Escuela de Doctorado
y Estudios de Posgrado
Universidad de La Laguna

MÁSTER UNIVERSITARIO EN DESARROLLO DE VIDEOJUEGOS

Trabajo Fin de Máster

“Void”: construyendo un prototipo de videojuego en tercera persona situado en el exterior y con enemigos reactivos.

“Void”: developing a third person video game prototype in an outdoor environment with reactive enemies.

Emerson Erikson Pereira Almada

D./Dña. **José Ignacio Estévez Damas**, con N.I.F. 43786097P, profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

C E R T I F I C A

Que la presente memoria titulada:

""Void": construyendo un prototipo de videojuego en tercera persona situado en el exterior y con enemigos reactivos."

ha sido realizada bajo su dirección por D. Emerson Erikson Pereira Almada con N.I.E Z0044176-K.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firma la presente en San Cristóbal de La Laguna, a 7 de julio de 2023.

Agradecimientos

En primer lugar, quiero agradecer a mi tutor Jose Ignácio Estevez Damas, por su tiempo, dedicación, ayuda y supervisión que me brindó desde el primer día y en este proyecto y por haber facilitado mi experiencia en esta universidad y este máster.

También quiero agradecer a mi familia, a la familia Canuto y a mis amigos por el apoyo, la comprensión y el cariño que me han brindado durante estos meses.

Y finalmente a los maestros por el aprendizaje.

Resumen

El objetivo de este proyecto ha sido el desarrollo de un prototipo de videojuego en tercera persona usando el motor Unreal Engine 5.1, haciendo uso especialmente de las herramientas del motor para el desarrollo de paisajes realistas así como de la implementación de árboles de comportamiento para controlar las acciones de los enemigos y sus animaciones.

Así, utilizando las herramientas para paisajes del motor, se ha creado la base para un mundo creíble que presenta diversos elementos para terrenos, vegetación y efectos atmosféricos. Además, el prototipo demuestra el uso de árboles de comportamiento para controlar el ciclo de vida y las reacciones de los enemigos, permitiendo la toma de decisiones a partir de estados del juego. También se han incorporado animaciones integradas con el árbol de comportamiento, proporcionando reacciones y movimientos fluidos y realistas a los enemigos.

Finalmente se ha experimentado con el nuevo sistema de "retargeting" de animaciones para Unreal Engine, aprovechando animaciones disponibles en librerías de terceros como Mixamo.

Palabras clave: Unreal Engine, Tercera persona, Árbol de comportamiento, Animaciones

Abstract

The objective of this project has been the development of a third-person video game prototype using the Unreal Engine 5.1 engine, making special use of the engine's tools for the development of realistic landscapes as well as the implementation of behavior trees to control the actions and animations of enemies.

Thus, using the engine's landscape tools, the foundation has been created for a believable world that features different elements for terrain, vegetation, and atmospheric effects. In addition, the prototype demonstrates the use of behavior trees to control the life cycle and reactions of enemies, allowing decisions to be made based on game states. Animations have also been incorporated into the behavior tree, providing fluid and realistic reactions and movement to enemies.

Finally, we have experimented with the new animation "retargeting" system for Unreal Engine, taking advantage of animations available in third-party libraries such as Mixamo.

Keywords: Unreal Engine, Third Person, Behavior Tree, Animations

Índice

Índice **5**
Índice de figuras **6**
Capítulo 1
Introducción **8**
1.1. Conceptos previos 8
1.1.1 La cámara 8
1.1.2 Paisaje 9
1.1.3 Árboles de comportamiento 11
1.1.4 Animaciones 12
1.2. Objetivo General 16
1.3. Objetivos Específicos 16
1.4. Estado Actual del Tema 16
Capítulo 2
Documento de Diseño de Videojuegos **22**
2.1. Concepto 22
2.2. Mecánica del juego 23
2.3. Estados del juego 23
2.4. Progreso del juego 24
Capítulo 3
Herramientas Empleadas **25**
3.1. Unreal Engine 25
Capítulo 4
Desarrollo del proyecto **28**
4.1. Paisaje 28
4.2. Árboles de comportamiento 32
4.3. Animaciones 35
Conclusiones y Líneas futuras **41**
Summary and Conclusions **42**
Bibliografía **43**

Índice de figuras

Figura 1 - Ibert Bierstadt, The Oregon Trail (1867), óleo. Extraído de [22] 17

Figura 2 - Imagen de RDR2 extráida de [22]. 18

Figura 3 - Captura de The Legend of Zelda: The Breath of Wild 19

Figura 4 - Captura de Horizon Zero Dawn extraída de [24]. 20

Figura 5 - Estado del Juego 23

Figura 6 - La ventana de Unreal 25

Figura 7 - El paisaje con las luces añadidas 29

Figura 8 - El material base y una de las funciones del material nieve 30

Figura 9 - El paisaje sin material, con material y lleno de elementos. 30

Figura 10 - El paisaje en el semi boscoso. 31

Figura 11 - El paisaje en el ambiente nevado. 31

Figura 12 - Un ambiente boscoso. 32

Figura 13 - La pizarra y sus claves. 33

Figura 14 - Un árbol de comportamiento de un enemigo. 34

Figura 15 - El controlador de inteligencia artificial 34

Figura 16 - Los zorros persiguen al personaje. 35

Figura 17 - El asset IK Retargeter, en él que se pueden ver la pose del modelo siendo ajustada a la otra pose. 36

Figura 18 - En este blend space se mezclan las animaciones referentes a la locomoción del personaje. 37

Figura 19 - El blueprint de animación de un enemigo. 38

Figura 20 - El blueprint de animación del jugador con sus estados y transiciones. 38

Figura 21 - La mezcla entre la animación apuntar y la de caminar. 39

Figura 22 - Aim Offset. 40

Figura 23 - El blueprint de animación del personaje. 40

Capítulo 1
Introducción

Este proyecto se subdivide en cuatro partes:

1. En la primera parte describiremos los objetivos que se esperan alcanzar, y además se presenta un análisis resumido sobre el estado actual del tema.
2. En la segunda parte, se incluye un documento de diseño del videojuego (GDD), describiendo los elementos básicos de este proyecto.
3. La tercera parte, aborda las técnicas y herramientas utilizadas para desarrollar el prototipo.
4. En la última parte, se presentan las conclusiones y líneas futuras.

En este trabajo de fin de máster, se conceptualiza y desarrolla el prototipo de un videojuego, que hace uso de herramientas y técnicas vistos durante el curso, siendo el enfoque elegido, la creación de un ambiente realista con un jugador en tercera persona y varios tipos de enemigos, controlando sus acciones y animaciones en los diferentes estados del juego. Todo el proyecto se basa en la utilización del motor Unreal Engine 5.1, incluyendo sus sistema de desarrollo.

El producto que he desarrollado tiene como inspiración la obra de un amigo escritor [1], de modo que usaré los conocimientos adquiridos para trasladarla a un videojuego, desde su conceptualización hasta el estado de prototipo.

1.1. Conceptos previos

Los juegos en tercera persona son juegos donde el personaje que representa al jugador es observado desde fuera, en lugar de utilizar el punto de vista del propio personaje, lo que correspondería a un juego en primera persona. Normalmente, el personaje es visto desde una perspectiva por encima del hombro o por detrás.

1.1.1 La cámara

Existen tres tipos de cámaras en este tipo de juegos:

1. **Cámara fija:** La cámara está colocada en una posición determinada y el jugador no puede cambiarla. Está programada para que cada vez que entres en una zona determinada, la cámara esté exactamente en el mismo punto. Los primeros juegos de Resident Evil son un buen ejemplo de ello: utilizaban ángulos de cámara fijos para crear tensión.

2. **Cámara con seguimiento:** Aquí, la cámara sigue al jugador mientras se mueve, el jugador no tiene ningún control sobre ella, la cámara se mueve a medida que se controla al personaje. como es el caso de la serie original de Crash Bandicoot, y aunque funciona en niveles lineales como los que presentan esos juegos, no es lo bastante flexible para entornos 3D abiertos.

3. **Cámara interactiva:** Este es el tipo de cámara en tercera persona más común hoy en día, las cámaras interactivas siguen a tu personaje pero también se permite su control. Normalmente, si el periférico es un "gamepad" se utiliza un stick analógico para cambiar la orientación de la cámara, mientras que si usamos PC con teclado y ratón, se utiliza este último. Este tipo de cámara es la más común porque proporciona el mayor control al jugador pero no son perfectas, y esto tiene un impacto en la jugabilidad y experiencia del juego porque un sistema de cámaras mal hecho puede atascarse en los objetos del mundo o dificultar la visión de lo que intentas ver [2]

En determinados juegos, la cámara interactiva tiene ventajas sobre la cámara en primera persona pues el personaje del jugador es visible por lo que acciones como: saltar, trepar o ponerse a cubierto son visibles. Por otra parte, tienen un campo de visión más amplio [3], de modo que el jugador mediante el control de la cámara puede ver al personaje principal en su totalidad y también puede contemplar el paisaje 3D desde diferentes perspectivas.

Por ello, en situaciones como "combates", la vista en tercera persona ofrece al jugador una mejor visión del conjunto, ya que puede ver todo el entorno, incluso qué oponentes están detrás o al lado de él pudiendo defenderse mejor. También permite mostrar mejor la relación entre el protagonista y el entorno permitiendo interacciones más intensas en juegos en los que el mundo del juego es una parte importante de la jugabilidad [4].

1.1.2 Paisaje

No solo la cámara influencia la jugabilidad, el mundo donde se desarrolla el juego también tiene un papel importante, ya que un mundo atractivo y creíble ayuda a establecer el tono del juego, fomenta la inmersión e invita a su exploración. Sus elementos constitutivos, como el terreno, la vegetación y los efectos

meteorológicos, son las piezas fundamentales que contribuyen a la atmósfera del juego y a la inmersión. [5][6].

Para crear un paisaje realista en un videojuego normalmente se hace uso de referencias de paisajes existentes en el mundo real, determinando las características del terreno, los elementos del entorno, la atmósfera, después se genera un **mapa de alturas**, que es una representación bidimensional de un terreno tridimensional, donde cada píxel del mapa corresponde a un valor de altura específico, o sea es una imagen en escala de grises donde los colores más claros representan elevaciones más altas y los colores más oscuros representan elevaciones más baja [7].

A continuación, cuando se genera el terreno base, son aplicadas texturas y materiales para dar vida al paisaje. Las texturas son imágenes 2D que tienen detalles como color, patrones, rugosidad, transparencia u otros atributos visuales que se aplican a las superficies de objetos 3D utilizando sus coordenadas UV añadiendo realismo y detalle a los objetos mientras que los materiales definen cómo interactúa la luz con la superficie de un objeto sus propiedades visuales, como el color, la reflectividad, la aspereza, la transparencia. La mayoría de los materiales toman una textura como parámetro y controlan el color de los objetos y el aspecto opaco o reflectante de la superficie[8][9].
Después se coloca la vegetación en terreno para aumentar el realismo y en seguida se añaden al paisaje los efectos ambientales, como las masas de agua, los efectos meteorológicos y la iluminación para mejorar la inmersión y la atmósfera. Todo esto siendo un proceso iterativo[10][11].

