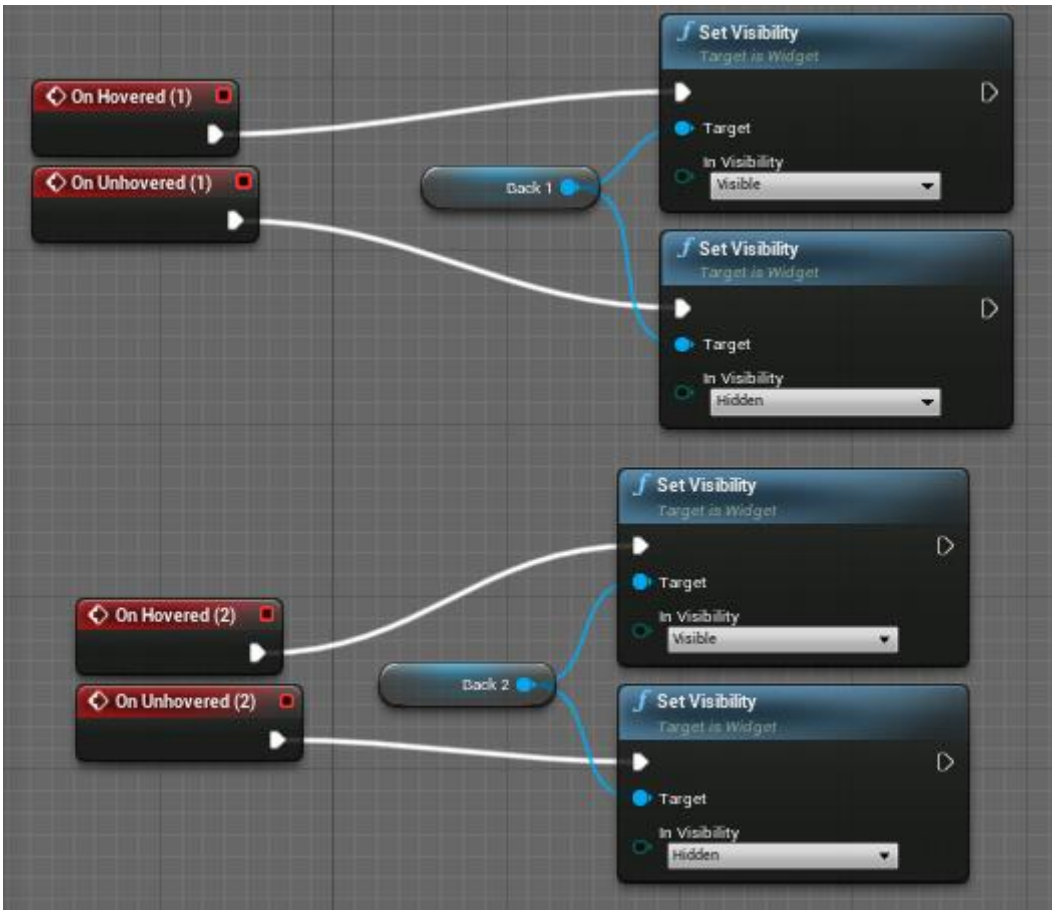


Ilustración 59: Hover de las teclas

- *CheckNumber*: se trata de la funcionalidad más importante de este apartado, cuando el usuario hace click en el botón de llamar se activa esta funcionalidad. En primer lugar se bloquea al botón de cerrar el móvil y se empieza a reproducir un sonido de conectando. Posteriormente se hace un switch para comprobar que numero se a marcadao. Y en función de este se realiza una acción u otra.

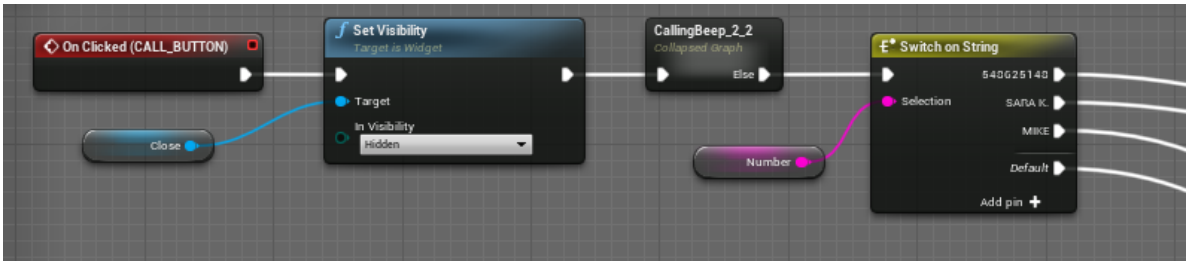


Ilustración 60: Funcionalidad de llamada 1

Dado que es posible llamar a varios todos los personajes en todos los niveles es necesario que se compruebe desde que nivel y momento del juego se está llamando para cada personaje. De esta manera los personajes tendrán diálogos distintos en función del momento en el que se les llame. Para ello después de comprobar el numero marcado se comprueba el nivel desde el que se llama.

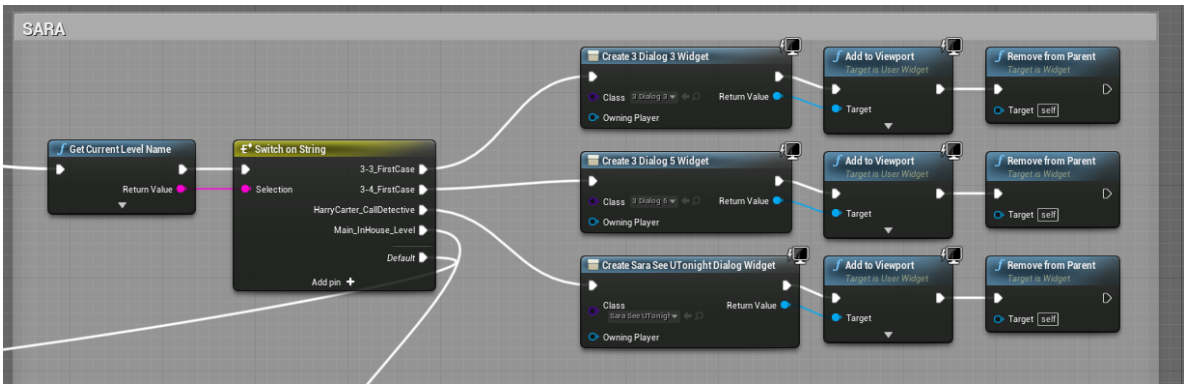


Ilustración 61: Funcionalidad de llamada 2 (Sara)

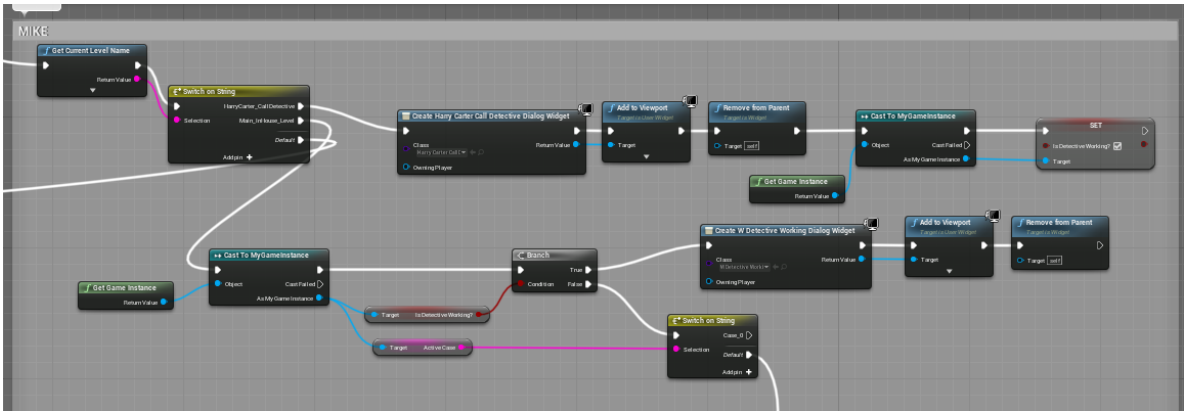


Ilustración 62: Funcionalidad de llamada 3 (Mike)

Como se ve, la complejidad de las acciones que se llevan a cabo depende del personaje.

En caso de que no exista un dialogo para ese personaje en ese momento concreto, los switch redirigirán por default y se mostrará un mensaje de “NO RESPONSE” en la pantalla del móvil.

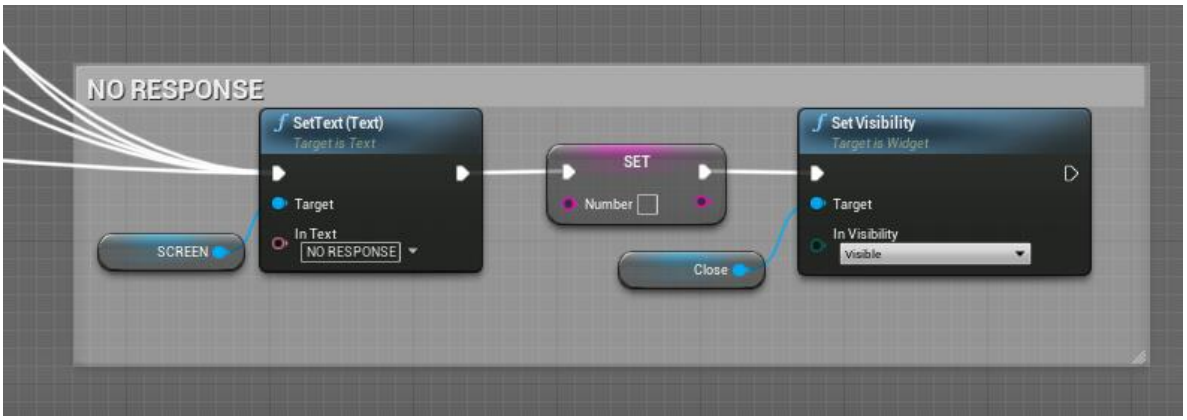


Ilustración 63: Funcionalidad de llamada 4

Dado que es inviable para el usuario recordar los números de todos los personajes se ha creado un sistema de contactos, cada vez que se conoce a un personaje relevante se añade su número a

contactos. Este sistema se accede desde el botón con dos rayas del móvil y su funcionalidad es la siguiente.

Existen dos botones extras en el móvil, subir y bajar, estos recorren la lista de contactos que se encuentra guardada en MyGameInstance dado que debe ser global a todos los niveles y guardarse entre partidas.

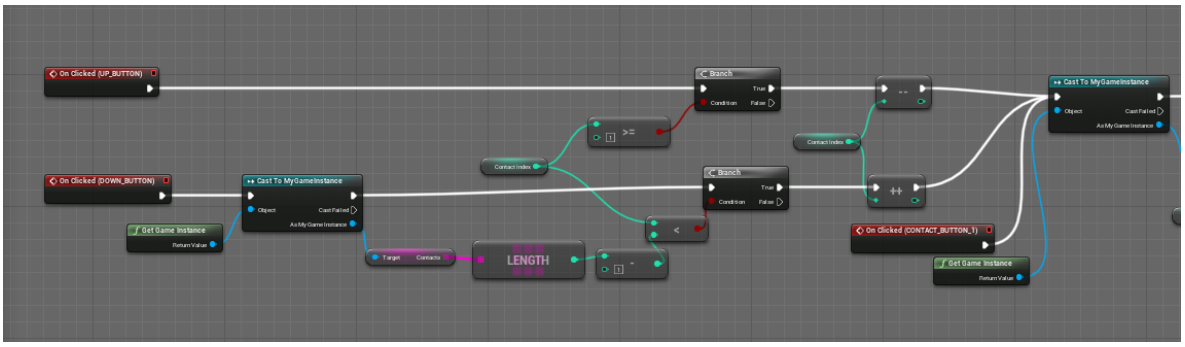


Ilustración 64: Funcionalidad de contactos 1

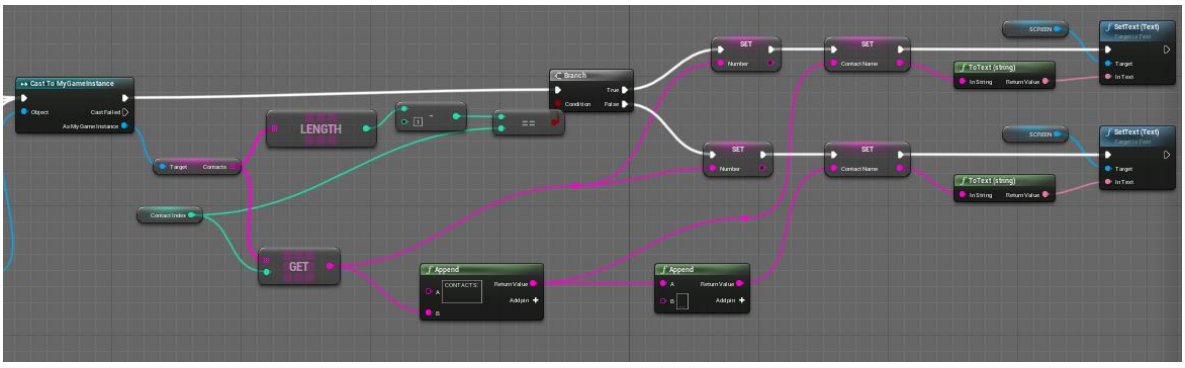


Ilustración 65: Funcionalidad de contactos 2

4.6.2.6 Glosario

El glosario es accesible desde el menú de pausa y desde el libro que se encuentra en la pantalla de investigación. Se trata de un libro con términos legales para informar al jugador. Básicamente tiene los botones necesarios para pasar página y para cerrar.

El botón de cerrar simplemente elimina la interfaz de la pantalla, mientras que los botones de pasar página comprueban la página actual y la siguiente o la anterior para cambiar la imagen (pagina) que se va a mostrar.

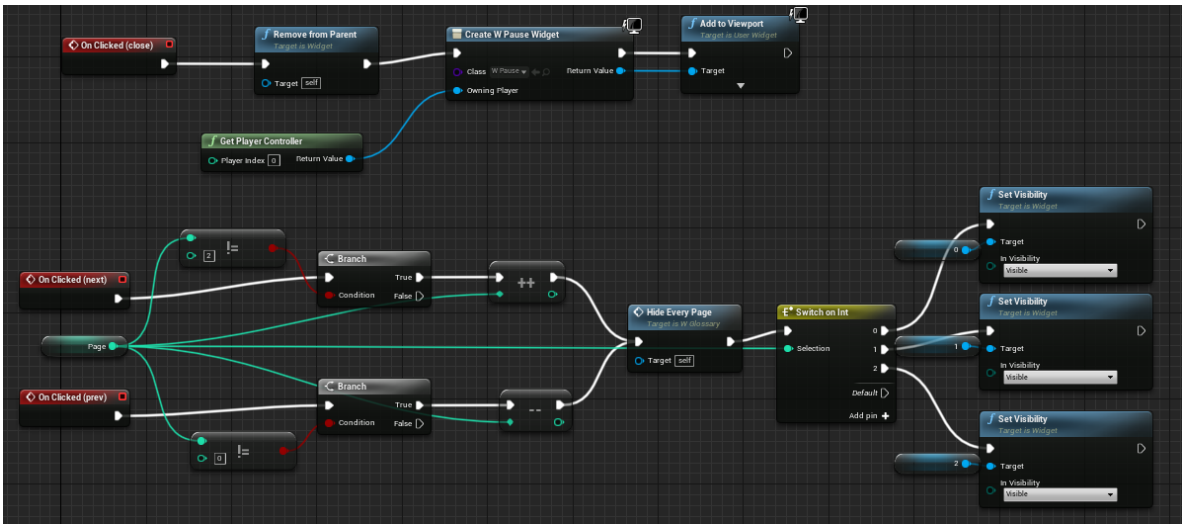


Ilustración 66: Funcionalidad del glosario

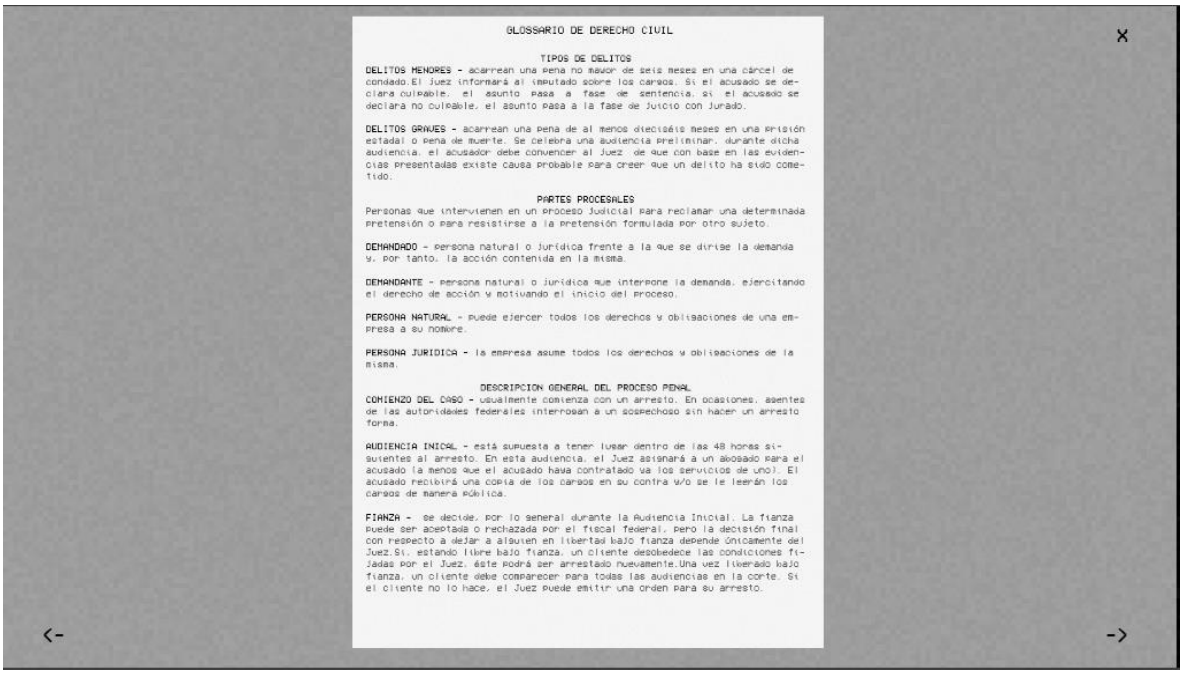


Ilustración 67: Glosario

4.6.2.7 Mapa

El mapa es accesible desde el nivel de investigación y nos permite recorrer las calles ficticias de Florida en busca de información.

Ilustración 68: Mapa

El mapa está compuesto de dos tipos de accesos:

- Callejero: listado de calles.
- Lugares importantes: marcados en el mapa con un punto de localización.

A través de cualquiera de los dos se puede acceder a otros lugares. La manera de acceder a ellos es haciendo click sobre ellos, para ello cada uno tiene asociado un *Onclick* que carga en nivel correspondiente. Dado que no todas las calles tienen información relevante existe un método que genera lugares proceduralmente para dar al usuario la sensación de viajar. De esta manera se genera un *RandomLevel*, con información irrelevante. En el caso de los lugares y calles con información importante existen niveles y diálogos programados completamente.

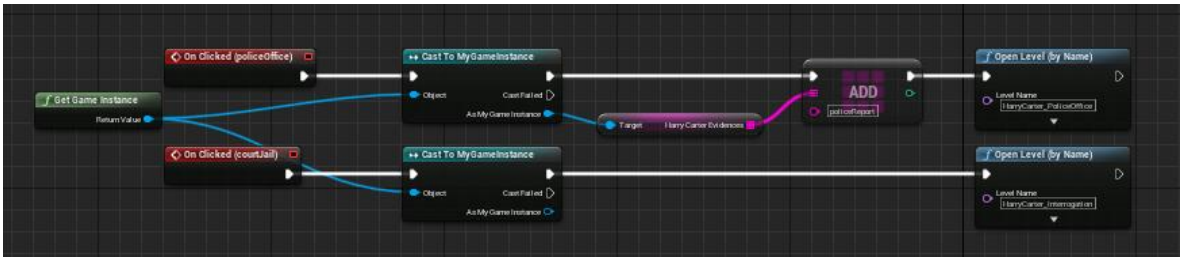


Ilustración 69: Cargando un nivel con información importante y desbloqueando una pista

Para las calles se ha generado un *DataSet* (explicado en la sección “Ordenador” de este mismo apartado) con más de mil calles. Para mostrarlas por pantalla se recorre dicho data set y se van añadiendo solamente las calles (dado que el *DataSet* contiene más datos que se explicarán posteriormente) a un array.

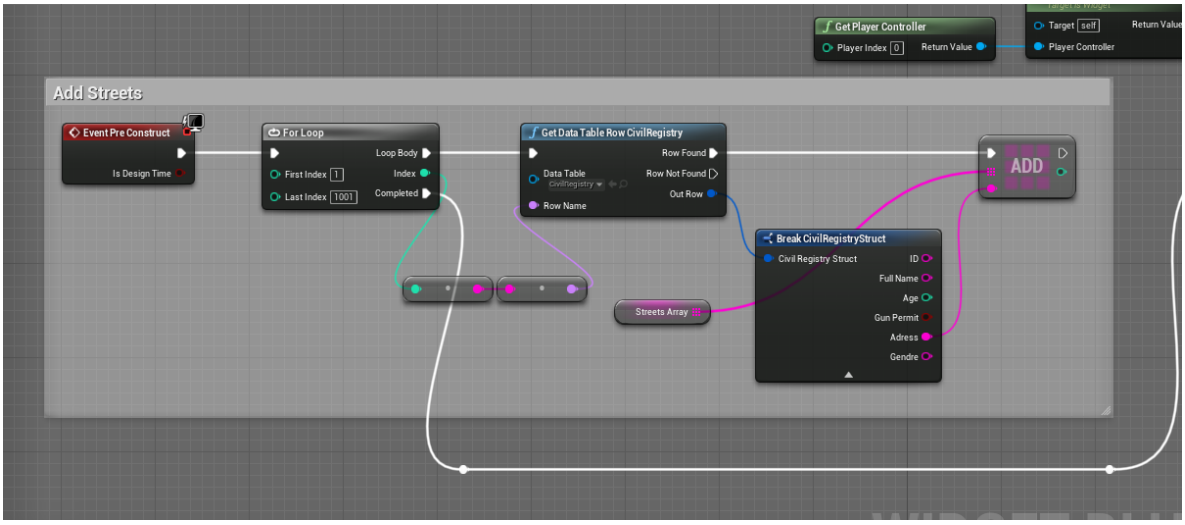


Ilustración 70: Se añade las calles del dataset al array

Posteriormente se controla el idioma y se crea un botón por cada elemento del array, es decir, un botón para cada calle del *DataSet*, que se va añadiendo a una *scrollBox* donde finalmente se mostraran todas las calles.

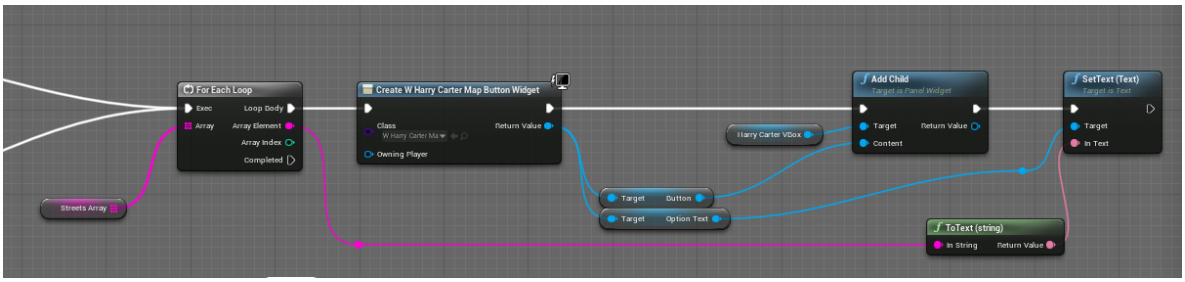


Ilustración 71: Se crea un botón por cada calle

Para que la búsqueda de las calles en el callejero sea más sencilla se ha programado un explorador para la *scrollBox* que permite buscar las calles tecleando su nombre.

