Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

GRADO EN INGENIERÍA EN TECNOLOGÍAS Y SERVICIOS DE TELECOMUNIACIÓN

ÁREA DE INGENIERÍA TELEMÁTICA

Simulador de un vehículo industrial

D. Martínez Varela Olai

TUTOR/ES: D. Xicu Xabiel García Pañeda

FECHA: Febrero 2024

RESUMEN

El trabajo presenta un simulador de un vehículo industrial, una excavadora, en un entorno natural. La intención es alcanzar de una forma fidedigna el movimiento y aspecto del vehículo mediante la aplicación de las físicas lo más realistas posibles. Además, se ha implementado la posibilidad de coger y transportar esferas, simulando tierra por el mapa.

El grueso del proyecto esta soportado por el motor de videojuegos Unity, que presenta su versión gratuita para el desarrollo de videojuegos a un nivel amateur. Lo que permite el desarrollo de simulaciones en 3D o 2D. También contaremos con una botonera y volante, de la marca Saitek, para PC que nos ayudará con parte de los controles necesarios para el correcto desarrollo del simulador y cuatro pantallas para tener una visión de 360º para una experiencia más inversiva.

INDICE

Figuras

Tablas

Catálogo de Conceptos

1. **Canvas**. Elemento de Unity que gestiona la interfaz se denomina "Canvas". Aunque sigue siendo un "GameObject" como cualquier otro, la distinción radica en que posee un componente llamado "Canvas", y automáticamente se considera que sus hijos son elementos de esta interfaz.

2. **Assests**. Cualquier archivo o recurso empleado en el desarrollo de un proyecto de juego. Los assests son elementos fundamentales que se utilizan para construir y dar vida a un juego en Unity, desempeñando un papel crucial en la creación y visualización de gráficos, modelos 3D, texturas y otros elementos necesarios para la experiencia del juego.

3. **DLC**. “Downloadable Content" o contenido descargable, se trata de un tipo de contenido adicional o complementario que se puede descargar, y añadir al videojuego, software o aplicación después de su lanzamiento inicial. Pueden ser de acceso gratuito o no.

4. **GameObject**. Elementos esenciales en Unity que representan personajes, accesorios y el entorno del juego. Por sí mismos, no ejecutan ninguna funcionalidad, pero actúan como contenedores para Components, que son los encargados de implementar la verdadera funcionalidad del objeto.

5. **Components**. Elementos operativos de los objetos y comportamientos en un juego en Unity. Actúan como los componentes esenciales de cada GameObject, proporcionando funcionalidades específicas, como gráficos, lógica de juego, físicas entre otras. Estos componentes se combinan y ensamblan en un GameObject para definir su comportamiento (fisicas) y apariencia en el mundo del juego.

6. **Prefad**. Plantilla que sirve de base para la creación de nuevas instancias del objeto en la escena. Cualquier modificación realizada en un prefab se reflejará de inmediato en todas las instancias generadas a partir de él. Sin embargo, también es posible sobrescribir componentes y ajustes de manera individual para cada instancia.

7. **Input Manager**. Sistema que gestiona la entrada de dispositivos, como teclados, ratones, controladores y pantallas táctiles, en un juego. Permite asignar configuraciones de entrada, definir botones virtuales y establecer acciones asociadas a eventos específicos, facilitando así el manejo de la interacción del usuario con el juego.

8. **Scene**. Es el conjunto elementos y recursos que componen una parte de tu proyecto Unity. En resumen, una Scene, en Unity, es como una "escena" de película, representan un entorno o situaciones específicas del juego.

9. **Game Engine.** Software que la creación, desarrollo y ejecución de videojuegos. Está diseñado para facilitar el desarrollo de videojuegos al proporcionar herramientas para la creación, visualización y gestión de contenido interactivo en tiempo real.

10. **API.** Conjunto de reglas y herramientas que permiten la comunicación de diferentes aplicaciones informáticas entre sí y compartan funcionalidades y recursos como datos de una manera estructurada y controlada.

11. **IDE.** Software que proporciona las herramientas y las funciones integradas que ayudan a facilitar el desarrollo de software. Un IDE típicamente incluye un editor de código, herramientas de depuración, compilación y ejecución, gestión de proyectos y otras utilidades que ayudan a los programadores a escribir y probar el código

12. **Display.** Hace referencia a la interfaz visual de usuario del juego o la aplicación que estas creando

13. **Scripts.** En nuestro contexto se trata de un componente indispensable para la programación de movimientos y comportamiento de los objetos en un juego. En los juegos son adjuntados un objeto de la escena y controlan sus funcionalidades, en resumen, os scripts son piezas de código que permiten dar vida y funcionalidad a los objetos dentro de un juego creado en Unity.

1. Memoria

En este documento se trata de recoger las pautas que se han seguido para el correcto desarrollo del proyecto. Desde la fase inicial hasta la implementación final, haciendo hincapié en los aspectos claves y las decisiones que se han ido tomando en cada etapa del proyecto.

También se recoge en el documento información sobre las herramientas y tecnologías empleadas en la construcción y desarrollo del proyecto con el fin de así de dotar al lector de un contexto compresible y completo.

1.1.- INTRODUCCIÓN

En el mundo en el que vivimos estamos rodeados de tecnología que nos dota de lo necesario para la creación de entornos virtuales seguros para la formación en el uso de maquinaria industrial. En una primera instancia, la creación de simuladores como el descrito en este documento, ayudara a la reducción de accidentes en un escenario real.

Por lo general, el grueso de los vehículos industriales presenta serios problemas relacionados con la seguridad. La causa de estos incidentes puede ser desde un simple accidente de tráfico, vuelcos, fallo en los frenos, riesgo de incendio, vuelcos entre muchos otros.

Una de las causas de estos accidentes suele estar relacionados con el tamaño y el peso, los vehículos industriales suelen tener una conducción compleja y las maniobras que se realizan con ellos son más complejas, en resumen, su conducción es más difícil que la conducción de los vehículos tradicionales. Además, siempre hay que tener en cuenta la topografía del terreno y la disposición de la carga para evitar vuelcos, representes un mayor peligro para el operador o el resto de las personas que se encuentren en las cercanas del vehículo.

Otro de factor habitual en los accidentes relacionados con los vehículos industriales es el fallo en el sistema de frenos. Este tipo de accidentes es muy preocupante, ya que causan una pérdida total del control del vehículo y suelen propiciar siniestros muy graves. Este tipo de accidentes suelen ir acompañados de incendios. Esto se debe a que este tipo de vehículos funcionan mediante la utilización de combustibles líquidos o gases inflamables que al entrar en contacto con una fuente de ignición (una chispa) pueden originar un incendio.

Los vuelcos son otro de los accidentes habituales con este tipo de vehículos. Es necesario conocer bien la topografía del área de construcción. Otra de las causas principales de los accidentes por vuelco es una mala colocación de la carga o que la misma exceda el peso máximo recomendada. También es necesario que el conductor cuente con la experiencia necesaria, para evitar vuelco, ante un desnivel del terreno, u otros accidentes de la topografía del terreno.

Aunque ya existen y son utilizados sistemas de seguridad activa como la detección de objetos mediante sensores de proximidad instalados en el exterior del vehículo o la utilización de luces de información para una detención mayor y más rápida de los peligros o conocer el estado de vehículo, como la temperatura del motor.

Además de los sistemas de seguridad activa, los vehículos industriales también cuentan con los sistemas de seguridad pasiva con los cinturones que son de obligatoria utilización o el diseño de los chasis de los vehículos que están diseñados para absorber la mayor parte posible del golpe del accidente y contar con la mayor estabilidad posible. Estos sistemas pasivos juegan un papel fundamental de protección del ocupante u ocupantes del vehículo en caso de choque o vuelvo del mismo.

Aun contando con todos estos sistemas de reducción de accidentes o de sus causas, no es suficiente y siempre hay que seguir la lucha para reducir los accidente al máximo. Los simuladores de conducción de los vehículos industriales juegan un papel fundamental en esta contienda.

La utilización de simuladores en la formación de nuevos conductores, o en la educación en buenos hábitos de los conductores ya más experimentados juega un papel fundamental en la reducción de accidentes como se ha comentado con anterioridad.

Cuando hablamos de un simulador nos referimos, a un software o sistema diseñado especialmente para imitar o reproducir una situación o experiencia real. El fin que persiguen los simuladores es prestar una representación fidedigna de la realidad permitiendo así que el usuario viva una serie de experiencias o situaciones en un entorno controlado.

En este simulador de una excavadora se pretende presentar una representación lo más realista posible. El desarrollo del simulador ha sido llevado a cabo mediante la herramienta Unity. Lo que nos ha permitido dotar al simulador de propiedades físicas, controles y movimientos. Todas

estas características que nos brinda Unity permiten que el usuario del simulador se familiarice con un entrono realista de la retroexcavadora y la aprenda a conducir de una forma más sencilla y realista.

Con el desarrollo de este simulador he notado que se trata del escenario perfecto para la prueba de nuevos sistemas de aviso de peligros o de reducción de accidentes. Ya que la prueba y despliegue de estos nuevos sistemas de reducción de peligros y accidentes en un entorno virtual son una gran alternativa de prueba que reduce en gran medida los peligros que causa la prueba de nuevas tecnologías en la vida real. Además, se trata de una medida de abaratar los costes de investigación y contraión de estos nuevos sistemas de detección de peligros que un mucho caso no acaba resultando efectivos y son desechados.

En resumen, un simulador de una excavadora presenta una alternativa fácil y divertida para la familiarización de los principales controles de la excavadora. Además, se trata de una alternativa segura para la prueba de situaciones potencialmente peligrosas o la prueba de nuevas tecnologías reducción de peligros.

1.2.- MOTIVACIÓN

La motivación que me empuja a realizar el desarrollo de este simulador es la necesitad continua que demanda la sociedad de mejorar la seguridad del manejo de maquinaria industrial.

Las excavadoras, como la que presenta el simulador, son utilizadas en la mayoría de los entornos industriales, especialmente en obras. Aunque el personal cualificado para el manejo de este tipo de maquinaria es escaso debido al complejo control que presentan estas maquinarias, especialmente en situaciones de una compleja topografía del terreno.

Contar con un simulador como este permitirá a los operarios contar con un entorno realista y seguro para poder así realizar horas de practica sin generar gastos de combustibles ni desgaste en la maquinaria real. Además, de permitir familiarizarse con nuevas mecánicas a los conductores ya más experimentados.

También la practica en la actividad de cavar, principal función de las excavadoras presenta una gran ventaja para aprender donde esta los limites seguros y recomendados que presentan estas máquinas para un uso seguro.

Por último, otro de los objetivos del simulador es dotar al usuario de un lugar donde poder hacer pruebas o ajustes de una forma rápida y segura, lo que conllevara un aumento en la eficiencia operativa y la productividad en términos generales.

1.3.- OBJETIVOS Y ALCANCE DEL PROYECTO

Con este desarrollo se pretende alcanzar una serie de objetivos, que son los siguientes:

- Desarrollar un simulador de excavadora.
- Ofrecer un movimiento realista y dinámico.
- Permitir la articulación del brazo y cucharon de la excavadora.
- Acceder a la rotación de la cabina de la excavadora.
- Implementar la capacidad de recoger tierra con el cucharon y poder introducirla en un recipiente.

1.4.- SÍNTESIS DE LAS CAPACIDADES DEL SIMULADOR

Las capacidades principales que presenta el simulador se pueden resumir en:

- La capacidad de conducción de una retroexcavadora en primera persona
- Movimiento en el brazo y en cucharon.
- Función de frenar, acelerar y girar como si se tratara de un vehículo realista.
- Competencia de rotación de la cabina que forma parte de la excavadora.
- Existencia de una posición de trabajo en la que solo se podrá utilizar las funciones de giro de cabina y las que correspondan al movimiento del brazo y el cucharon. En esta posición no será posible el movimiento de aceleración o frenado.
- Capacidad de controlar el simulador atreves de mandos realistas.
- Se cuenta con 4 cámaras, para así dotar de una experiencia de 360º. Con esto se pretende realizar una experiencia más realista e inversiva.
- Posibilidad de transporte de partículas simulando tierra para simular la acción de cavar.

1.5.- EXCAVADORA

La excavadora es un vehículo de construcción que se utiliza generalmente para trabajos relacionados con la excavación. Se trata de un vehículo industrial muy extendido y popular de los empleados en ambientes industriales y es fácilmente diferenciable. [Wikipedia- La enciclopedia viva]

Las partes principales de una excavadora son:

Figura 1.1 Excavadora [Imagen genérica de excavadora]

Las partes más destacadas y que hemos dotado de funcionalidad son:

- Cucharon: pieza en forma de cuchara que es encuentra en el extremo del brazo de la excavadora y será la encargada de hacer la acción de excavar.
- Cabina: parte donde se encuentra el usuario del vehiculó y además cuenta con opción de rotar sobre sí misma.
- Brazo: parte que conecta la pluma con el cucharon. Se encarga de dar profundidad al movimiento de cavar a la máquina.

- Pluma: conecta la cabina con el brazo y se encarga de dotar de altura a la acción de cavar.
- Oruga: se encarga de dotar a la máquina de la capacidad de movimiento.

En el caso del simulador utilizaremos un prefab que cuenta con las principales que forman una excavadora.