El paisaje es un enorme asset cargarlo a la vez en la memoria, impacta en el rendimiento del juego, para evitar que esto se suceda, se utiliza una técnica que consiste en dividir el mundo en fragmentos menores y cargar sólo los fragmentos relevantes del mundo y descargar los demás fragmentos de la memoria de forma automática aumentando el rendimiento y permitiendo una gestión más eficaz de la memoria y del nivel de detalle [12].

El Paisaje también influencia el movimiento del jugador y de los enemigos en el mundo como también ellos lo tienen en cuenta el terreno en su planificación, decisiones y acciones, tradicionalmente la inteligencia artificial no tenía en cuenta el terreno siendo guiado por algunas sugerencias estáticas del diseñador de niveles [13][14].

1.1.3 Árboles de comportamiento

El Árbol de comportamiento es una técnica utilizada en el desarrollo de videojuegos. Proporciona una estructura jerárquica para definir y controlar los comportamientos de los personajes no jugadores (PNJ / NPC) o agentes de inteligencia artificial en un juego y consiste en una estructura en forma de árbol compuesta por nodos, cada uno de los cuales representa un comportamiento específico o un proceso de toma de decisiones.

Algunos de los componentes clave de un árbol de comportamiento son:

1. Primer nodo o nodo raíz, en él se produce la primera división en función de la variable más importante, es el punto de partida del árbol de comportamiento y es responsable de iniciar el proceso de toma de decisiones.
2. Nodos compuestos, es un nodo que puede tener uno o más hijos. Procesarán uno o más de estos hijos en una secuencia del primero al último o en orden aleatorio dependiendo del nodo compuesto particular en cuestión, y en algún momento considerarán que su procesamiento está completo y pasarán el éxito o el fracaso a su padre, a menudo determinado por el éxito o el fracaso de los nodos secundarios.
   Incluyen:
   a. Nodo de secuencia: ejecuta una lista de comportamientos hijo del nodo en el orden que se han definido. Cuando el comportamiento que está ejecutándose actualmente termina con éxito, se ejecuta el siguiente. Si termina con fallo, él también lo hace. Devolverá Success si todos se han ejecutado con éxito.
   b. Nodo selector: tiene una lista de comportamientos hijos y los ejecuta en orden hasta encontrar uno que tiene éxito. Si no encuentra ninguno, termina con fallo. El orden de los hijos del selector proporciona el orden en el que se evalúan los comportamientos.
   c. Nodo paralelo: ejecuta a la vez todos sus comportamientos hijos. Esta ejecución no debe necesariamente ser paralela (en varios núcleos), sino que puede realizarse entrelazada; lo importante es que ocurre simultáneamente dentro de la misma iteración del bucle de juego. Los nodos paralelos pueden tener como política de finalización la del nodo secuencia (acabar si todos los hijos lo hacen), o la de los selectores (acabar si uno de sus nodos hijos lo hace).
3. Nodos decoradores, como un nodo compuesto, puede tener un nodo secundario. A diferencia de un nodo compuesto, específicamente solo

pueden tener un solo hijo. Su función es transformar el resultado que reciben del estado de su nodo hijo, terminar el hijo o repetir el procesamiento del hijo, según el tipo de nodo decorador.
Incluyen:
    a. Nodo condicional: comprueba una condición y determina si ejecutar sus nodos secundarios.
    b. Nodo Inversor: Invierte el resultado de su nodo hijo, convirtiendo un éxito en un fracaso y viceversa.
    c. Nodo repetidor: Ejecuta repetidamente su nodo secundario un cierto número de veces o indefinidamente.
4. Nodos de tareas u hojas: Son tareas que modifican de alguna forma el entorno del NPC, por ejemplo moverlo, atacar a un enemigo, saltar, etc. Las acciones se sitúan en los nodos terminales (hojas) de los árboles.

Los árboles de comportamiento proporcionan un marco potente y flexible para diseñar comportamientos de IA en los juegos.

Los desarrolladores pueden crear sistemas complejos de toma de decisiones que permiten a los NPC reaccionar, adaptarse y participar en diversas actividades según reglas predefinidas.

Los árboles de comportamiento al organizar los nodos jerárquicamente y definir condiciones y acciones, ayudan a crear un método eficiente para diseñar algoritmos y sistemas complejos que sean tanto *modulares*, que es el grado en que los componentes de un sistema pueden separarse en bloques individuales y luego recombinarse como también *reactivos* que es la capacidad de un algoritmo para reaccionar a los cambios de manera eficiente. Este enfoque modular mejora el juego al dar vida a los NPC, permitiéndoles interactuar con el entorno y responder dinámicamente a las situaciones del juego [15][16[17][18].

1.1.4 **Animaciones**

La animación es un proceso en el que se toman múltiples instantáneas y se manipulan mostrándolas en rápida sucesión, lo que crea la ilusión de que se están moviendo. La animación utiliza una combinación de imágenes e ilusión óptica para crear la apariencia de movimiento. El cerebro humano retiene de forma natural una imagen durante un poco más de tiempo del que realmente existe ante los ojos. Cuando este principio científico se combina con el proceso de animación (una rápida sucesión de imágenes) nuestro cerebro interpreta estas imágenes como un movimiento continuo. Gracias a ello los videojuegos pueden

presentar elementos dinámicos evolucionando en nuestra pantalla, ya que el monitor refresca la imagen suficientemente rápido para lograr el efecto.

Sin embargo, es preciso que el software modifique el “frame buffer” de video con un fotograma con la suficiente rapidez sin romper demasiado la cadencia, ya que en ese caso se percibirá en el videojuego falta de suavidad en los movimientos e incluso “saltos” y “cortes” no deseados.

Para lograr esta eficiencia en el refresco del “frame”, se utilizan técnicas que varían dependiendo de la tipología del videojuego. En videojuegos 2D, tradicionalmente se crean animaciones utilizando los denominados “sprites” que son matrices representando al actor del videojuego, en diferentes fases de su animación. Cambiando el sprite del personaje en el “frame” se consigue que éste parezca animado [19].

Sin embargo, esta técnica no sería adecuada para juegos en 3D donde la cámara puede ver a los objetos animados desde muchas perspectivas diferentes, ya que obligaría a tener una cantidad de sprites excesiva y sería muy poco flexible. Debemos tener en cuenta entonces que en los juegos 3D los actores se representan por mallas formadas por vértices y aristas, que permiten definir matemáticamente figuras en tres dimensiones. Además de la malla, el actor cuenta con texturas, que visten la Figura para pintar sus detalles. El pipeline de procesamiento gráfico, que hoy en día es implementado mediante sistemas de hardware especializados (GPUs), procesa toda esta información a gran velocidad para crear en cada frame la imagen bidimensional de esta estructura 3D. Es decir, el modelo 3D es proyectado desde un punto de vista determinado en lo que se denomina “renderizado”. Por lo tanto, aquí se hace necesaria otra técnica de animación diferente: la animación basada en esqueletos o animación esquelética. Se trata de un método de animación por computadora usado para simular por ejemplo animales vertebrados, o en general actores que pueden realizar desplazamientos y rotaciones de sus partes, manteniendo ciertas restricciones.

En este método, el personaje u objeto se representa al menos, mediante dos estructuras:

- La piel o malla, que es la representación de la superficie utilizada para diseñar un personaje.
- El esqueleto, que es una serie de partes jerárquicas e interconectadas, denominadas huesos.

Estas dos estructuras no son independientes, sino que se establece una relación de influencia entre los elementos del esqueleto y los vértices de la malla. Esta relación hace que el desplazamiento o rotación de un hueso afecte a los vértices de la malla en mayor o menor medida, deformándola.

Por lo tanto, la animación de una malla 3D mediante esta técnica, requiere en primer lugar de la construcción del esqueleto asociado al modelo 2D o 3D y de establecer la influencia entre los huesos y las diferentes partes de la malla. A continuación, se utiliza un programa de animación digital para crear en el tiempo una secuencia de puntos, donde en cada uno de ellos se adapta la posición, la escala y la rotación de los elementos del esqueleto para introducir movimiento. Esta secuencia temporal es lo que denominamos un asset de animación para una malla esqueletizada. Su creación es un proceso tedioso y delicado que actualmente se realiza por artistas experimentados que se apoyan en técnicas matemáticas para la interpolación de las transformaciones y/o mediante sistemas de captura de movimiento donde un actor real es provisto de balizas que permiten digitalizar la posición de las partes de su cuerpo y recoger esta información en tiempo real para trasladarla al esqueleto digital.

En resumen, el proceso general paso a paso para la animación de personajes 3D es el siguiente:

- Crear un modelo 3D.
- Construir un esqueleto para el personaje u objeto, teniendo en cuenta la jerarquía de sus partes en el movimiento.
- Determinar los parámetros que controlan cuánta influencia tiene el movimiento de un hueso sobre una parte específica de la malla.
- Transformar estos huesos utilizando un software y/o hardware de animación digital para poder cambiar su posición, escala y rotación.
- Registrar los movimientos de los huesos a lo largo de una línea de tiempo.

Además de la malla y el esqueleto existen otros elementos que facilitan e influencian la creación de una animación. Por ejemplo un “rig” o estructura de control para actuar sobre el esqueleto manteniendo ciertas restricciones. La analogía pueden ser los artefactos de palos y cuerdas utilizados por los titiriteros para mover sus muñecos.

- Mediante un rig, se puede utilizar cinemática directa e inversa sobre los huesos. La cinemática directa hace que un hueso siga directamente un control de movimiento, mientras que la cinemática inversa calcula el movimiento de una cadena de huesos bajo ciertas restricciones de modo que uno de los huesos llegue a encontrarse en una posición y rotación establecida. Esto se aplica principalmente a los brazos, las piernas y las

colas para garantizar que los codos y las rodillas apunten en la dirección correcta para conseguir un movimiento realista.
- Mediante el rig, se aplican restricciones a ciertos huesos cuando sea necesario, de modo que sólo se muevan en una dirección para obtener resultados suaves y realistas.

La aplicación de estas técnicas de animación resuelven problemas importantes:

- Se consigue aplicar movimiento independiente sobre un hueso, pero manteniendo las relaciones jerárquicas, Los huesos pueden moverse por sí solos para crear el efecto deseado para el personaje u objeto. Cualquier hueso relacionado también se moverá a través de la estructura jerárquica del esqueleto y con las restricciones del rig.
- Define la animación en movimientos óseos simples. Si se utilizara solo la malla poligonal, los animadores tendrían que definir sus animación vértice por vértice, un proceso muy laborioso. En cambio, con el rigging pueden hacer esto mediante simples movimientos óseos.
- Se pueden aplicar restricciones a huesos específicos para crear movimientos realistas, pudiendo guardarse para replicarlos en el futuro[20][21].