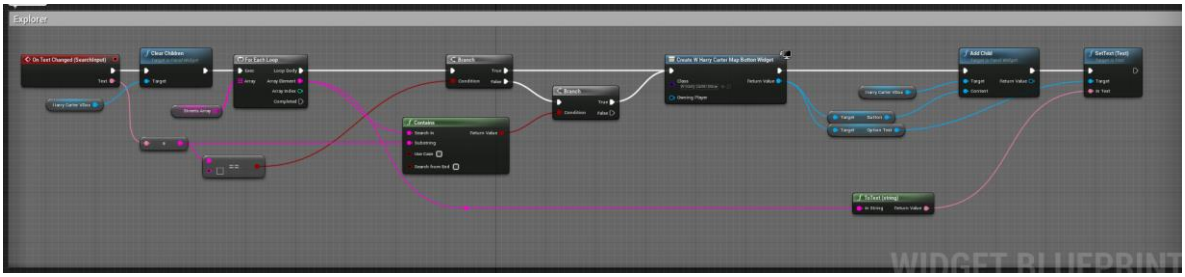


Ilustración 72: Explorador de calles

4.6.2.7.1 Botones del callejero

Los botones que se generan en la *scrollBox* tienen como única función cargar el nivel concreto en función del nombre de la calle. En caso de ser una calle sin información, cargar un *RandomLevel* generado proceduralmente. Como se ve en la siguiente ilustración las calles importantes añaden pistas para ser posteriormente utilizadas en la negociación o juicio.

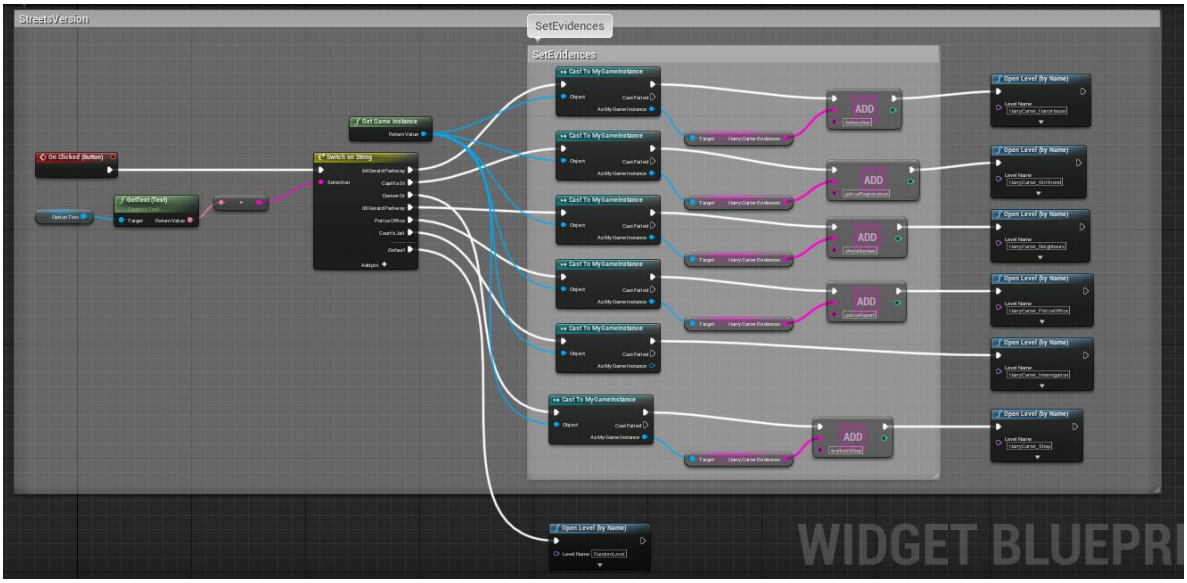


Ilustración 73: Carga de niveles en función de la calle

4.6.2.7.2 Generando un nivel procedural

Para la generación de este tipo de niveles existe un único nivel llamado *RandomLevel*, este añade un dialogo aleatorio y un fondo generado de forma procedural. Para ellos existieran varios *sprites* que se iran superponiendo para generar un nivel nuevo cada vez que se entre al *RandomLevel*. Para ello se coge un valor aleatorio de cada uno de los arráis de *sprites* y se muestra en pantalla.

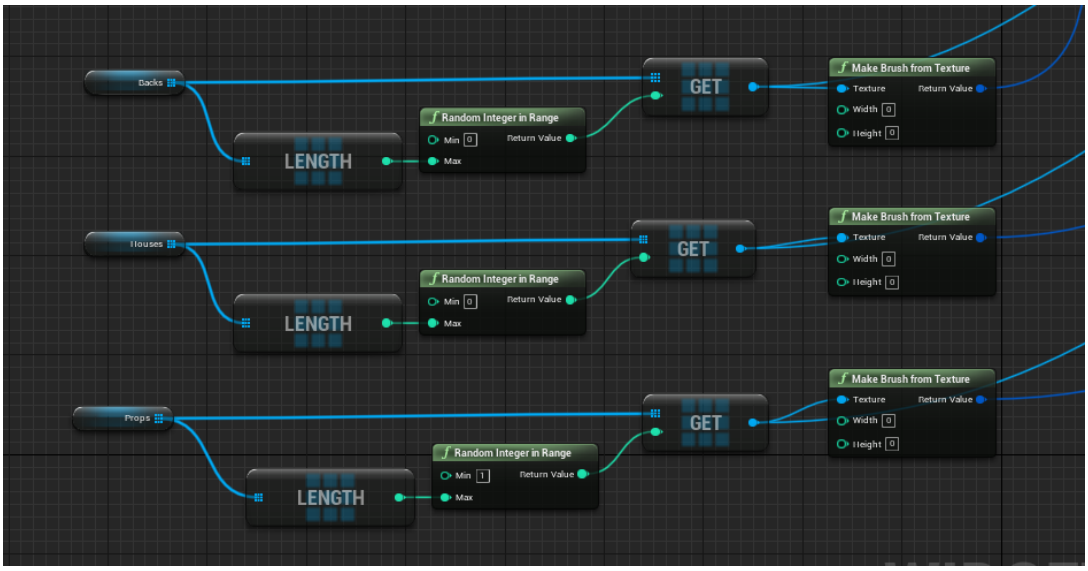


Ilustración 74: Seleccionando valor aleatorio

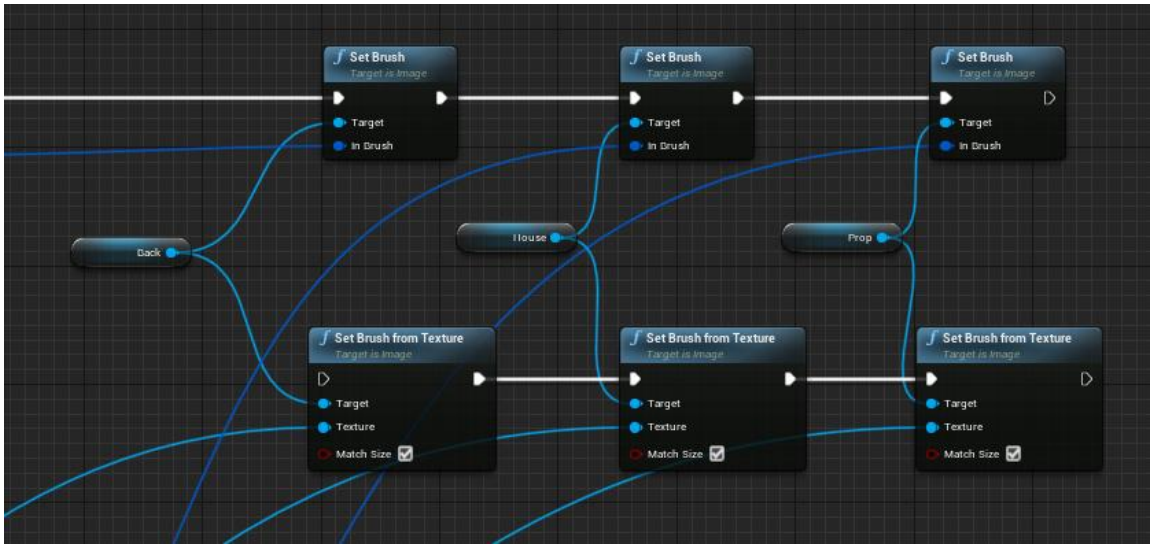


Ilustración 75: Mostrando los sprites elegidos

4.6.2.8 Ordenador

El ordenador es accesible desde el nivel de investigación y permite el acceso a dos bases de datos ficticias para buscar información para poder seguir la investigación. Al igual que todas las interfaces de usuario tienen una sección para el idioma y los típicos botones de cerrar, volver, etc.

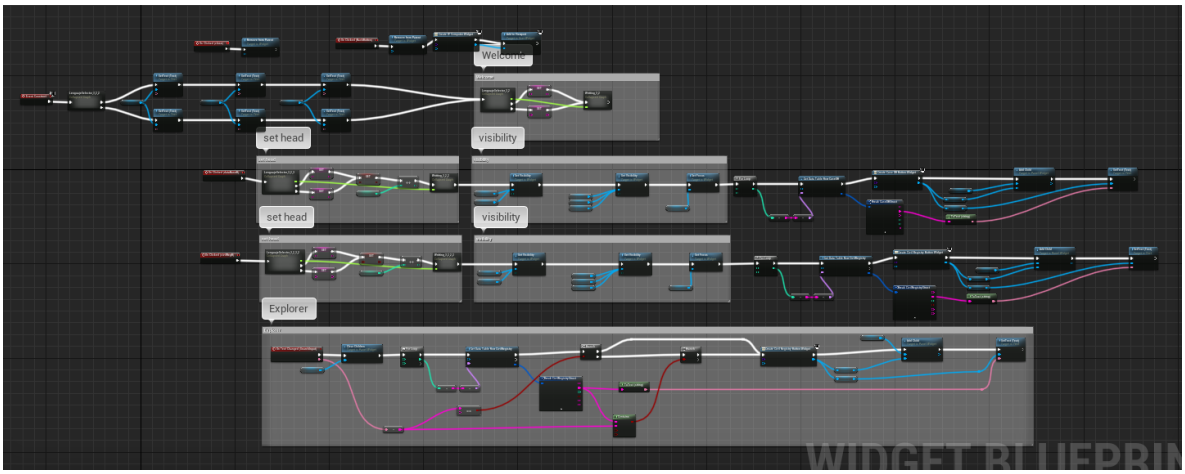


Ilustración 76: Vistazo general al ordenador

Para explicar mejor el funcionamiento del ordenador es necesario conocer cómo se han implementado las bases de datos.

4.6.2.8.1 Las bases de datos

Para estas se han generado *DataSets* (el segundo data set está en desarrollo) con la información que se requiere. Para el manejo de este tipo de estructuras Unreal Engine 4 nos proporciona dos tipos de Assets, *DataTables* y *Structures*.

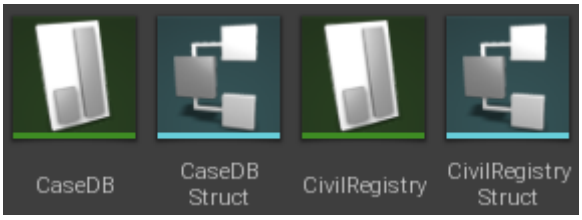


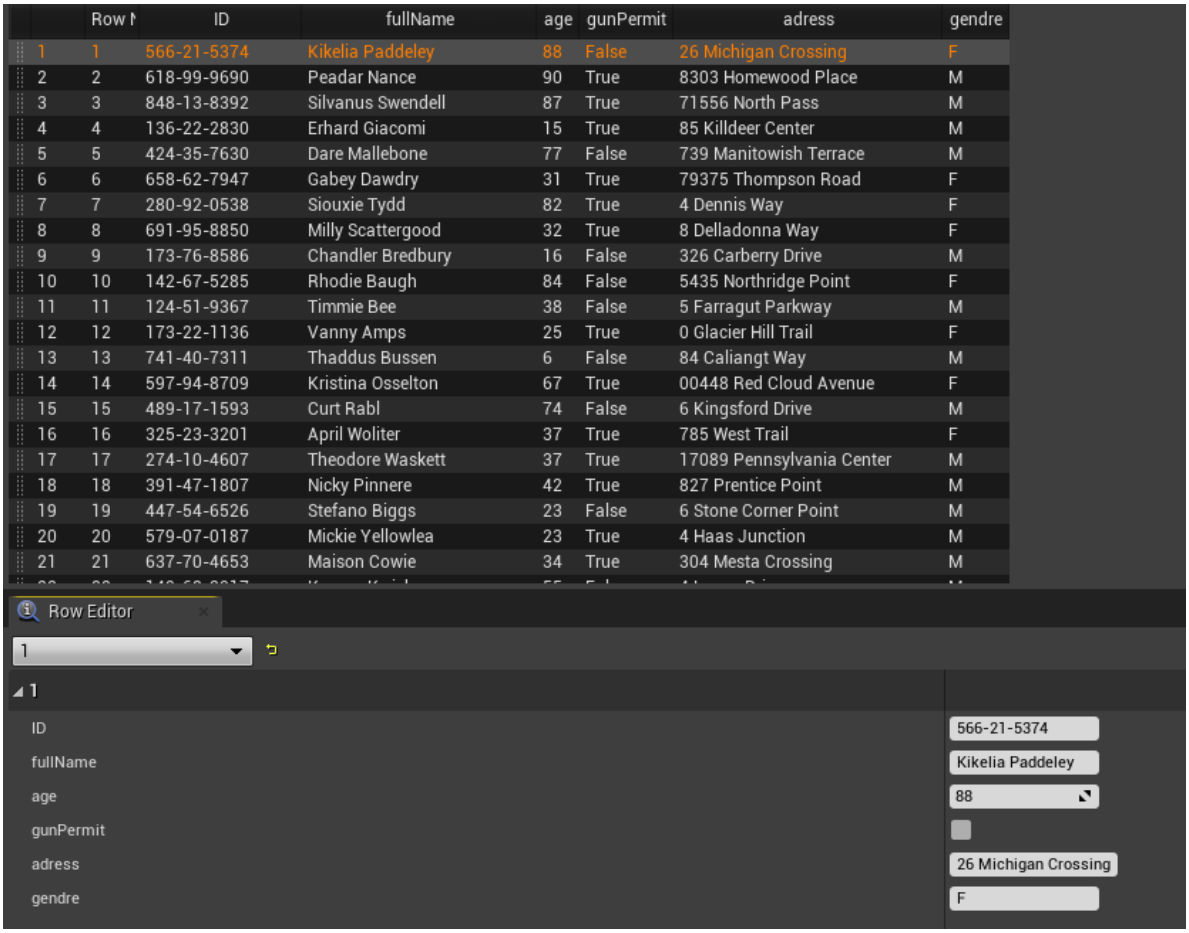
Ilustración 77: DataTable y Structure

Se han creado estructuras con los campos que tiene el data set para poder tratar la información de forma más cómoda. De esta forma solo es necesario tener un tipo de dato para cada entrada de la *dataTable*. Dado que el funcionamiento de ambos *DataSets* y sus respectivas tablas y estructuras es idéntico se procederá con la explicación de uno de ellos.

A continuación una imagen de la estructura que se usará para el registro civil. Como se ve consta de seis campos cada uno con su tipo correspondiente.

Ilustración 78: Estructura del registro civil

Y a continuación podemos ver el *DataSet* importado en una *DataTable*.



Row N	ID	fullName	age	gunPermit	adress	gendre
1	566-21-5374	Kikelia Paddeley	88	False	26 Michigan Crossing	F
2	618-99-9690	Peadar Nance	90	True	8303 Homewood Place	M
3	848-13-8392	Silvanus Swendell	87	True	71556 North Pass	M
4	136-22-2830	Erhard Giacomi	15	True	85 Killdeer Center	M
5	424-35-7630	Dare Mallebone	77	False	739 Manitowish Terrace	M
6	658-62-7947	Gabey Dawdry	31	True	79375 Thompson Road	F
7	280-92-0538	Siouxie Tydd	82	True	4 Dennis Way	F
8	691-95-8850	Milly Scattergood	32	True	8 Delladonna Way	F
9	173-76-8586	Chandler Bredbury	16	False	326 Carberry Drive	M
10	142-67-5285	Rhodie Baugh	84	False	5435 Northridge Point	F
11	124-51-9367	Timmie Bee	38	False	5 Farragut Parkway	M
12	173-22-1136	Vanny Amps	25	True	0 Glacier Hill Trail	F
13	741-40-7311	Thaddus Bussen	6	False	84 Caliangt Way	M
14	597-94-8709	Kristina Osselton	67	True	00448 Red Cloud Avenue	F
15	489-17-1593	Curt Rabl	74	False	6 Kingsford Drive	M
16	325-23-3201	April Woliter	37	True	785 West Trail	F
17	274-10-4607	Theodore Waskett	37	True	17089 Pennsylvania Center	M
18	391-47-1807	Nicky Pinnere	42	True	827 Prentice Point	M
19	447-54-6526	Stefano Biggs	23	False	6 Stone Corner Point	M
20	579-07-0187	Mickie Yellowlea	23	True	4 Haas Junction	M
21	637-70-4653	Maison Cowie	34	True	304 Mesta Crossing	M

Ilustración 79: DataTable del registro civil

Para poder importar el *DataSet* se hace uso de un fichero tipo JSON que contiene toda la información.


```
[
  {
    "Name": 1,
    "ID": "566-21-5374",
    "fullName": "Kikelia Paddeley",
    "age": 88,
    "gunPermit": false,
    "adress": "26 Michigan Crossing",
    "gendre": "F"
  },
  {
    "Name": 2,
    "ID": "618-99-9690",
    "fullName": "Peadar Nance",
    "age": 90,
    "gunPermit": true,
    "adress": "8303 Homewood Place",
    "gendre": "M"
  },
  {
```


Ilustración 80: Fichero JSON

4.6.2.8.2 Registro civil

Haciendo uso de todo lo mencionado anteriormente se puede mostrar toda la información a través de la pantalla del ordenador ficticio haciendo click en el botón de "Registro civil". Al hacer click en este se accede a un listado completo de las personas que contiene el *DataSet* y haciendo click en cada una de ella se puede acceder a toda su información formateada para ser legible.

Ilustración 81: Listado del registro civil

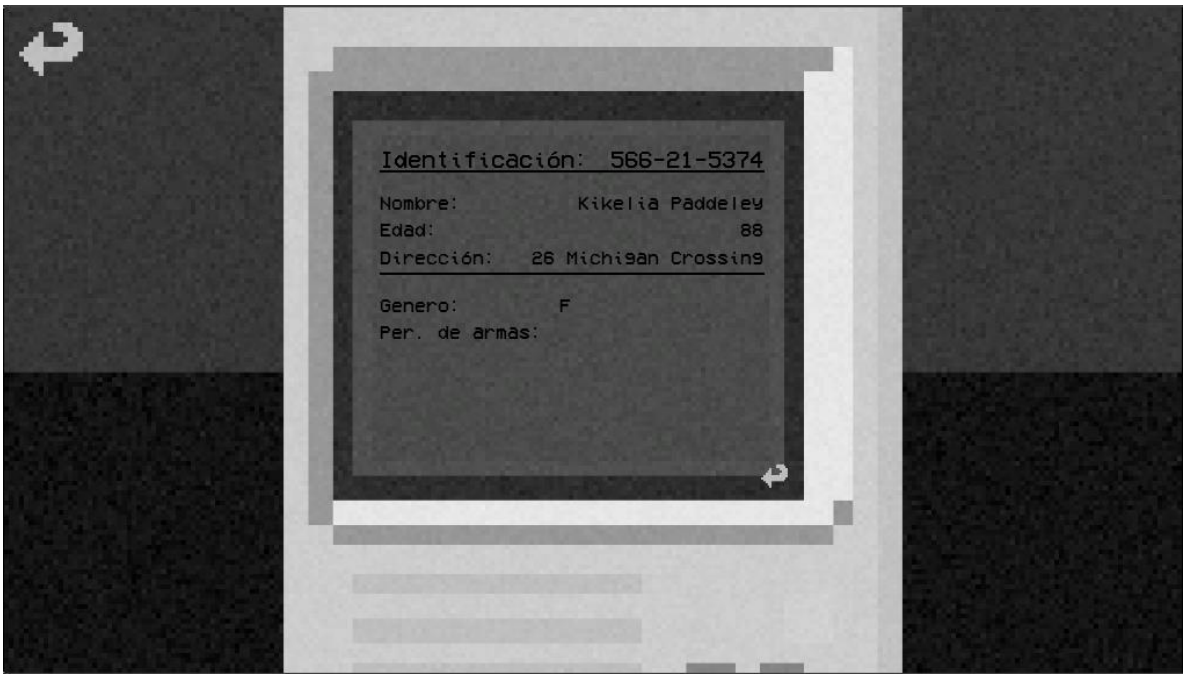


Ilustración 82: Salida para una persona concreta

Este sistema funciona igual que el del callejero usado en el mapa. Se recorre la *DataTable* obteniendo el nombre de cada entrada para crear un botón y añadirlo a la *scrollBox*. Para posteriormente en el código del botón añadir la funcionalidad concreta. A demás se ha generado el mismo explorador que en el mapa para facilitar la búsqueda en la base de datos.

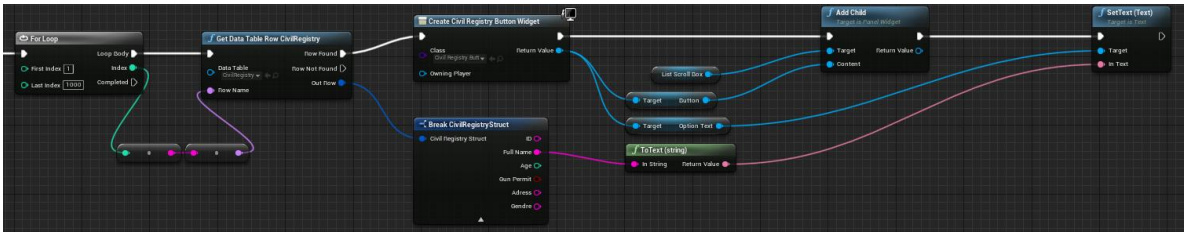


Ilustración 83: Funcionamiento del registro civil

En este caso el funcionamiento de los botones es algo más complejo que en el mapa dado que será necesario obtener toda la información de la entrada de la tabla y formatearla. En primer lugar se busca en toda la tabla el nombre en el cual se ha hecho click (*OptionText*) y cuando se encuentra se guardan todos los datos de esta entrada en variables.

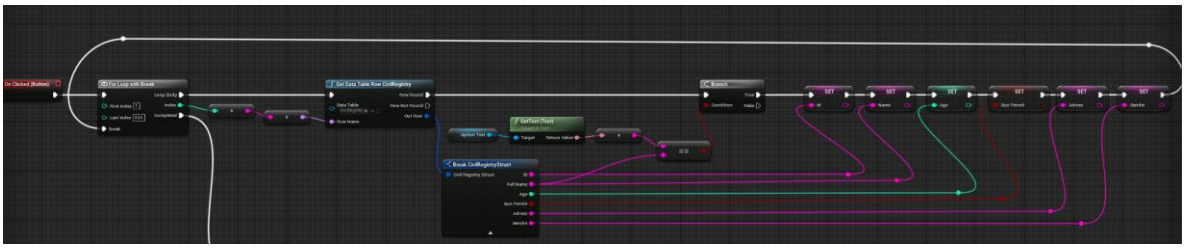


Ilustración 84: Buscamos y guardamos la entrada

Una vez se tienen los datos solo sería necesario hacer visibles todos los campos (*labels* de texto) que se van a mostrar. Y añadir a estas la información que acabamos de recoger. Para ellos se hace uso de los nodos *SetVisibility* sobre los componentes que se van a mostrar, se pone el idioma que está configurado con *SetText* y por último se rellenan los campos con la información recogida, de nuevo, con *setText*.