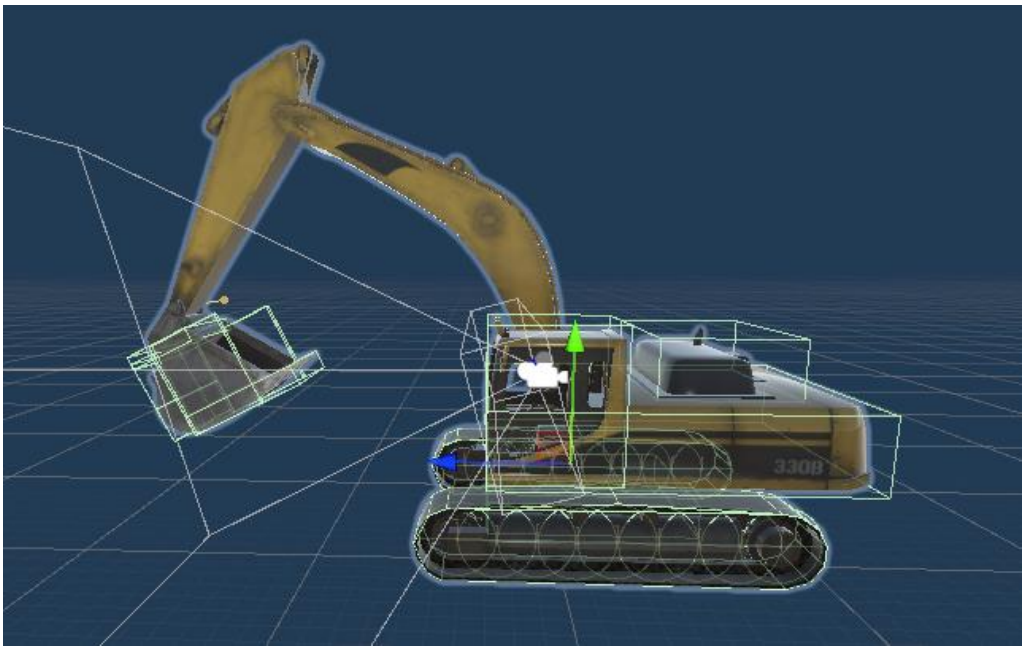

Figura 1.2 Prefab Excavadora [Comunidad github- Pateman16]

1.6.- APLICACIÓN DE UNITY EN SIMULADORES

La decisión de usar Unity se debe a que se trata de un "motor de videojuegos" o "game engine" basado en el lenguaje de programación C# que además presenta una versión gratuita, bastante completa para desarrolladores aficionados. Unity ya cuenta con numerosas rutinas de programación destinada a facilitar el trabajo de los desarrolladores, además, de contar con una amplia comunidad y numerosos DLC o paquetes descargables en versión gratuita, que ayudan al desarrollador de entornos interactivos como los videojuegos o los simuladores.

Unity destaca por presentar una interfaz intuitiva y fácil de utilizar. Esto lo hace una herramienta para el desarrollo de videojuegos perfecta para desabolladores con menos experiencia. Su interfaz visual y la amplia documentación que presenta lo convierten en un

entorno de trabajo perfecto para que personas poco experimentadas alcancen diseño y desarrollos de proyectos con una alta calidad.

También, destaca la facilidad que presenta Unity para escalar los desarrollos a varias plataformas como móvil, Tablet y realidad virtual entre otras. Este es factor de peso en la elección debido a la situación en la que se encuentra la industria actual que presenta numerosas plataformas y el objetivo es llegar a la mayor audiencia posible.

Otro gran punto a favor que nos brinda Unity es la gran comunidad de desarrolladores. Esta comunidad nos brinda desde documentación muy útil para tener unos conocimientos básicos para nuestra primera toma de contacto, hasta tutoriales ya más enfocados en aspectos más específicos del entorno. También nos presenta la posibilidad de compartir nuestros proyectos de forma online para así recibir "feedback" de otros usuarios de la herramienta para alcanzar un resultado final mejor en nuestro proyecto.

Pero, aunque Unity cuente con todas estas ventajas para desarrolladores amateur, también se trata de una herramienta utilizada en el ámbito más profesional de la industria de desarrollo de videojuegos y simuladores. Podemos encontrar muchos videojuegos famosos presentes en numerosas plataformas como prueba del gran potencial y calidad que presenta este motor de desarrollo.

Algunos pueden ser:

- Pokémon Go: Indudablemente, Pokémon Go generó una revolución en la industria de los videojuegos móviles a escala global. Este juego, diseñado para dispositivos móviles, fusiona la realidad virtual al integrar a los Pokémon con la ubicación real del jugador, manteniendo su capacidad para atraer a millones de usuarios de diversas edades a aventurarse fuera de casa para capturar a estas adorables criaturas. Este título fue desarrollado utilizando la plataforma Unity.

  [Niantic- Pokémon Go]

- CarX Rally: se trata de un videojuego de temática de vehículos que intenta simular la experiencia vivida en las competiciones de rally en la vida real. En este simulador de conducción se puede experimentar al máximo la experiencia de dirigir cada vehículo y competir contra otros pilotos como si fuera la vida real. [Racing simulator- Carx Rally].

- Farming Simulator 19: se trata del simulador agrícola más famosos del mercado. En este simulador te permite dirigir y gestionar tu propia granja y cultivos. Además, de los recursos propios del videojuego existe una numerosa oferta de DLC que pueden cambiar tu experiencia de juego con cada partida. [Giants Softwarw- Farming Simulator]

- Microsoft Flight Simulator X: Uno de los simuladores de vuelo más realistas y visualmente impresionantes, permite a los usuarios explorar el mundo con detalles precisos. Con este simulador puedes vivir la experiencia de un piloto. [Micrisift - Microsoft Flight Simulator X]

- Touch Surgery: se trata de una aplicación móvil y realidad virtual que proporciona a los profesionales de la salud, una experiencia interactiva de prácticas en los procedimientos quirúrgicos [Medtronic – Touch Surgery]

1.7.- ETAPAS DEL PROYECTO

El proyecto de desarrollo ha transcurrido en diferentes partes:

1. Planificación
2. Producción
3. Pruebas
4. Lanzamiento
5. Documentación

1.7.1.- Planificación

Se trata de la fase donde nace la idea del desarrollo del proyecto, y se realiza antes de diseño general de los personajes y escenarios. La finalidad de esta primera etapa es dar respuesta a cuestiones fundamentales del proyecto como:

- Donde se desarrollan los escenarios.
- A que publico dirigirse.
- Para que plataformas desarrollarlo.
- Tipo de videojuego o simulador será.
- Desatollarlo en 2D o 3D.
- Mecánicas clave para tener en cuenta.

Con el planteamiento de todas estas cuestiones técnicas para el desarrollo del proyecto surgen algunas dudas que pueden causar ciertos problemas para la viabilidad del proyecto, como pueden ser:

- Plazos temporales con lo que contamos.
- Costes de producción.
- Forma de monetizar el resultado final.
- Recursos con los que contamos para el inicio del proyecto.
- Capacidad para realizar mantenimientos tras el lanzamiento final.
- Numero necesario de personas involucradas.

Esta etapa se ha estimado que ocupara en un espacio temporal de unas 95 horas.

1.7.2.- Producción

Es aquella etapa del proyecto en la que se aplican todos lo recurso en el desarrollo del proyecto. Es una de las partes más densa del desarrollo del proyecto. Durante el desarrollo de esta fase se pretende producir los entregables finales del proyecto, aunque aún pueden sufrir modificaciones si no cumple los objetivos deseados o si no pasan la fase de prueba.
Esta etapa se ha estimado que ocupara en un espacio temporal de unas 220 horas.

1.7.3.- Pruebas

Se trata de una de las etapas más claves del desarrollo del proyecto. Esta fase se realizará un control de calidad del producto final. Esta fase se encarga en detectar los errores del software para su corrección.

Además, tan bien común la actuación de “testers” personas encargadas de probar el juego final como si se tratara de un usuario genérico. Estas figuras juegan en papel fundamental en la detención de fallos en el juego y ayudan equilibrar la dificultad del juego. Tras esta fase estaremos preparados para lanzar una fase alfa.

Esta etapa se ha estimado que ocupara en un espacio temporal de unas 200 horas.

1.7.4.- Lanzamiento

Tras realizar todos los cambios y mejoras necesarios que se han detectado en la fase anterior de pruebas. Esta fase de lanzamiento se trata de poner nuestra versión alfa a un público externo al involucrado al proyecto. El producto del lanzamiento será mejorado y sufrirá mantenimientos mediante DLC hasta que se transforme en la versión final deseada.

Esta etapa se ha estimado que ocupara en un espacio temporal de unas 100 horas.

1.7.5.- Documentación

Se trata de la etapa final del proyecto. Se objetivo es recopilar en documentos toda la información relacionada con el proyecto. Estos documentos tienen que ser claros, concisos y de fácil compresión para persona no especializadas.

Esta etapa se ha estimado que ocupará en un espacio temporal de unas 150 horas, aunque será desarrollada además durante el resto de las etapas de una forman un segundo plano.

1.8.- ROLES

En todos los proyectos al existir varias necesidades es necesario la involucración de diferentes personas que desarrollen roles diferentes unos de otros. En este caso todos los roles lo han realizado la misma persona, aunque esto no suele ser lo más habitual.

Los roles más comunes son:

- Diseñador.
- Programador.
- Tester.
- Planificador.
- Marketing.
- Director.

1.8.1.- Diseñador

Persona encargada de los aspectos estéticos y creativos del videojuego. Encargado tanto de los escenarios como en el diseño de los menús. También de los aspectos de iluminación y los factores visuales del interfaz. Este rol juega un papel fundamental en la etapa de desarrollo o de producción.

1.8.2.- Programador

Rol encargado en plasmar las ideas y los aspectos técnicos en el código. Esta persona realiza la función de traducir los aspectos técnicos creados por el diseñador en código operativo y funcional. Además, realiza soporte en la toma de decisiones de los diseñadores para arrojar luz sobre las capacidades funcionales del código. Este rol es fundamental en las etapas de producción, pruebas.

1.8.3.- Tester

Persona encargada de realizar las pruebas del código previamente diseñado por el desarrollador. Esta persona es el filtro de calidad por lo que representa un punto de gran responsabilidad. Es la persona encargada de dar de paso el código para la fase de lanzamiento y es muy relevante en la fase de pruebas.

1.8.4.- Planificador

Persona encargada de realizar el estudio inicial del proyecto y marcar en una primera estancia los objetivos que el videojuego juego debe alcanzar. También es el encargado de realizar el alcancé del proyecto, así como, la planificación temporal del proyecto, aunque esta varie durante el desarrollo por aspecto ajenos a su control. Este rol participa en la etapa de planificación.

1.8.5.- Marketing

Rol encargado de la distribución y la parte comercial del proyecto: Desempeña un papel activo después del lanzamiento del videojuego. Mantiene comunicación directa con los usuarios y es el encargado de trasladar el "feedback" de los usuarios al resto de personas del equipo. Desempeña su función en la etapa de lanzamiento.

1.8.6.- Director

Personal encargado de ser la cabeza del equipo. Este rol es el encargado de tener una idea general de todo proyecto y es la persona encargada de tomar las decisiones finales y más relevantes. Esta persona no estará centrada en ninguna etapa concreta del proyecto, irá acompañando al proyecto a través de toda su vida pudiendo tener así la visión global del mismo y tendrá el deber de documentar el proceso completo.

1.9.- CRONOGRAMA

En la mayoría de los proyectos en las diferentes etapas son realizadas por personas con roles diferentes. Aunque en este proyecto será realizado por una sola. En la tabla inferior recogeremos un pequeño resumen de las horas necesarias para cada tarea y al final el cómputo total de horas que ha llevado realizar este proyecto.

Etapas	Duración (horas)
Planificación	95
Producción	220
Pruebas	200
Lanzamiento	100
Documentación	150
TOTAL	**765 horas**

Tabla 1.1 Planificación temporal

1.10 DIAGRAMA DE GANTT

El diagrama de Gantt es una herramienta comúnmente empleada para la gestión de todo tipo de proyectos. Se emplea para la visualización del estado de las tareas, plazos y descansos que forman el proyecto. En este caso, hemos utilizado recuadros para representar la unidad de una semana y hemos destacado en azul las semanas en las que se ha realizado cualquier tipo de tarea relacionada. Cabe destacar que dos tareas pueden ser desarrolladas de forma simultánea, como pasa la cuarta semana de marzo en la que se empezó la producción si haber acabado la planificación total del proyecto. Además, existen meses en los cuales el proyecto ha sido abandonado por diferentes cuestiones como periodos vacacionales o por la participación en otros proyectos.

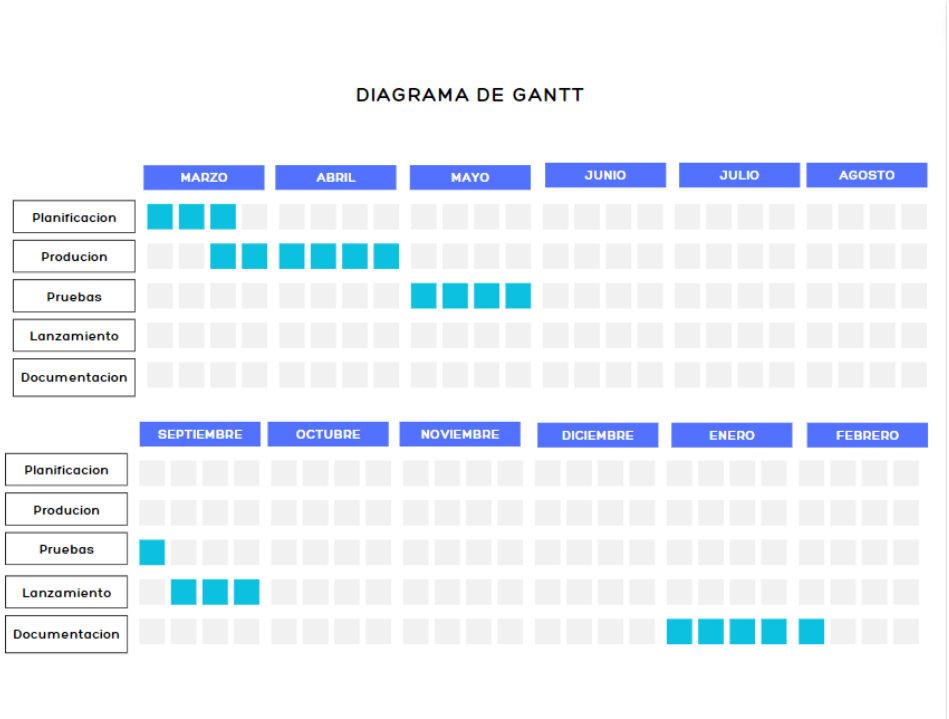

Figura 1.2 Diagrama de Gantt del Proyecto [Canva]

2. Presupuesto

2.1.-ESTIMACION DE COSTES

El proceso de desarrollo de cualquier videojuego es apasionante e implica numerosos recursos. A causa de esto es preciso realizar una estimación de costes en el momento que sean conocidos todos los recursos que van a ser necesarios para el correcto desarrollo del proyecto, se trata de una acción esencial para estudiar la rentabilidad del proyecto.