Si quisiéramos reutilizar una animación de un personaje en otro personaje, podemos utilizar un método llamado *Retargeting de animaciones*, que es el proceso de reutilizar animaciones existentes para su uso entre varios personajes, eliminando la necesidad de crear animaciones completamente nuevas compartiendo los assets de animación entre los personajes.

Hay dos formas de Retargeting de Animaciones, una en la que el esqueleto del personaje con el que se quiere compartir animaciones usa el mismo esqueleto que el personaje para el que se creó originalmente las animaciones. La otra implica un objeto intermediario llamado Rig que permite reorientar animaciones desde el Esqueleto de un personaje y pasar la información de los Huesos de ese Esqueleto a un Esqueleto Diferente usando el Rig que ambos Esqueletos comparten.

Las animaciones están vinculadas a un esqueleto, que también almacena datos sobre las proporciones iniciales de la malla esquelética utilizada para definir el esqueleto. Estos datos se almacenan como datos de traslación de huesos y el retargeting sólo redirecciona el componente de traslación del hueso siendo la

rotación proveniente de los datos de animación.

Dado que la malla esquelética original se utiliza para definir las proporciones del esqueleto, cualquier otra malla esquelética que utilice ese esqueleto y que tenga proporciones diferentes necesitará un retargeting para funcionar correctamente, Sin ella, las mallas esqueléticas de diferentes proporciones utilizarían los datos de traslación de la malla original, lo que provocaría a diferentes errores como el mapeo óseo incorrecto, poses desalineadas y distorsionadas y recortes y saltos de animación [21]

1.2. Objetivo General

El objetivo principal de este proyecto es el desarrollo de un prototipo de un videojuego haciendo uso de herramientas aprendidas durante el curso académico. El videojuego pretende ser una experiencia inspirada en la obra literaria de un autor independiente.

Se estudian especialmente algunas técnicas relativas a juegos 3D en tercera persona, como por ejemplo:

- "Retargeting" de animaciones.
- Control de actores mediante árboles de comportamiento.
- Técnicas de construcción de terrenos exteriores 3D.

1.3. Objetivos Específicos

Los objetivos definidos para este proyecto son:

1. Crear un paisaje con diferentes ambientes.
2. Animar diferentes personajes en la escena, utilizando el estado del juego para cambiar y combinar estas animaciones.
3. Controlar las diferentes acciones de los personajes mediante árboles de comportamiento.

1.4. Estado Actual del Tema

En la actualidad existen muchos juegos donde las técnicas empleadas en el prototipo que he construido son empleadas de maneras increíblemente sofisticadas. Destacaron en su lanzamiento juegos como Horizon Zero Dawn

(paisajes e IA de los enemigos), The Legend of Zelda: Breath of the Wild, Read Dead Redemption 2, The Last of Us II, y muchos otros. Estos juegos son relativamente recientes, porque las mejoras en el hardware de las consolas y los PCs, con sus potentes GPUs y nuevas técnicas de renderizado, han posibilitado realmente la creación de paisajes tan detallados.

En particular Read Dead Redemption 2 (RDR2) ejemplifica muchos aspectos de la tecnología utilizada en la creación de paisajes. Una de las primeras lecciones es que este videojuego, en realidad no pretende generar un paisaje realista, sino la pintura de un paisaje realista, manteniendo el dinamismo de los elementos de la naturaleza.

En [22] podemos observar algunas pinturas que reflejan el ambiente de RDR2. En la Figura 1 tenemos una de ellas.

Figura 1 - Ibert Bierstadt, The Oregon Trail (1867), óleo. Extraído de [22]

Figura 2 - Imagen de RDR2 extráida de [22].

En la Figura 2 se puede ver el videojuego RDR2 mostrando el estilo de pintura al óleo. Vemos cómo los videojuegos no intentan definir un paisaje realista sino que establecen un estilo estético sobre el que tratan de crear los paisajes.

Lo vemos también en The Legend of Zelda: Breath of Wild, donde el estilo artístico es diferente y más impresionista (Figura 3)

Figura 3 - Captura de The Legend of Zelda: The Breath of Wild

En las dos figuras presentadas observamos una profusión de elementos: el propio suelo irregular con sus caminos, colinas, barrancos, árboles, rocas, vegetación etcétera. Pero dar realismo a la escena no es la función más importante de estos elementos. En realidad, gracias a ellos es habitual que el paisaje forme parte de las mecánicas del juego. Lo más sencillo de entender es cuando determinados obstáculos del paisaje impiden el paso del personaje, pero hay muchos otros ejemplos.

En Horizon Zero Dawn, el personaje necesita los arbustos y la vegetación para esconderse y aproximarse a los enemigos sin que estos detecten su presencia. En la Figura 4 podemos observar como un tipo de vegetación determinado que es suficientemente alto permite al jugador ocultarse.

En RDR2, las pisadas en el terreno son creadas por los personajes y son pistas que el jugador puede aprovechar [23]. Por lo tanto, vemos como un paisaje en un videojuego no debe ser simplemente un artificio para poner a prueba el hardware gráfico del sistema, sino que tiene que tener un objetivo más allá del puro realismo, ya sea estético, como parte de la historia que se cuenta o bien como elemento de las mecánicas que permiten progresar.

Figura 4 - Captura de Horizon Zero Dawn extraída de [24].

En cuanto a las técnicas utilizadas hay que decir que son bastante sofisticadas tratando de exprimir todas las posibilidades del hardware moderno. En [24] podemos encontrar un análisis de los complejos procesos empleados en RDR2. Sin entrar a fondo en este análisis podemos apreciar dos tipos de técnicas.

- Actuaciones dentro del proceso de renderizado. Se realizan a lo largo de las diferentes etapas del pipeline gráfico de la GPU. Una escena requiere múltiples aplicaciones del pipeline, para tratar diferentes aspectos de la escena. Por ejemplo, en RDR2 se calcula un cubo del entorno para procurar iluminación apropiada a ese entorno sobre las geometrías que se renderizan. Tras obtener el cubo se aplica para en el proceso de iluminar la escena. Se emplea un G-Buffer para calcular por separado diferentes propiedades de la imagen como el albedo, la respuesta de los materiales metálicos, etcétera y todo se combina en el frame final.
- Actuaciones sobre el frame ya renderizado, a nivel de píxel para crear efectos. Estas técnicas son mucho más económicas computacionalmente pero son esenciales para lograr el aspecto final deseado en las fases de posprocesamiento.

Una de las tecnologías centrales de IA que está presente en gran parte de los videojuegos modernos es el árbol de comportamiento, fueron popularizados en gran parte por Halo 2 y ahora es frecuente su uso en los juegos actuales como Alien Isolation, Far Cry 4, BioShock Infinite, Spec Ops: The Line, Far Cry Primal, Halo 3, Tom Clancy's The Division .

Tradicionalmente, cuando se diseñaron por primera vez los árboles de comportamiento, la idea era que cada frame del sistema ejecutaría una actualización desde la raíz llamada tick. Luego, el sistema recorre hacia abajo el árbol para encontrar el nodo activo y vuelve a verificar si otros nodos deberían estar activos en el camino hacia abajo, pero es costoso, principalmente cuando el árbol empieza a tornarse grande, por lo que actualmente el árbol conserva una referencia al nodo activo y procesa el tick y al finalizar el comportamiento en este nodo, volverá a evaluar qué hacer. Adicionalmente un concepto introducido por Halo fue hacerlos más orientados a eventos, significando que el árbol pueda cambiar rápidamente de un nodo determinado para volver a evaluar dónde se encuentra en el árbol en caso de que ocurra un evento en el mundo del juego.

Además de todo esto, se introdujo un componente conocido como pizarra, que se usa para almacenar datos útiles que el árbol usa como parte del comportamiento de múltiples nodos. Con la pizarra almacenando las informaciones se minimiza la necesidad de cálculos repetidos y la reutilización de la misma pizarra en múltiples árboles, posibilitando la compartición de la información entre diferentes agentes de IA.

Su popularidad en la industria, se debe a su escalabilidad, legibilidad visual, su reusabilidad y sus optimizaciones. En este sentido, el paso del procesamiento de ticks a la utilización de anulaciones basadas eventos ha permitido que los árboles de comportamiento se optimicen más con el tiempo y el uso de pizarras ayuda a mantener la memoria al mínimo, ya que permite que los datos se compartan entre nodos, así como entre diferentes IA que ejecutan el mismo árbol de comportamiento.

El uso de anulaciones basadas en eventos fue fundamental para resolver los problemas de rendimiento que tenía el juego Spec Ops: The Line al transferirlo a las consolas [25][26[27][28][29].

Capítulo 2
Documento de Diseño de Videojuegos

Un documento del diseño se mostrará a continuación:

2.1. Concepto

Título	Void
Género	Acción y aventura, videojuego de disparos en tercera persona
Plataforma	PC
Sinopsis de jugabilidad y contenido	Salva al mundo de misteriosas criaturas que amenazan con la extinción a la humanidad. Void es un juego de acción y aventura en el que el jugador asume el papel de un soldado meta humano, que tiene la misión de resolver el misterio de la aparición de unos monstruos que amenazan a la humanidad. Esto le obligará a explorar el mundo en búsqueda de pistas y combatir a los diferentes enemigos.
Público	Jugadores casuales de entre 15 y 30 años
Mecánica	En Void el jugador recibe misiones y explorará regiones, luchando contra enemigos, resolviendo acertijos y obteniendo pistas para comprender el mundo y las criaturas y en el proceso adquiere nuevos ítems y habilidades. En el prototipo, el objetivo será explorar e interactuar con el mundo y combatir los enemigos que aparezcan.
Tecnología	Unreal Engine 5.1.

2.2. Mecánica del juego

Cámara	El juego será en tercera persona.
Periféricos y Controles:	Teclado, ratón, altavoz/auriculares **Mover el personaje:** las teclas WASD. **Saltar:** Espacio. **Apuntar**: Clic derecho del ratón. **Disparar:** Clic izquierdo del ratón.

Siendo que el juego es corto y se puede completar en 10-15 minutos, no hay un sistema de guardado y carga.

2.3. Estados del juego

Un estado del juego se refiere al lugar en donde se encuentra el jugador durante el juego, es decir, si el jugador está en el Menú Principal, si está jugando un Juego Multijugador, si está en el Menú de Pausa. Los diagramas representan visualmente las relaciones entre los estados.