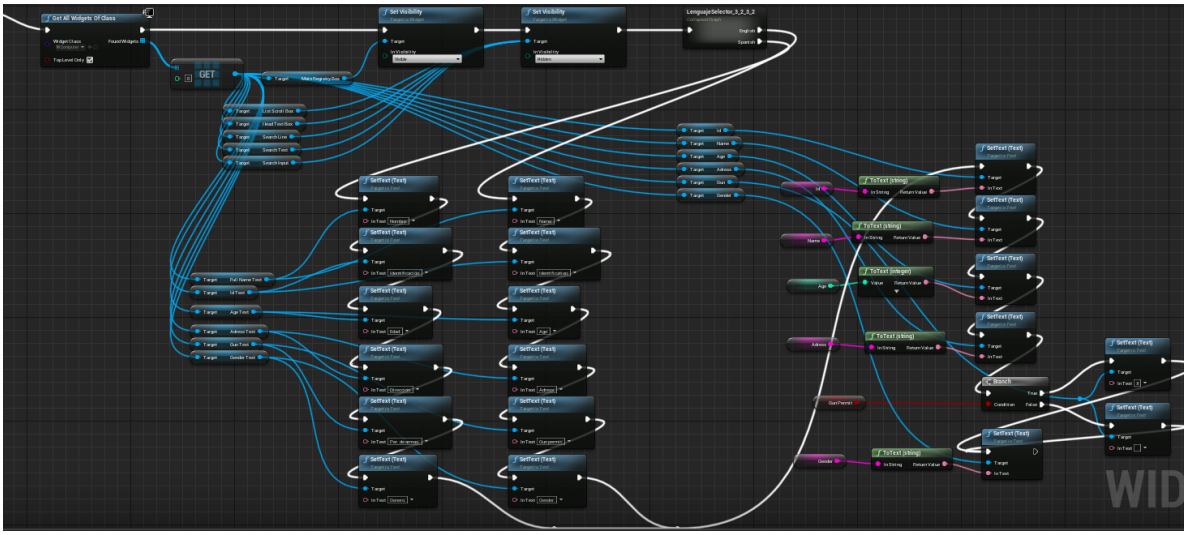


Ilustración 85: Rellenando los campos con los datos

4.6.3 Diálogos

Los diálogos son una característica critica del proyecto dada la gran importancia que tienen en el desarrollo de la historia. Por ello ha sido muy importante desarrollar un sistema de diálogos completo e intuitivo que permita al usuario disfrutar del juego. El sistema de diálogos tiene varias características destacables, como la posibilidad de saltar los diálogos en cualquier momento y la toma de decisiones para desviar la conversación al gusto del jugador.

4.6.3.1 Sistema de representación de diálogos

Para la representación de los diálogos se han utilizado los típicos bocadillos /globos de dialogo con el nombre y una imagen del personaje que habla.

Ilustración 86: Ejemplo de globo de dialogo

Para generar una sensación más característica de los videojuegos y en especial de las aventuras más narrativas se ha decidido crear un sistema por el que las letras se escriben de una en una simulando el tecleo de un texto escrito a máquina. Para esto existe una función “Writting” encargada de realizar esta tarea.

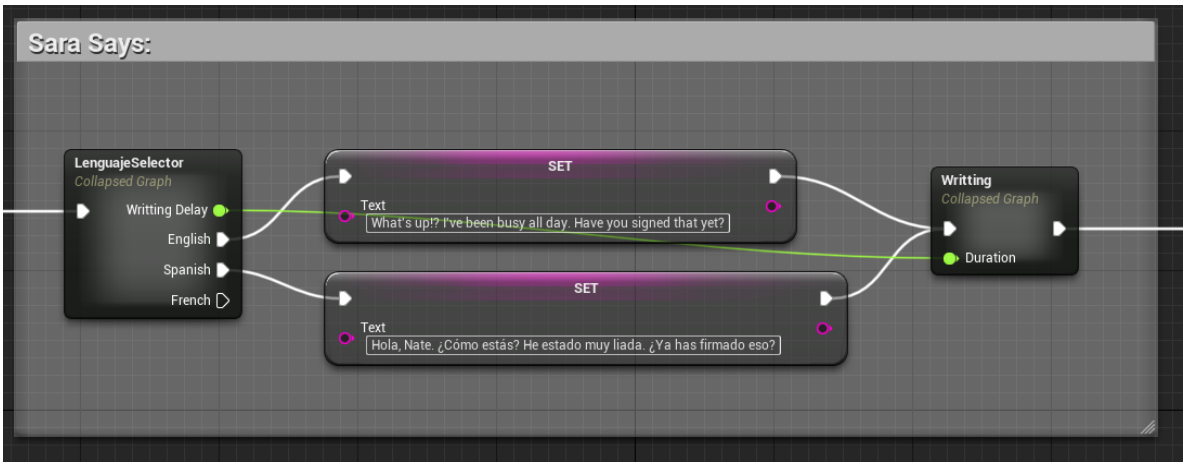


Ilustración 87: Esquema básico de dialogo

Para cada dialogo, antes de empezar a escribirse se selecciona el idioma, lo que permite cambiar el idioma en cualquier momento del juego sin tener que ir al menú principal y recargar la partida. Una vez seleccionado el texto que se va a escribir, se procede a escribirlo en “Writing”.

Antes de explicar el funcionamiento de los diálogos es necesario dejar claro las variables que están involucradas en este proceso. Las funciones involucradas se explicarán a lo largo del apartado.

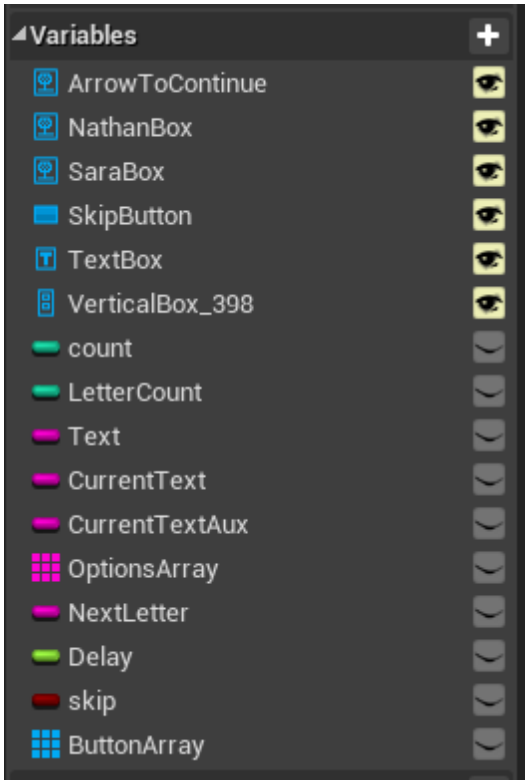


Ilustración 88: Variables necesarias para los diálogos

- *ArrowToContinue*: se trata de un Sprite de una pequeña flecha que se dibuja abajo a la derecha cuando el dialogo a concluido y se puede pasar al siguiente.
- *xxxxxxBox*: son los Sprites con el globo de dialogo de cada personaje.
- *SkipButton*: es un botón invisible que ocupa toda la pantalla, sirve para saltar el efecto de tecleo y mostrar el dialogo completo sin tener que esperar.
- *VerticalBox*: se trata de la caja donde se colocarán los botones de las opciones de dialogo si las hubiera.
- *Count*: es un contador que cuenta los click que se han hecho sobre el *SkipButton* para saber cuándo cambiar de dialogo exactamente.
- *LetterCount*: va contando las letras del dialogo que se va escribiendo.
- *Text*: se trata del propio texto completo que se va a escribir.
- *CurrentText*: se trata del texto que está escrito en cada momento.

- CurrentTextAux: sirve para tener guardado el texto con el siguiente carácter que se va a escribir.
- OptionsText: contienen las opciones de dialogo.
- *NextLetter*: contiene la siguiente letra que se va a escribir.
- *Delay*: se trata del tiempo que tardará en escribirse cada letra.
- *Skip*: es una variable de tipo *Boolean* para comprobar si el usuario ha saltado el dialogo.
- *ButtonArray*: el array de botones que se crea con las opciones de dialogo que se desean.

En un primer vistazo al sistema de diálogos podemos ver dos funciones que hacen referencia al sonido de cada letra, “PlayTypingSound” hace sonar un audio “mudo” para limpiar el audio antes de empezar a escribir y “PlayOneType” reproduce un “beep” por cada letra que se escribe. También podemos ver la función “ClearVariables”, ejecutada al final del todo, que como su nombre indica restaura el valor de todas las variables para el próximo diálogo.

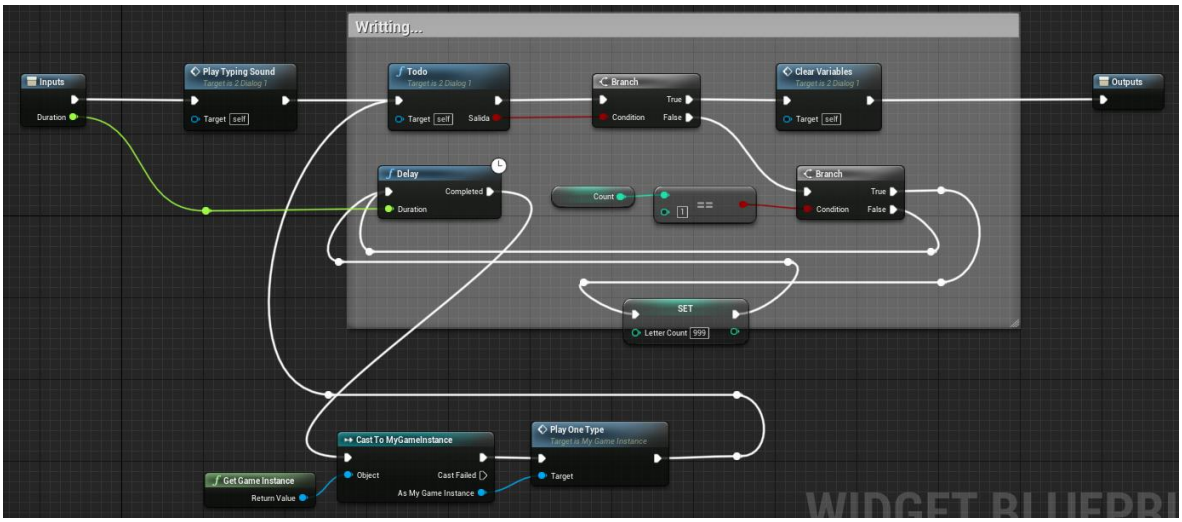


Ilustración 89: Contenido de "Writting"

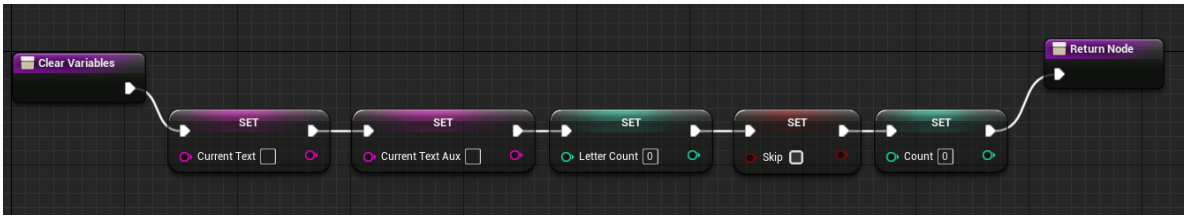


Ilustración 90: ClearVariables

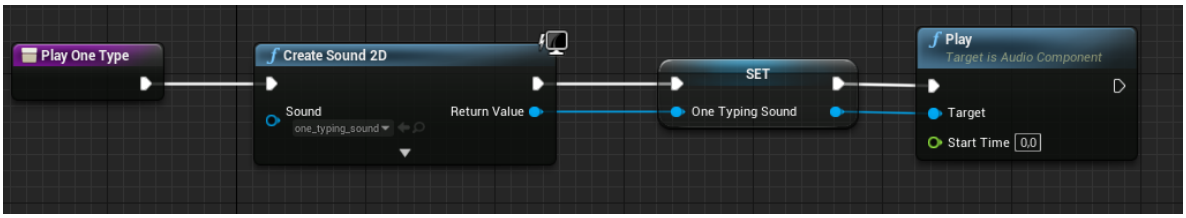


Ilustración 91: PlayOneType

Después de limpiar el sonido, como se ve en la ilustración 74, se pasa a la función "Todo". Esta es la encargada de la escritura en sí misma, mientras que la anterior mantiene el juego en un bucle hasta que se pase al siguiente diálogo.

Lo primero que se hace en la función "Todo" es esconder la flecha de continuar dado que aún no se puede pasar al siguiente dialogo. Y posteriormente empieza a escribir en la función "WriteWithDelay".

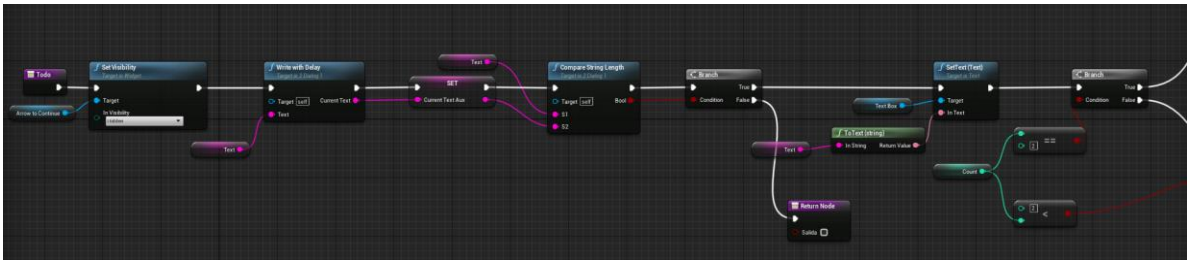


Ilustración 92: Todo 1

En "WriteWithDelay" se empieza creando un array de caracteres del texto que se va a escribir y se comprueba que su longitud sea mayor que el contador de letras, si no es así la función retorna. En caso positivo obtenemos la letra correspondiente al contador y la guardamos en "NextLetter", casteamos a MyGameInstance, dado que de ahí se obtiene el *delay* de cada letra y en función del carácter que se va a escribir se le aplica un *delay* u otro, de esta manera se puede conseguir que los puntos u otros caracteres tengan una pausa mayor. Después de *settear* el *delay* se añade a *currentText* la letra que se va a escribir y este es el texto que se añade a la vista. Por supuesto se aumenta el contador de las letras para poder escribir la siguiente.

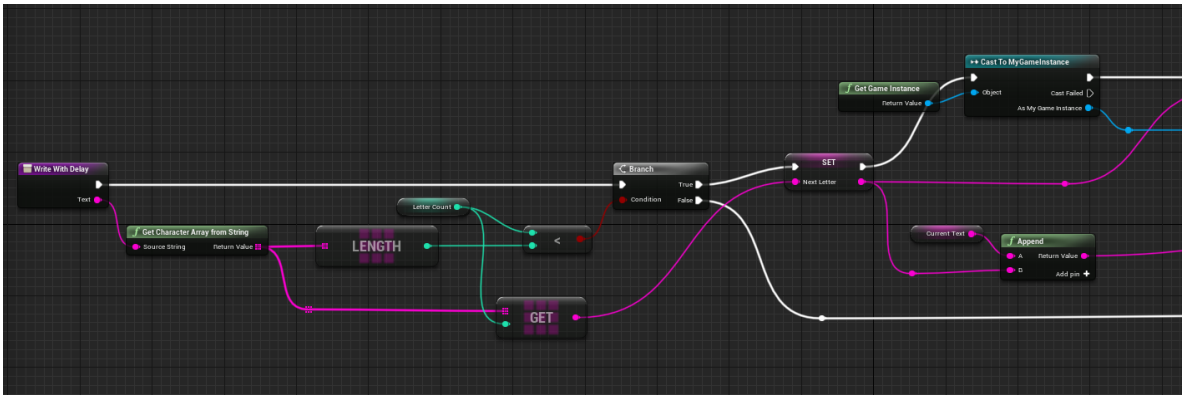


Ilustración 93: WriteWithDelay 1

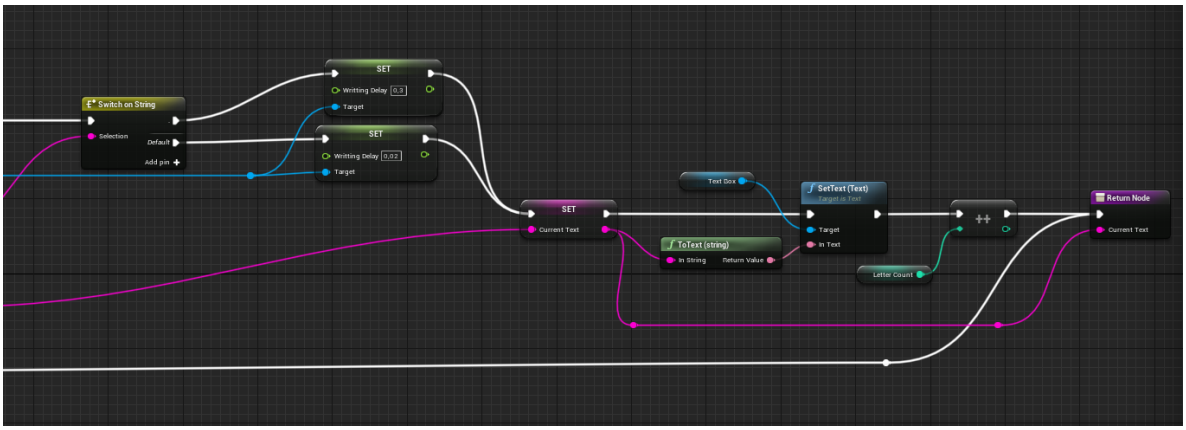


Ilustración 94: WriteWithDelay 2

Al salir de “WriteWithDelay”, volviendo a “Todo”, obtenemos el texto que llevamos escrito y el texto completo, de esta forma podemos comprobar si las longitudes coinciden en “CompareStringLength”.

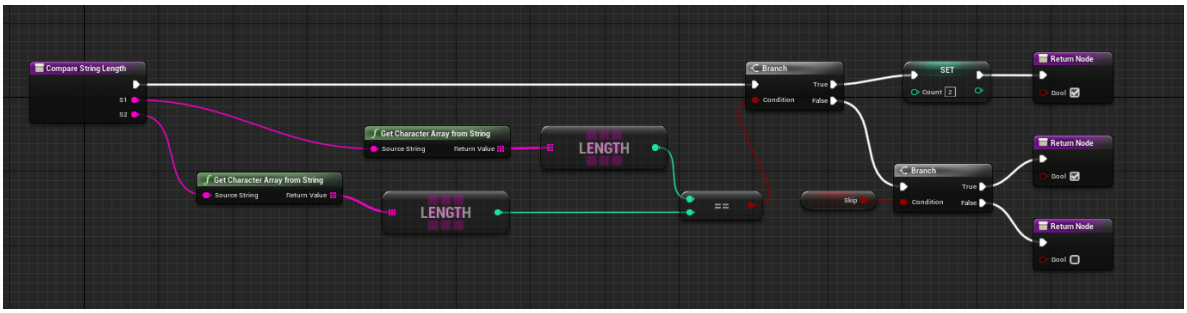


Ilustración 95: CompareStringLength

Si estas no coinciden devolvemos false en “Todo”, si no continuamos la ejecución, pintando por pantalla el texto completo. En este momento hay que comprobar cuantas veces se ha pulsado el botón de *Skip* y en función de ellas pasar al siguiente dialogo o esperar a la siguiente pulsación. Para ello “Todo” devuelve un valor booleano, si este es True quiere decir que hemos acabado, limpiamos variables y pasamos al siguiente dialogo. Si el valor es False hay que mantenerse en el bucle hasta que el usuario pulse de nuevo el botón *Skip*.

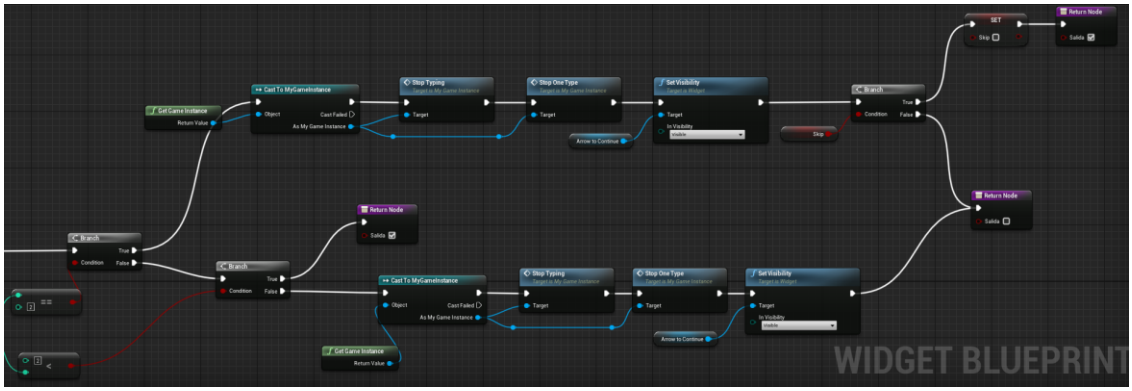


Ilustración 96: Todo 2

Como se ve en la ilustración 81, dependiendo del valor de *count* (veces que se pulsa el botón *skip*) se mostrará o no la flecha de continuar y se dejará de reproducir el sonido del tecleo, con *SetVisibility* y *StopTyping*/*StopOneType* respectivamente.

En el grafo general del dialogo existe un *OnClick* para el botón de *Skip* que indica que el usuario ha pasado el dialogo con un *Boolean* y aumenta el contador *count*.

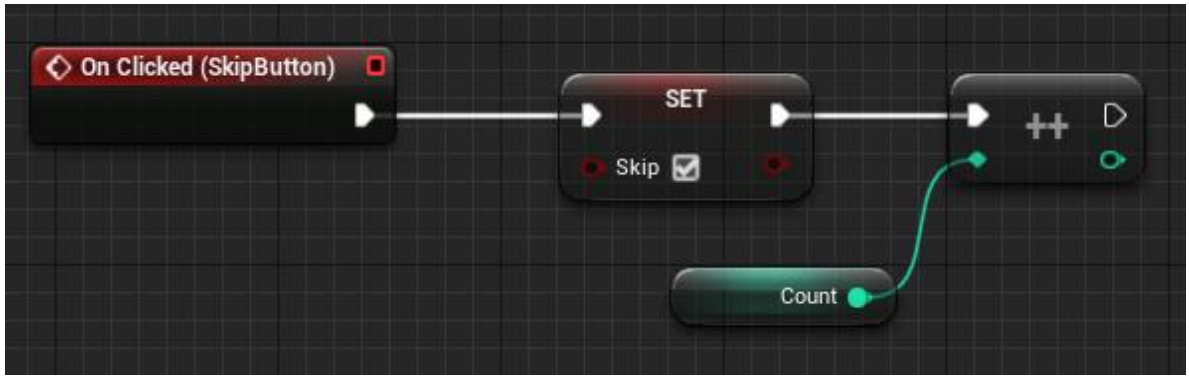


Ilustración 97: OnClick SkipButton

Resumen de funcionalidad de “Skip”:

- Un click antes de acabar de escribir el texto supondría saltar el tecleo, escribir el texto completo de golpe, mostrar la flecha de continuar y quedarse esperando a un segundo click para saltar al siguiente dialogo.
- Si se acaba de escribir el texto y no se ha hecho click, se muestra la flecha de continuar y se espera a un único click para saltar el dialogo.

El paso de un dialogo a otro es tan simple como usar la función “ChangeTalkingBox” para cambiar los Sprites de los bocadillos de un personaje a otro, seleccionar el idioma, indicar el texto y volver a llamar a “Writting”.

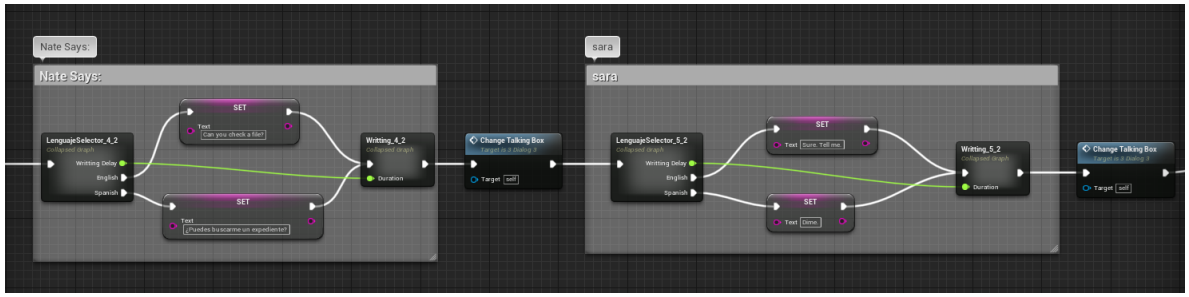


Ilustración 98: Cambio de diálogos

4.6.3.2 Sistema de toma de decisiones

Para el sistema de decisiones primero se escribe un texto que indique la decisión a tomar, esto se realiza al igual que los diálogos normales. Y posteriormente se crea el array de opciones para el idioma seleccionado.