La estimación de costes del proyecto no solo está compuesta por los costes monetarios, sino que también implica los recursos humanos necesarios como, el tiempo requerido para los procesos o la compra de software. También, se deberá tener en cuenta las necesidades específicas del proyecto como como la plataforma de despliegue o las metas comerciales, entre muchas otras.

En lo relativo a los recursos los costes estimados se reflejan en:

Personal	Días	Horas	€/h	Salario
Diseñador	19	110	25,00 €	2750€
Programador	38	220	25,00 €	5500€
Tester	9	50	17,00 €	850€
Marketing	9	50	20,00 €	1000€
Planificador	16	95	20,00 €	1900€
Director	26	150	30,00 €	4500€
TOTAL:		765	----------	**15600€**

Tabla 2.1 Estimación costes humanos (El cálculo de días, se tienen en cuenta los días no laborales)

Para el proceso de desarrollo del simulador se ha requerido un ordenador, en nuestro caso, ROG Strix G513IH [PC Componente A] con un valor aproximado de 879 euros. Para los aspectos visuales se necesita 4 pantalla TCL [PC Componente B] con un precio aproximado 407 euros y para la interfaz de usuario se ha adquirido un volante y pedales Logitech G Saitek [PC Componente C»], con un precio de alrededor de 240 euros. Por último, debemos tener en cuenta el coste de extras como, cables, switch y la infraestructura de la cabina cuya suma aproximada será de unos 350 euros.

Materia	Unidades	€/Und	Precio Total
ROG Strix G513IH	1	879	879€
Logitech G Saitek	1	240	240€
Pantallas	4	407	1628€
Extras	1	350	350€
TOTAL:			**3097€**

Tabla 2.2 Estimación costes materiales

Con la idea de su futura comercialización también incluiremos la tasa de despliegue de la plataforma elegida, es este caso hemos elegido el despliegue en la plataforma Steam [Steam] que se trata de una de las principales plataformas de videojuegos para PC. El coste de esta tasa será de 100 dólares, aproximadamente 92 euros, aunque tomaremos el valor del dólar y el euro 1:1.

Por últimos deberemos tener en cuenta aspectos como ellos gastos generales que suponen un 13% en proyectos de Ingeniería, el 6% del beneficio industrial y por último el Impuesto sobre el Valor Añadido (IVA) del 21%.

2.2.-DESGLOSE DETALLADO

Presupuesto estimado humano	15.600€
Presupuesto estimado en material (hardware)	3.097€
Licencia (software)	100€
Total costes estimados	**18.797€**
Gastos generales (13%)	2.443,61 €
Beneficio industrial (6%)	1127,82€
Total sin IVA	22.368,43€
IVA (21%)	4.697,3703€
TOTAL ESTIMADO	**27.065,8003 euros**

3. Gameplay

3.1.- CONCEPTO

Un Gameplay es el conjunto de actividades, experiencias e interacciones que realiza y experimenta un usuario al participar en un videojuego o un simulador. Este incluirá toda la regla, controles, aspectos visuales y mecánicos además de los aspectos mecánicos y las dinámicas generales del videojuego.

El Gameplay trata de mostrar las interacciones que se realizan entre los jugadores y el mundo virtual que se crea en el videojuego, explicando como los elementos que lo forman se combinan para formar la experiencia de juego única que se ofrece al jugador.

El intentar conseguir la mayor calidad de Gameplay es el un objetivo fundamental para alcanzar una mejor experiencia de juego para el usuario, ya que está directamente relaciona con el disfrute e inversión del jugador, serán fundamentales para alcanzar una efectividad de la experiencia de juego. En resumen, el Gameplay se trata de la esencia de la interactividad y participación entre el usuario y el mundo virtual que se crea en un videojuego.

3.2.- GAMEPLAY DEL SIMULADOR

Como meta final del simulador es replicar las funcionalidades principales de una excavadora industrial, para la posibilidad de aplicaciones en formación en las nociones principales de la empleabilidad de estos vehículos.

El usuario se encontrará en un asiento dotado de los mandos necesarios para su jugabilidad. Además, también hay que tener en cuenta que las pantallas estén a la distancia adecuada para mayor disfrute del simulador y una correcta experiencia de juego. La estructura planteada para este simulador es la que se puede apreciar en la siguiente imagen.

Figura 3.1 Estructura simulador

Al comenzar el videojuego la primera visión que tendremos será el menú principal del videojuego, desde este menú podremos realizar las diferentes acciones:

- Trasladarnos a la escena de créditos
- Trasladarnos la escena de opciones
- Comenzar el juego
- Salir de juego

La imagen principal de la escena solo se mostrará en la pantalla principal (la que se encuentra enfrente del usuario), mientras que las demás estarán en negro. La visión que se espera en esta pantalla es la mostrada a continuación.

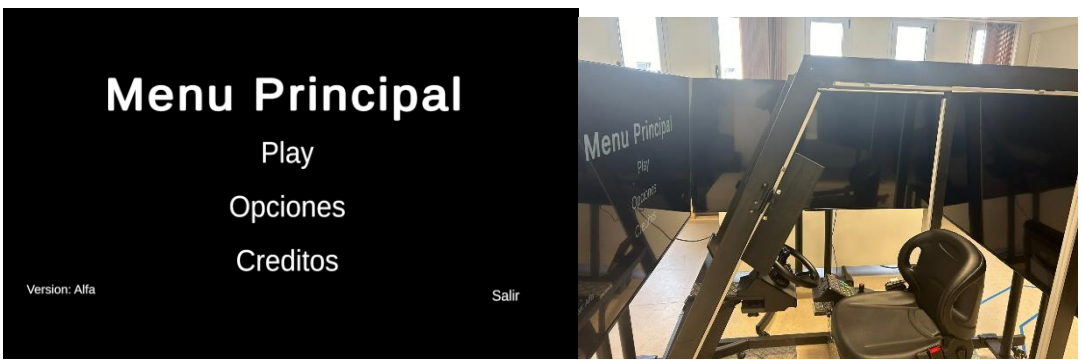

Figura 3.2 Visión Menu_Principal

El texto con la versión y el título de la pantalla se tratan de simple información mientras que el resto de las elecciones nos moverán las diferentes escenas salvo la selección de “Salir” que finalizara la ejecución del simulador. La selección de “Opciones” nos trasladada a una escena la cual presentara en frontend con elementos interactivos para regular el brillo y el volumen, aunque el backend necesario para realizar las acciones no está implementado, pero se prevé realizarlo en una actualización futura. Para poder regresar al menú principal deberemos seleccionar la funcionalidad de “Atrás”.

Figura 3.3 Visión Pantalla Principal en Opciones

Si seleccionamos “Creditos” nos trasladaremos a una escena la cual quiere dar créditos de las personas implicadas de forma directa e indirectamente con la realización del proyecto, además, de agradecer al usuario el uso del videojuego. Nueva mente para regresar al menú principal tendremos que pulsar “Atras”.

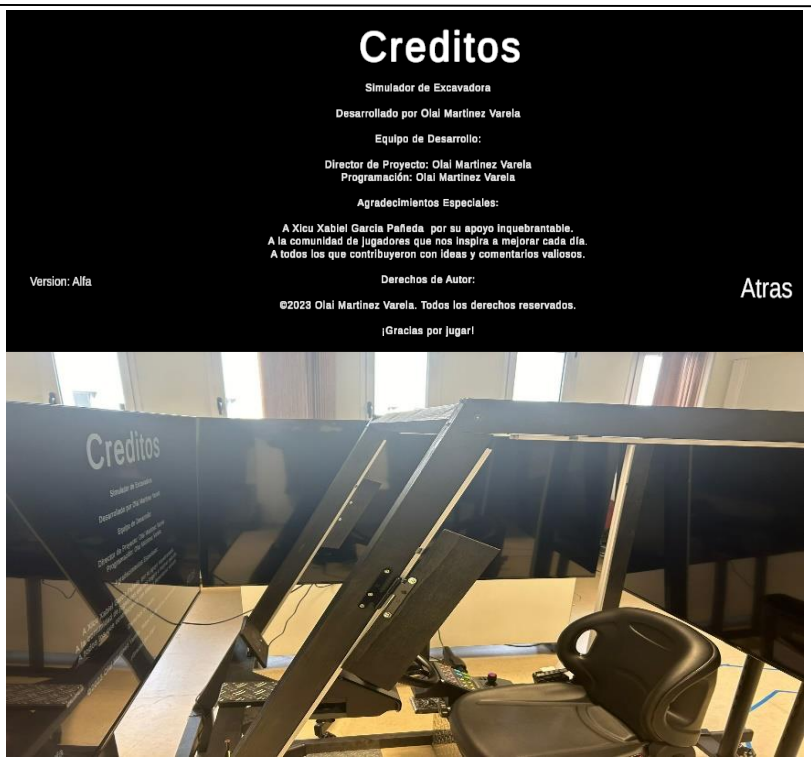

Figura 3.4 visión Pantalla Principal en créditos

Por último, nos queda la posibilidad de seleccionar la opción de “Jugar”, la cual dará inicio al juego y nos trasladar al mundo virtual creado para esta simulación. En este caso las cuatro pantallas, que componen el simulador, presentaran cuatro visiones diferentes con la intención de dotar al usuario de una visión 360º y así aumentar la experiencia inversiva.

Figura 3.5 Visión en Jugar

En esta pantalla contaremos con la posibilidad de ver una imagen diferente en cada pantalla pudiendo así hacernos una idea al completo de nuestro alrededor y ayudarnos así en funciones de la condición como en la conducción de reversa o a la hora de chocar con diferentes elementos de nuestro alrededor.

Figura 3.6 Visión Izquierda, Trasera y Derecha

Para facilitar la explicación del Gameplay lo haremos solo con la visión delantera y principal del jugador así que el resto de las imágenes empleadas para la explicación estarán tomadas desde este punto de vista.

En el inicio del simulador nos encontraremos a cierta distancia de la zona de trabajo y tendremos que usar los mando que nos proporción el volante y los pedales para dirigir el vehículo industrial hasta la zona designada para el trabajo.

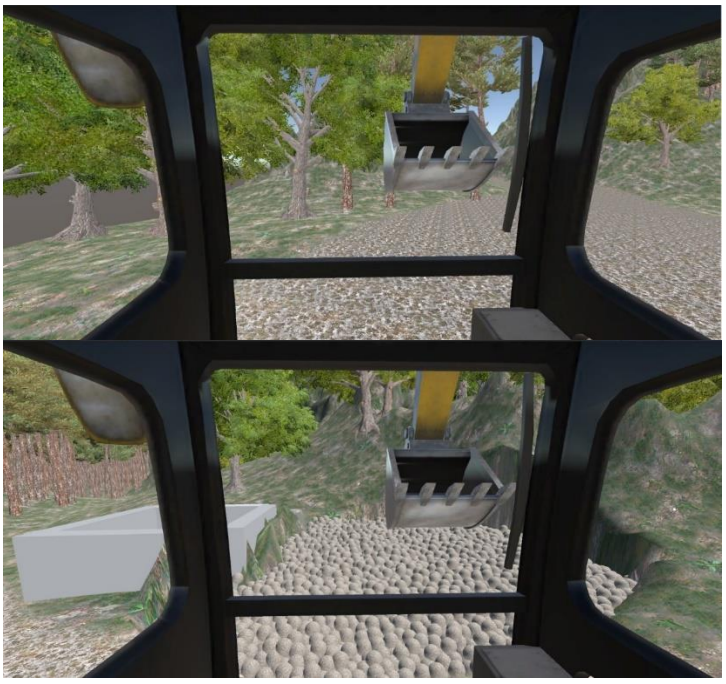

Figura 3.7 Movimiento de zona de inicio a zona de trabajo

Tras llegar a la zona de trabajo apretaremos el botón que activa el modo trabajo nos dispondremos a realizar las acciones de excavación utilizando las funcionalidades que nos ofrece el brazo de nuestra excavadora. En este caso combinaremos la botonera con el volante para realizar las acciones de excavación descritas en las imágenes de a continuación.