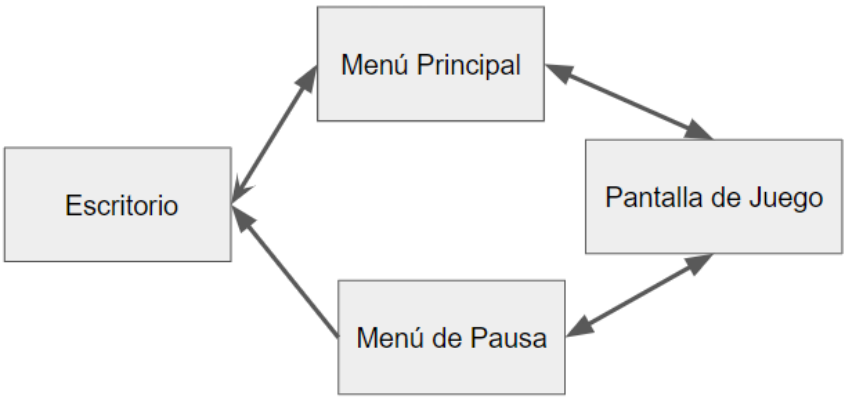

Figura 5 - Estado del Juego

Tendremos un menú principal con los siguientes elementos:

- Play, comienza un nuevo juego.,
- Exit, termina el juego y saldríamos de la aplicación;

El menú de pausa teria los elementos:

- Resume, continua el juego.
- Exit, termina el juego y saldríamos de la aplicación;

como podemos observar en la figura 5, al iniciar el juego se muestra el menú principal donde podemos iniciar un juego y pasaremos a la pantalla de juego. De este estado podemos transicionar al menú principal en el caso de que el jugador muera y al menú de pausa donde podrá salir del juego hacia al escritorio o continuar a jugar y volver a pantalla de juego.

2.4. Progreso del juego

Es una lista secuencial o por un diagrama de flujo de los eventos o niveles que el jugador debe de pasar para progresar en el juego.

El progreso del juego se desarrolla del siguiente modo::

1. Ir a la cabaña.
2. Recoger el artefacto especial.
3. Encontrar el camino para el bosque.
4. Derrotar a los enemigos antes de que estos eliminen al personaje del jugador.

Capítulo 3
Herramientas Empleadas

En esta sección, hablaremos de las herramientas utilizadas en el desarrollo de este proyecto.

3.1. Unreal Engine

Unreal Engine es un motor de juego creado por la compañía Epic Games, mostrado inicialmente en el shooter en primera persona Unreal en 1998.
Al inicio se desarrolló principalmente para los shooters en primera persona, se ha utilizado con éxito en una variedad de otros géneros, por su código escrito en C++, el motor presenta un alto grado de portabilidad y es una herramienta utilizada actualmente por muchos desarrolladores de juegos [30].

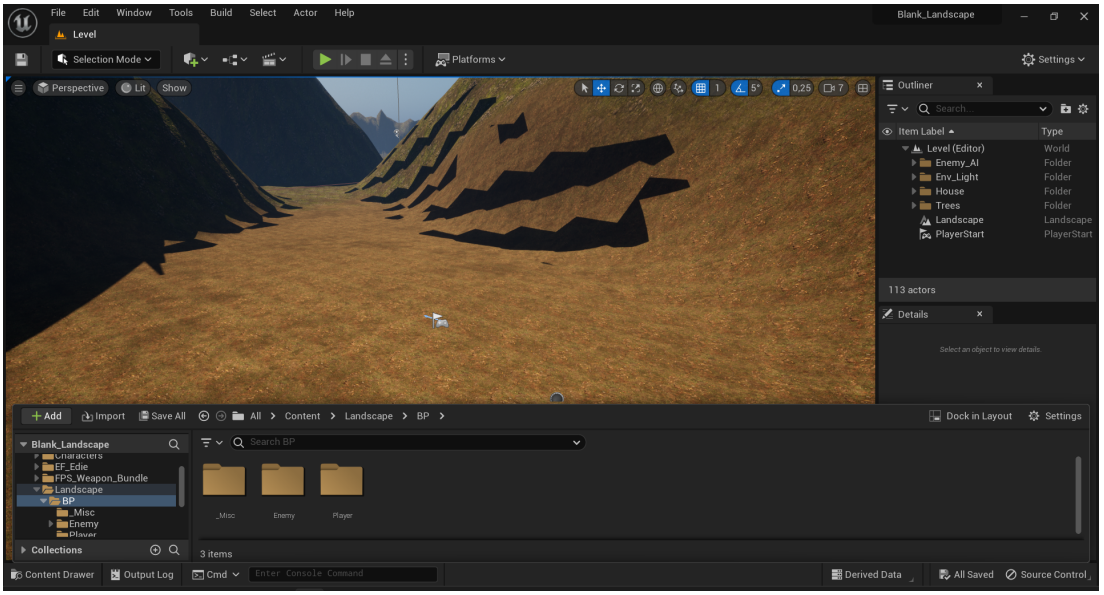

Figura 6 - La ventana de Unreal

El motivo por el que elegí este motor fue por mi deseo de realizar un proyecto en con este motor, dada su popularidad y porque en muchos aspectos representa el estado del arte de la tecnología de videojuegos. Está diseñado para juegos para diferentes modos de juego, por ejemplo juegos en primera persona o en tercera persona. La elección de los “templates” de partida ayuda a construir prototipos

porque incluye una gran cantidad de elementos necesarios para el juego. Además dispone de una gran cantidad de "assets" ya preparados en el denominado "*starter content", lo que* ayuda al desarrollador. Además cuenta con la tienda de Epic Games y una pasarela integrada a Quixel, sitios donde se pueden descargar infinidad de assets, personajes, texturas, etcétera, tanto gratuitos como de pago. La documentación es extensa, pero dado el tamaño del motor es complicada de manejar y algunas veces no está suficientemente clara y actualizada, pero dado el gran número de usuarios existen muchos videos de aprendizaje y artículos explicando cómo implementar funcionalidades. Por todo ello, me gustaría aprender a fondo el funcionamiento de este motor y deseo en el futuro trabajar con el mismo en más proyectos.

Unreal Engine puede manejar más polígonos y más efectos de partículas, permite utilizar shaders más complejos y avanzados y viene de serie con mejores herramientas en términos de calidad gráfica. También tiene su Game Framework, con eso debemos entender que tiene característica como: simulación física, render, sonido, gestión de la entrada, animación, efectos de partículas y muchas cosas más, que podemos usar como queramos, trae abstracciones de mayor nivel, muy comunes en videojuegos, como: game managers, level managers, controlador de personajes, controlador de IA, sistemas de salud o de daño, o ayuda a la animación de apuntado del arma. El motor también prescribe una arquitectura, lo que ahorra trabajo y sirve de guía a la hora de saber dónde colocar cada cosa.

El único pero es que su game framework está muy orientado al tipo de juegos que se suelen hacer con ese motor - como shooters y juegos de acción-aventura en primera o tercera persona—, lo que quiere decir que si se pretende hacer un juego de ese tipo o similar, como un walking simulator— el desarrollo con Unreal Engine es bastante cómodo. Por el contrario, las dificultades se pueden incrementar notablemente si se pretende desarrollar otros género [31].

También proporciona un conjunto de herramientas, editores y sistemas, entre ellos tenemos [32]:

- El editor del mundo, es una herramienta que permite a los desarrolladores construir y diseñar mundos de juego. Posee una interfaz intuitiva y un conjunto de funcionalidades, que permite la creación, ubicación y manipulación de assets, terreno, iluminación (Figura 6). siendo modular permite crear assets reutilizables , y colocarlos fácilmente en el mundo del juego fomentando la eficiencia, la coherencia, y reutilizabilidad de los assets ahorrando tiempo y esfuerzo en el proceso de desarrollo [33].

- El editor de Blueprint permite a los usuarios crear y manipular scripts visuales llamados Blueprints. Estos consisten en nodos que representan acciones, eventos, variables y condiciones, que se pueden conectar para crear una lógica de juego compleja. Una de las mayores ventajas del editor de blueprints es su accesibilidad y su integración a la perfección con otros sistemas de Unreal Engine, como el sistema de animación, el sistema de física y el editor de materiales. Se puede también mezclar los blueprints con códigos en C++ usando entornos como Visual Studio, jet Rider u otros entornos [34][35].

- El editor de materiales, proporciona una interfaz poderosa y fácil de usar para crear y editar materiales. Su sistema basado en nodos, vista previa en tiempo real, amplia biblioteca de nodos e integración perfecta con otras funcionalidades del motor ahorra tiempo y facilita el desarrollo [36].

- El editor de paisajes, es una herramienta que permite a los desarrolladores crear entornos exteriores amplios y detallados. Con su interfaz intuitiva y funciones avanzadas permite a los diseñadores esculpir terrenos, agregar follaje y crear paisajes exteriores proporcionando herramientas para controlar los aspectos visuales del terreno, como la iluminación, la oclusión ambiental y integracion con otros sistemas del motor, como el sistema de follaje, la simulación física y el pipeline de renderizado [37].

- Los editores de animación, están compuestos por editores de assets de animación especializados que cuentan con herramientas de animación sólidas que puede usarse para trabajar con malla esqueletizadas y otros assets de animación, posibilitando crear animaciones de personajes, interacciones dentro de los niveles entre otros. Uno de los assets más importante es el Animation Blueprint. Se trata de un blueprints especializado que controla la animación de una malla esquelética durante la simulación o el juego. Dentro del Editor de blueprints de animación, se pueden crear máquinas d estado jerárquicas para definir las transiciones entre animaciones, combinar animaciones por mezcla, controlar los huesos de un esqueleto o crear una lógica que definirá la pose de animación final para que una malla esquelética la use[38][39].

Capítulo 4
Desarrollo del proyecto

En esta sección, una vez que hemos presentado los conceptos y el documento de diseño que rige nuestro proyecto, presentaremos el proyecto en sí y cómo transcurrió su desarrollo.

4.1. Paisaje

Desde mi punto de vista, el paisaje es uno de los elementos del juego más importante. Me acuerdo de la primera vez que jugué The Legend of Zelda: Ocarina of Time, y quedé fascinado por su paisaje, pasear por Hyrule montado en un caballo, explorar los diferentes territorios de Hyrule. También me causó impresión el mundo detallista y frenético del Grand Theft Auto: San Andreas.

Para este prototipo tuve como referencia principal un lugar cerca en mi barrio de Ponta d' Agua en Cabo Verde, el lugar es un valle con rocas y que cuando llueve se llena de vegetación y se pueden observar diferentes animales, entonces quise transmitir la sensación que como niño sentía cuando paseaba por ahí. A esta referencia le añadí elementos de otros lugares, como montañas con nieve y sitios boscosos para que el paisaje final tuviera diferentes ambientes.

Obtenida esta referencia, usé el motor Unreal Engine para implementarlo. A partir de un proyecto de tercera persona, añadí un nuevo nivel vacío donde en el modo paisaje, creé un nuevo paisaje de tamaño medio con una resolución de 505 x 505, 63 quads, una sección, y un tamaño de componentes 8x8 totalizando 64 componentes según las recomendaciones de Epic [40].