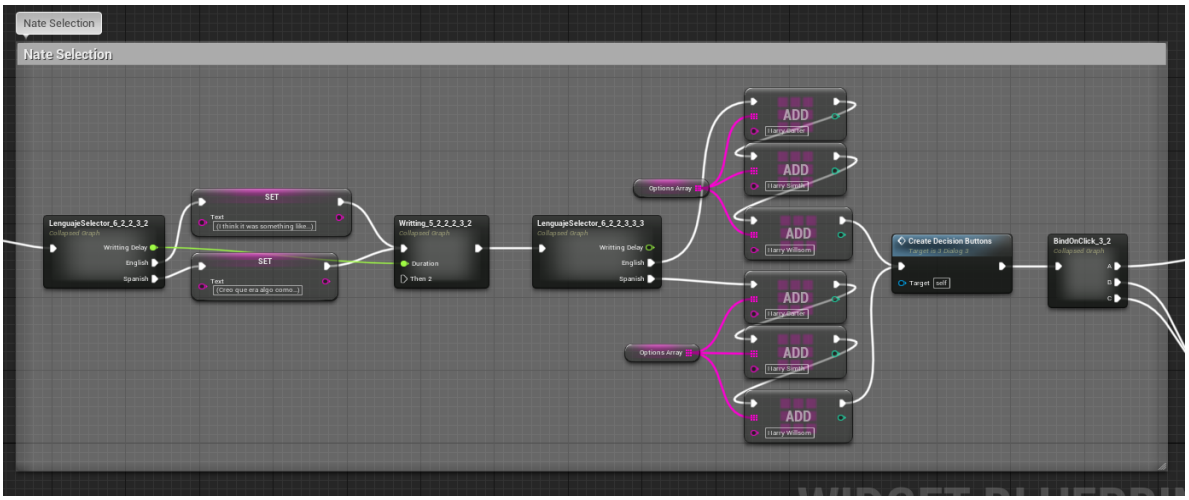


Ilustración 99: Generando las opciones de decisión

La función “CreateDecisionButton” es la encargada de convertir el array de *string* en un array de botones y añadirlo a la *VerticalBox*. Para ello recorre el array de *string* creando un *W_DialogSelectionButton*, dándole los valores necesarios, como el texto que va a contener, y por último añadiéndolo a la *VerticalBox* y al *ButtonArray*. Al completar de recorrer el array y crear los botones se debe esperar a la selección del usuario por lo tanto se debe bloquear el botón de *skip* y no mostrar la flecha de continuar. También se vacía el *OptionsArray* por si existieran más decisiones futuras y se tuviera que volver a rellenar.

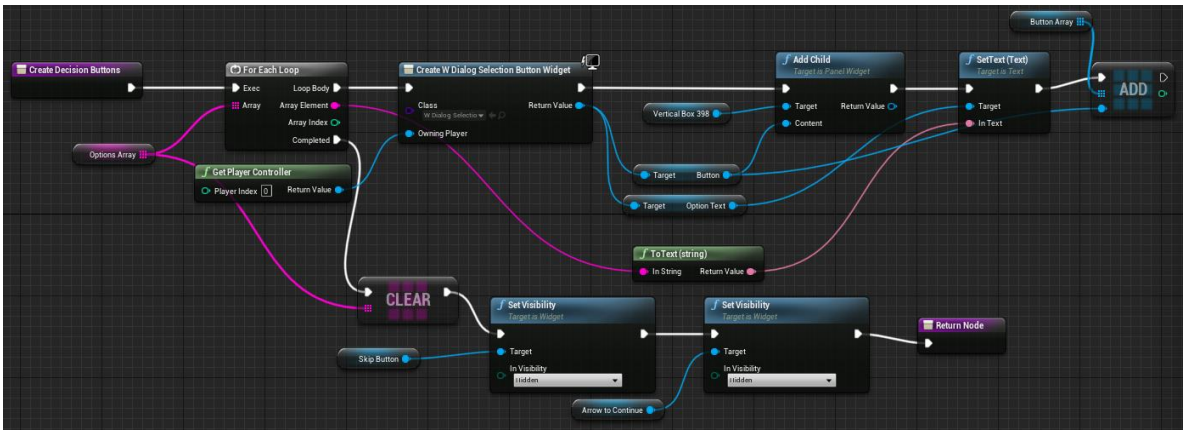


Ilustración 100: CreateDecisionButton

Una vez creados los botones se deben *bindear* correctamente para que cada uno realice la opción adecuada. Estos botones se podrían programar manualmente y dejar sus funciones *hardcodeadas* pero se ha preferido crear un meto más extensible que se pueda reutilizar a lo largo de todo el proyecto. Para ello se crea el “BindOnClick” que se ve en la ilustración 84. De esta manera se genera un bloque de código que nos devuelve la ejecución que debe ejecutarse en función del botón que se pulse. De esta forma queda mucho más claro a la hora de volver a utilizar este sistema de toma de decisiones.

Para conseguirlo se hace uso del bloque de código “BindOnClick”, donde se asigna un evento por cada uno de los botones que hemos añadido a *ButtonArray*, todos los eventos recorren dicho array para vaciar de botones la *VerticalBox* y al terminar cada uno toma un camino distinto, que determinará que se ejecuta a continuación.

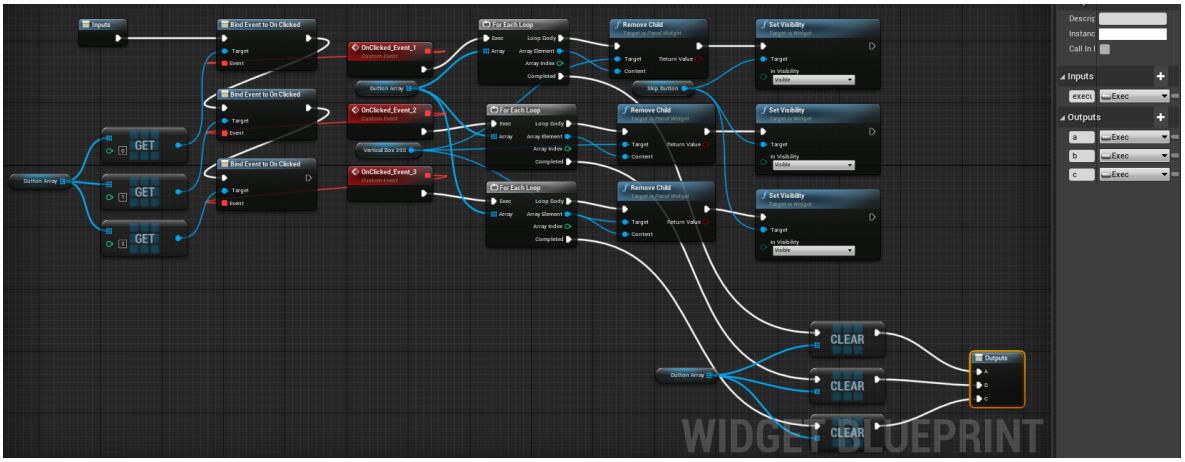


Ilustración 101: BindOnClick

Como se ve en la ilustración anterior primero se *bindea* con un evento, por lo que existen tres eventos (en rojo), uno para cada botón. Posteriormente se vacían la *VerticalBox* y el array de botones y en el nodo de output podemos ver tres salidas de tipo *exec* que son las tres salidas que dirigen a las distintas ejecuciones (en este caso B y C dirigen a la misma ejecución).

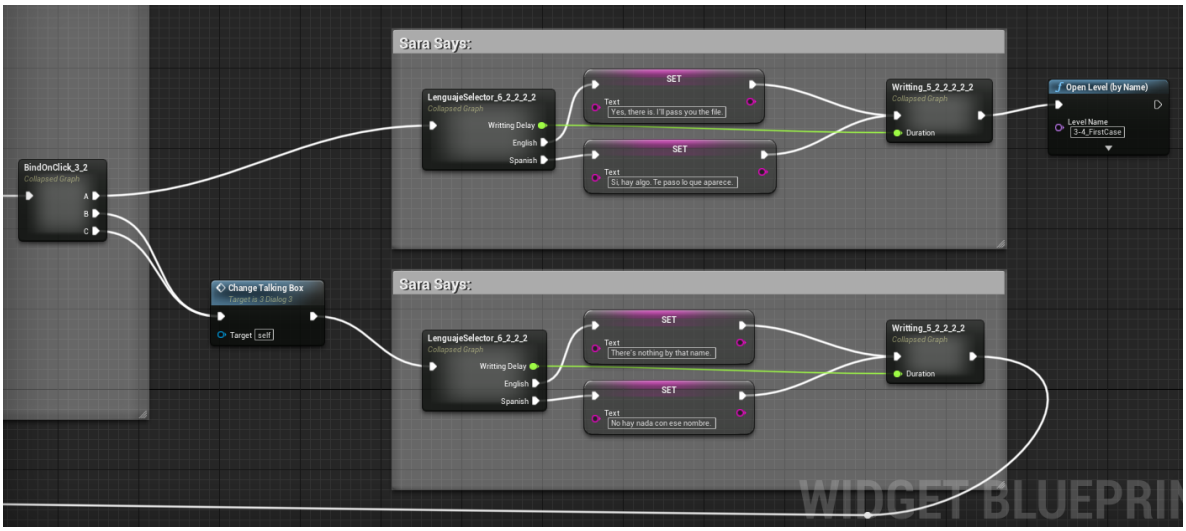
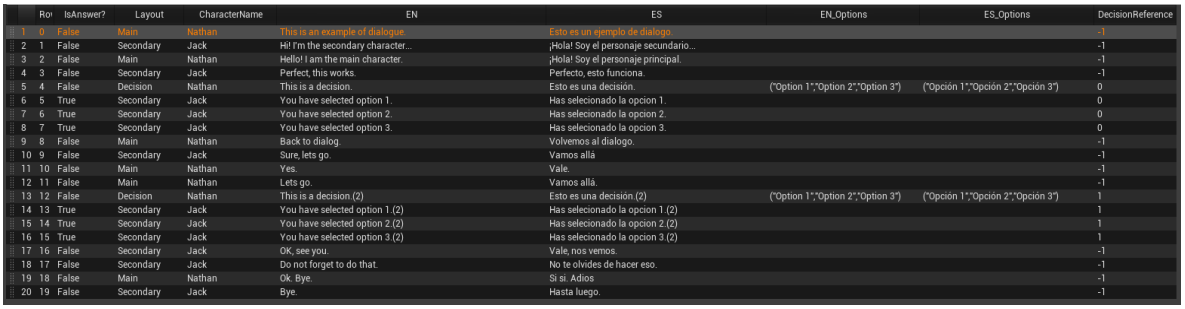


Ilustración 102: Salidas a distintas ejecuciones de BindOnClick

4.6.3.3 Sistema de diálogos actualizado

Después de acabar el sistema de diálogos anteriormente explicado se observó gran potencial en él. Por lo que se creó una mejora sustancial del mismo, la cual permitía gestionar el dialogo completo a través de una tabla. Esta tabla permite de forma muy sencilla configurar los nombres de los personajes, el layout que se desea para cada personaje, si se trata de una selección de

opciones o un dialogo normal, si es la primera permite añadir las opciones que aparezcan, etc. Por supuesto es configurable para todos los idiomas que se desee. Además, permite modificar la interfaz de los diálogos de forma muy sencilla a través del editor de interfaces de UE4.



Ro	IsAnswer?	Layout	CharacterName	EN	ES	EN_Options	ES_Options	DecisionReference
0	False	Main	Nathan	This is an example of dialogue.	Esto es un ejemplo de dialogo.			-1
1	False	Secondary	Jack	Hi! I'm the secondary character...	¡Hola! Soy el personaje secundario...			-1
2	False	Main	Nathan	Hello! I am the main character.	¡Hola! Soy el personaje principal.			-1
3	False	Secondary	Jack	Perfect, this works.	Perfecto, esto funciona.			-1
4	False	Decision	Nathan	This is a decision.	Esto es una decisión.	("Option 1","Option 2","Option 3")	("Opción 1","Opción 2","Opción 3")	0
5	True	Secondary	Jack	You have selected option 1.	Has selecionado la opcion 1.			0
6	True	Secondary	Jack	You have selected option 2.	Has selecionado la opcion 2.			0
7	True	Secondary	Jack	You have selected option 3.	Has selecionado la opcion 3.			0
8	False	Main	Nathan	Back to dialog.	Volvemos al dialogo.			-1
9	False	Secondary	Jack	Sure, lets go.	Vamos allá			-1
10	False	Main	Nathan	Yes.	Vale.			-1
11	False	Main	Nathan	Lets go.	Vamos allá.			-1
12	False	Decision	Nathan	This is a decision.(2)	Esto es una decisión.(2)	("Option 1","Option 2","Option 3")	("Opción 1","Opción 2","Opción 3")	1
13	True	Secondary	Jack	You have selected option 1.(2)	Has selecionado la opcion 1.(2)			1
14	True	Secondary	Jack	You have selected option 2.(2)	Has selecionado la opcion 2.(2)			1
15	True	Secondary	Jack	You have selected option 3.(2)	Has selecionado la opcion 3.(2)			1
16	False	Secondary	Jack	OK, see you.	Vale, nos vemos.			-1
17	False	Secondary	Jack	Do not forget to do that.	No te olvides de hacer eso.			-1
18	False	Main	Nathan	Ok. Bye.	Si si. Adios			-1
19	False	Secondary	Jack	Bye.	Hasta luego.			-1

Ilustración 103: Tabla de gestión de diálogos

Con algunos retoques de UI y optimización, y una exhaustiva documentación del funcionamiento del sistema se creó un proyecto paralelo para ofrecer el sistema en la UNREAL MARKETPLACE, la tienda de assets de Unreal y Epic. Se crearon varias interfaces distintas a las del videojuego principal para ofrecer variedad a posibles compradores y se envió a UNREAL MARKETPLACE para una revisión, en esta revisión el sistema se aceptó y ahora forma parte de la lista de artículos disponibles en la tienda de Epic.

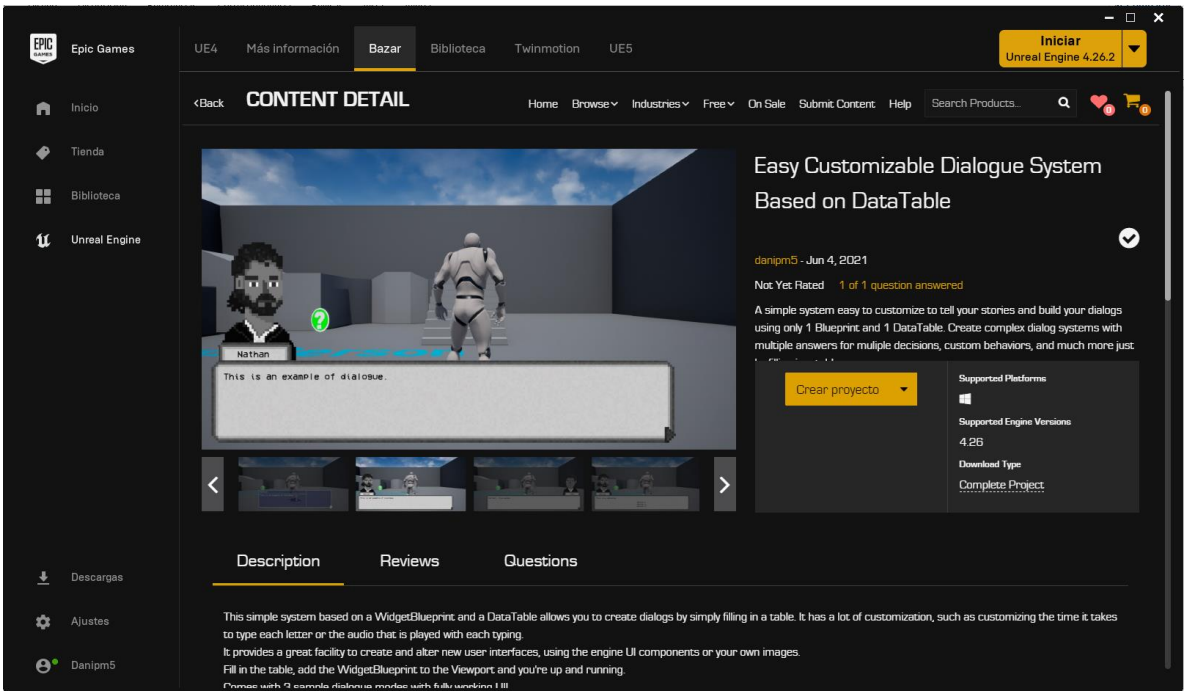


Ilustración 104: Easy Customizable Dialogue System Based on DataTable

4.6.4 Negociaciones (General)

Las negociaciones son una de las partes más importantes del proyecto ya que es gran parte de la jugabilidad del videojuego y en ella se incluye la implementación de una inteligencia artificial. En esta parte del informe se explicará en detalle su funcionamiento y el desarrollo.

Las negociaciones consisten en un dialogo de propuestas introducidas por el usuario y otras generadas por el fiscal, una inteligencia artificial capaz de tomar la decisión de aceptar o no propuestas de sentencia y rebajar o aumentar la condena en función de varios valores. Entre estos valores hay varios se generan al principio de la negociación de forma aleatoria con el fin de generar algo más de realidad en el comportamiento de la IA. Otros de los valores se calculan en función de las decisiones que toma el usuario y las propuestas que este realiza.

Con el fin de hacer más amena la negociación, también se implementa un sistema a través del cual se pueden presentar las pistas que se han descubierto a lo largo del juego, a mayores existe un sistema de charla que se puede usar para rebajar o aumentar el nivel de enfado del fiscal. Todas estas opciones afectan al comportamiento de la inteligencia artificial de una manera u otra.

4.6.4.1 Interfaz de las negociaciones

En la zona marcada en verde se encuentra la zona en la que se pueden modificar los datos para realizar una propuesta al gusto del jugador. Para cerrar una propuesta existe un botón invisible en la zona en la que firmaría el abogado defensor, igual en la propuesta del fiscal

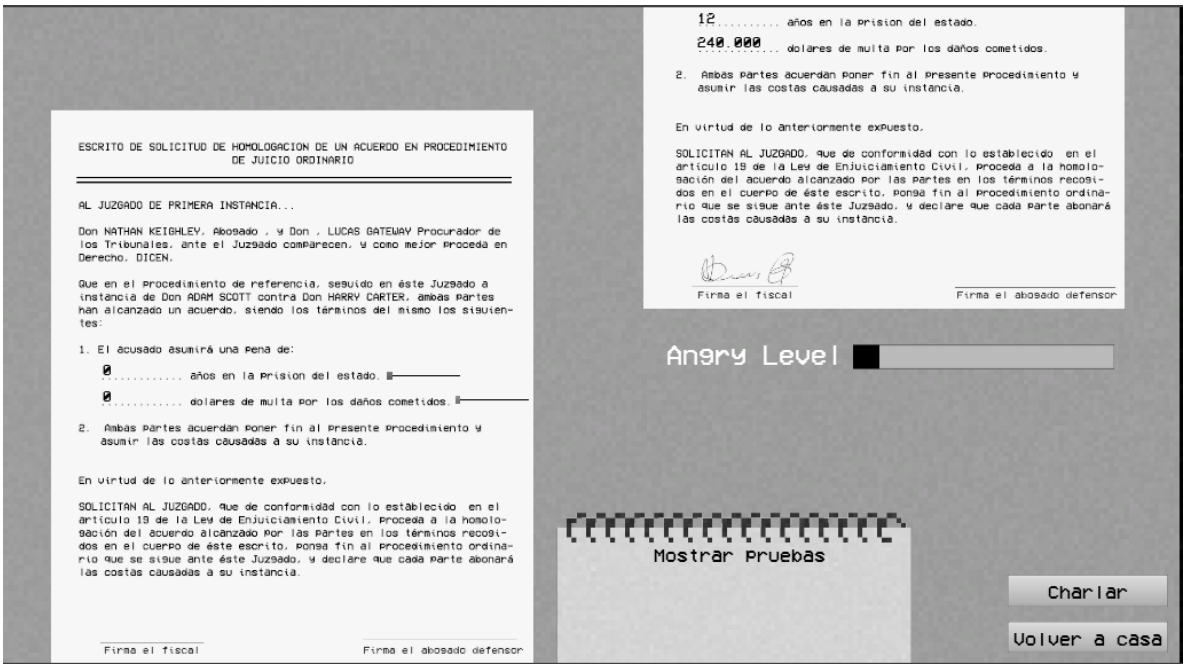


Ilustración 105: Interfaz de negociaciones

Haciendo click sobre la libreta que sobresale por la parte inferior de la pantalla se puede acceder a las pistas que se pueden proponer para intentar que el fiscal rebaje la condena.

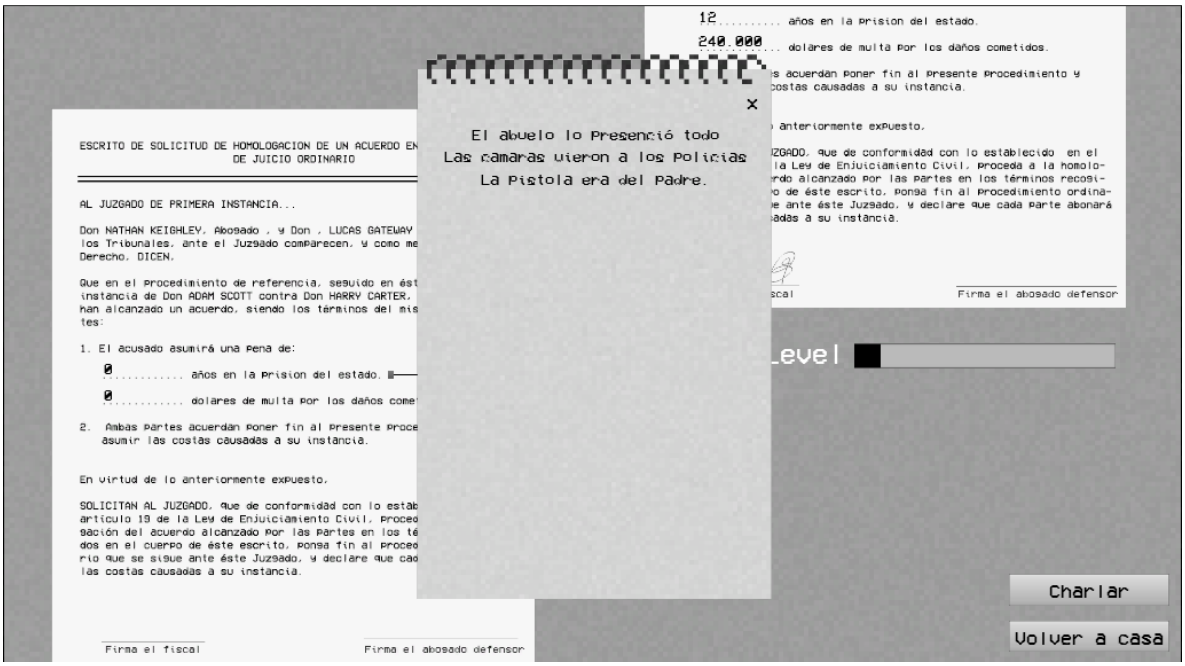


Ilustración 106: Interfaz de negociaciones 2

De igual manera se puede acceder a las opciones de charla en el botón que indica “Charlar”, para volver a casa para aplazar la negociación y seguir investigando se puede hacer click en el botón “Volver a casa”, situado abajo a la derecha.

Toda la interfaz y su funcionalidad está situada en un *BlueprintWidget* llamado “HarryCarter_AINegotiate”. Los componentes más relevantes de esta interfaz son las dos entradas de texto y dos sliders para indicar las propuestas del jugador, la barra de enfado y los botones que muestran los selectores de pistas y de charlar. Cada uno de ellos se explicará en detalle posteriormente. Una parte destacable de esta interfaz es la cantidad de eventos que pueden lanzar varias animaciones.