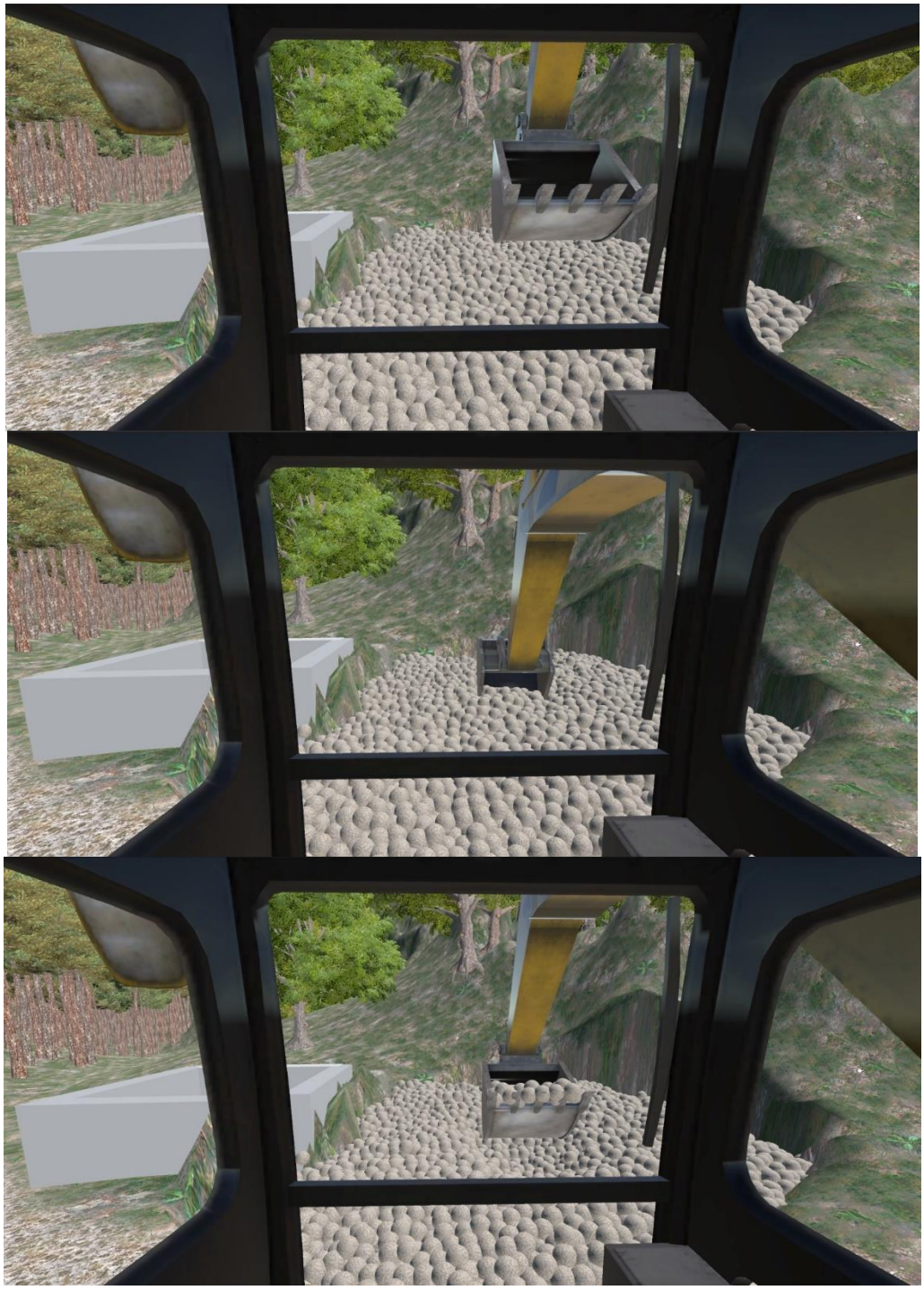

Figura 3.8 Movimiento del brazo

A continuación, con cazo cargado apretaremos el botón que activa el modo de conducción y nos dirigiremos a la zona destinada a la basculación de carga.

Figura 3.9 Basculación de la carga

Tras esta última acción se volverá a realizar las acciones anteriores para repetir la acción de excavación. Debemos tener en cuenta que los movimientos descritos se tratan del procedimiento básico y que siempre está abierto a mejoras como la colocación en un punto donde la posibilidad de rotación de la cabina ayude a reducir el tiempo de la excavación. Por ultimo y para finalizar la simulación contaremos con un menú de "Pausa" accionado con un botón independiente.

En este menú tendremos la opción de hacer una pequeña interacción en nuestra simulación ya que mientras nos encontremos en el no correrá el tiempo en el mundo virtual o bien la opción de salir y finalizar nuestra simulación.

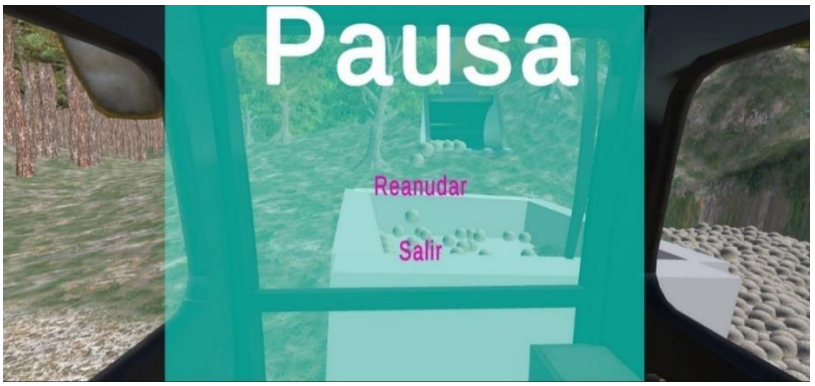

Figura 3.10 Menu Pausa

4. Documentos técnicos

4.1.- FRONTEND

Se trata de la parte del proyecto que se encarga de la parte de interactuar con el usuario y engloba todo loque forma parte de la experiencia del usuario. El frontend, en el contexto Unity en el que nos encontramos, engloba los elementos que forman parte del entorno de desarrollo y que son los encargados del control, y el desarrollo de las interacciones del usuario con el juego.

La carpeta principal donde los usuarios administran y modifican el contenido fundamental es la carpeta “Assests”. Esta carpeta contiene las recursos y estructuras fundamentales para la creación del simulador.

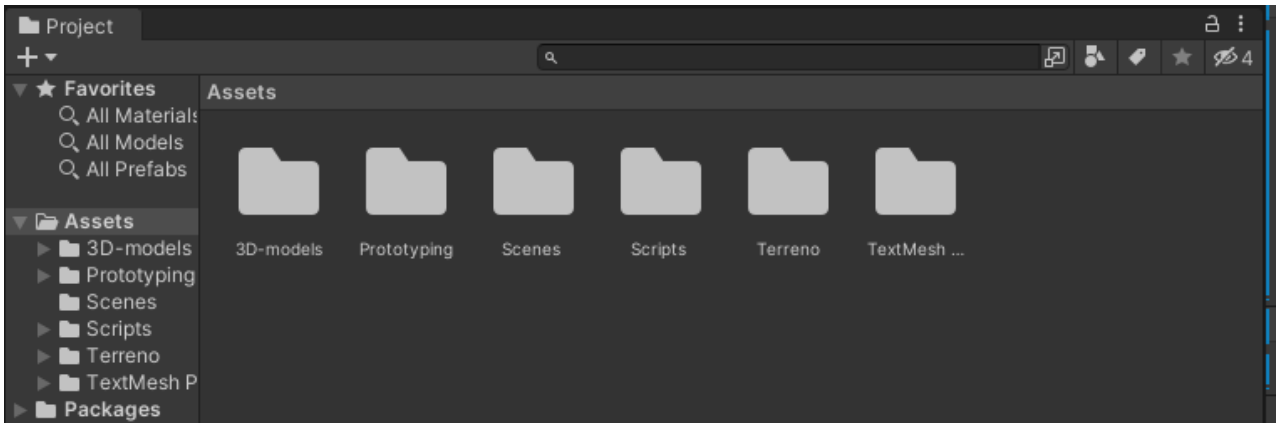

Figura 4.1 Carpeta Assests

A su vez la carpeta de Assests está formada por las carpetas que contiene materiales fundamentales para el funcionamiento del simulador, y serán explicada individualmente. Las carpetas que la forman son:

- 3D-models
- Prototyping
- Scenes
- Scripts
- Terreno
- TextMesh Pro

4.1.1.- 3D-models

En esta carpeta se encuentran los componentes utilizados para la implementación de la excavadora. Desde su modelo 3D, los materiales y las texturas que la forman. Está carpeta y sus componentes han sido tomada de otro proyecto Unity de la comunidad de programadores GitHub. [Comunidad GitHub A»]

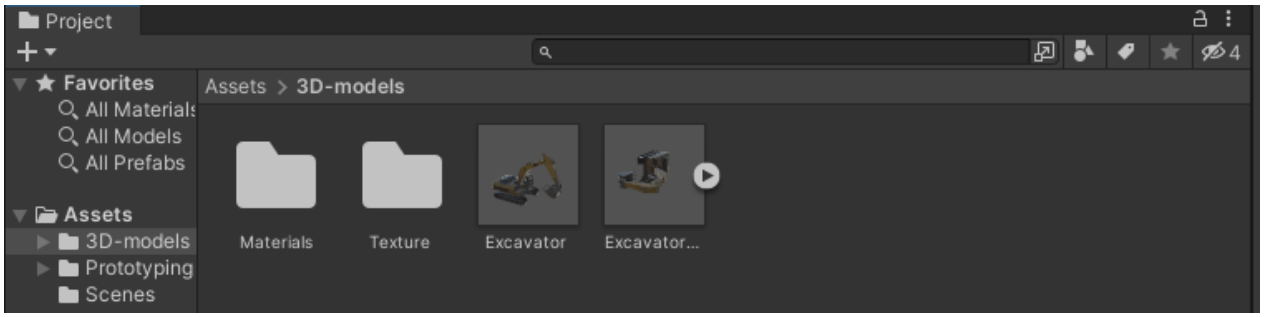

Figura 4.2 Interior de la carpeta 3D-models

En el interior de estas carpetas se encuentran los elementos fundamentales para la construcción de un proyecto.

Materials. En ella no encontramos lo “shader” que se emplean para dotar de presencia la superficie de un elemento y que estarna ligados a una textura. Las opciones que emplearemos dependerán del “shader” que deseamos utilizar en cada caso.

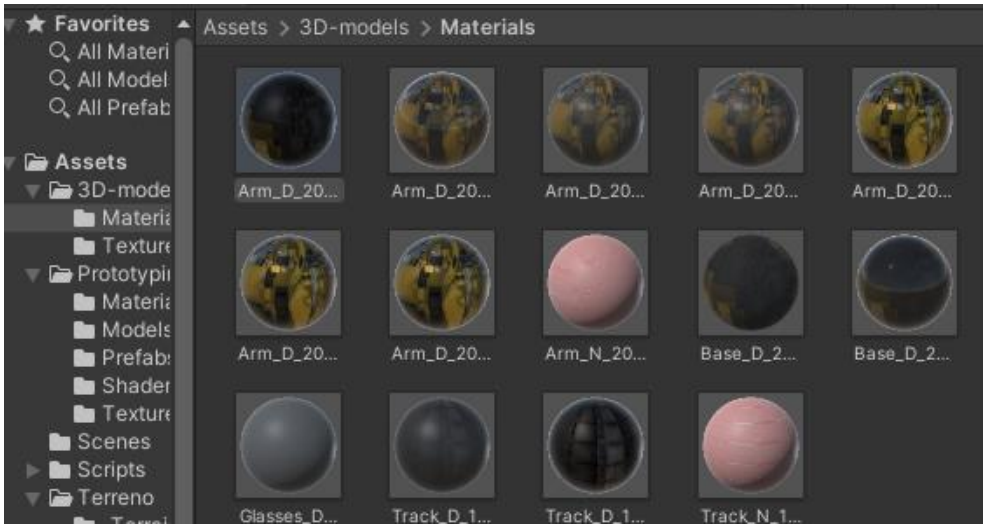

Figura 4.3 Interior de la carpeta Maetrials

Texture. Se trata de imágenes en formato bipmat que se aplican sobre un objeto 3D para agregarle más detalles como patrones, rugosidad y materiales. Se emplean para mejorar el aspecto del objeto y crear entornos más realistas. Se pueden aplicar texturas a través de materiales, que controlan cómo interactúa la luz con la superficie del objeto, lo que afecta su apariencia visual.

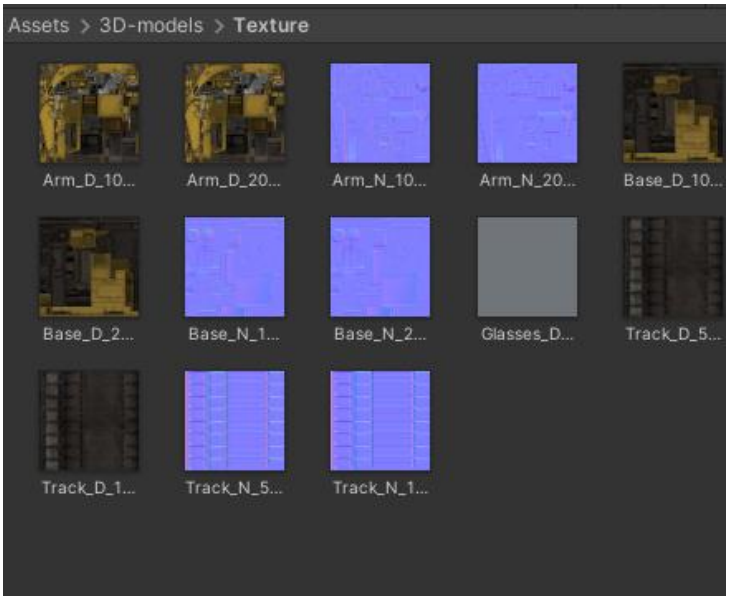

Figura 4.4 Texturas 3D-models

También encontramos el modelo 3D de la excavadora que están compuestos por una estructura de malla que define su forma y geometría, y pueden estar adornados con texturas, materiales y animaciones para crear una representación visual más detallada y realista.

Por último, nos encontraremos el prefabs de la excavadora. Un prefabs una instancia reutilizable de un objeto o conjunto de objetos que se guarda como un activo en el proyecto. Los prefabs permiten crear y modificar objetos de manera eficiente al proporcionar una plantilla que puede ser instanciada múltiples veces en la escena sin necesidad de recrearlas desde cero cada vez.