Hecho esto obtenemos un mapa de alturas y un paisaje plano sin ninguna iluminación. Entonces usando el *Mezclador de luz ambiental* añadimos una *luz atmosférica* y *Atmósfera del cielo* y obtenemos un cielo azul, después se agregó una luz solar que nos posibilita tener iluminación indirecta derivada de la atmósfera en el paisaje y por fin añadimos el *Exponential Height Fog*.

Después definí todas las fuentes de luz como "móviles", de forma que la iluminación completa se calcula en tiempo real, y activé el atributo *captura en tiempo real* de la luz del cielo para que cuando se altere cualquier propiedad de las luces, que la escena se actualice. También activé el atributo de la luz direccional que permite que se coordine la representación de la esfera solar con la

dirección de la luz direccional que ilumina el paisaje, tomando también el color adecuado en función de su posición respecto al horizonte (Figura 7).

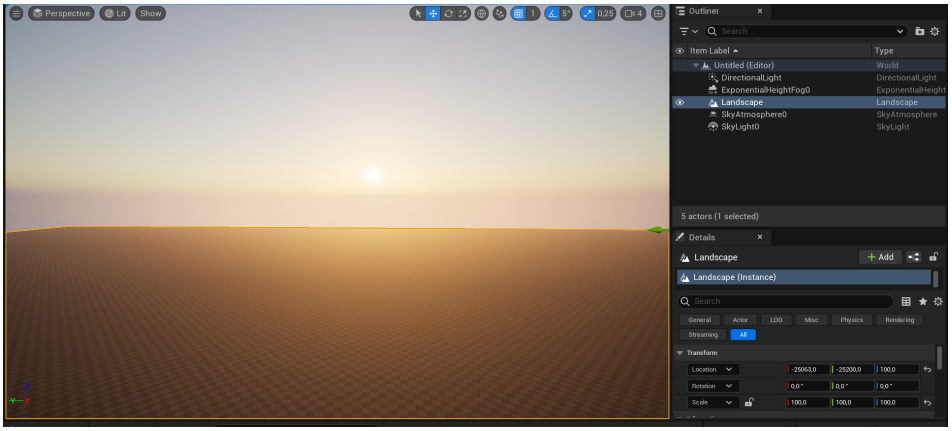

Figura 7 - El paisaje con las luces añadidas

A continuación, esculpí el terreno, construyendo montañas y valles, alterando los valores del mapa de alturas. Habiendo obtenido el terreno base, pasé a crear materiales para aplicar al paisaje. Las texturas utilizadas para crear los materiales las obtuve de la librería Quixel, utilizando la herramienta Bridge[41] del editor. Descargadas las texturas, creé un material base para el paisaje y para modularizar, desarrollé funciones de material, una para cada ambiente. Estas funciones de material se mezclan para obtener el material final.

Las funciones de materiales toman como valores de entrada las texturas que son procesadas utilizando nodos de materiales con parámetros, para terminar construyendo las entradas del nodo *Make Material Attributes* [42] para generar el shader del material con las propiedades deseadas (Figura 8).

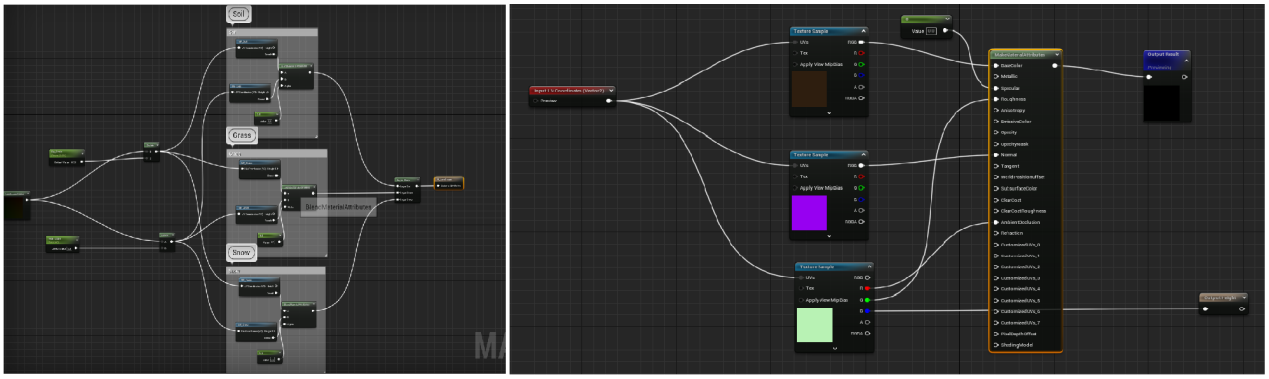

Figura 8 - El material base y una de las funciones del material nieve

A continuación, aplicamos los materiales al terreno y lo llenamos con árboles, rocas y otros elementos para aumentar su realismo (Figura 9). Las árboles fueron generados usando el software Tree It [43], las rocas fueron obtenidas del Quixel Bridge.

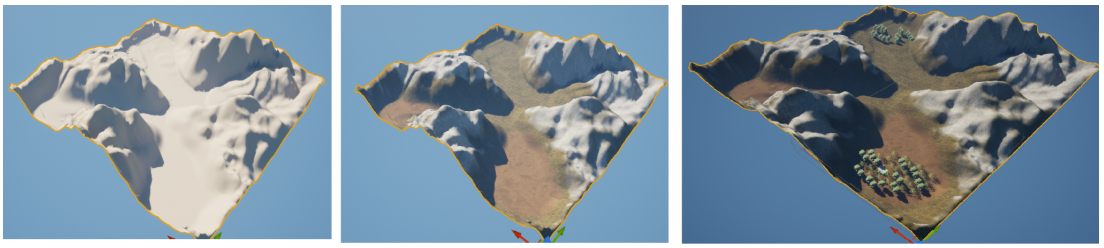

Figura 9 - El paisaje sin material, con material y lleno de elementos.

En la Figura 10, 11 y 12 observamos el paisaje desde diferentes puntos.

Figura 10 - El paisaje en el semi boscoso..

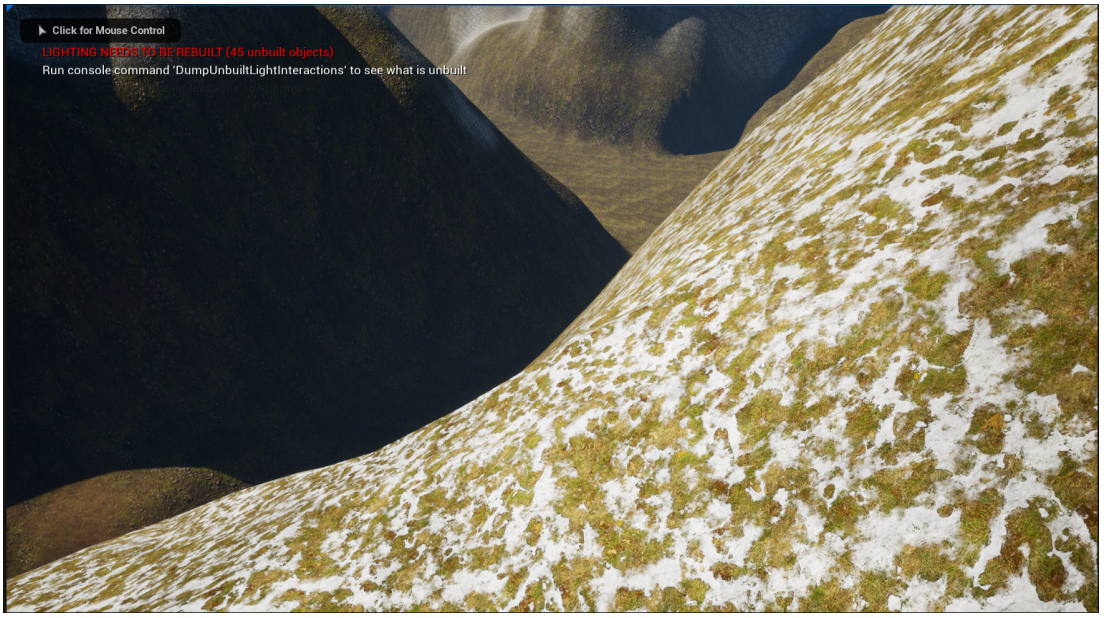

Figura 11 - El paisaje en el ambiente nevado.

Figura 12 - Un ambiente boscoso.

4.2. Árboles de comportamiento

Con el paisaje terminado, programamos la inteligencia artificial de los enemigos y las acciones a realizar mediante un árbol de comportamientos.

Todos los enemigos en el juego comparten 2 tareas bases que son la patrulla y la persecución.

Para construir el árbol tenemos que crear 3 assets:

- Una pizarra, que se utiliza para almacenar información (llamada Claves de la Pizarra) que el Árbol de Comportamiento necesita conocer para poder tomar decisiones informadas, es esencialmente el cerebro de nuestra IA. Todo lo que queremos que nuestra IA sepa tendrá una clave de pizarra a la haremos referencia.
- El árbol de comportamiento, se utiliza para ejecutar ramas que contienen la lógica, para determinar qué ramas deben ejecutarse.
- El controlador de inteligencia artificial, administra las acciones, el movimiento y las respuestas del personaje de IA a los estímulos dentro del mundo del juego actuando como un puente entre el árbol de

comportamiento y el entorno del juego, permitiendo un comportamiento inteligente y autónomo para los NPC[44][45][46].

Los árboles de comportamiento en Unreal Engine ejecutan su lógica de izquierda a derecha y de arriba a abajo. El orden numérico de operación se puede ver en la esquina superior derecha de los nodos colocados en la figura.

Primero añadimos el componente *Pawn Sensing* [47] al personaje, lo que le permitirá oír y ver a otros actores derivados de la clase Pawn (personajes), después creamos claves en la pizarra para poder hacer un seguimiento del jugador: si el jugador fue visto, si el NPC tiene una línea de visión con el jugador. Estas claves serán actualizadas en los blueprints [35] de los agentes, por ejemplo la clave "*has Seen Player" será* cambiada a verdadera cuando el enemigo vea al jugador y a falso cuando se aleje a una determinada distancia de él, como vemos en la figura 13.