4.6.4.2 Enviando una propuesta

Este apartado se centrar en la funcionalidad necesaria para hacer funcionar el sistema de propuesta de sentencias. Los elementos más importantes de esta parte son las entradas de texto, y su sincronización con las sliders y viceversa.

Para la sincronización existen dos eventos, *OnTextChange* que se ejecuta cuando cambia el texto de una *TextBox* y *OnValueChange* que se ejecuta cuando cambia el valor de la slider.

En el caso de *OnTextChange*, cada vez que cambia el valor de la *TextBox* se establece el valor de “Years”, la variable que contiene el valor de los años que se proponen. Después de esto se *settea* el valor de la slider. En el caso de *OnValueChange* se sigue el mismo proceso, pero a la inversa, se *settea* el valor de la *textBox* después de dar valor a “Years”, dado que la slider puede dar valores decimales es necesario trucarlo antes de asignar su a valor a la variable.

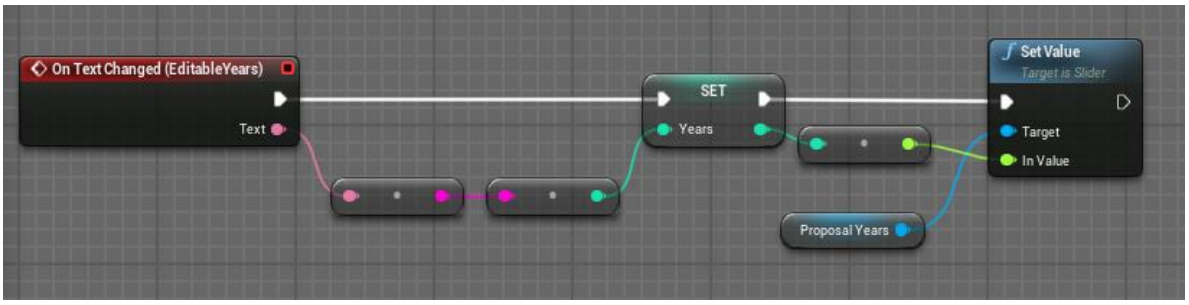


Ilustración 107: OnTextChange

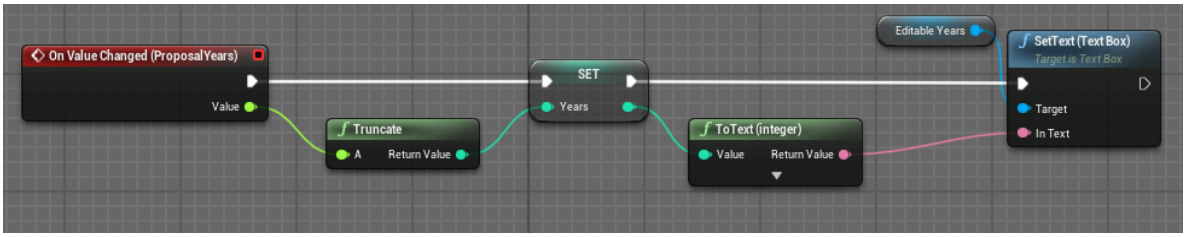


Ilustración 108: OnValueChange

Cuando los valores de la propuesta son los deseado se puede proponer haciendo click sobre el botón situando encima de la línea de firma de del abogado defensor. Este botón pondrá a True la variable que permite detectar que se ha realizado una propuesta, después de esto se llama al evento encargado de lanzar la animación de enviar la propuesta.

Ilustración 109: OnClick proposeB

El evento muestra la firma del jugador, reproduce un sonido y bloquea todas las entradas de datos para que no se puedan hacer más propuestas hasta que se responda la que se acaba de enviar, por último, se inicia la animación de enviar la propuesta.

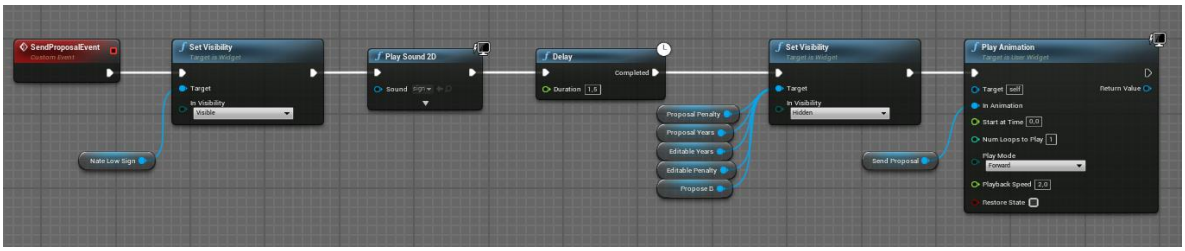


Ilustración 110: Evento SendProposal

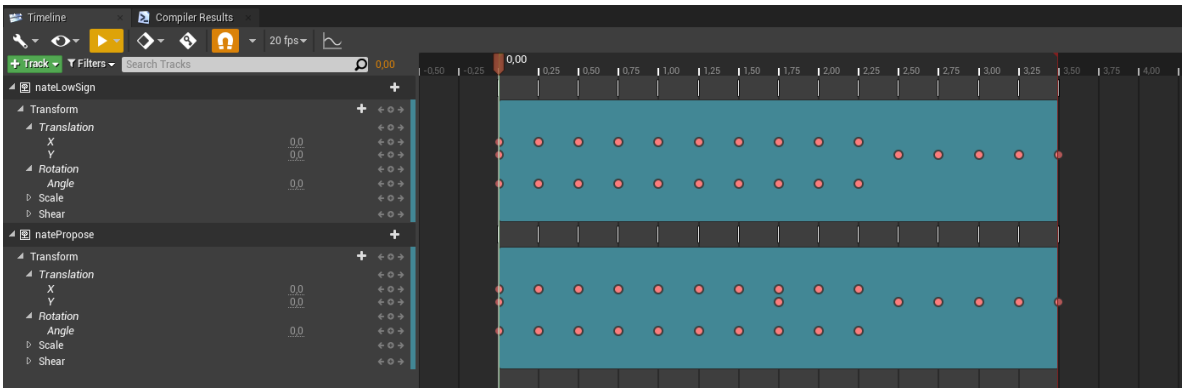


Ilustración 111: Timeline de la animación SendProposal

4.6.4.3 Mostrando una pista

Este apartado se centra en la funcionalidad necesaria para hacer funcionar el sistema de muestra de pistas. Para abrir el menú con las pistas desbloqueadas hay que acceder desde el botón posicionado encima del cuaderno situado en la parte inferior de la pantalla. Este mostrará un listado de las pistas desbloqueadas y algunas falsas añadidas para aumentar la dificultad de la negociación. Dado que se puede entrar y salir de la negociación en cualquier momento es necesario mantener un registro de las pistas que ya han sido utilizadas y las que no para que no se pueda aprovechar una pista infinitas veces. Para ellos se hace uso de dos arrays, HarryCarterEvidences y HarryCarterEvidencesAux, el primero se va rellenando cada vez que se encuentra una pista a lo largo del juego y el segundo se iguala al primero en el momento de iniciar la primera negociación y se usa de registro, eliminando de este las pistas que ya han sido utilizadas, mientras tanto el primero de los arrays siempre contiene todas las pistas desbloqueadas.

Una vez se tiene el array auxiliar y se han añadido las pistas falses se puede establecen los textos según el idioma y se pueden empezar a crear los botones de las pistas, para ello se usa el array auxiliar y en función de los strings que este contenga se crean unos botones de tipo "W_HarryCarterShowEvidenceButton" u otros dentro de un bucle *for* para posteriormente añadirlos a un *ScrollBox*.

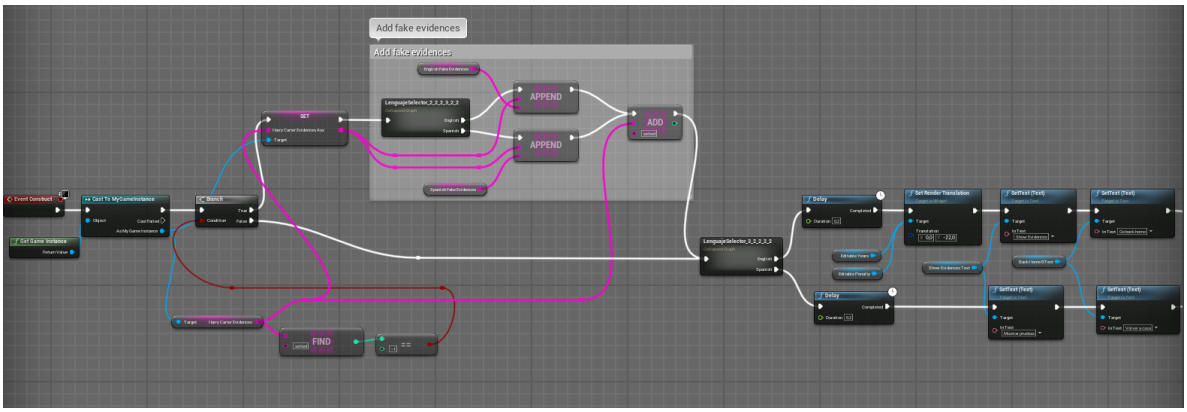


Ilustración 112: Gestión de array de pistas 1

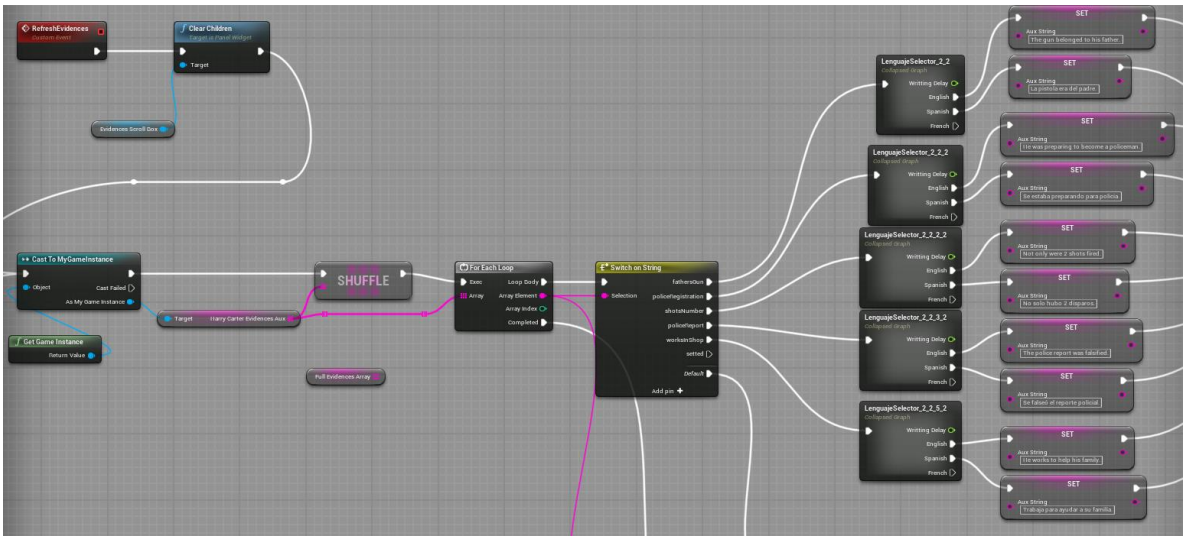


Ilustración 113: Gestión de array de pistas 2

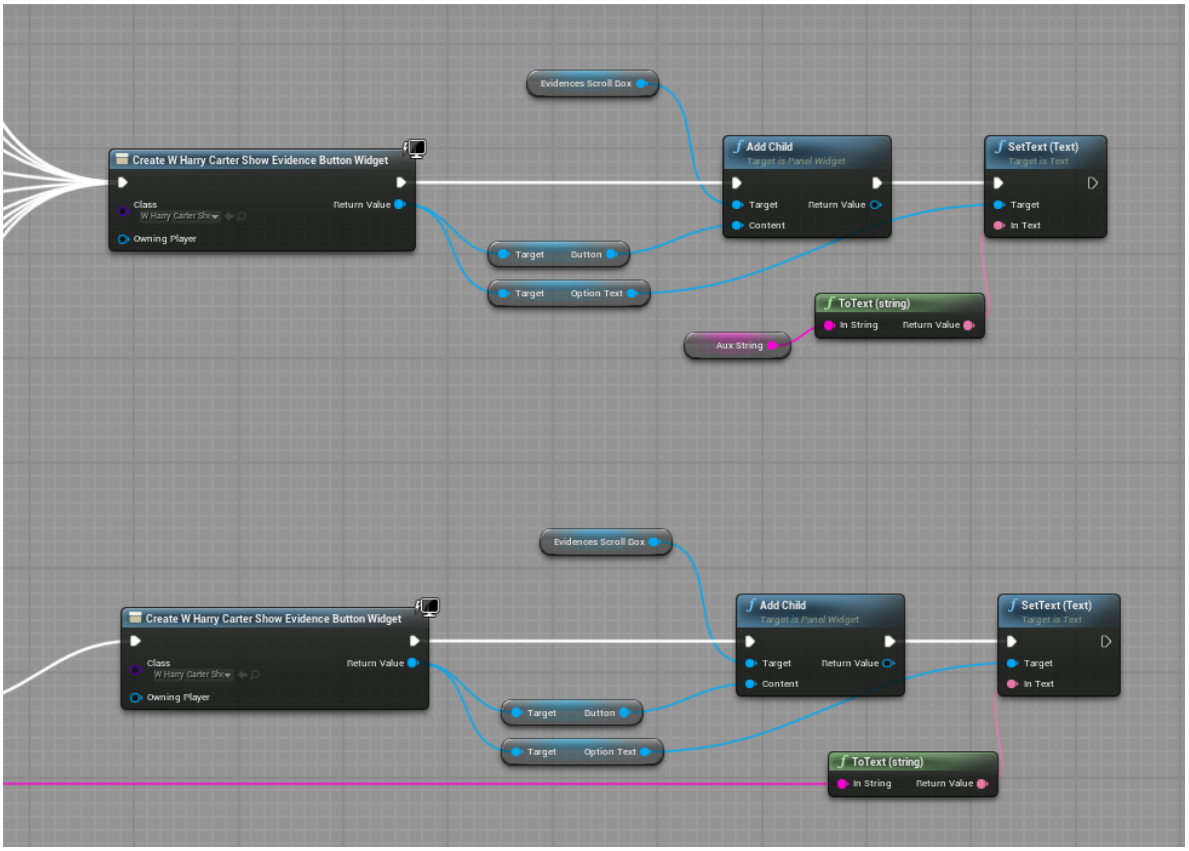


Ilustración 114: Gestión de array de pistas 3

Después de añadir las pistas se sigue un proceso idéntico con el array de "charla", "HarryCarterTalking".

Una vez creados todos los botones es el usuario podrá pulsar cualquiera de ellos, cuando se recibe una pulsación de cualquiera de los botones se redirige la funcionalidad al EventGraoh de

"W_HarryCarterShowEvidenceButton", donde aparte de encontrar los hover de los botones podemos ver como se pone a True una variable para indicar que se ha enviado una pista y posteriormente, haciendo uso del parámetro interno al botón "Text" obtenemos cuál de los botones se ha pulsado y de esta forma lo podemos borrar del array auxiliar para que no se pueda pulsar de nuevo. Por último, se cierra el cuaderno de pistas a la espera de una respuesta del fiscal.

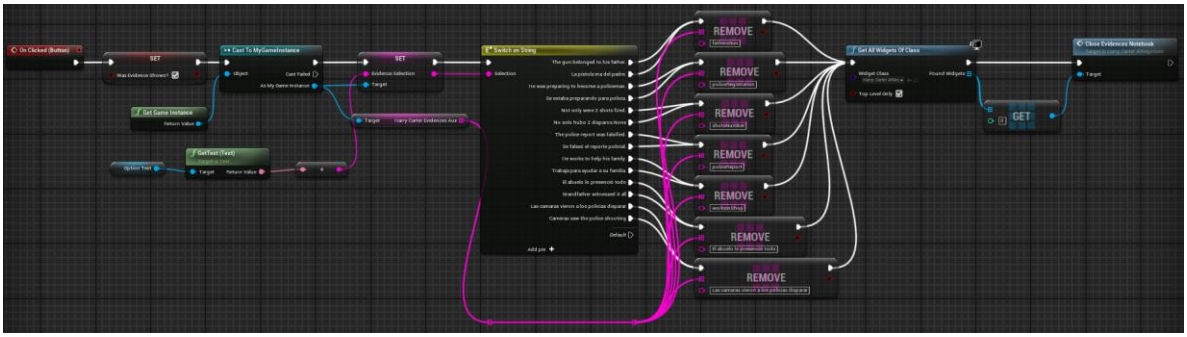


Ilustración 115: W_HarryCarterShowEvidenceButton

4.6.5 Convenciendo al fiscal

Este apartado engloba la funcionalidad de la opción de "Charlar" con el fiscal, el sistema es idéntico al de las pistas, expuesto en el apartado anterior. Se cargan los botones en un ScrollBox y se redirige la funcionalidad de estos a otro widget donde se detecta que botón se ha pulsado. En ambos casos, pistas y charla, se accede al listado de botones haciendo click en el lugar correspondiente, haciéndose visibles los componentes necesarios para poder elegir la opción deseada.

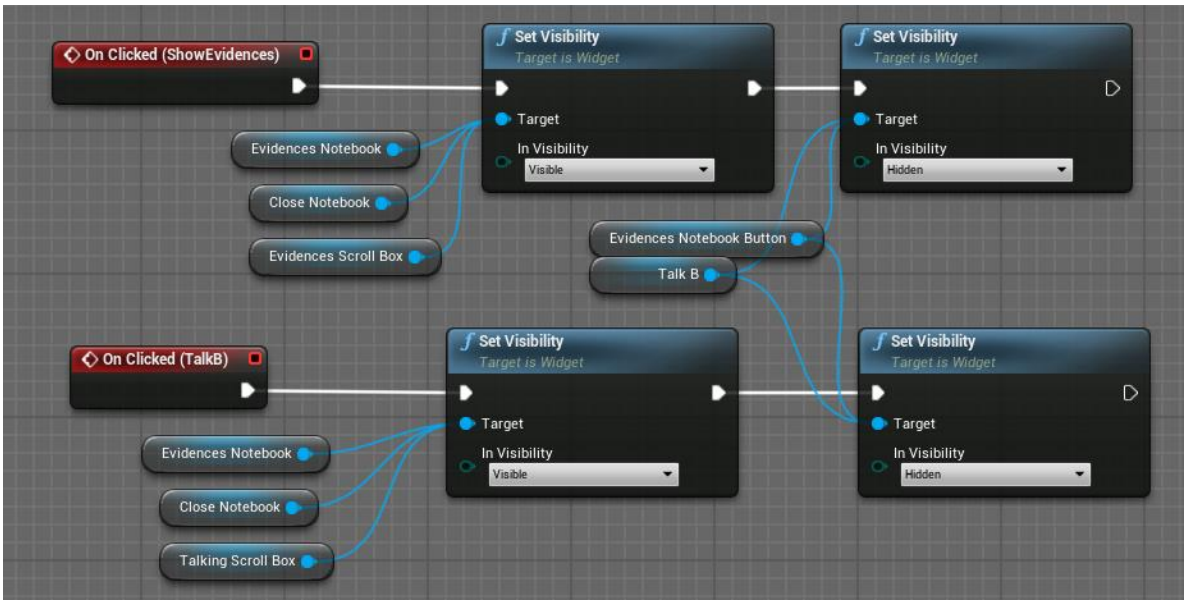


Ilustración 116: Onclicks para mostrar las pistas y opciones de charla

4.6.6 Negociaciones (IA)

Una de las partes más interesantes de las negociaciones es que el fiscal reacciona de forma inteligente gracias a una inteligencia artificial que controla sus acciones. Esta IA está programada en el sistema de Inteligencia artificial de Unreal Engine y ha permitido crear un comportamiento completo para las interacciones requeridas por las mecánicas que se esperaban para esta parte del proyecto.

La inteligencia artificial es capaz de tomar la decisión de aceptar o declinar una propuesta de sentencia en función de varios parámetros que se explicaran posteriormente, además es capaz de reaccionar a las propuestas hechas por el jugador y alterar la propuesta del fiscal en función de su nivel de enfado y la cercanía o lejanía de la propuesta del usuario a la propuesta del fiscal. También es capaz de detectar cuando se ha mostrado una pista y reaccionar a ella con una aumento o decremento de la sentencia, por supuesto tiene un comportamiento similar en el caso de la "charla". Otra característica interesante es que es capaz de calcular tiempos entre las interacciones del jugador pudiendo suponer que su comportamiento está siendo más o menos agresivo y en función de esto remodelar su comportamiento.

4.6.6.1 Generando una IA en Unreal Engine 4

Algunas teorías sobre inteligencia artificial en Unreal Engine 4 y *BehaviorTrees*. En UE4, hay muchas formas de implementar la lógica de un rol inteligente. Un método muy simple es usar *EventTick* para implementar un cierto algoritmo para que el rol pueda realizar una operación u otra. Esto se puede lograr directamente aquí, pero como necesitamos hacer las acciones que realizarán los Actores, el estado en el que pueden estar y lo que pueden hacer en cada caso, se volverá más complicado. Tal lógica será más tediosa. Esta es una de las razones por las que UE4 promueve la creación de un *BehaviourTree* (árbol de comportamiento) para poder definir el comportamiento del NPC en función de las condiciones en las que se encuentra.

4.6.6.2 BehaviourTree y Blackboard

La creación de inteligencia artificial (IA) en Unreal Engine 4 (UE4) para los personajes puede hacerse de diferentes maneras. Se puede usar Blueprint Visual Scripting para indicar a un personaje para que "haga algo" como reproducir una animación, moverse a un lugar específico, reaccionar cuando es golpeado por algo y más. Pero cuando quieras que los personajes de la IA piensen y tomen sus propias decisiones, los árboles de comportamiento son la mejor elección.

Los Árboles de Comportamiento se crean de una manera similar a la de Blueprint, añadiendo y conectando una serie de nodos que tienen alguna funcionalidad. Mientras que un Árbol de Comportamiento ejecuta la lógica, un activo separado llamado *Balckboard* se utiliza para almacenar información (llamada *BlackBoard Keys*) que el *BehaviourTree* necesita conocer para tomar decisiones informadas. Un flujo de trabajo típico sería crear una Pizarra, añadir algunas *BlackBoard Keys* y, a continuación, crear un *BehaviourTree* que utilice el activo de la *BlackBoard*.

4.6.6.3 Elementos utilizados para la IA del proyecto

Para la realización de la inteligencia artificial de este proyecto se han utilizado los siguientes elementos.

Ilustración 117: Elementos de la IA

4.6.6.3.1 AIController

Los *Controllers* son actores no físicos que pueden poseer un *Pawn* para controlar sus acciones. Un *PlayerController* es utilizado por los jugadores para controlar a los *Pawn*, mientras que un *AIController* implementa la inteligencia artificial para los *Pawn* que controlan. Los controladores toman el control de un *Pawn*. En esta clase se inicia el árbol de comportamiento.

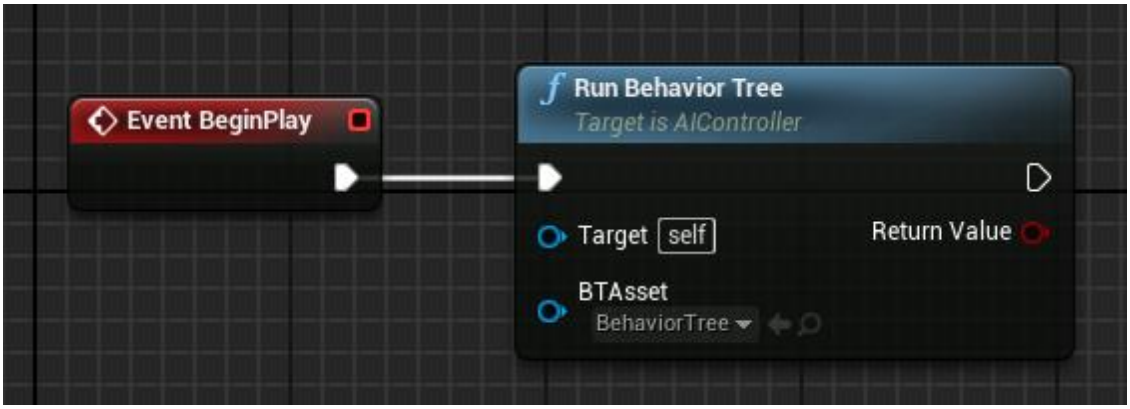


Ilustración 118: AIController

4.6.6.3.2 AIProsecutorPawn

La clase Pawn es la clase base de todos los actores que pueden ser controlados por jugadores o IA. Un Pawn es la representación física de un jugador o entidad de la IA dentro del mundo. Esto no sólo significa que el Peón determina el aspecto visual del jugador o entidad de IA, sino también cómo interactúa con el mundo en términos de colisiones y otras interacciones físicas. Esto puede ser confuso en ciertas circunstancias, ya que algunos tipos de juegos, como es el caso que aborda este proyecto, pueden no tener una malla de jugador o avatar visible dentro del juego. En cualquier caso, el Pawn sigue representando la ubicación física, la rotación, etc. de un jugador o entidad dentro del juego. En esta clase se establece cual va a ser el controlador que posea a el Pawn, que será el AIController mencionado anteriormente.