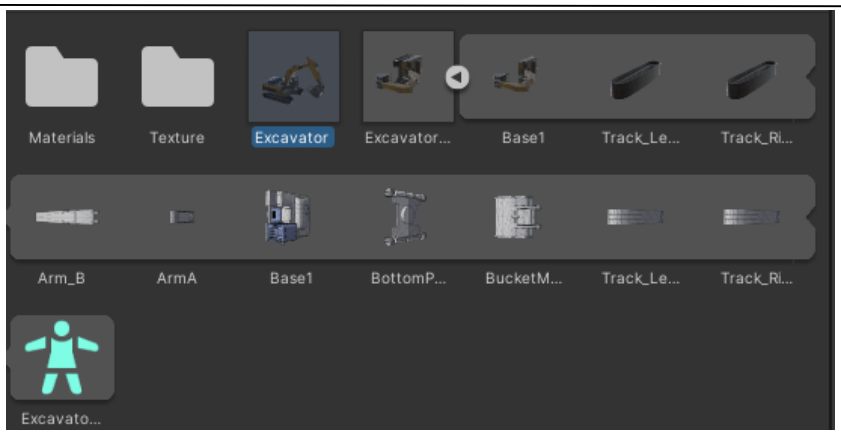

Figura 4.5 Modelos y prefabs de la excavadora

4.1.2.- Prototyping

Se trata de un conjunto de recursos, de versión gratuita, obtenidos de la Unity Asset Store [Unity Asset Store B], empleados para la formación de algunos elementos empleados en la "scene" del juego como los elementos que usaremos como contenedore. Aunque en este caso no contemos con carpeta de Materials y Texture no nos harán falta porque se utilizarán de otras carpetas. En su interior encontramos:

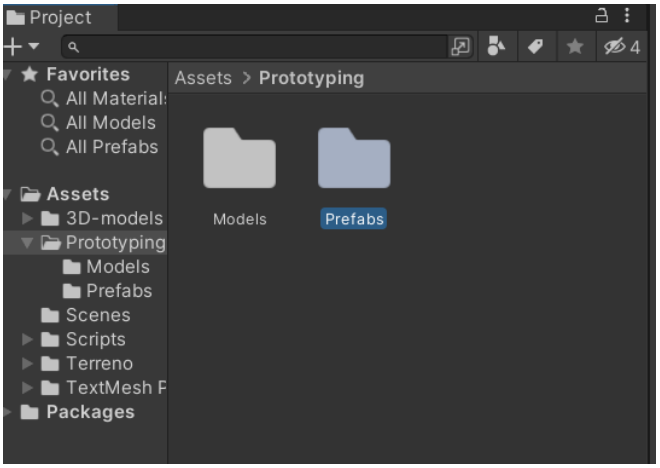

Figura 4.6 Carpeta Prototyping

4.1.2.1 Models

En Unity, los "models" se refieren a los objetos tridimensionales que componen los elementos visuales de una escena. En este caso nuestra carpeta contendrá los modelos 3D como rampas, paredes, suelos o escaleras.

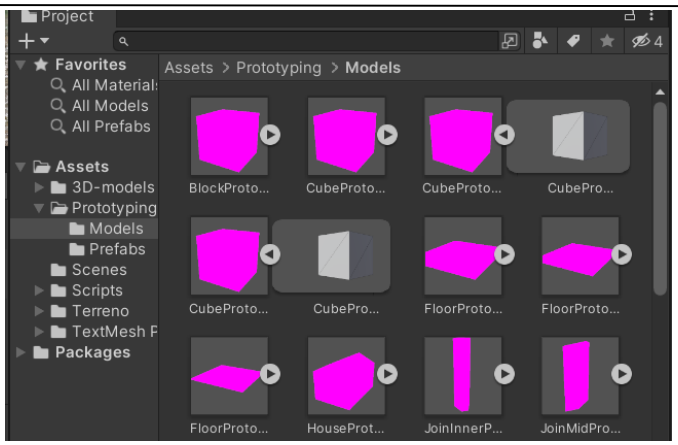

Figura 4.7 Interior de la carpeta Models

Los modelos en Unity se crean utilizando software de modelado 3D externo, como Blender o 3ds Max, y luego se importan al proyecto, aunque en nuestro caso no hemos utilizado estos programas, hemos optado por la descarga de paquetes gratuitos de la Unity Asset Store.

4.1.2.2 Prefabs

Un "Prefab" es un objeto previamente configurado que facilita la creación rápida y sencilla de elementos que se usan de una forma repetitiva en una misma escena. Se trata de una especie de plantilla que encapsula la configuración completa de un objeto, incluyendo su geometría, materiales, componentes, propiedades y jerarquía.

Cuando se realiza cualquier edición en un "Prefab", los cambios se aplican de inmediato a todas las instancias generadas a partir de él. Sin embargo, también existe la posibilidad de realizar modificaciones en componentes aislados ya, permitiendo adaptar ciertos aspectos según las necesidades específicas.

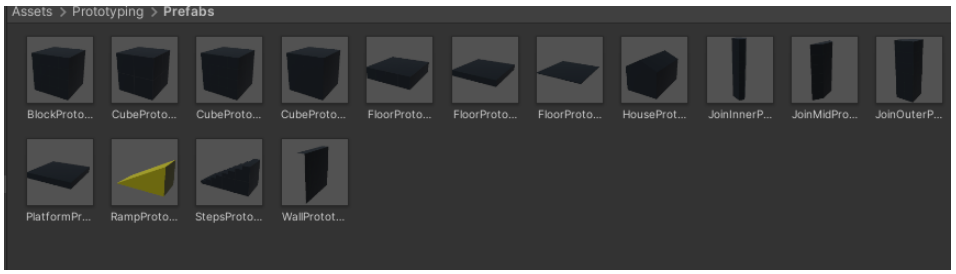

Figura 4.8 Ejemplo Prefabs

4.1.3.- Scene

En esta carpeta se recoge el grueso del proyecto. Engloba todas las escenas que forman el proyecto y por las cuales se puede navegar de una forma u otra.

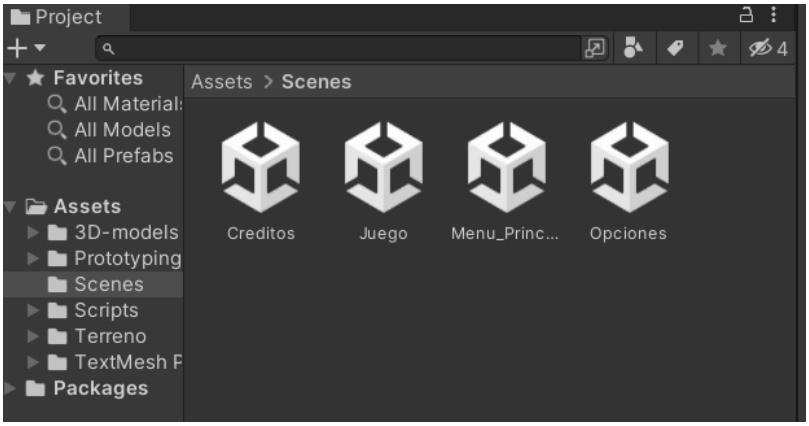

Figura 4.7 Carpeta Scene

Mediante la llamada de construcción “Build Settings” se establece el orden de llamada de las escenas, también se indica que el simulador ha sido diseñado para la plataforma PC sin importar el Sistema Operativo que sea utilizado. [«Manual BuildSettings]

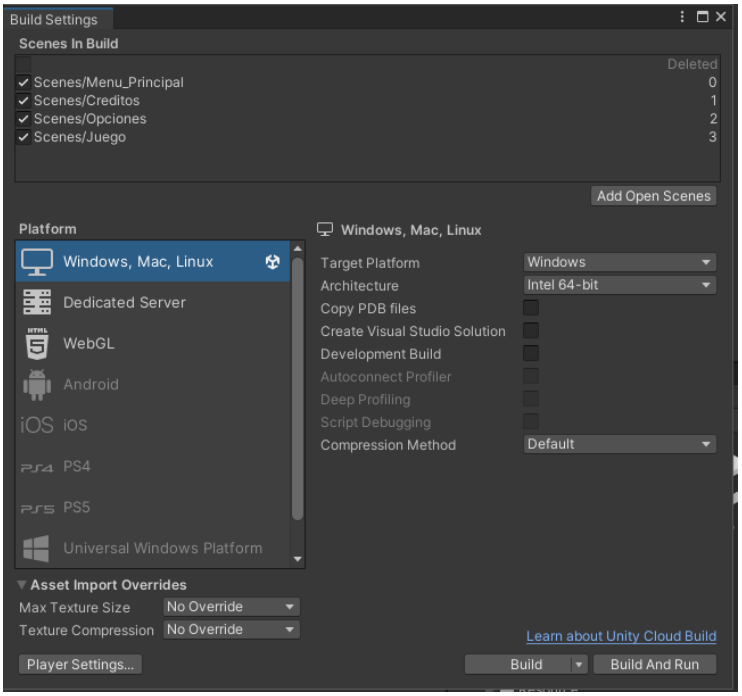

Figura 4.8 Build Settings

4.1.3.1.- Menú Principal

La escena de Menú principal se trata de una pantalla inicial que se muestra al jugador cuando inicia el simulador. Desde esta pantalla se pretende que los jugadores puedan acceder a varias opciones que le permitan iniciar el juego, ajustar configuraciones u otros servicios que se consideren necesarios. Por lo tanto, el menú principal se trata de la puerta de entrada a la experiencia de juego a los usuarios como ose puede apreciar en la siguiente figura.

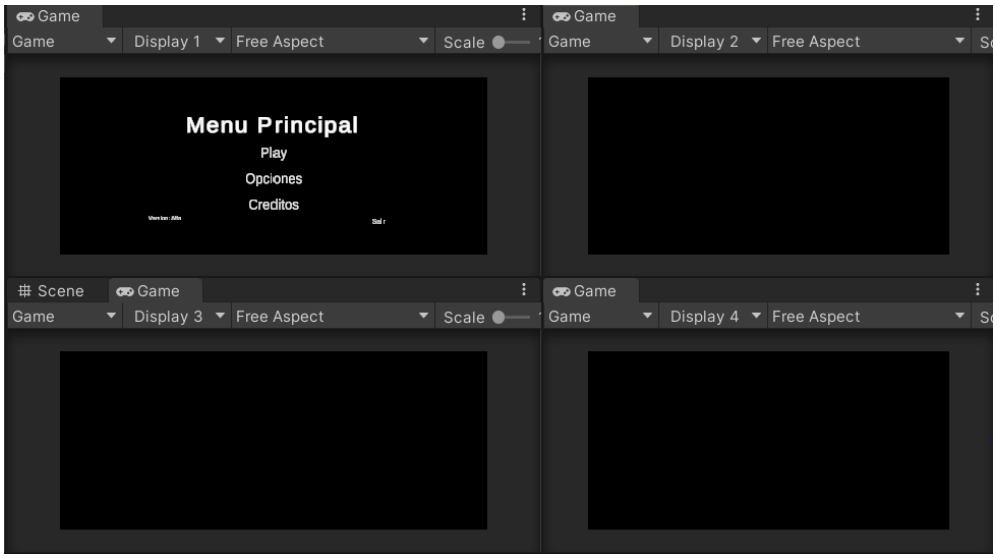

Figura 4.9 Scene Menu Principal

La escena cuenta con cuatro componentes para controlar la cámara que mostrará las diferentes visiones que usuario podrá disfrutar en cada pantalla y otro elemento que sirve como controlador de las funcionalidades como los botones de la escena, será el portador de los scripts que son utilizados en esta escena. También cuenta con un objeto canvas que contara con los textos y los elementos funcionales, en este caso botones que forman las escenas. Dentro del elemento canvas contaremos con dos textos independientes, uno para el titulo y otro para indicar la versión. También contamos con un boto independiente para salir del juego. Por último, contamos con un panel de botones que agrupa el texto y el botón del resto de opciones que tenemos para elegir dentro de la escena. Tal y como se puede observar en la imagen de a continuación.

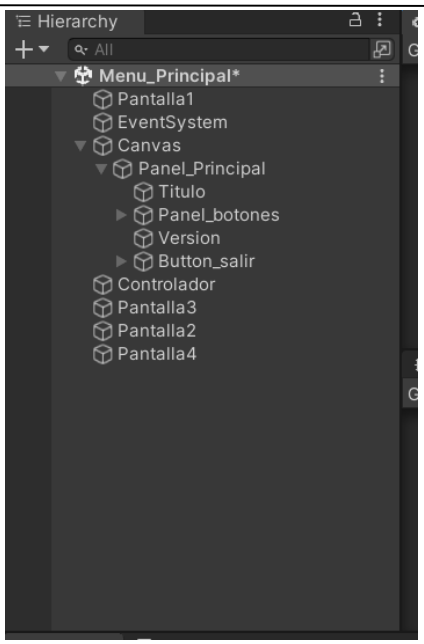

Figura 4.10 Scena Menu Principal

4.1.3.2.- Juego

Se trata de la escena principal del simulador, el corazón por así decirlo, en ella se llevarán a cabo la mayor parte de las acciones que forman el simulador. Desde la escena "Juego" el usuario realizara las interacciones propias del simulador, y es la encargada de controlar las interacciones con el entorno, las mecánicas y la jugabilidad. Además, esta escena gestiona gran parte de la lógica del juego.

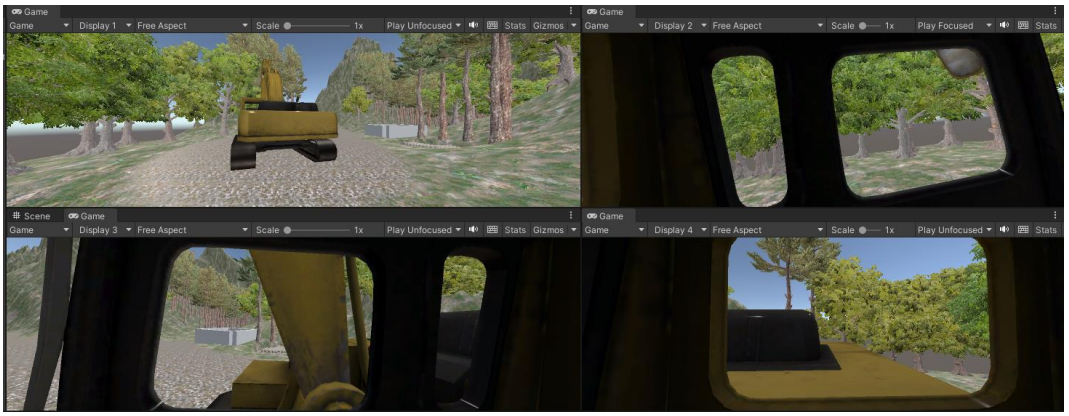

Figura 4.11 Scena Juego

La escena cuenta con números elementos como se puede apreciar en la siguiente imagen y aunque con todos ellos se construirá la experiencia de juego los elementos más relevantes son: la excavadora, la pantalla de pausa, el terreno, las cajas y la tierra.