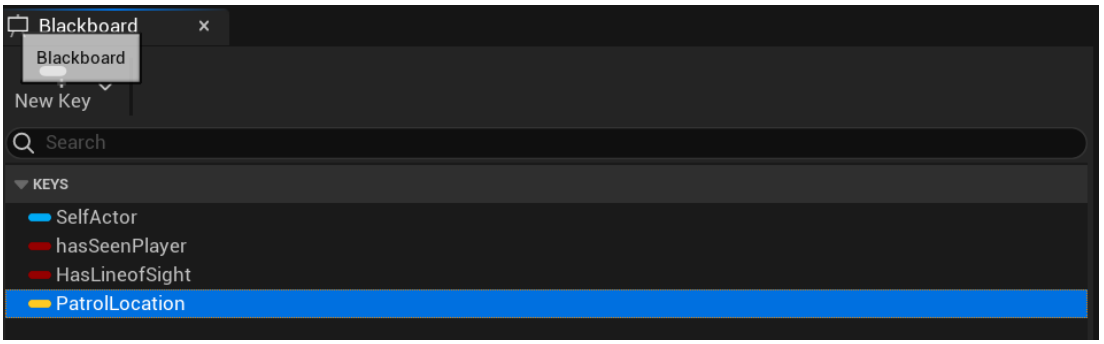

Figura 13 - La pizarra y sus claves.

En el árbol de comportamiento, tenemos el nodo raíz, un selector y dos ramas, ambas ramas tienen una secuencia y una condición. La secuencia *Look Around* se ejecuta si la clave *hasSeenPlayer* es falsa y *Chase Player* si es verdadera, el último nodo en ambas secuencias es un nodo hoja, donde fueron creadas 2 tareas (Figura 14) :

- Una para que el agente haga una patrulla, que consiste en elegir un punto aleatorio a una determinada distancia del jugador y mover el agente hacía ahí.
- Otra para perseguir, que consiste en mover el agente hacia la posición del jugador.

Además, dependiendo del enemigo se implemente una tarea que hace que huya si su salud descienda por debajo de un valor predeterminado, de modo que mientras no se recupere no atacará el jugador.

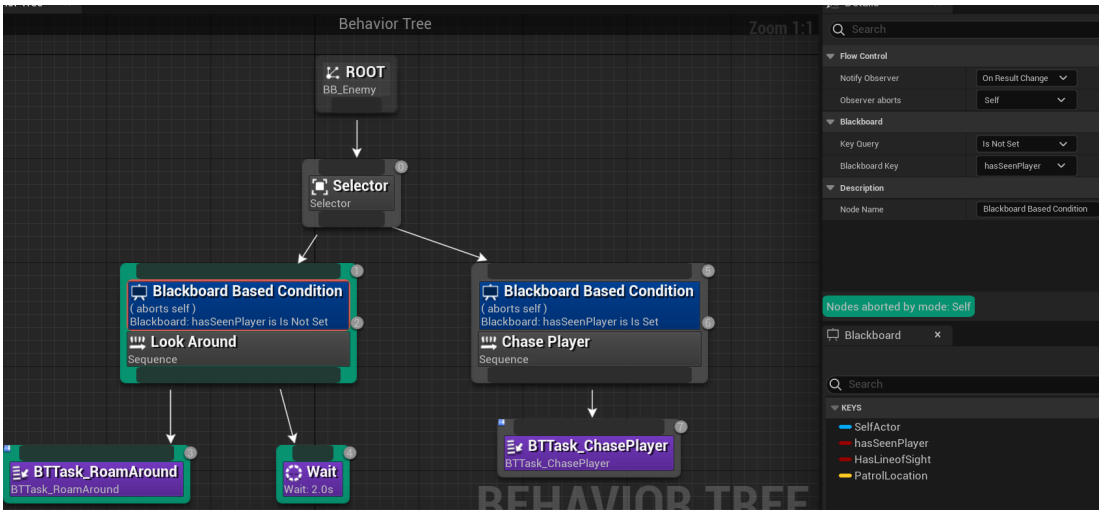

Figura 14 - Un árbol de comportamiento de un enemigo.

En el controlador de inteligencia artificial, se ordena la ejecución el árbol de comportamiento que creamos (Figura 15).

Figura 15 - El controlador de inteligencia artificial

Hecho esto, ajustamos algunas propiedades de los agentes, como dejar que el controlador gestione su rotación *Yaw*, haciendo que el enemigo gire adecuadamente y añadimos un *NavMeshBoundsVolume* a la escena para que los agentes puedan caminar por el paisaje (Figura 16).

Figura 16 - Los zorros persiguen al personaje.

4.3. Animaciones

Con los agentes de inteligencia artificial moviéndose por el paisaje tenemos que ajustar sus acciones a las animaciones.
Para este proyecto los personajes de los agentes y del jugador fueron obtenidos en el bazar del Unreal [48] y en el portal Mixamo, que es una plataforma que tiene disponible una amplia biblioteca de personajes, incluyendo las animaciones y los esqueletos [49].

La utilización de librerías de animaciones para personajes ahorra mucho tiempo de trabajo, y es muy útil en el prototipado de juegos, pero hay que solventar una dificultad: la animación importada está realizada para un esqueleto diferente al del personaje que tenemos en Unreal y para aplicarla hay que modificar las referencias a los huesos en el esqueleto original con las correspondientes al nuevo esqueleto sobre el que queremos aplicarla. Este proceso se denomina "Retargeting". Afortunadamente, Unreal cuenta con una herramienta que permite automatizar en algunos casos este proceso.

Para esto es necesario crear 2 assets:

- Un *IK Rig*, que es un sistema que proporciona un método para crear Solvers de forma interactiva que realizan automáticamente la edición de

poses para mallas esqueléticas aplicando la tecnología de cinemática inversa [50]. La cinemática inversa permite cambiar la pose, a partir de cambiar un hueso y actualizar automáticamente la cadena huesos con dependencias con el primero, manteniendo ciertas restricciones. Por ejemplo, si se mueve la mano del personaje a un punto, el resto de huesos del brazo debe seguirla, sin crear posturas antinaturales en el brazo.

- Un *IK Retargeter,* que es una herramienta que permite el mapeado de animaciones usando cinemática inversa de un esqueleto a otro, preservando las posiciones y movimientos deseados de las extremidades en diferentes personajes o modelos [51].

Para esto creamos el asset *IK Rig* asociado al esqueleto donde queremos utilizar las animaciones y después creamos cadenas de huesos que indican el hueso inicial y el hueso final. Esto proporciona flexibilidad en la reorientación de personajes con estructuras óseas muy diferentes.

Creamos cadenas para la cabeza, la columna vertebral, los brazos izquierdo y derecho y piernas izquierda y derecha y la espina como la raíz del Retargeting. Después creamos el *IK Retargeter* del modelo del que queremos copiar las animaciones seleccionamos el esqueleto del personaje donde queremos copiar la animación. Finalmente editamos las pose para que ambas coincidan en lo posible, girando los brazos en la posición y rotación deseadas (Figura 17).

Figura 17 - El asset IK Retargeter, en él que se pueden ver la pose del modelo siendo ajustada a la otra pose.

Finalmente seleccionamos las animaciones deseadas y hacemos el retargeting con un esqueleto origen y otro destino.

Obtenidas todas las animaciones que vamos a usar, creamos un *Blend Spaces 1D* [52] para nuestro personaje. La herramienta BlendSpace1D (Figura 18) permite mezclar animaciones usando un parámetro: por ejemplo, las animaciones de cuando el personaje esté parado, caminando y corriendo mediante una variablel que en este caso sería la velocidad con la que el personaje se desplaza por el paisaje.

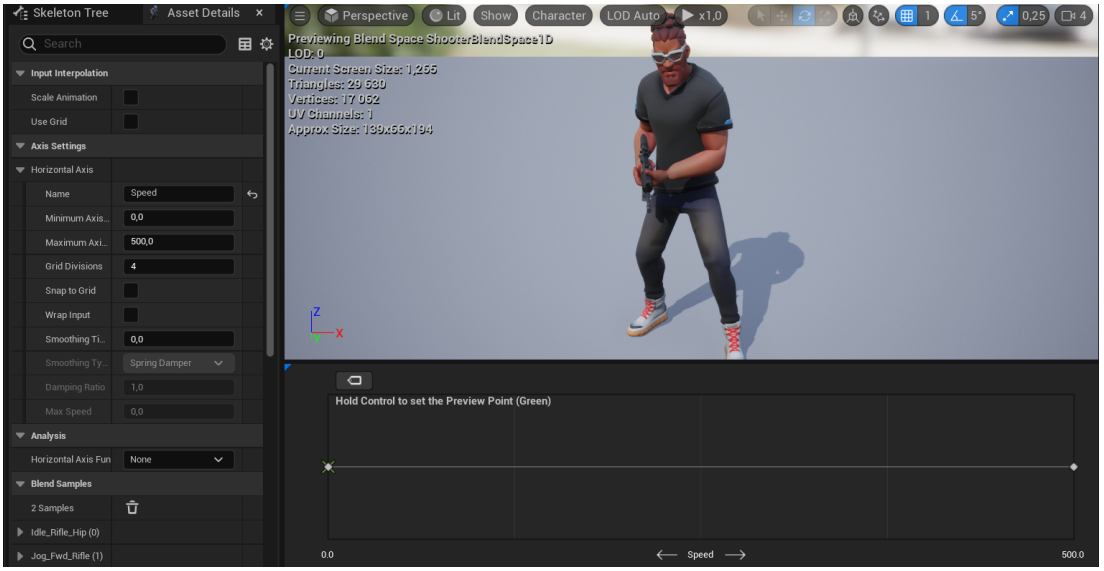

Figura 18 - En este blend space se mezclan las animaciones referentes a la locomoción del personaje.

Después, creamos un blueprint de animación para poder controlar las animaciones de la malla esqueletizada [53] y obtenemos la velocidad del personaje y lo usamos como parámetro de entrada para el blend space (Figura 19).

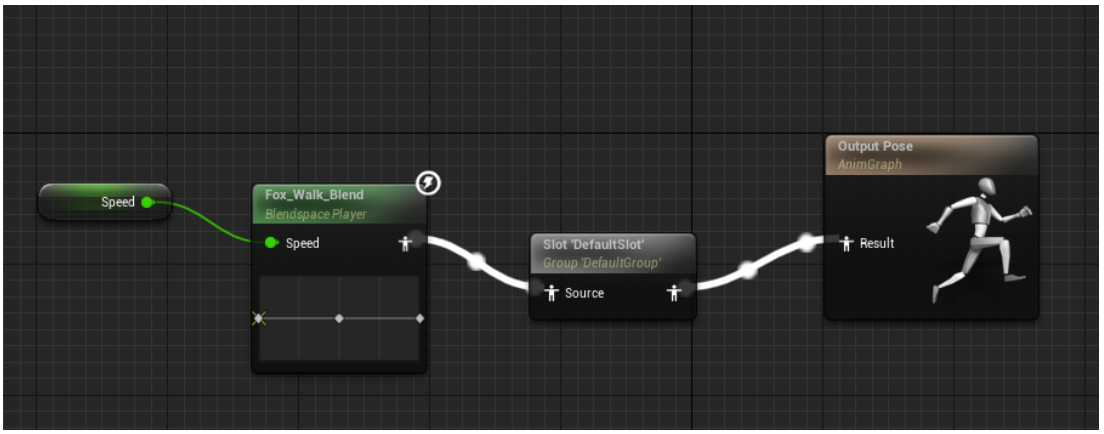

Figura 19 - El blueprint de animación de un enemigo.