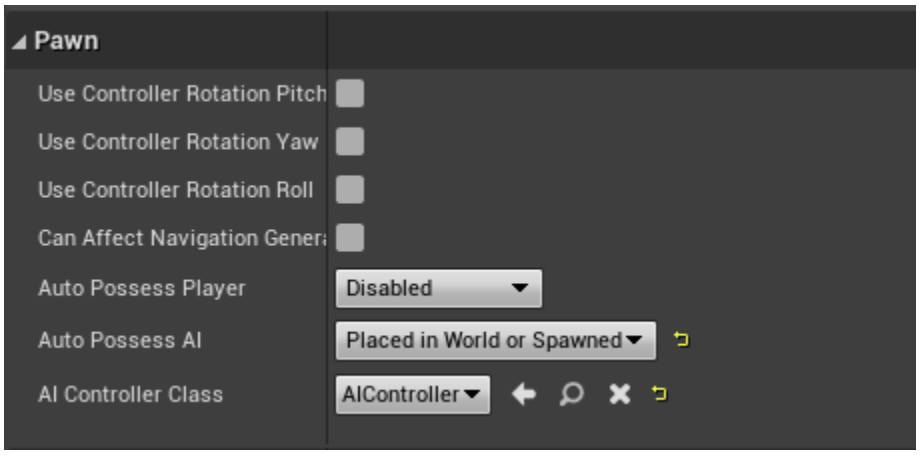


Ilustración 119: Asignación del AIController al Pawn

4.6.6.3.3 BlackboardData

En este apartado se explicará en detalle el desarrollo del *Blackboard* que se usa en el proyecto. Para el funcionamiento de la inteligencia artificial, el *Blackboard* aloja varias variables que se usaran en el *BehaviourTree* para realizar la toma de decisiones. Las variables son las siguientes.

- *WasProposalSent?:* sirve para comprobar si una propuesta se ha enviado.
- *WasEvidenceShown?:* sirve para comprobar si una pista se ha enviado.
- *WasTalkingStarted?:* sirve para comprobar si se ha iniciado una "charla"
- *FriendshipLevel:* valor aleatorio que se inicia al principio de la negociación para darle más variedad a esta.
- *VolumeOfWork:* valor aleatorio que se inicia al principio de la negociación para darle más variedad a esta.
- *InterestLevel:* valor aleatorio que se inicia al principio de la negociación para darle más variedad a esta.
- *AngryLevel:* mantiene el nivel de enfado del fiscal.
- *SelfActor:* se trata del propio actor.
- *AreValuesInit?:* comprueba que los valores necesarios al principio de la negociación han sido iniciados.
- *Secs:* guarda los segundos que pasan entre interacción e interacción.
- *NotWaiting?:* comprueba si se está esperando por la respuesta de la IA.
- *TimeAverage:* guarda la media que de tiempo entre interacciones.
- *AcceptedPenalty:* sirve para indicar si la propuesta ha sido aceptada o rechazada.

- *ReadingText:* indica que se encuentra en un estado en el que se le está mostrando texto al usuario y se debe esperar a que el usuario acabe.
- *WasProseProposalAccepted?:* sirve para indicar si se ha aceptado la propuesta del fiscal.

Ilustración 120: Variables del Blackboard

4.6.6.3.4 BehaviourTree

En este apartado se explicarán en detalle los procedimientos que se han seguido para la creación del árbol de comportamiento, a lo largo de la explicación se irán explicando en detalle todas las tareas que se ejecutan. En la parte superior del árbol se encuentran el nodo raíz y la secuencia de iniciación, tras este podemos se encontrar el primer condicional, donde que se seleccionara qué camino seguir en función de si los valores iniciales han sido iniciados, para ello se añade el *decorator* que se muestra en azul en la siguiente imagen.

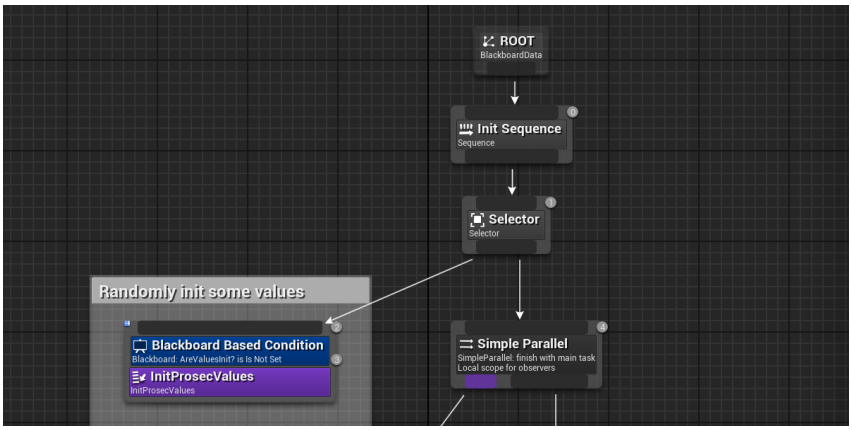


Ilustración 121: BehaviourTree 1

En morado, "InitProsecValues", vemos la primera tarea que se va a ejecutar. Dado que esta y todas las tareas que se comentaran posteriormente necesita hacer uso de la Blackboard, esta se guarda en una variable para tenerla más a mano.

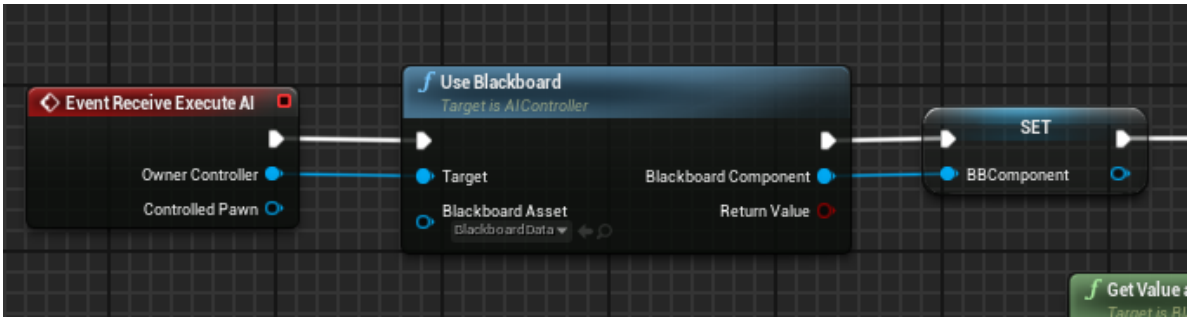


Ilustración 122: Guardando la blackboard en una variable

Posteriormente se comprueba si se ha aceptado alguna propuesta y si no es así se inician todos los valores que se van a usar en la negociación. Todos los valores se inician de la misma manera, con el nodo *SetValueAs****** pasando por el parámetro *keyName*, en nombre de la variable de la Blackboard. En caso de los valores aleatorio se selecciona uno entre cero y diez gracias al nodo *RandomInRange*.

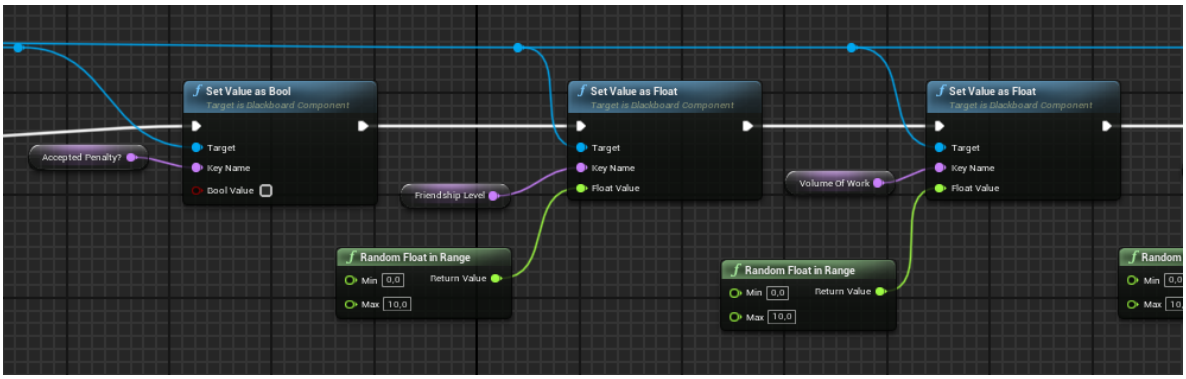


Ilustración 123: Iniciando valores

Al finalizar el inicio de todos los valores se finaliza la ejecución con el parámetro success a true para poder pasar al siguiente nodo.

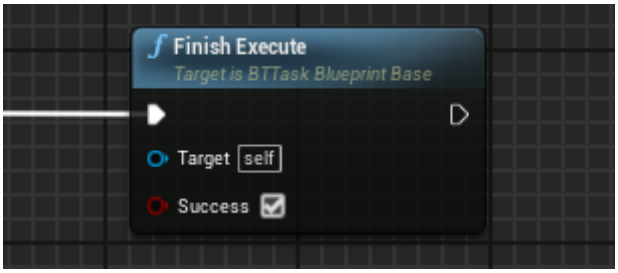


Ilustración 124: Success a true

En el simpleParallel se ejecutan ambas tareas a la vez, la que vemos a la izquierda en la ilustración 108 se trata del temporizador que cuenta el tiempo entre interacciones. Para ello se comprueba si no se está esperando por ninguna acción. Y si se está esperando, se espera un segundo y se incrementa el contador de segundos, así continuamente hasta que la IA detecta una acción. En ese momento entraría por la rama de True.

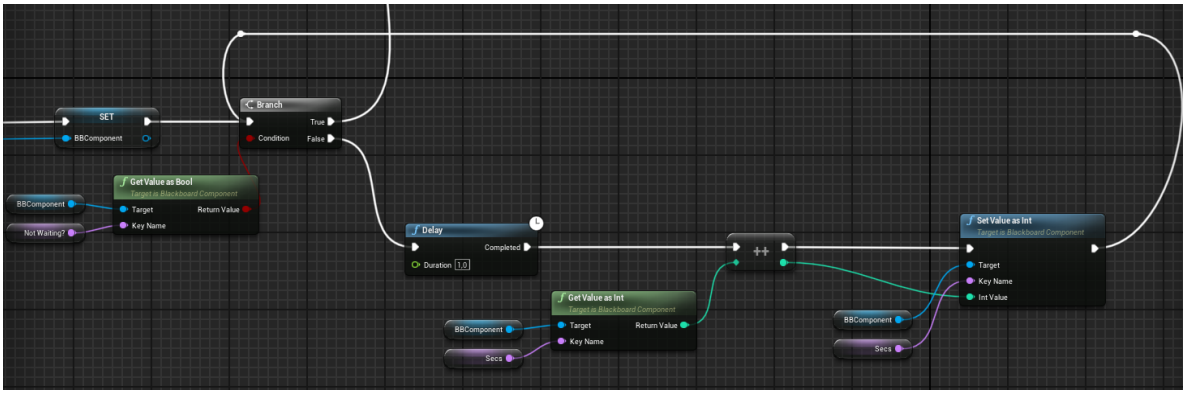


Ilustración 125: Rama de false del timer

Al entrar en esta rama se obtienen el tiempo que se ha estado esperando y se le restan siete segundos que es lo que aproximadamente tardan las animaciones. Y luego se añade este nuevo cálculo un array de tiempos que se usará para calcular la media de todos los tiempos. Esta media se guarda en la *blackboard* y se vuelve a poner la espera a falso.

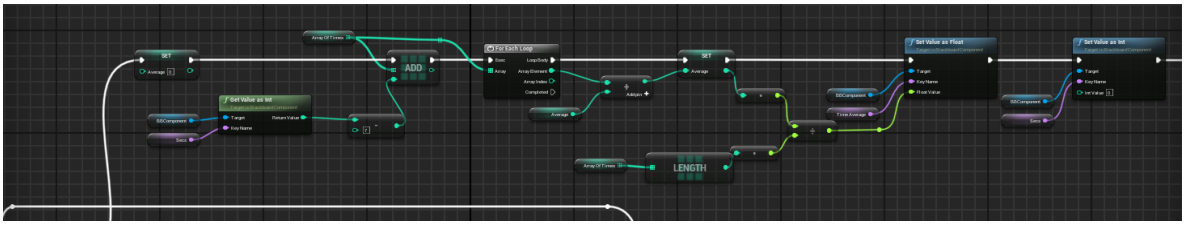


Ilustración 126: Rama true del timer 1

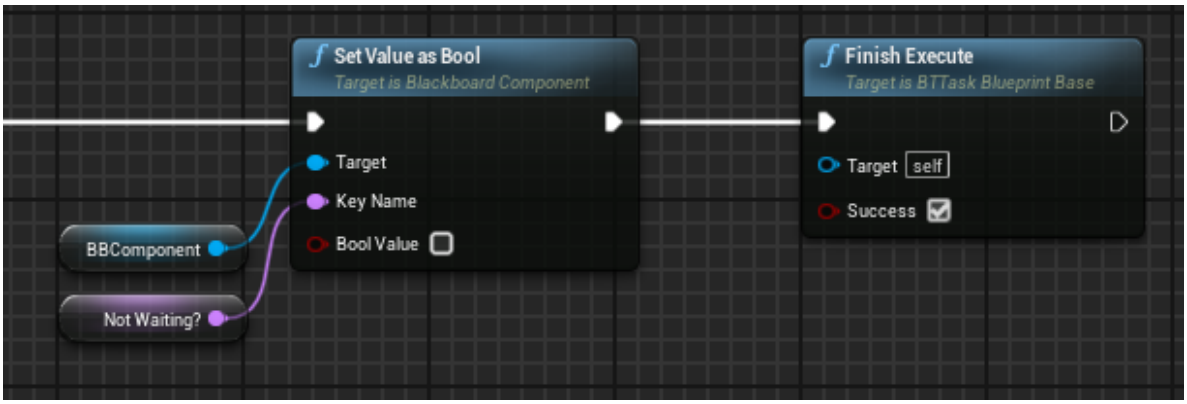


Ilustración 127: Rama true del timer 2

A la vez que está corriendo el contador de segundos se ejecuta en paralelo la otra rama del *SimpleParallel*, que es una *Sequence* llamada “WatingProposal”.

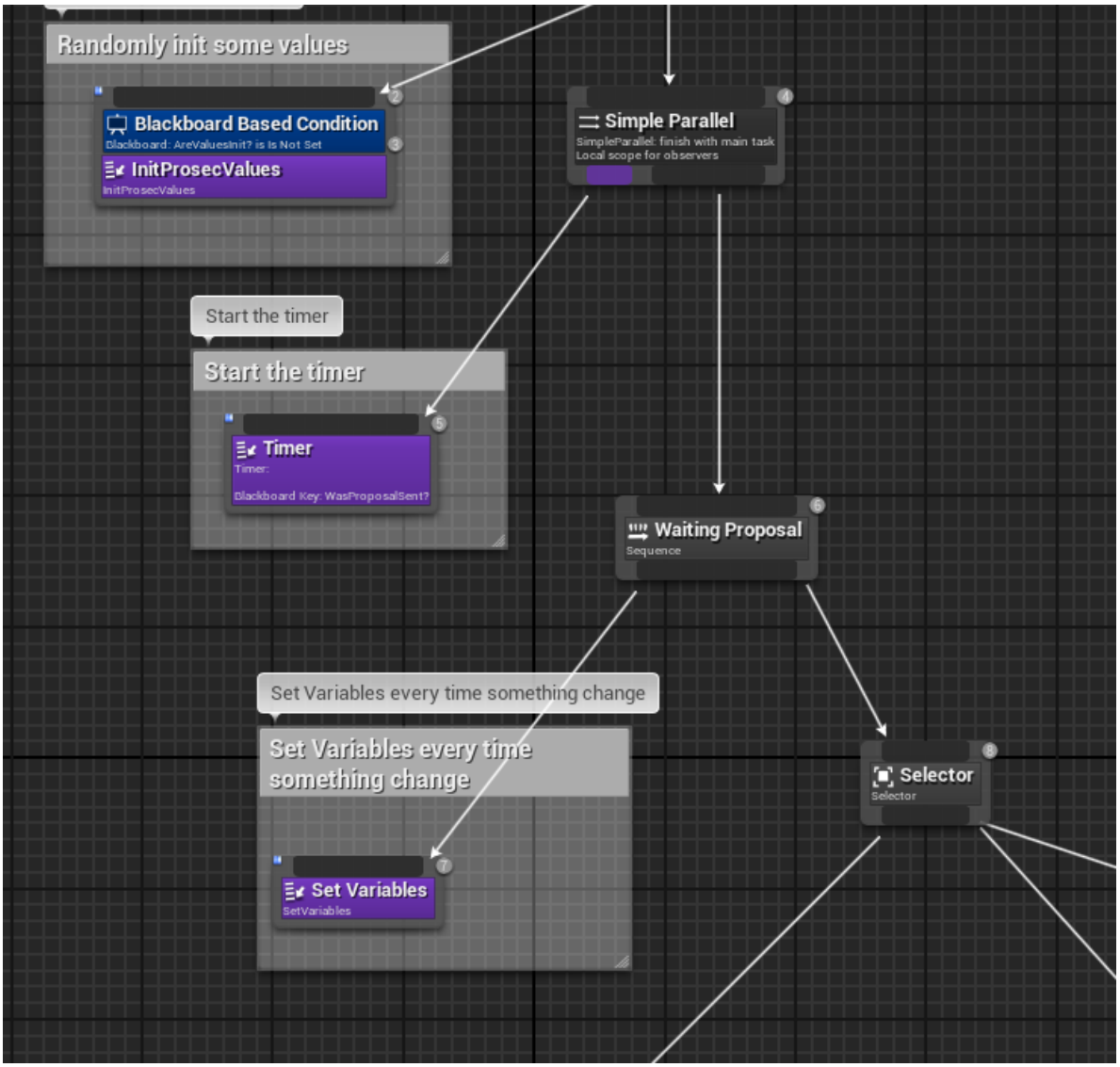


Ilustración 128: Ramas de SimpleParallel

En esta secuencia encontramos una tarea que es encargada de restaurar el valor de todas las variables cada vez que ocurre algún cambio. Y posteriormente seleccionar que tipo de interacción se ha realizado, propuesta, pista o "charla".

Si vemos la tarea *SetVariables* podemos ver como se obtienen los datos del fiscal de una estructura llamada *ProsecutorState*, que contiene la pena en años y en dinero y el nivel de enfado que se inició anteriormente. Es necesario guardar estos datos en *MyGameInstance* para que se mantengan en las próximas negociaciones del mismo caso. Para ello se genera este bloque de código donde se obtienen los valores de la estructura y se igualan a las variables que se usan en el *WidgetBlueprint* para mostrar los datos al usuario.

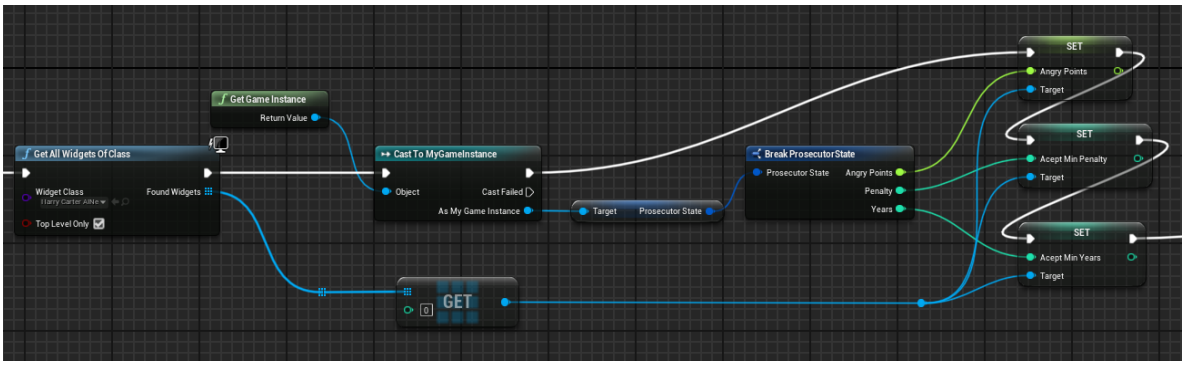


Ilustración 129: Obtención y asignación de datos de la estructura

Después de esto se resetean las variables:

- WasProposalSent?
- WasEvidenceShown?
- WasProsecProposalAccepted?
- WasTalkingStarted?

Todas ellas se resetean de la misma manera, como se muestra en la siguiente ilustración.

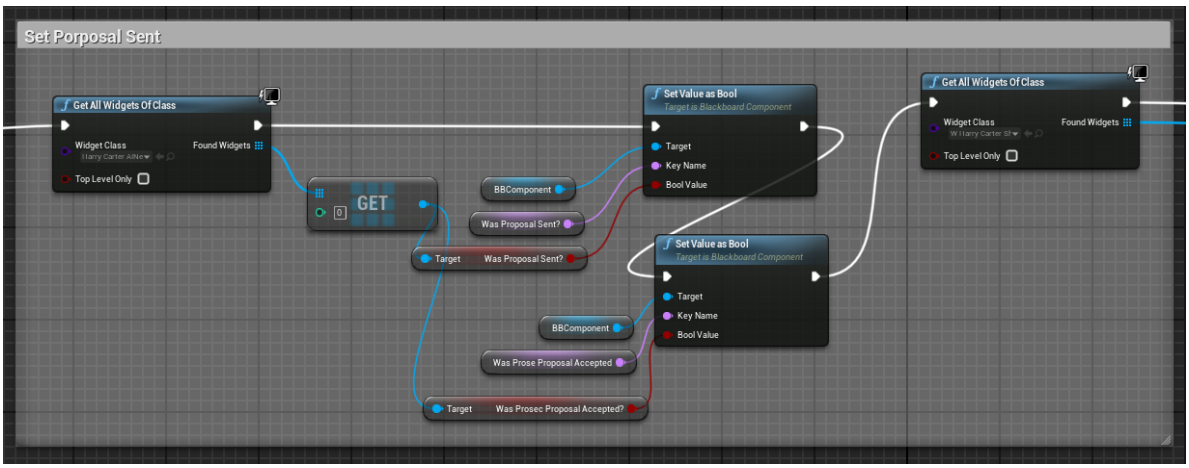


Ilustración 130: Reseteo de variables para la negociación

Tras resetear las variables se llega al selector, donde se elige que interacción a llevado a cabo el usuario.

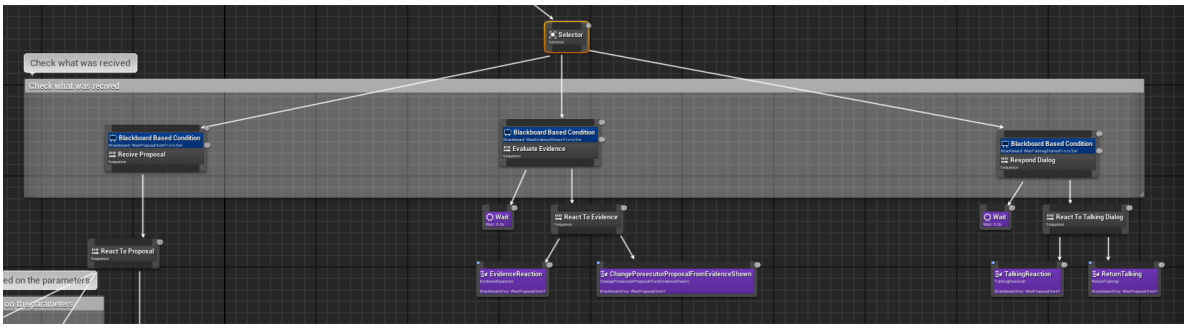


Ilustración 131: Selector de interacciones

Como se ve en la anterior ilustración el árbol se divide en 3 ramas, una para las propuestas, una para las pistas y otra para las opciones de charla. La IA es capaz de decidir cuál es la opción que ha seleccionado el usuario gracias a los *decoratos* que comprueban el estado de las variables Was*****? contenidas en el *blackboard*. Cada una de las ramas dirige la ejecución a las acciones determinadas para cada interacción.