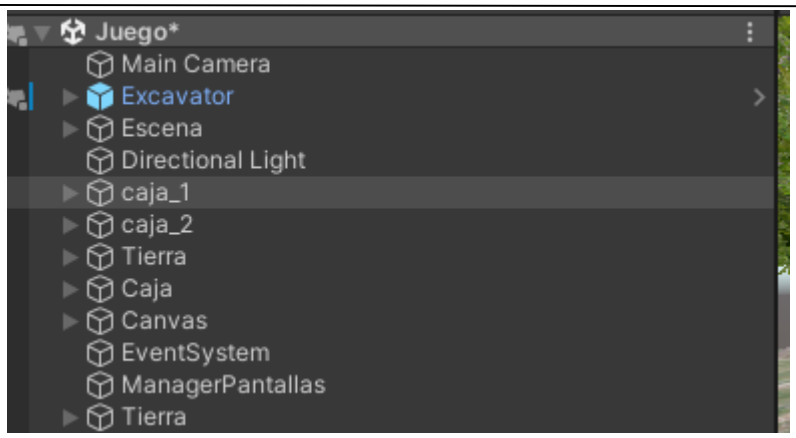

Figura 4.12 Elementos de la Scena Juego

La excavadora corresponde al elemento" Excavator" de nuestra escena y se trata de la maquina industrial principal de nuestro simulador. Se trata de un modelo 3D obtenido de la comunidad de desabolladores de Unity.

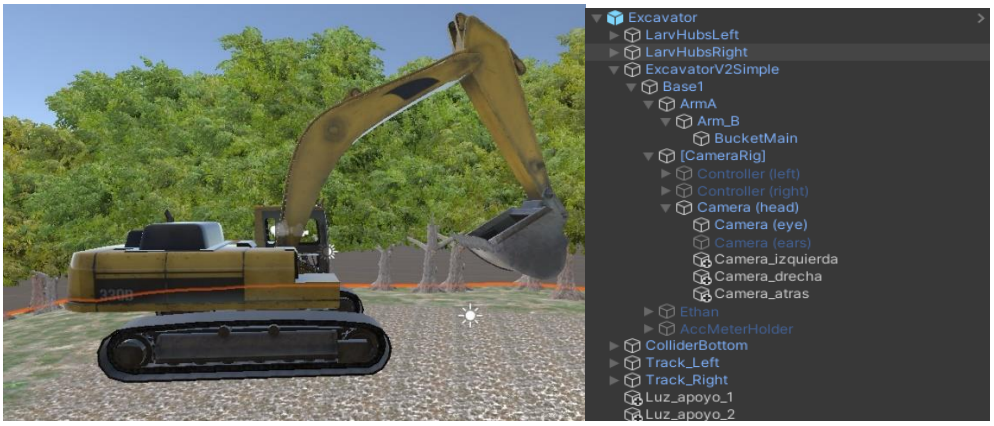

Figura 4.13 Elemento Excavator y sus partes

Las partes principales que se encargan del movimiento son las "LarvHubs", que se trata de un conjunto de elementos "Wheel Collider" [Manual Unity B] puestos en fila para formar una especie de oruga para dotar a la excavadora de movimiento. El "Wheel Collider" es un componente que Unity nos presenta ya tiene implementado y nos ayuda a simular los comportamientos de las ruedas de vehículos. Nos ayuda a controlar la física y la interacción de las ruedas con el suelo y la dirección de una forma sencilla. Son especialmente útiles para crear simulaciones realistas de vehículos en juegos y aplicaciones, ya que permiten a los desarrolladores la capacidad de controlar el movimiento y la dinámica de los vehículos de manera precisa. Las opciones de configuración son muy variadas y sencillas, como se pueden apreciar en la imagen de a continuación.

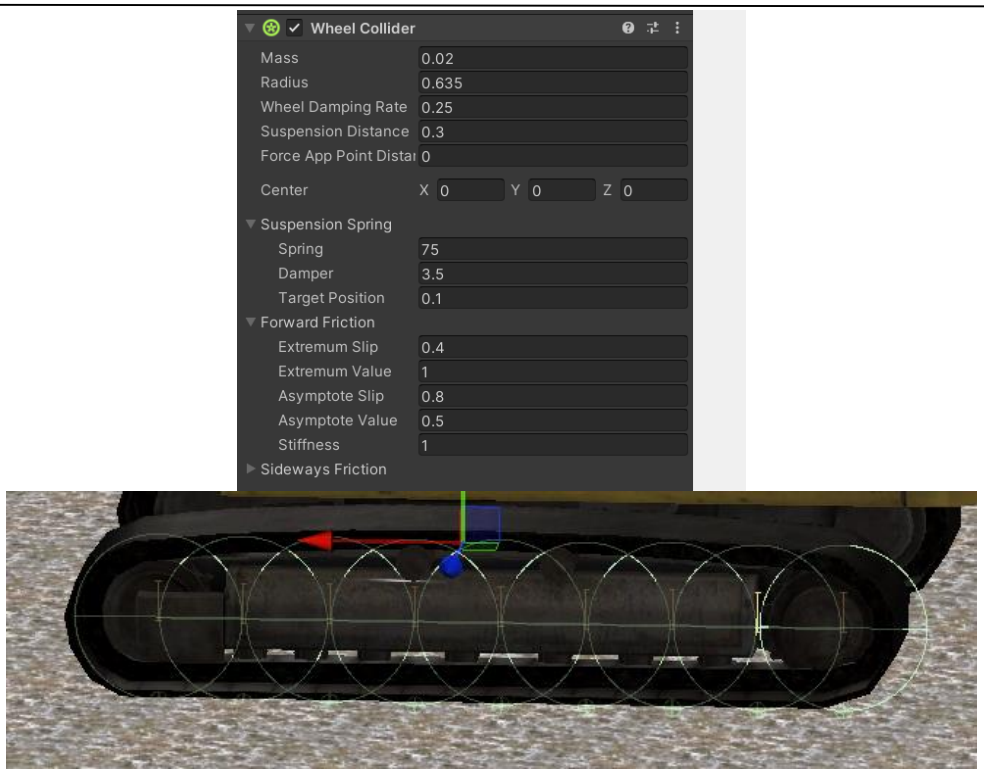

Figura 4.14 Ejemplo LarvHubs y Wheel Collider

El siguiente elemento importante en la estructura de la excavadora es la “Base1” que será el elemento encargado de llevar la lógica de gran parte de las acciones del simulador. Además, será padre de los elementos que forman el brazo del vehículo industriar y de los elementos cámaras, como se aprecia en la imagen de a continuación que nos dotara de la posibilidad de una visión de 360ª.

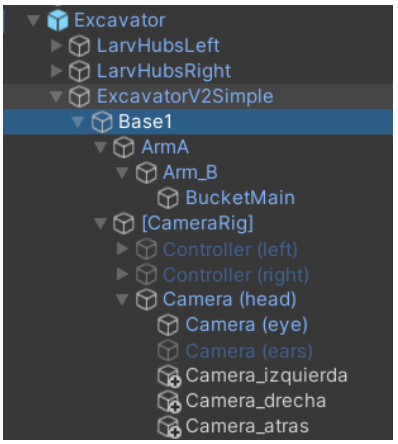

Figura 4.15 Ruta e Hijos de Base1

Este elemento también contendrá algunos scripts para funcionalidades y un elemento “Hinge Joint” [Manual Unity C]que nos permitirá más adelante dotar a la cabina de una rotación.

El "Hinge Joint" se emplea para simular una articulación del tipo bisagra entre dos objetos permitiendo que estos objetos roten alrededor de un eje común, El "Hinge Joint" también proporciona opciones para controlar la limitación del ángulo de rotación con un tope de +180 o -180 y la fricción en la articulación, lo que lo hace un componente muy útil, que Unity nos ofrece ya implementado, para simular mecanismos y estructuras articuladas.

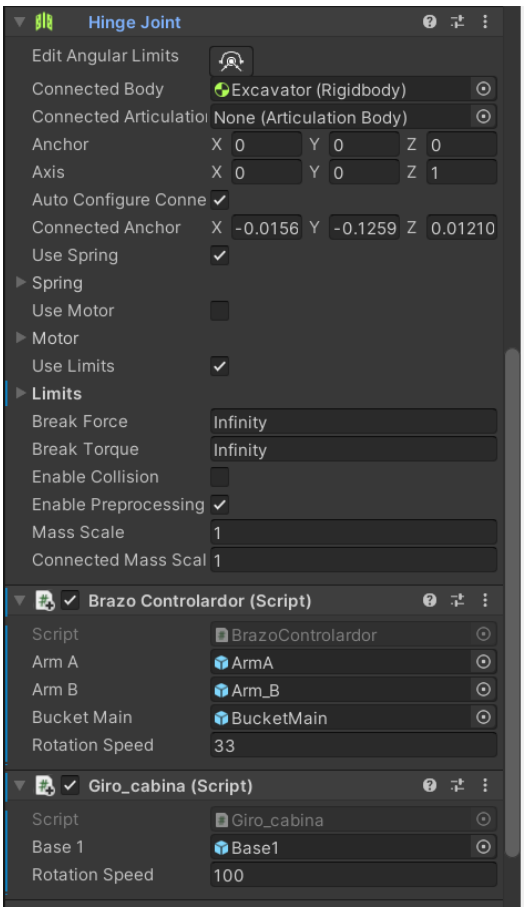

Figura 4.16 Elementos Principales de Base1

A continuación del elemento “Base1” nos encontraremos los elementos “ArmA”, “ArmB” y “BucketMain” que serán los encargados de articular el brazo de la excavadora y contarán con un elemento “Hinge Joint” cada uno para que a través de rotaciones se pueda simular el movimiento del brazo. Además, los componentes están relacionados unos con otros para realizar un movimiento más uniforme como se aprecia en la siguiente imagen.

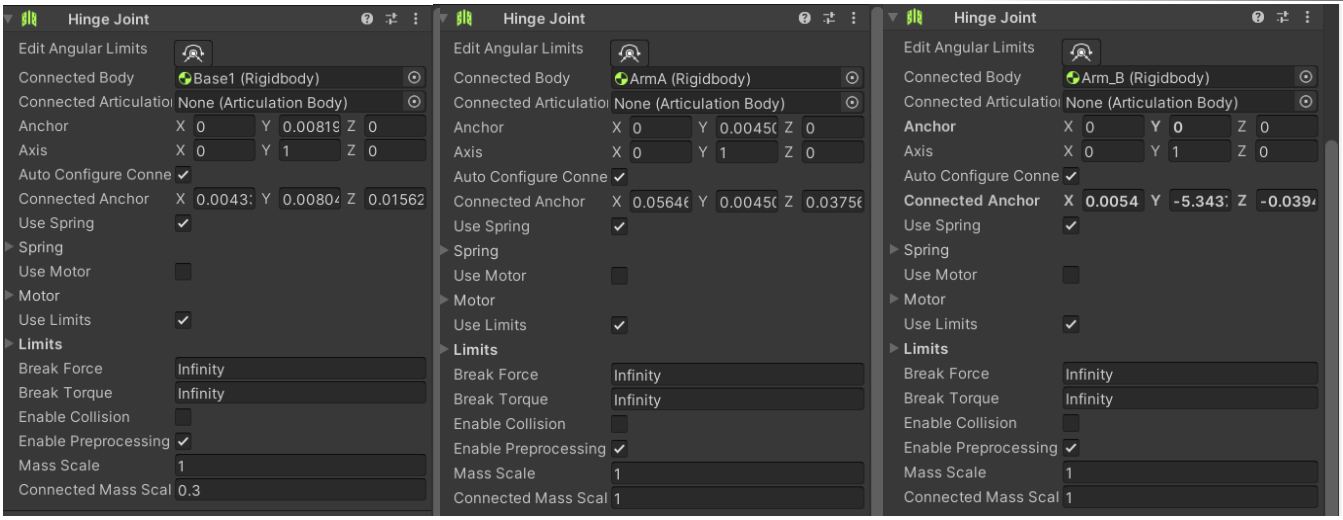

Figura 4.17 Elementos Hinge Joint de “ArmA” “ArmB” y “BucketMain”

Por último, del elemento “Base1” cuelgan elementos cámaras que forman la visión de 360ª dividendo la visión en cuatro display. El control de la imagen relacionada con cada display estará controlado por un script para que los display se encienda y se muestren las imágenes correspondientes de una forma automática.

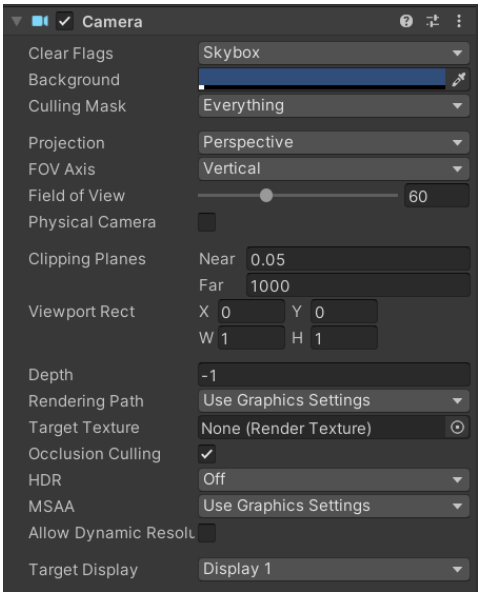

Figura 4.18 Ejemplo de Elemento Cámara

Además de estos elementos y componentes todos los elemento que forman la excavadora cuentan con “Rigidbody” [Manual Unity D] , “Box Collider”`[Manual Unity E] y “Shader”[Manual Unity F] para dotar a la excavadora de peso colisión y aspecto propio de una maquinaria industrial.