En cada uno de los blueprints de animación, tenemos que crear estados y definir las condiciones de transición entre ellos. Tomando el personaje del jugador como ejemplo, tenemos un estado de animación *locomotion* donde tenemos dos estados uno para cuando el jugador se encuentre moviéndose y otro para cuando está apuntando su arma y la transición entres ellos depende de si el jugador está apuntando su arma o no (Figura 20).

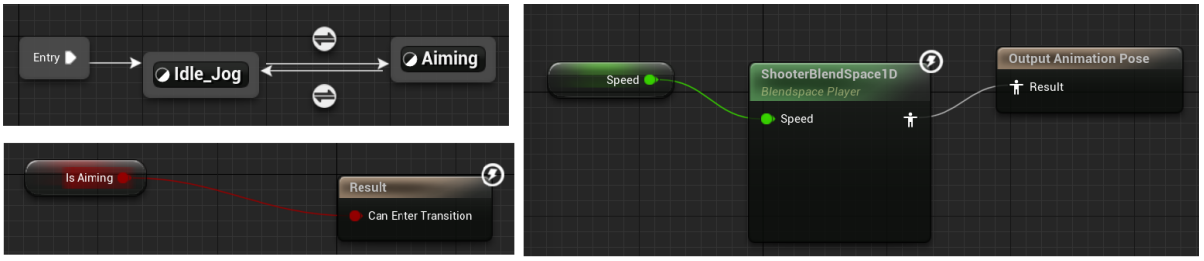

Figura 20 - El blueprint de animación del jugador con sus estados y transiciones.

Al apuntar y mover el jugador, sus piernas permanecían estáticas porque la animación no se está reproduciendo, para resolver esto, usamos el nodo *Layered Blend per Bone* que permite controlar individualmente y combinar las transformaciones óseas durante la animación, lo que resulta en un control detallado sobre los movimientos e interacciones de los personajes [39]. Definimos el hueso donde queremos aplicar la transformación y conectamos las animaciones al nodo y obtenemos el resultado en la figura 21 :

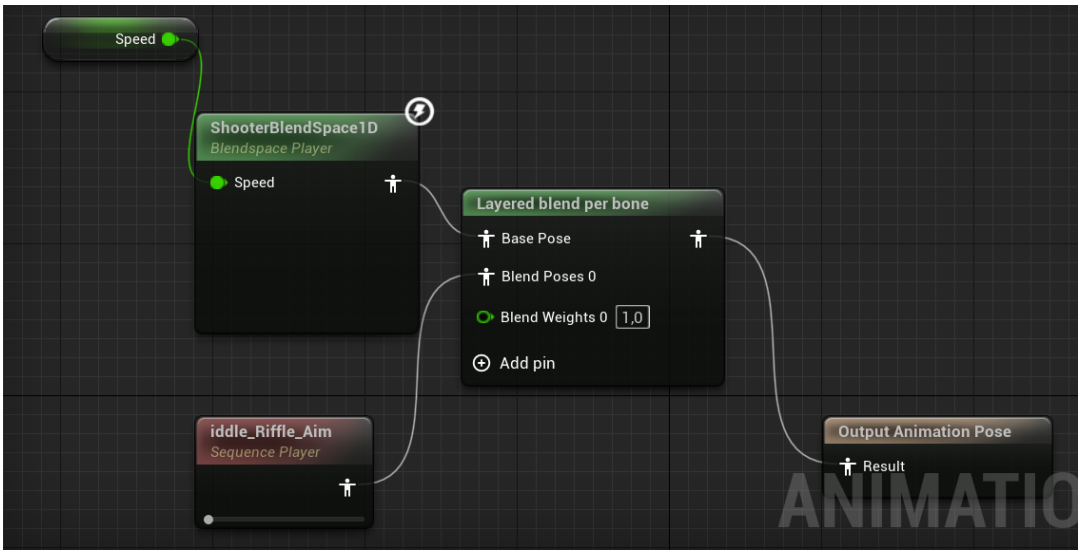

Figura 21 - La mezcla entre la animación apuntar y la de caminar.

Para ajustar cuando el personaje apunta hacia al cielo o hacia el suelo creamos un asset del tipo *Aim Offset 1D*, que es un espacio de mezcla especializado en contener animaciones aditivas del espacio de malla, que por lo general se usan para crear armas u otros espacios combinados para mirar y apuntar [54].
En él definimos los valores mínimos y máximos que queremos girar el torso que es -90 y 90 grados respectivamente y mezclamos las animaciones en que el personaje está apuntando el arma arriba, en frente y al suelo, como hicimos en Blend Space 1D (Figura 22).

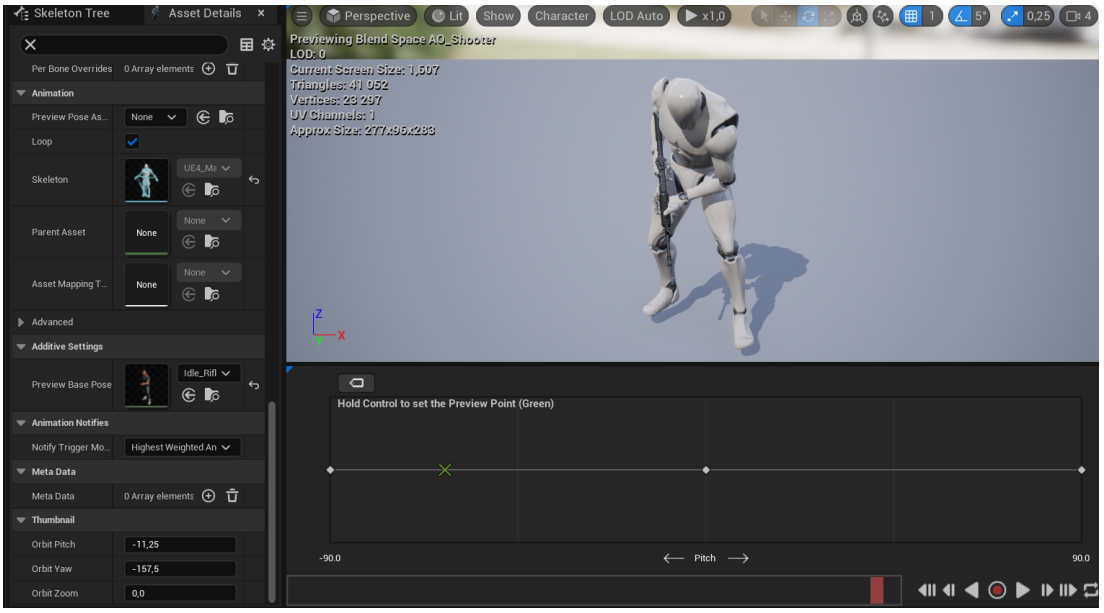

Figura 22 - Aim Offset.

Hecho esto obtenemos el valor del Pitch del personaje y lo usaremos como parámetro de entrada para el nodo Aim Offset y quedaría como na figura 23:

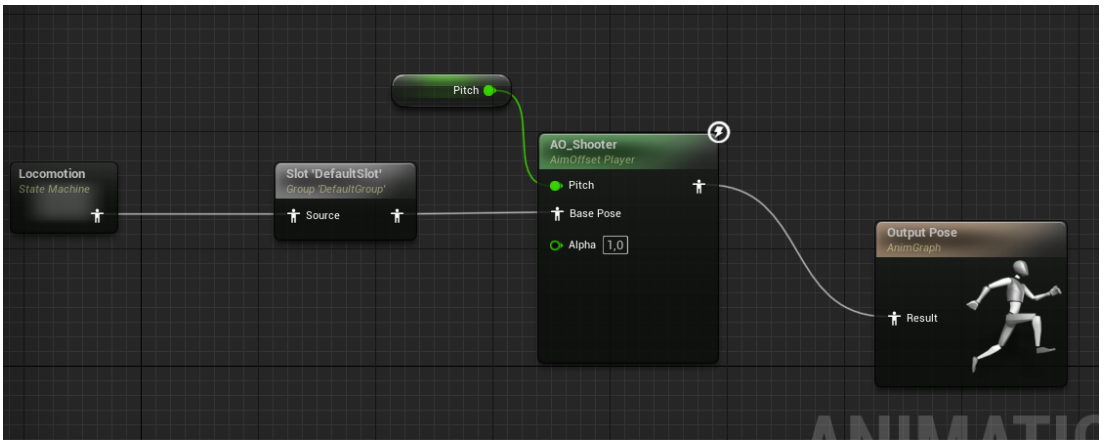

Figura 23 - El blueprint de animación del personaje.

Las animaciones y su correcta aplicación añaden realismo y fluidez al movimiento de los diferentes personajes y usando los blend spaces eliminamos la necesidad de crear estados para cada movimiento del personaje, permitiendo transiciones suaves y eficientes y flexibilidad y control de las diferentes animaciones.

Conclusiones y Líneas futuras

En este proyecto se ha conseguido desarrollar un prototipo de un videojuego que teniendo en cuenta los objetivos definidos se considera logrado. considero que un juego en tercera persona es una forma adecuada de adaptar la obra en que me basé, por permitir que se cree una conexión con el personaje al ver sus movimientos y sus animaciones y también por permitir al jugador explorar e interactuar con el mundo creando una narrativa cinematográfica.

He ganado en experiencia y entendimiento de cómo algunos componentes y partes del motor interactúan entre sí y como modularizar assets. Es un sentimiento agradable y de orgullo conceptualizar un prototipo y desarrollarlo en un proyecto del que existe un apego personal y es más ambicioso de lo que hice en los proyectos de las asignaturas del máster, reuniendo conceptos y técnicas de las diferentes asignaturas en este proyecto.

Uno de los problemas de este desarrollo fue encontrar assets, desde armas hasta animaciones. Buscar personajes que se adecuaban al estilo deseado y que tenían las características técnicas adecuadas fue muy difícil.

En el futuro se pretende mejorar y añadir más funcionalidades, animaciones y acciones a los personajes.

Las mejoras serían:

- Mejorar el *mood* del paisaje, añadiendolo más ambientes.
- Añadir más enemigos al juego y hacer que trabajen en conjunto para lograr las tareas.
- Crear tareas más complejas para los enemigos.
- Añadir animaciones de combate.
- Añadir un inventario;
- Añadir items y armas convencionais y mágicas.;
- Implementar un NPC que ayudaría al jugador a combatir los enemigos.
- Añadir más niveles y misiones.

Pretendo continuar en la adaptación del libro, añadiendo funcionalidades que creo interesantes y después implementarlo en conjunto con mi amigo ya que él es diseñador gráfico y también pretende adaptar el libro.