4.6.6.3.5 Propuesta de pena (IA)

El primer caso, que el usuario decida realizar una propuesta, se inicia con una secuencia de acciones que consiste en esperar 4 segundos, simulando que el fiscal está pensando cómo responder y una tarea que es la encargada de toda la inteligencia detrás de la toma de decisiones referente a los propios valores introducidos por el usuario y los esperados por la máquina. Toda esta lógica se encuentra en una clase "RespondToProposal" la cual tiene un evento llamado "RespondInFunctionOf" que es el principal encargado de realizar la tarea de tomar decisiones.

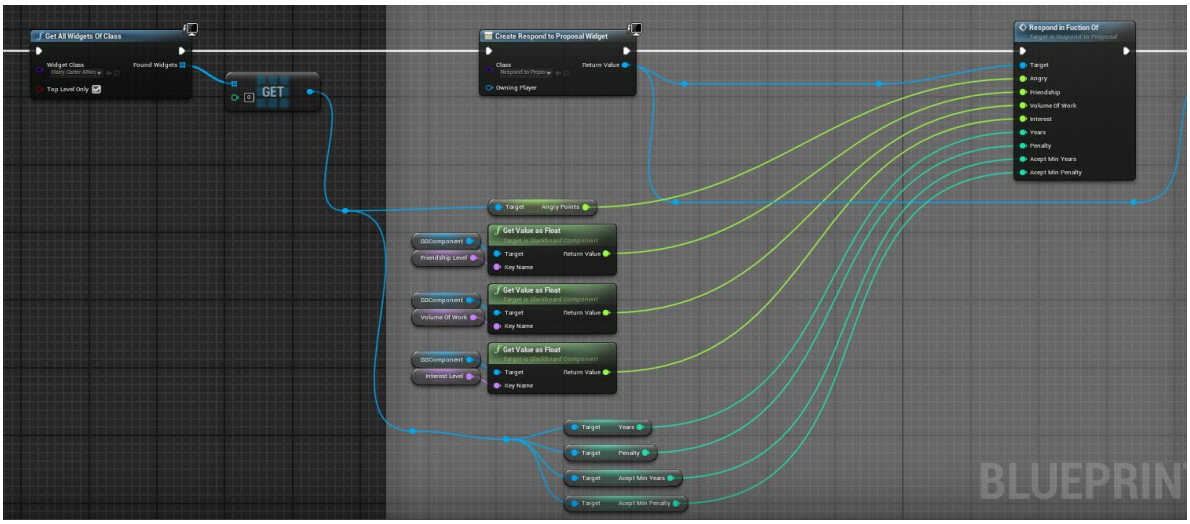


Ilustración 132: Tarea RespondToProposal

Para que el evento "RespondInFunctionOf" pueda tomar decisiones necesita que se le pasen todos los datos como se ve en la anterior ilustración, donde se le pasan ciertos valores de la blackboard así como los que el usuario a introducido para su propuesta.

Si vemos el evento “RespondInFunctionOf” lo primero que vemos es como los valores recibidos por los parámetros del evento son asignados a variables del mismo nombre para poder tratarlas de forma más cómoda en el grafo. Lo primero que se calcula es un valor de aleatoriedad vasado en los tres valores que se muestran en la ilustración que aparece a continuación.

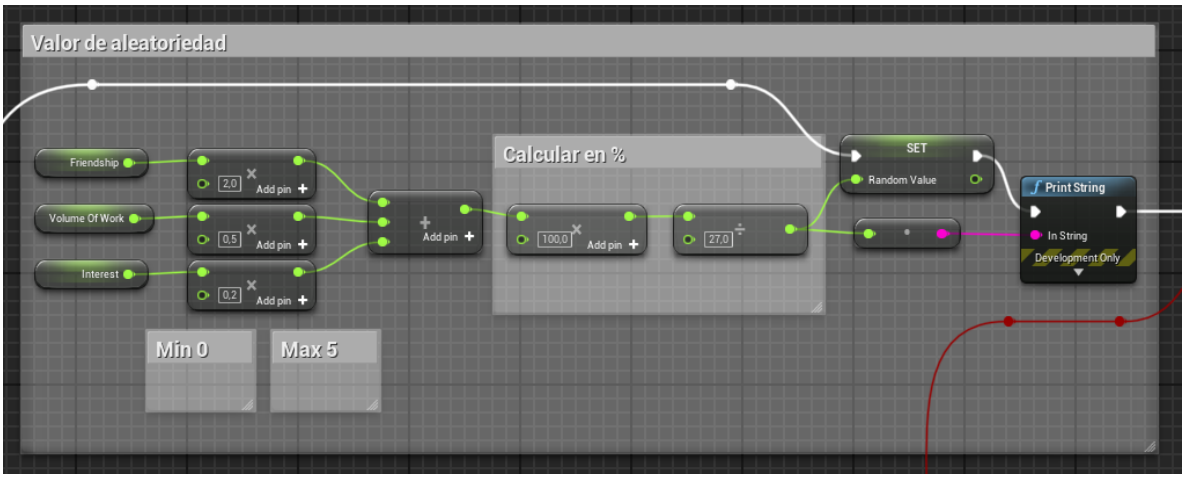


Ilustración 133: Calculo de valores de aleatoriedad

Este cálculo nos da un valor aleatorio en forma de porcentaje que se usara posteriormente en la toma de decisiones.

Finalizando con este cálculo se tiene un condicional sencillo que comprueba si los valores introducidos por el usuario son mayores que los que espera la IA, si es así se aceptará la propuesta y si no se empezará a maquinar una reacción con sus respectivas consecuencias. Si el caso es el último se empieza por calcular la diferencia entre la propuesta del usuario y la de la IA, además se divide entre 6 la propuesta de la IA para tener 6 rangos de “reacción”, y estos niveles se guardan en un array para los años y la multa. Posteriormente se comprueba en que rango se encuentra la propuesta del usuario (años y multa) y en función de esta se le asignan una serie de puntos.

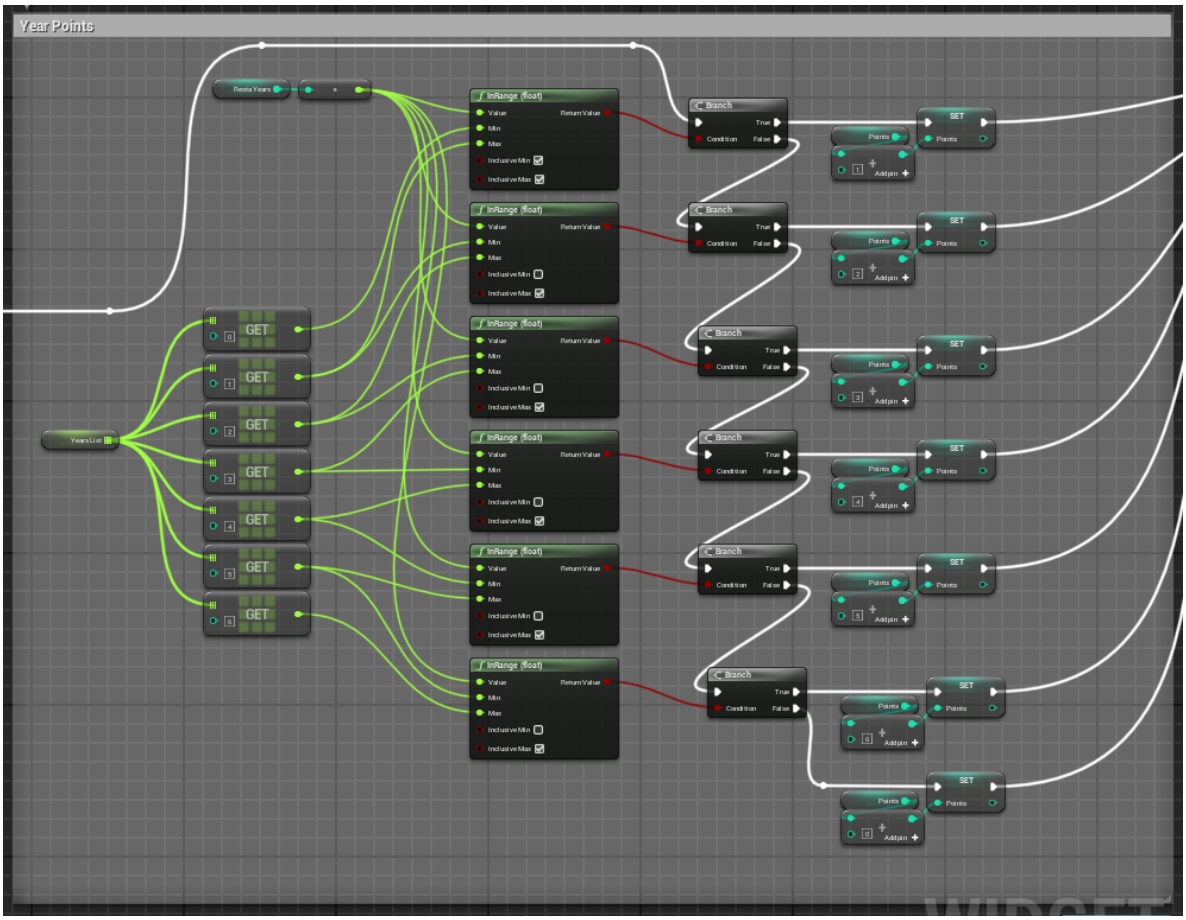


Ilustración 134: Asignación de puntos en función del rango

Después de calcular los puntos de ambos rangos se pasa por una función matemática (sujeta a ajustes) que valora los valores de aleatoriedad, el nivel de enfado, los puntos y el tiempo entre propuestas y va a una función que va a decidir qué valor año/multa merece según el valor de salida de dicha función. Para ello se han calculado los valores máximo y mínimo de la función y se ha dividido en rangos de igual valor, dependiendo el valor de salida de esta función se accede a un rango u otro. Dependiendo de la complejidad que se desee se pueden ajustar hasta 28 rangos distintos, pero para el caso práctico se establece este valor a 7 rangos. Cada rango establece un valor de año/multa con el cual se toma la decisión final de aceptar o no la propuesta del usuario.

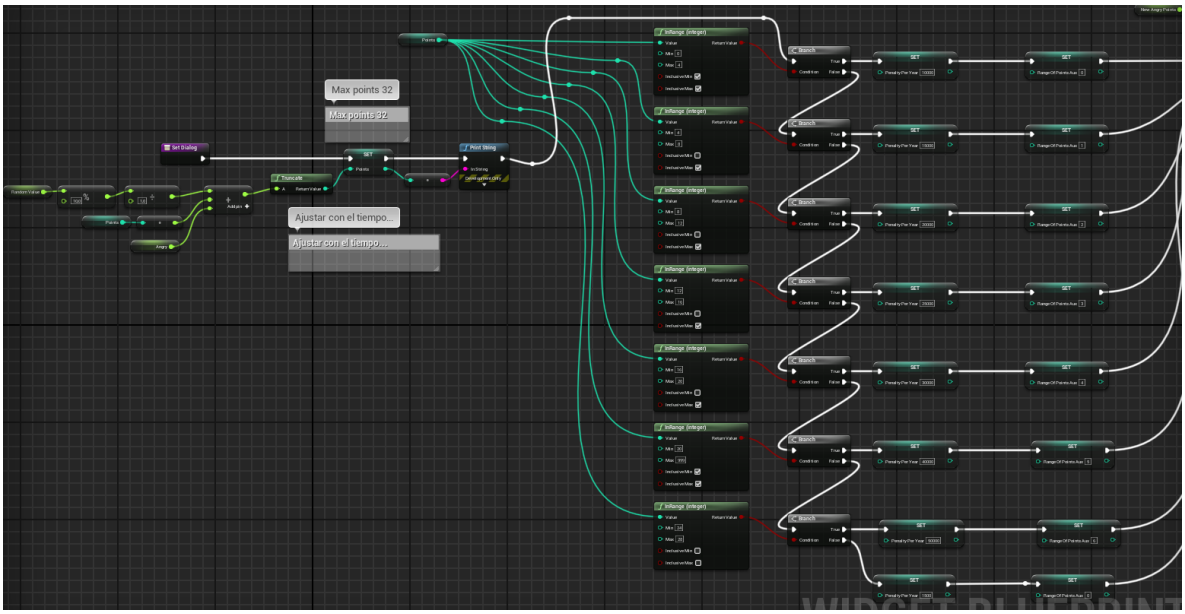


Ilustración 135: Función y decisión de rango

Pero antes de nada se debe comprobar si se ha superado el número máximo de puntos de enfado y si es así cerrar y bloquear la negociación, obligando al jugador a ir a juicio.

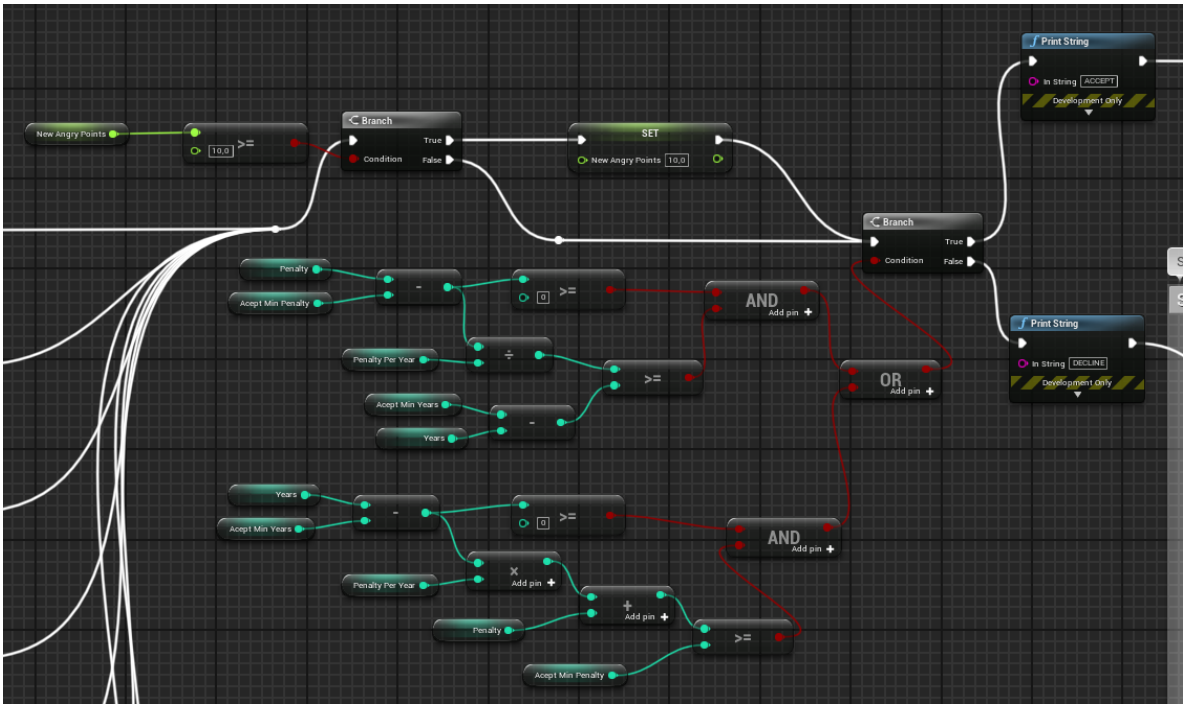


Ilustración 136: Decisión de aceptar o declinar la propuesta

Por último, y de nuevo en función del rango por el que se ha entrado anteriormente se selecciona el dialogo que va a decir el fiscal, así como el aumento de la pena y el nivel de enfado.

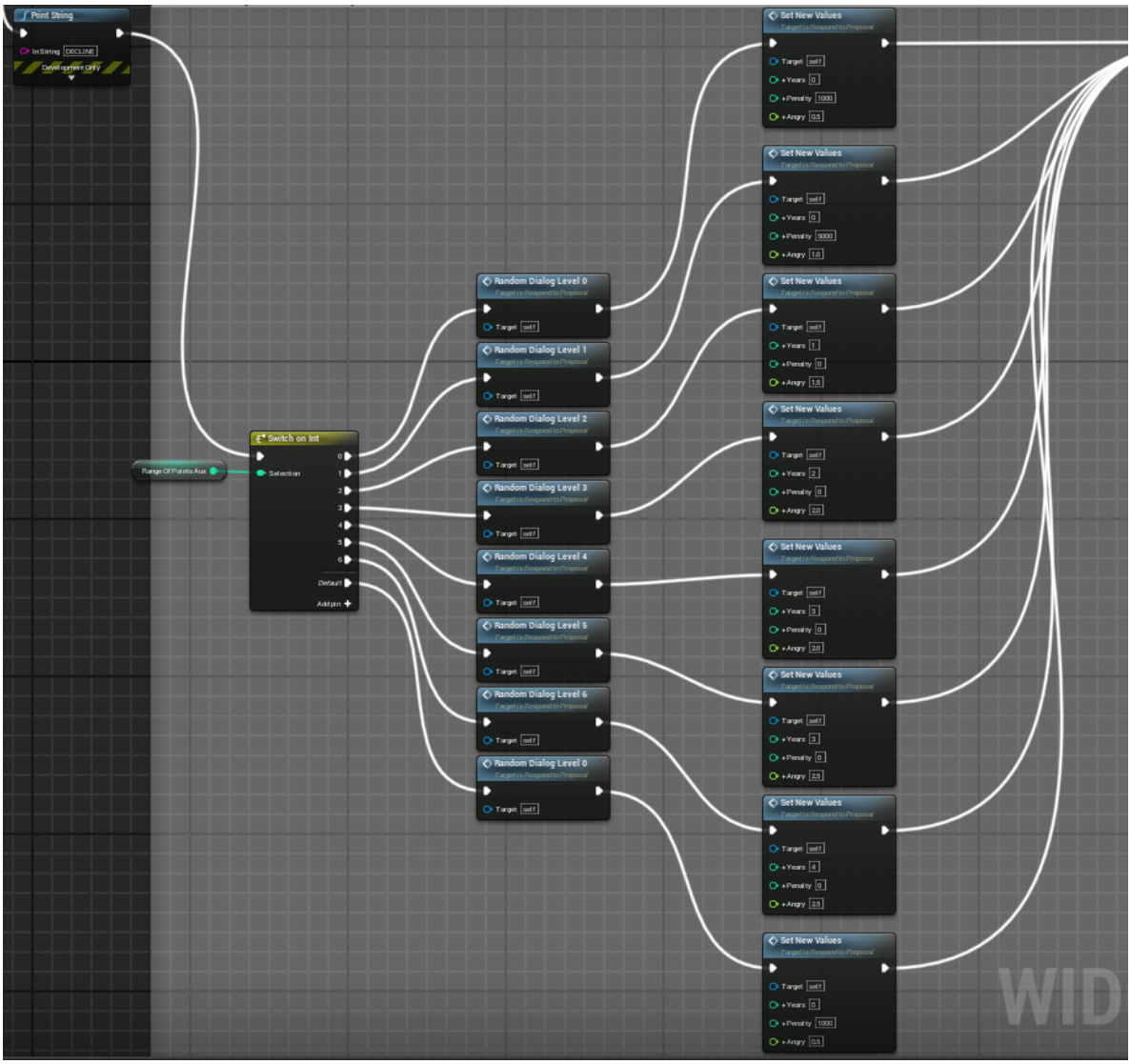


Ilustración 137: Decisión de dialogo del fiscal

Los diálogos se deciden en las funciones RandomDialogLevelX, que contienen hasta 4 líneas de diálogo diferentes lo que hacen hasta 28 diálogos distintos. Pero configurable hasta un máximo de 28 rangos por (suponiendo) 4 líneas de dialogo por rango podríamos programar hasta 112 opciones de dialogo y sus convenientes cambios en la condena.

Si en cualquiera de las comprobaciones se acepta la propuesta se llama a la función "AceptPenalty" en la cual solo se cambia el valor de *Accepted?* a true.

Al salir de esta función ya se conoce la decisión de la IA, por lo que sí es se ha aceptado la propuesta se selecciona el dialogo de "Cerrar el trato" y en caso contrario se guardan los nuevos valores en la estructura "ProsecutorState" de "MyGameInstance" y se procede a mostrar el dialogo seleccionado al igual que se hace en los diálogos normales.

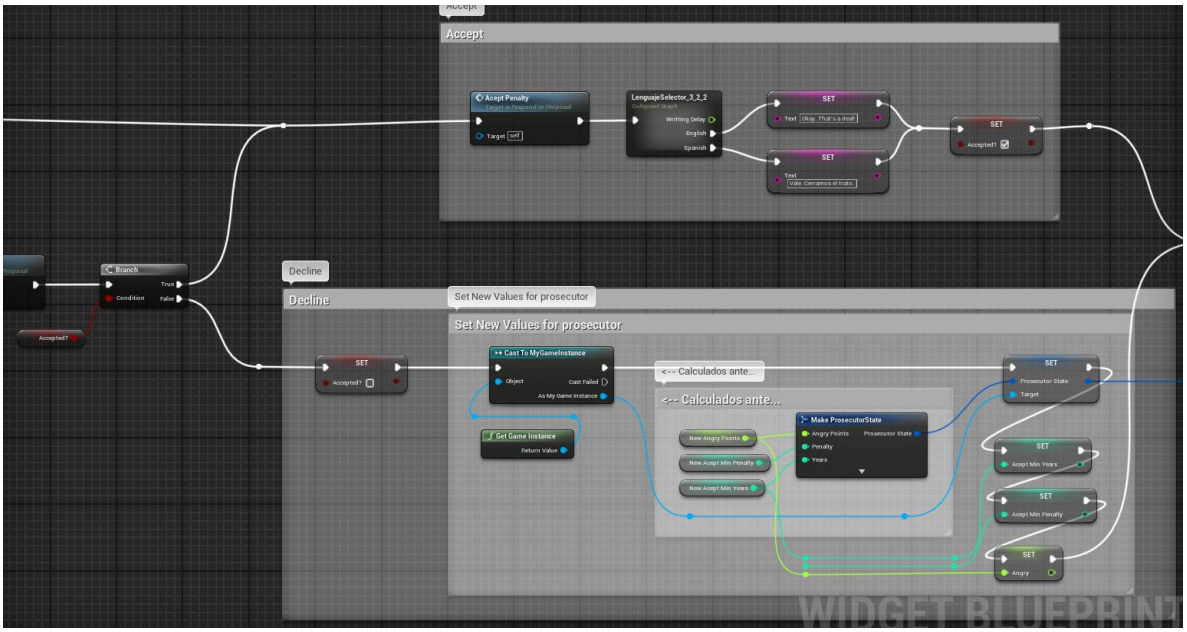


Ilustración 138: Propuesta aceptada o rechazada

Si al final del progreso se alcanza el nivel máximo de enfado se muestra un dialogo de enfado y se bloquean todas las opciones de negoción obligando al jugador a arriesgar todo en un juicio.

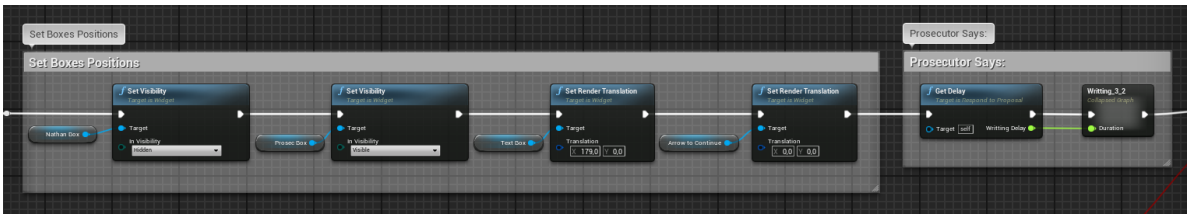


Ilustración 139: Imprimiendo dialogo de respuesta

Volviendo al árbol de comportamiento, después de tomar la decisión tenemos tres opciones que realizaran una u otra acción. Todas ella lanzan una animación y en caso de que se acepte la propuesta cierran la negociación de forma satisfactoria y con ello se cierra el caso.