El "Rigidbody" es un componente que se emplea para simular la física de un objeto dentro del juego. Al incluir un Rigidbody en un objeto, este responde a fuerzas físicas como la gravedad. En consecuencia, el objeto realizara las acciones principales de movimiento, giro o reaccionar con otros elementos de una forma realista.

El "Box Collider" es un elemento que establece un rectángulo en forma de caja alrededor de un objeto tridimensional. Este componente es esencial para detectar colisiones y permitir las relaciones físicas entre los objetos de la escena de un juego.

Por último, los “Shader” como ya hemos comentado anteriormente se utilizan para controlar la apariencia visual de los objetos en una escena tridimensional. A continuación, mostramos una configuración de cada uno para mostrar un ejemplo.

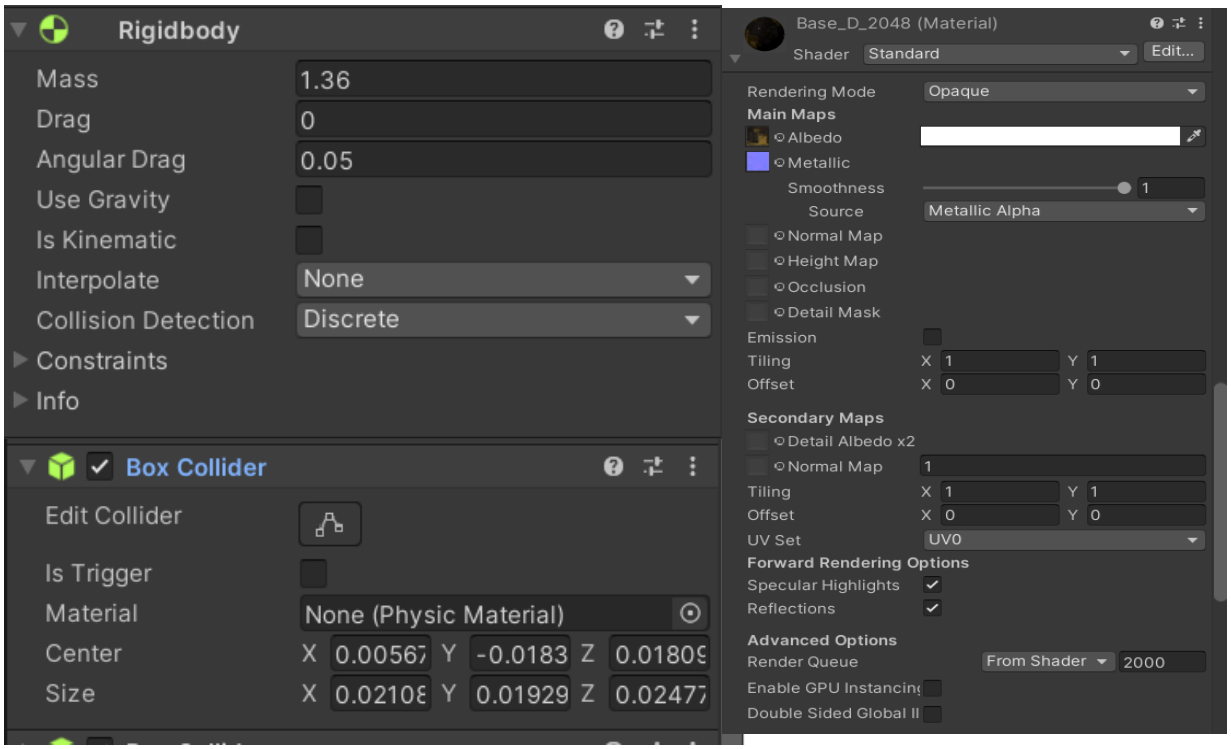

Figura 4.19 Configuración "Rigidbody” “Box Collider" y “Shader”

Otro elemento fundamental de la escena es “Terrain”, se trata de ser el encargado de formar el escenario que forma nuestro mundo virtual. Mediante este elemento dotaremos al mundo virtual de su topografía y los elementos naturales que lo forman como árboles o flores.

Figura 4.20 Elementos Terrain

Los componentes principales que lo forman son “Terrain” [Manual Unity G], que se trata de un editor del escenario con el que los os desarrolladores crear, editar y personalizar terrenos tridimensionales para sus juegos y aplicaciones. Este editor proporciona una serie de herramientas y características para colocar flora, aplicar texturas y modificar la topografía a su elección.

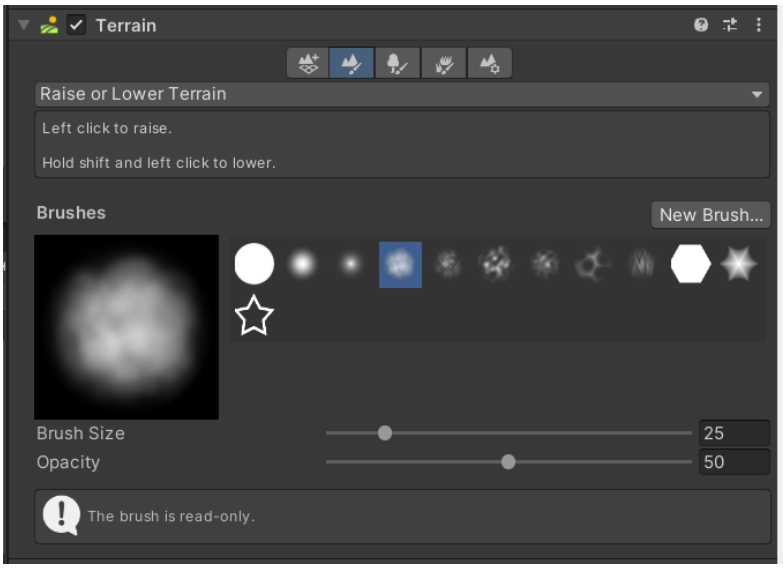

Figura 4.21 Herramienta Terrain

Como se puede apreciar en la imagen anterior contamos con diferentes opciones como: escultura de terreno, plantación de vegetación y pintura de texturas.

"Terrain Collider" se utiliza para dotar a los objetos físicos la posibilidad de que interactúen con el terreno. Este componente permite que el terreno y otros elementos, colisionen y reaccionen te de acuerdo con las leyes de la física del motor de juego Unity.

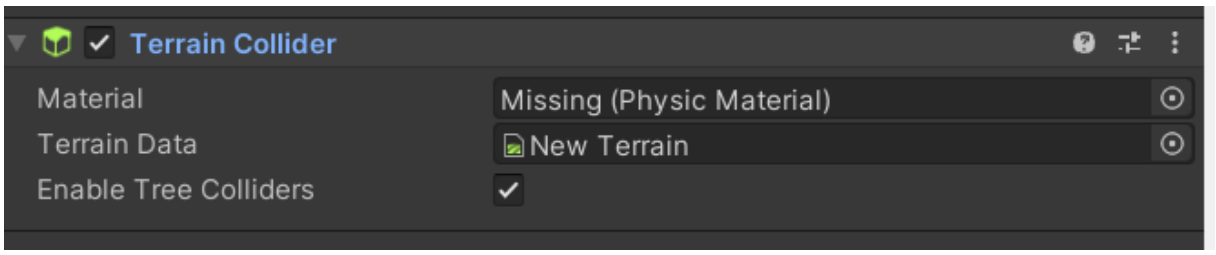

Figura 4.22 Componente Terrain Collider

Otro elemento de la escena seria la pantalla de pausa implementada mediante un elemento “Canvas” que será el padre de los elementos que formaran esta pantalla como textos o botones. Los hijos del “Canvas” están inicialmente desactivados y se activaran durante el desarrollo del juego mediante una acción manual que realice el usuario.

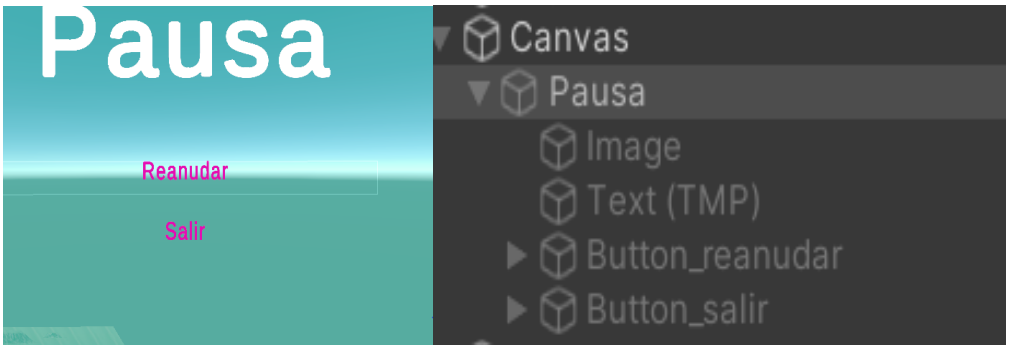

Figura 4.23 Componentes Canvas

Por último, encontraremos los elementos “Cajas” y “Tierra”. El primero se trata un conjunto de paredes, con comentes comunes como “Rigidbody", "Box Collider" y “Shader” mencionados anteriormente para ser funcionales, que serán utilizados para depositar la tierra excavada. La “Tierra” en cambio se trata de una matriz de elementos esfera que nos ofrece Unity, que serán los encargados de simular la tierra que debe de ser excavada por nuestra maquina industrial.

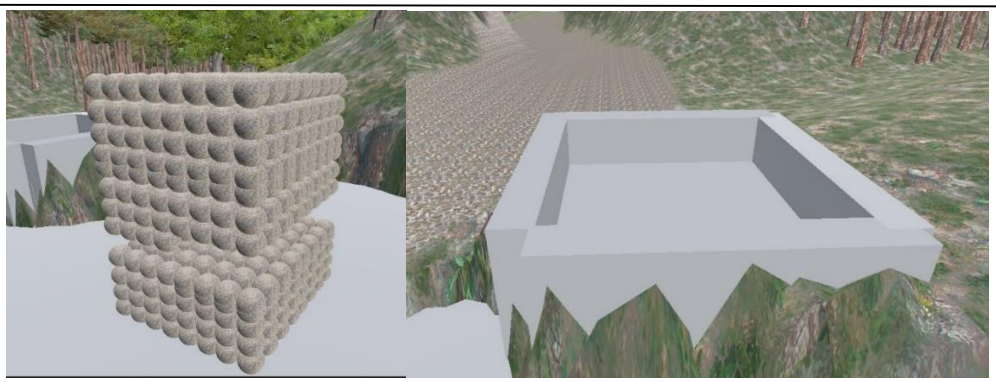

Figura 4.24 Ejemplos de “Cajas” y “Tierra”

4.1.3.3.- Créditos

La escena de créditos trata de dar un poco de información de las personas implicadas de forma directa en el proyecto como, diseñadores, programadores y otros profesionales y a personas indirectas como, personas que se consideren importantes por sus opiniones o apoyo.

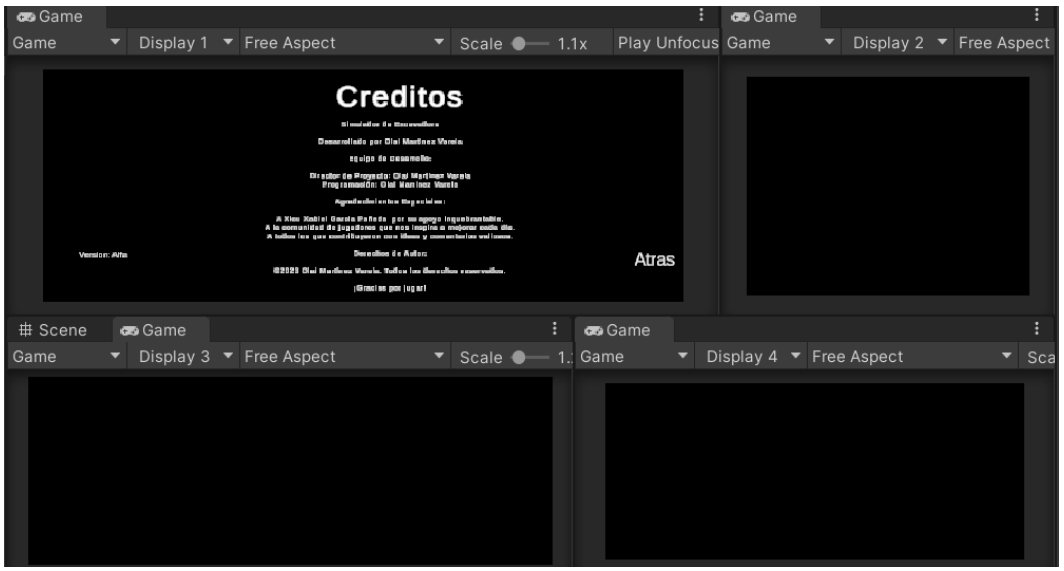

Figura 4.25 Scena Creditos

La escena cuenta con cuatro componentes para controlar la cámara para cada pantalla que actuaran igual que en la escena de menú principal y otro elemento que sirve como controlador de las acciones de la escena, será el portador de los scripts que son utilizados en esta escena. Además, contará con un canvas que cuenta con un panel principal que recoge el texto que se quiere transmitir y el título de la escena, además de los elementos funcionales, en este caso el botón atrás. En la imagen de a continuación se muestra los componentes que forman esta imagen,

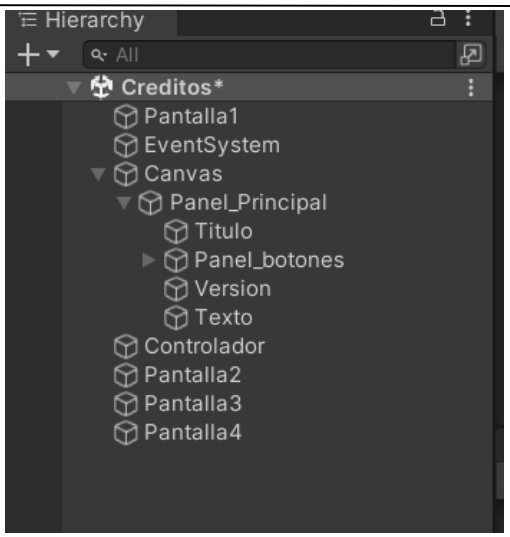

Figura 4.26 Elementos de la Scena Créditos

4.1.3.4.- Opciones

La escena de opciones trata de dar más oportunidades al jugador para poder modificar la experiencia de juego. Estas escenas añaden un elemento interactivo y de toma de decisiones. Estas opciones pueden variar desde el cambio de aspecto o del nivel de algunos aspectos del juego. En este caso se aun añadido dos sliders para controlar el volumen y el brillo. Actualmente no son funcionales y su implementación queda para una versión futura. Pero se ha decidido dejar ya la implementación del backend principal de la escena.