Summary and Conclusions

Based on the defined objectives, I managed to achieve the intended results developing a prototype of a video game. I consider that a third person game is a suitable mode to adapt the book on which I based the prototype, because it allows a connection to be created with the character by seeing their movements and animations and also because it allows the player to explore and interact with the world creating a cinematic narrative.

I have gained experience and understanding of how some components and parts of the engine interact with each other and how to modularize assets. It was a pleasant feeling of pride to conceptualize a prototype and develop it into a project to which there is a personal attachment and which is more ambitious than what I did in the master's subject projects, bringing together concepts and techniques from the different subjects in this project.

One of the issues I faced in development was finding assets, from weapons to animations to characters that fitted the envisioned style and had the intended technical features.

In the future I intend to improve and add more features, animations and actions to the characters.

The improvements would be:

- Improve the mood of the landscape, by adding more environments.
- Add more enemies to the game and make them work together to accomplish the tasks.
- Create more complex tasks for the enemies.
- Add combat animations.
- Add an inventory;
- Add conventional and magical items and weapons;
- Implement an NPC that would help the player to fight the enemies.
- Add more levels and missions.

I intend to continue adapting the book, by adding features that I find interesting and then implementing it together with my friend since he is a graphic designer and also intends to adapt the book.

Bibliografía

[1] Monteiro, G. *Void*. En desarrollo.

[2] Stegner B.,. *First-Person Games vs. Third-Person Games: What Are the Differences?*. Make Use Of. https://www.makeuseof.com/first-person-games-vs-third-person-games-differences/

[3] PlamzDooM. *Third-Person Perspective (Concept*). Giant Bomb. https://www.giantbomb.com/third-person-perspective/3015-464/

[4] Buehler, A. *Third Person Camera View in Games - a record of the most common problems in modern games, solutions taken from new and retro games*. Game Developer. https://www.gamedeveloper.com/design/third-person-camera-view-in-games---a-record-of-the-most-common-problems-in-modern-games-solutions-taken-from-new-and-retro-games

[5] Dillon C,. *4 Reasons Why Gaming Terrain Is So Important*. Spikey Bits. https://spikeybits.com/2019/10/4-reasons-why-gaming-terrain-is-so-important.html

[6] Meier A,. *The Evolving Landscapes of Video Games*. Hyperallergic. https://hyperallergic.com/78850/the-evolving-landscapes-of-video-games/

[7] *Mapa de alturas*. ImageToStl. https://imagetostl.com/es/glosario/mapa-de-alturas

[8] *Texturing and Materials*. Game Dev Insider. https://gamedevinsider.com/making-games/game-artist/texturing-and-materials/

[9] Admin. *Difference Between Materials and Textures*. Arena Sayajigunj. https://arena-sayajigunj.com/difference-between-materials-and-textures/

[10] Tokarev k,. *Creating Natural Landscapes for Games*. 80 Level. https://80.lv/articles/creating-natural-landscapes-for-games/

[11] Tokarev k,. *Creating realistic virtual landscapes*. 80 Level. https://80.lv/articles/creating-realistic-virtual-landscapes/

[12] *World Partition*. Unreal Engine. https://docs.unrealengine.com/5.0/en-US/world-partition-in-unreal-engine/

[13] van der Sterren W,. *Terrain Reasoning for 3D Action Games*. Game Developer. https://www.gamedeveloper.com/programming/terrain-reasoning-for-3d-action-games

[14*] TTRPG Terrain: Exploring the Impact of Geography. Game Mechanics*. https://creativegamemechanics.com/2023/04/19/ttrpg-terrain-geography-dd-pathfinder-game-mechanics/

[15] Simpson C,. *Behavior trees for AI: How they work*. Game Developer. https://www.gamedeveloper.com/programming/behavior-trees-for-ai-how-they-work

[16] Sagredo-Olivenza I,. Gómez-Martín M. Flórez-Puga G. edro A. González-Calero P. *Implementación de nodos consulta en árboles de comportamiento*. Departamento de Ingeniería del Software e Inteligencia Artificial Universidad Complutense de Madrid, España. https://ceur-ws.org/Vol-1394/paper_14.pdf

[17] Ferrero. R,. *Qué son los árboles de decisión y para qué sirven*. Máxima Formación. https://www.maximaformacion.es/blog-dat/que-son-los-arboles-de-decision-y-para-que-sirven/#01

[18] Josh N,. *The role of behavior trees in robotics and AI*. Allerin. https://www.allerin.com/blog/the-role-of-behavior-trees-in-robotics-and-ai

[19] *Quick Introduction To Skeletal Animation for Video Game Development*. Night Quest Games. https://www.nightquestgames.com/quick-introduction-to-skeletal-animation-for-video-game-development/

[20] *What Is Rigging in Animation? Skeletal Animation Explained*. Adobe. https://www.adobe.com/uk/creativecloud/animation/discover/rigging.html

[21] *Animation Retargeting*. Unreal Engine 5.2 Documentation. https://docs.unrealengine.com/5.2/en-US/animation-retargeting-in-unreal-engine/

[22] Gies A,. *How landscape painting inspired the world of Red Dead Redemption 2*. Polygon. https://www.polygon.com/red-dead-redemption/2018/10/26/18024982/red-dead-redemption-2-art-inspiration-landscape-paintings

[23] Graphics Study: Red Dead Redemption 2 . I'm Geself. https://imgeself.github.io/posts/2020-06-19-graphics-study-rdr2/

[24] Wald H,. How the remnants of the old world in Horizon Forbidden West awakens a sense of wonder. Games Radar. https://www.gamesradar.com/how-the-remnants-of-the-old-world-in-horizon-forbidden-west-awakens-a-sense-of-wonder/

[25] AI and Games. *Behaviour Trees: The Cornerstone of Modern Game AI | AI 101*. Youtube. https://www.youtube.com/watch?v=6VBCXvfNlCM

[26] Isla, D. *GDC 2005 Proceeding: Handling Complexity in the Halo 2 AI*. Game Developer. https://www.gamedeveloper.com/programming/gdc-2005-proceeding-handling-complexity-in-the-i-halo-2-i-ai

[27] Valdes R,. *The Artificial Intelligence of Halo 2*. How Stuff Works. https://electronics.howstuffworks.com/halo2-ai.htm

[28] AI and Games. *The Behaviour Tree AI of Halo 2 | AI and Games #09*. Youtube. https://www.youtube.com/watch?v=NU717sd8oUc&t=230s

[29] Wibowo F,. *Implementation of Behavior Tree in Halo 2* . Readkong. https://www.readkong.com/page/implementation-of-behavior-tree-in-halo-2-itb-9414506

[30] Jensen k,. *25 Years Later: The History of Unreal and an Epic Dynasty*. PC Mag. https://www.pcmag.com/news/25-years-later-the-history-of-unreal-and-an-epic-dynasty

[31] Torres J,. (2023). *Motores de videojuegos*.

[32] *Tools and Editors in Unreal Engine*. Unreal Engine 5.0 Documentation. https://docs.unrealengine.com/5.1/en-US/tools-and-editors-in-unreal-engine/

[33] *Level Editor*. Unreal Engine Documentation. https://docs.unrealengine.com/4.26/en-US/BuildingWorlds/LevelEditor/

[34] *Blueprint Editor Reference*. Unreal Engine 5.1 Documentation .https://docs.unrealengine.com/5.1/en-US/user-interface-reference-for-the-blueprints-visual-scripting-editor-in-unreal-engine/

[35] *Unreal Engine Programming and Scripting*. Unreal Engine 5.1 Documentation. https://docs.unrealengine.com/5.1/en-US/unreal-engine-programming-and-scripting/

[36] *Unreal Engine Material Editor User Guide*. Unreal Engine 5.1 Documentation. https://docs.unrealengine.com/5.1/en-US/unreal-engine-material-editor-user-guide/

[37] *Editing Landscapes in Unreal Engine*. Unreal Engine 5.1 Documentation. https://docs.unrealengine.com/5.1/en-US/editing-landscapes-in-unreal-engine/

[38] *Animation Editors in Unreal Engine*. Unreal Engine 5.1 Documentation. https://docs.unrealengine.com/5.1/en-US/animation-editors-in-unreal-engine/

[39] Animation Blueprints in Unreal Engine. Unreal Engine 5.1 Documentation. https://docs.unrealengine.com/5.1/en-US/animation-blueprints-in-unreal-engine/

[40] *Landscape Technical Guide*. Unreal Engine 4.27 Documentation. https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/Landscape/TechnicalGuide/

[41] *Quixel Megascans*. https://quixel.com/megascans/home

[42] *Material Attributes Expressions*. Unreal Engine 4.27 Documentation. https://docs.unrealengine.com/4.27/en-US/RenderingAndGraphics/Materials/ExpressionReference/MaterialAttributes/

[43] *TreeIt. Tree Generator*. http://www.evolved-software.com/treeit/treeit

[44] *Behavior Tree Overview*. Unreal Engine 5.1 Documentation. https://docs.unrealengine.com/5.1/en-US/behavior-tree-in-unreal-engine---overview/

[45] *Behavior Tree Quick Start Guide*. Unreal Engine 5.1 Documentation. https://docs.unrealengine.com/5.1/en-US/behavior-tree-in-unreal-engine---quick-start-guide/

[46] *AI Controllers in Unreal Engine*. Unreal Engine 5.2 Documentation. https://docs.unrealengine.com/5.2/en-US/ai-controllers-in-unreal-engine/

[47] *Pawn Sensing*. Unreal Engine Documentation. https://docs.unrealengine.com/5.1/en-US/BlueprintAPI/AI/Components/PawnSensing/

48 *Unreal Engine Marketplace*. Store of UE Assets for Games and 3D Rendering. https://www.unrealengine.com/marketplace/en-US/store

[49] Sierra M,. *Mixamo y Blender: 2 soluciones para animar modelos 3D. Niixer.* https://niixer.com/index.php/2021/05/24/mixamo-y-blender-2-soluciones-para-animar-modelos-3d/

[50] *IK Rig Editor*. Unreal Engine 5.2 Documentation. https://docs.unrealengine.com/5.2/en-US/ik-rig-in-unreal-engine/

[51] *IK Rig Animation Retargeting in Unreal Engine*. Unreal Engine 5.2 Documentation. https://docs.unrealengine.com/5.2/en-US/ik-rig-animation-retargeting-in-unreal-engine/

[52] *Blend Spaces*. Unreal Engine 4.27 Documentation. https://docs.unrealengine.com/4.27/en-US/AnimatingObjects/SkeletalMeshAnimation/Blendspaces/

[53] *Blend Nodes*. Unreal Engine 4.27 Documentation. https://docs.unrealengine.com/4.27/en-US/AnimatingObjects/SkeletalMeshAnimation/NodeReference/Blend/

[54] *Aim Offset in Unreal Engine*. Unreal Engine 5.0 Documentation. https://docs.unrealengine.com/5.0/en-US/aim-offset-in-unreal-engine/