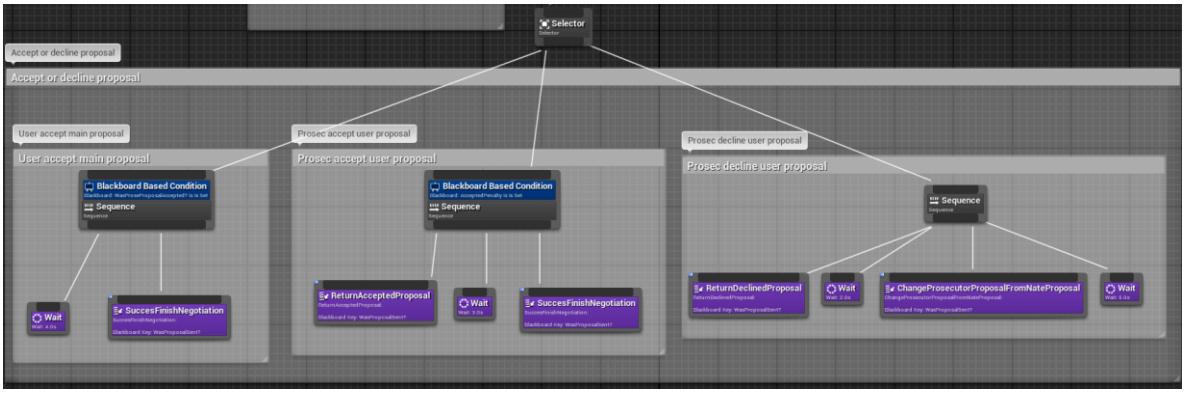


Ilustración 140: Opciones después de tomar decisión

Dado que las tareas de presentar animaciones son idénticas entre ellas solo se explicará el funcionamiento de una de ellas. Las tareas que tratan esta mecánica son:

- ReturnAcceotedProposal
- ReturnDeclinedProposal
- ChangeProsecutorProposalFromNateProposal

Todas ellas lanzan la nimacion de “HarryCarterAINegotiate” y establecen los valores de las variables de la *blackboard* para poder continuar.

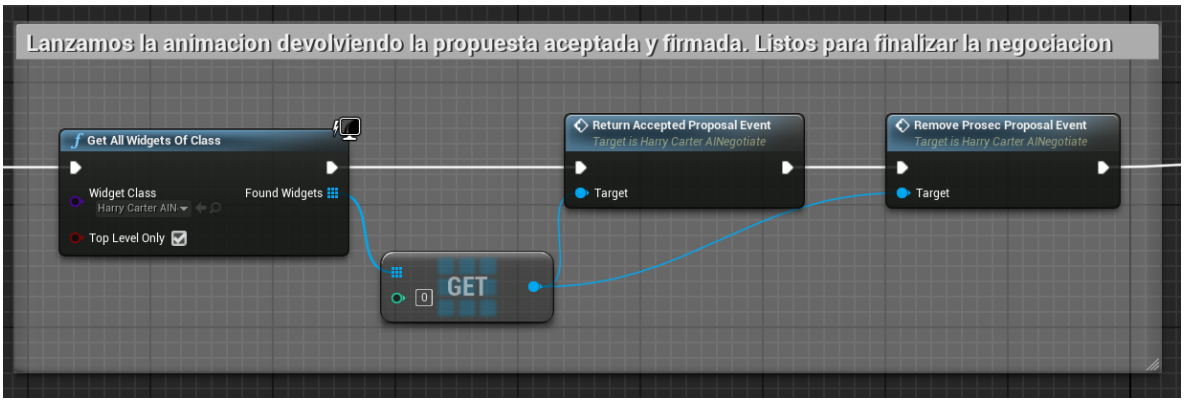


Ilustración 141: Lanzando la animación desde una Task

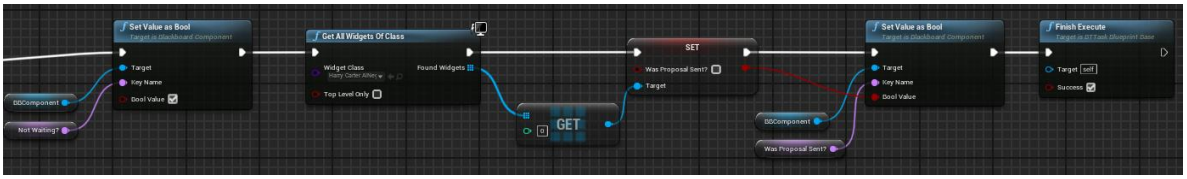


Ilustración 142: Estableciendo nuevos valores para continuar la ejecución

Si en cualquiera de estos tres casos se debe cerrar la negociación se llama a la tarea “SuccesFinishNegotiation”, lo que significa que se cierra el caso por lo que hay que mover el caso de abierto a cerrado en la libreta de casos, abrir el nivel principal y finalizar la ejecución sin *success* para que no pase a la siguiente rama del árbol.

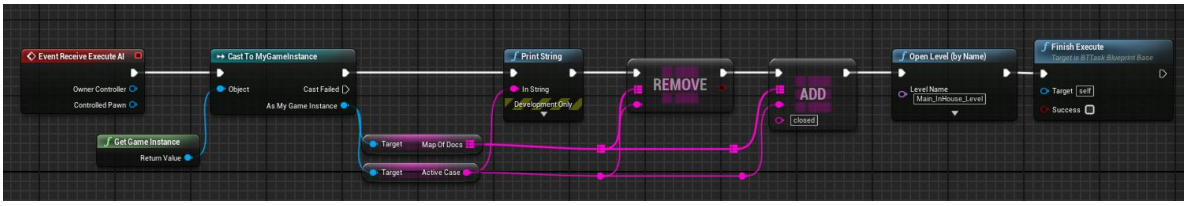


Ilustración 143: SuccesFinishNegotiation

4.6.6.3.6 Propuesta de pista (IA)

Si se detecta que se ha mostrado una pista se ejecuta la siguiente rama del árbol de comportamiento.

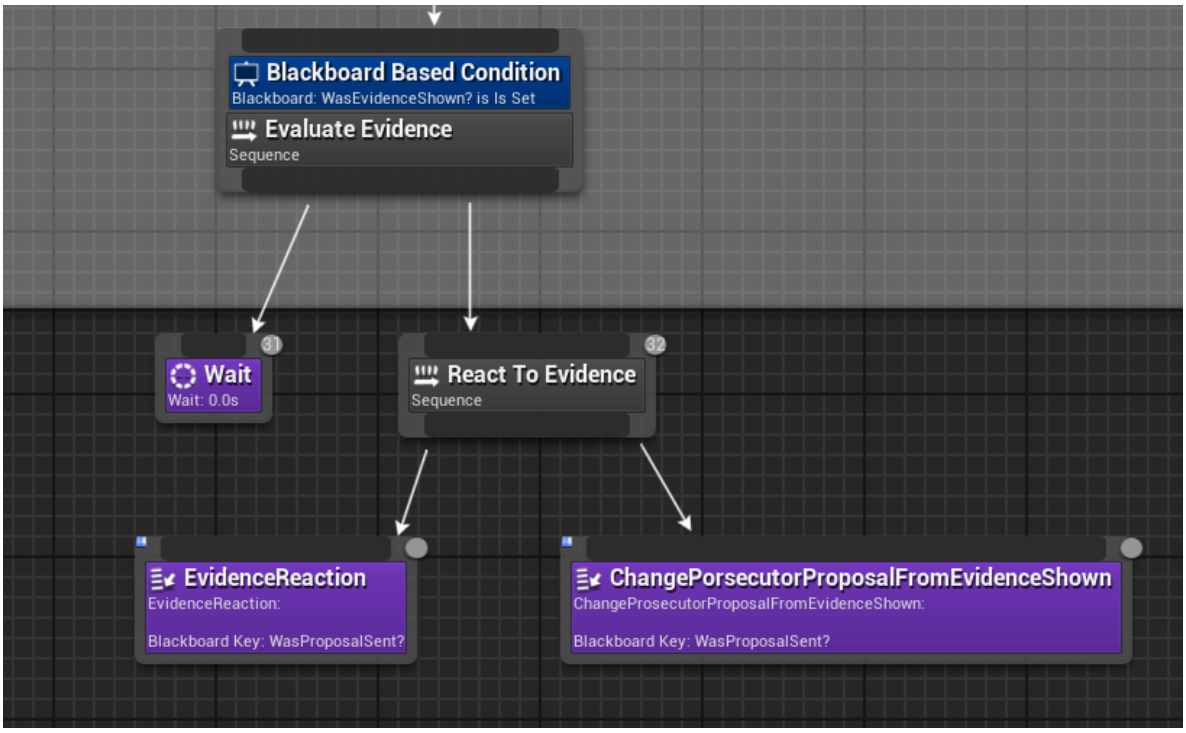


Ilustración 144: Rama pistas del árbol de comportamiento

Al igual que en el caso de las propuestas existe un evento en "HarryCarter_ReactToEvidence" que se encarga de detectar cuál de las pistas ha sido seleccionada y mostrar el dialogo correspondiente seguido de la variación de la pena. Para ello se realiza un *switch* con las opciones de los botones de las pistas y en cada opción se ejecuta una secuencia de operaciones distinta. Posteriormente se guardan los valores nuevos de la estructura del fiscal en *MyGameInstance*, se comprueba si se ha superado el nivel de enfado y se vuelve a la negociación.

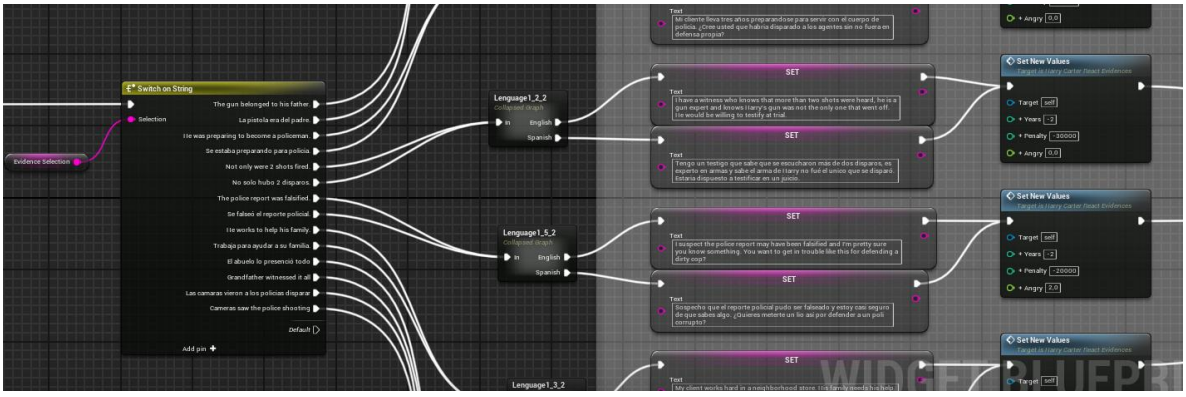


Ilustración 145: selección de dialogo en función de pista mostrada

Por supuesto se elimina la pista elegida como se explicó anteriormente.

4.6.6.3.7 Propuesta de charla (IA)

Si se detecta que se ha iniciado una charla se ejecuta la siguiente rama del árbol de comportamiento.

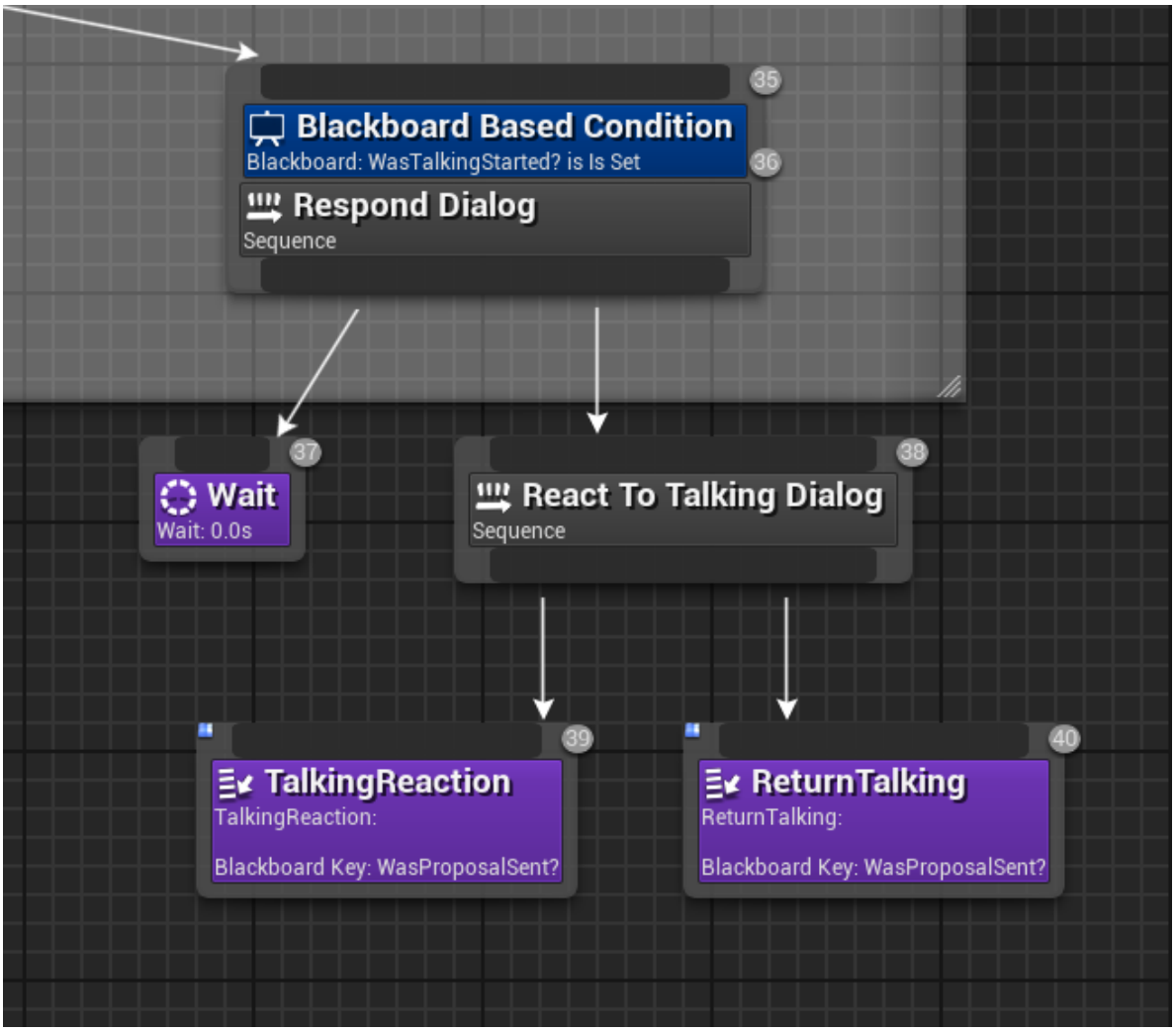


Ilustración 146: Rama charla del árbol de comportamiento

El funcionamiento de la elección de una opción de charla es idéntico al funcionamiento de las pistas por lo que queda explicado en el apartado anterior. La tarea ReturnTalking establece los valores de las variables de la *blackboard* a los necesarios para continuar la negociación como se ha explicado anteriormente.

5 CONCLUSIONES Y LÍNEAS DE TRABAJO FUTURAS

5.1 FUNCIONALIDAD DEL PROYECTO

Este proyecto tiene como objetivo el desarrollo de un videojuego con un acabado semiprofesional, abarcando todos los apartados del desarrollo de un proyecto de creación de una aplicación de ocio. Se ha pretendido abarcar gran parte de las fases que se tienen en cuenta a la hora de realizar un proyecto como este, desde la planificación temporal, la especificación y análisis de requisitos, el diseño del sistema, la documentación, hasta el propio desarrollo de la aplicación en un motor grafico orientado a la industria del videojuego profesional, la realización del arte mediante assets visuales, el guion o las pistas de audio.

5.1.1 Objetivos alcanzados

En este apartado se hará un resumen de los objetivos propuestos al final de proyecto, cuales han sido alcanzado y cuáles no.

- Gestión de diálogos y selección de opciones, era uno de los objetivos básicos para la realización de un videojuego del tipo aventura gráfica. Este objetivo se ha cumplido con creces generando un sistema de diálogos muy completo y fácil de usar.
- Gestión de casos y documentos varios, este objetivo era importante en la parte que incumbe a la historia que se narra en el videojuego, también se ha cumplido de forma satisfactoria.
- Gestión de documentos por caso, al igual que el objetivo anterior.
- Gestión de guardado de datos, este objetivo no era tan importante para la realización de una pequeña demo, pero si muy importante en la previsión de crear un videojuego largo y completo. Ha sido cumplido satisfactoriamente gracias a las facilidades de Unreal Engine 4 en el apartado de guardado y cargado de partidas.
- Gestión de cargado de datos, es un caso similar al anterior.
- Gestión de paso de niveles, este objetivo era fundamental dado que el videojuego está compuesto de varios niveles y era muy importante pasar entre ellos con los datos adecuados, se ha cumplido al completo.
- Gestión de negociaciones, este era uno de los objetivos más interesantes y a su vez más complicados dado que suponía la implementación de una IA capaz de reaccionar al comportamiento del usuario. Ha sido sin duda el objetivo más complejo de cumplir, pero finalmente se ha cumplido como se esperaba.
- Gestión de mapa y movimiento, dadas las características del videojuego era necesario crear un método que permitirá viajar al jugador de un lugar a otro, para ello se ha creado

un sistema sencillo y usable que permite al jugador moverse entre los distintos lugares con facilidad.

- Gestión de búsqueda de información, este era uno de los objetivos fundamentales dado que con la información recogida se podría interactuar de una manera u otra con la IA, dentro de este objetivo caben gran cantidad de funcionalidades, una de las más importantes fue la creación de una base de datos ficticia que permite al usuario buscar información sobre personajes.
- Gestión de desbloqueo de pistas y conversaciones, de nuevo uno de los objetivos que requería el tipo de juego e historia que se quería contar. Se ha cumplido con facilidad gracias a las facilidades de Unreal Engine 4.

5.1.2 Entrando en el mundo de los assets de Unreal Engine 4

Uno de los logros más importantes para mí ha sido un objetivo que no se reconoció al principio del proyecto, pero que a medida que se fue avanzado se convirtió en un objetivo secundario que me resultó interesante cumplir. Este es la creación del sistema de diálogos que fue aceptado para la venta al público de la UNREAL MARKETPLACE.

5.1.3 Tecnologías

5.1.3.1 Unreal Engine 4

La realización de este proyecto ha sido mucho más sencilla de lo que esperaba gracias a todas las facilidades que aporta el sistema de Blueprints de Unreal Engine 4. Por supuesto ha sido de gran ayuda el sistema de inteligencia artificial de UE4, un sistema que desde el primer momento ha sido muy intuitivo y sencillo de utilizar, con apenas unos días de practica te permite hacer una IA funcional orientada a personajes no jugables de videojuegos. Otra de las características que me ha llamado la atención de UE4 es lo sencillo que es su sistema de guardado y cargado de partidas que con dos sencillas clases te permite crear un sistema de guardado totalmente profesional y refinado.

Quizás una de las mayores dificultades que ha supuesto UE4 es la gran cantidad de información que transmite en su interfaz, pero con el tiempo fue sencillo acostumbrarse y empezar a usar su editor de forma fluida.

5.1.3.1.1 Sistema de Blueprints de UE

Unreal Engine propone una forma de programar diferente al resto. El sistema de nodos de UR4 desde fuera parece demasiado simple y con pocas posibilidades en un principio, pero una vez te sumerges un proyecto complejo y orientado a la industria del videojuego descubres que se trata de un sistema super complejo con unas capacidades altísimas con la ventaja de ser extremadamente sencillo de aprender y de utilizar. Este sistema está compuesto de una enorme cantidad de nodos y funciones que ahorran muchísimo trabajo al programador.

5.1.3.1.2 Sistema de IA de UE

Después de varias búsquedas de plugins y bibliotecas para realizar la IA incluida en el videojuego opté por el sistema de IA de UE4, un sistema aparentemente sencillo, pero con cierta complejidad, este sistema, basado en un árbol de comportamiento que actúa de hilo de ejecución y una Blackboard que actúa de cerebro de la IA, permite la creación de inteligencias artificiales complejas en poco tiempo. La sencillez que ofrece a la hora de crear el árbol de comportamiento es sorprendente y gracias a ella ha sido fácil aprender y crear la IA que está presente en el videojuego.

5.1.3.2 Photoshop

Dado que no es la primera vez que usaba esta herramienta ha sido muy sencillo adaptar los conocimientos que tenía a un proyecto como este. Esto me ha permitido crear las imágenes de forma rápida y sencilla.

5.2 CONCLUSIONES

En este apartado recogeremos las conclusiones obtenidas al largo del desarrollo del proyecto.

La principal motivación para el desarrollo de este proyecto era desarrollar por completo una demo de lo que podría ser un videojuego orientado al mercado, añadiendo ciertas pinceladas de inteligencia artificial para dotar al desarrollo de un interés más orientado a la ingeniería informática más pura. El desarrollo de un proyecto a esta escala ha sido una experiencia nueva que me ha permitido acercarme a lo que sería el desarrollo de una aplicación de ocio en la industria "real". Sirviendo como medio de aprendizaje de lenguajes como C++ orientado a Blueprints y metodologías de desarrollo de inteligencias artificiales en videojuegos mediante arboles de decisión.

De esta manera el proyecto ha servido como introducción al mundo del desarrollo de videojuegos y aplicaciones de ocio, así como a la inteligencia artificial. Además, se han afianzado ciertos conocimientos adquiridos durante el grado de ingeniería informática y ha servido para comprender la gran importancia de una buena gestión de proyecto e ingeniería de software.

Los objetivos planteados al inicio del proyecto han sido conseguidos de forma satisfactoria, consiguiendo crear una demo jugable capaz de dar una idea robusta de cómo podría ser un juego final listo para el mercado.

5.3 LÍNEAS DE TRABAJO FUTURAS

Dado el gran crecimiento de la industria del videojuego y en concreto, el alza de los juegos independientes, se plantea la continuación del desarrollo con el fin de crear nuevas mecánicas, pulir ciertas partes del desarrollo, crear nuevos niveles y finalizar la creación de un videojuego completo con la capacidad de generar beneficios.

Dado que este proyecto ha sido desarrollado con el PC en el punto de mira, otras líneas de trabajo pueden estar orientadas al desarrollo del videojuego para otras plataformas como consolas, modificando las interfaces y controles para que sean más accesibles.

Como conclusión, se ha construido una demostración de una aplicación de ocio que cumple con los requisitos previamente establecidos y tiene un acabado adecuado para el nivel que se espera de un desarrollo independiente. Los siguientes pasos serian la finalización de un producto completo y profesional orientado al mercado y a la industria del videojuego.

6 REFERENCIAS

[1] << Historia de los videojuegos >> [En línea]. Available: https://es.wikipedia.org/wiki/Historia_de_los_videojuegos

[2] << Aventura gráfica >> [En línea]. Available: https://es.wikipedia.org/wiki/Aventura_gr%C3%A1fica

[3] << Píxel Art >> [En línea]. Available: https://es.wikipedia.org/wiki/Pixel_art

[4] << Inteligencia artificial (videojuegos) >> [En línea]. Available: https://es.wikipedia.org/wiki/Inteligencia_artificial_(videojuegos)

[5] << Unreal Engine >> [En línea]. Available: https://es.wikipedia.org/wiki/Unreal_Engine

[6] << Artificial Intelligence – Unreal Documentation >> [En línea]. Available: https://docs.unrealengine.com/en-US/InteractiveExperiences/ArtificialIntelligence/index.html

[7] << Behavior Trees – Unreal Documentation >> [En línea]. Available: https://docs.unrealengine.com/en-US/InteractiveExperiences/ArtificialIntelligence/BehaviorTrees/index.html

[8] << Adobe Photoshop >> [En línea]. Avaliable: https://es.wikipedia.org/wiki/Adobe_Photoshop