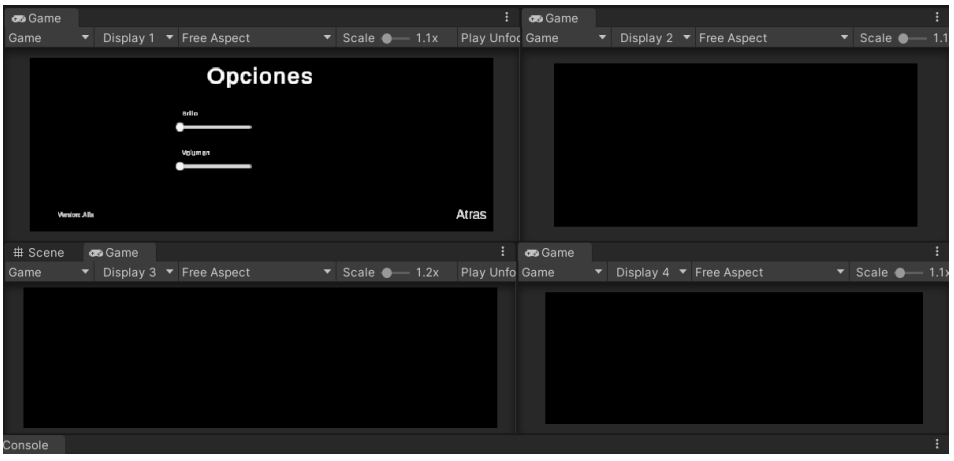

Figura 4.27 Scena Opciones

La escena cuenta con cuatro componentes para controlar la cámara para cada pantalla al igual que la escena anterior y el menú principal. Otro elemento que sirve como controlador de las acciones de la escena, será el portador de los scripts que son utilizados en esta escena. Por último, tenemos un elemento canvas con un panel principal que contara con los elementos

funcionales como los botones, en este caso un botón independiente que servirá para volver a la escena anterior y las sliders y su titilo para indicar que controla cada una, también contara con aspectos visuales como puede ser el texto de la versión y el título de la escena.

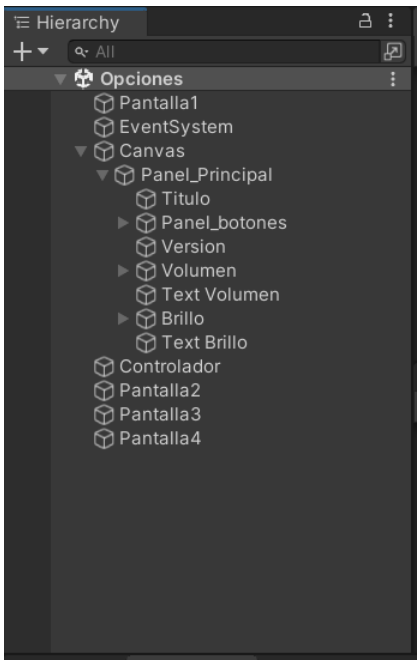

Figura 4.28 Elementos de la Scena Opciones

4.1.4.- Scripts

En esta carpeta se recogen todos los códigos necesarios para las diferentes funciones de nuestro proyecto. La información detallada de los scripts será comentada en apartados posteriores de la documentación, aunque un breve resumen es que han sido divididos en dos carpetas:

- Cámara
- Movimiento

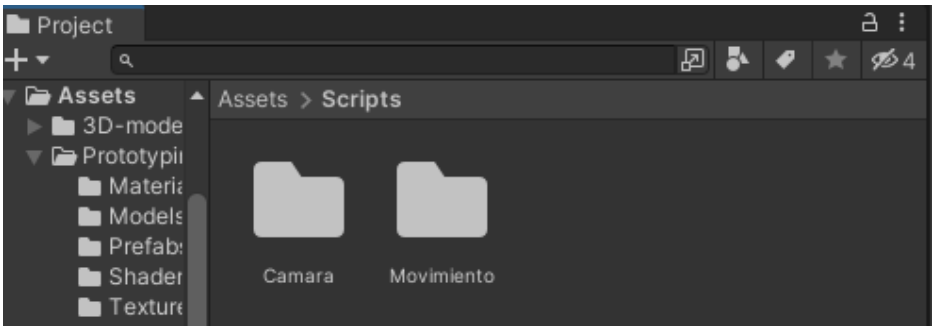

Figura 4.29 Carpeta Scripts

En la carpeta cámara se encuentran los scripts relacionados con las cámaras del jugador, la navegación por las distintas escenas del simulador y la configuración de los distintos display. Está formada por:

- Cámara_jugador
- Menús
- Pantallas

Figura 4.30 Carpeta Cámara

En la carpeta de movimiento se engloban los distintos scripts relacionados con todos los movimientos que puede realizar la excavadora. Está formada por:

- BrazoControlador
- Giro_cabina
- Movimiento

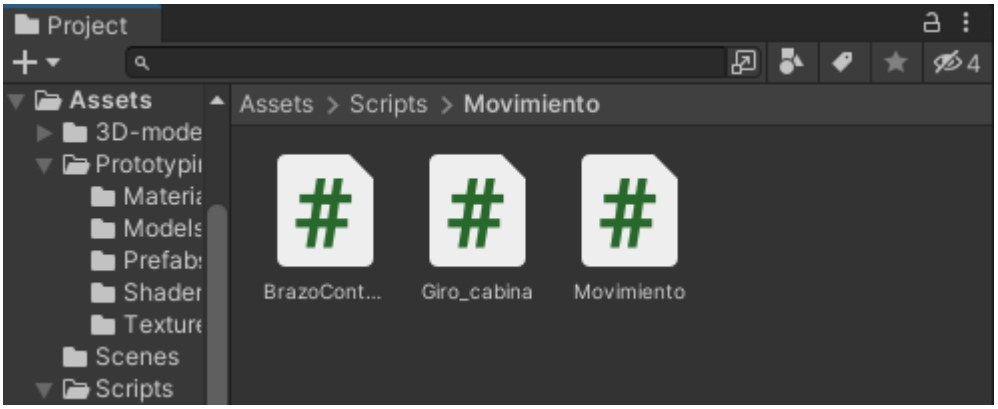

Figura 4.31 Carpeta Movimiento

2.1.5.- Terreno

En esta carpeta se engloban los recursos relacionados con la creación y el aspecto del mundo que ha sido creado. Desde la topografía de error, hasta las texturas empleadas. Algunos de estos recursos han sido obtenidos de miembros de la comunidad de desarrolladoras de GitHub, [Comunidad GitHub B] otros han sido obtenidos atreves de la Unity Assest Sotre como puede ser:

- Outdoor Gronud Texture [Unity Asset Store C]
- Terrain Textures Pack Free [Unity Asset Store D]
- Grass Flowers Pack Free [Unity Asset Store E]

Quiero destacar que todos estos paquetes son gratis y no se han utilizado al completo si no que se ha eliminado gran parte de su contenido y solo se ha mantenido en el proyecto los recursos utilizados de cada uno de ellos.

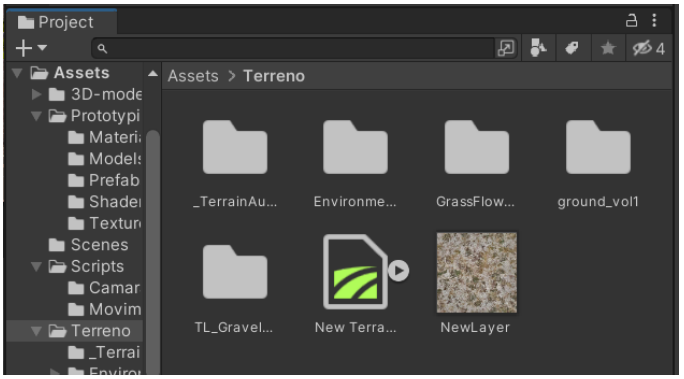

Figura 4.32 Carpeta Terreno

4.1.6.- TextMesh Pro

Se trata de una herramienta de descarga obligatoria para el tratamiento del texto en los proyectos. TextMesh Pro es una herramienta Unity, esencial para desarrolladores que buscan mejorar la calidad visual y la flexibilidad de representación de texto en sus proyectos. Con sus características avanzadas, optimización de rendimiento y soporte para múltiples idiomas, se convierte en una elección preferida para aplicaciones y juegos que requieren un control preciso sobre la apariencia del texto. [Manual TextMeshPro]

TextMesh Pro generalmente se integra en Unity como un paquete que puedes instalar desde el Package Manager. Una vez instalado, puedes reemplazar el componente de texto estándar en tus objetos con el componente "Text - TextMeshPro".

4.2.- BACKEND

Se trata de la parte del proyecto que engloba la lógica y la gestión de las acciones que se realizan por detrás de la escena. Algunas de las acciones que engloba son: manejar las solicitudes del frontend, procesamiento de datos, y realizar tareas relacionadas con las acciones funcionales del proyecto. Los usuarios no interactúan directamente con el backend, pero este está presente y juega una parte crucial para funcionamiento del proyecto.

4.2.1.- Lenguaje Empleado

Para todo el código encargado de gran parte de las funcionalidades del proyecto se utilizado C#. Es un lenguaje de programa ampliamente usado para el desarrollo de videojuegos en el entorno de desarrollo Unity. Hay varios motivos por los cuales la combinación de C# con Unity es la elección más popular entre los desarrolladores de juegos y aplicaciones. Aquí algunos de los motivos más destacados;

C# se trata del lenguaje principal de Unity. La API de Unity está preparada para ser utilizada con Unity, debido a esto la utilización de C# facilita la integración y desarrollo de videojuegos en esta plataforma.

Unity cuenta con una gran comunidad de desarrolladores que emplean C#. Esto implica que hay una gran cantidad de recursos, tutoriales, foros y activos disponibles como ayuda para superar problemas y aprender nuevas técnicas.

Se trata de un lenguaje de programación multiplataforma, lo que significa que el código escrito en C# para Unity puede ser compilado y ejecutado en diversas plataformas. Como pueden ser PC Mac y Android. Además, C# es un lenguaje de programación a objetos. Esto es muy útil en proyectos grandes y complejos ya que, facilita la organización y la estructuración del código.

Desde el punto de vista del rendimiento y la eficiencia C# es un lenguaje que combina la facilidad de uso con un buen rendimiento. Unity, junto con el compilador Just-In-Time (JIT) de

C#, ofrece un rendimiento eficiente, lo que es crucial para el desarrollo de juegos y aplicaciones en tiempo real.

En resumen, la combinación de Unity y C# ofrece una plataforma sólida y accesible para el desarrollo de juegos y aplicaciones interactivas, respaldada por una comunidad activa y recursos abundantes.

4.2.2 Editor de código fuente

El entorno de desarrollo integrado o IDE que he empleado para la edición código ha sido Microsoft Visual Studio. La elección de Microsoft Visual Studio se debe a que se trata de la elección más popular entre los desarrolladores, además presenta numerosas ventajas al ser combinado con Unity.

Visual Studio es la herramienta de desarrollo recomendada por Unity, ofrece una experiencia de desarrollo más coherente y fluida. También, se trata de un IDE con grandes capacidades y con un uso muy extendido pudiendo así ofrecer una amplia gama de recursos como, sintaxis autocompletado o depuración en tiempo real. Presenta la función IntelliSense de Visual Studio proporciona sugerencias de código y documentación en tiempo real mientras escribe, o la posibilidad de implementar plugins o extensiones con mejoras de funcionalidades incluso algunas específicas para Unity.

Destaca también por poder soportar otro tipo de lenguaje que no sean C# o por la compatibilidad con diferentes plataformas, permitiendo así trabajar tanto en Windows como en MacOs.

En conclusión, la combinación de Unity y Visual Studio ofrece una experiencia de desarrollo amena y eficiente, respaldada por una robusta integración entre ambas herramientas. Esto lo convierte en la elección más popular y eficiente para la creación de juegos o aplicaciones con Unity.

4.2.3 Scripts

Los scrpits son fundamentales para el correcto funcionamiento del simulador y las acciones que se realizan están directamente relacionadas con la experiencia que va a vivir el jugador. Los hemos agrupado en dos carpetas: Cámara y Movimiento

4.2.3.1.-Camara

Carpeta que engloba los scripts encargados de las cosas visuales del proyecto como, del seguimiento la cámara del jugador o la posibilidad de navegar por diferentes escenas. En esta carpeta se encuentra:

- Camara_jugador
- Menu
- Pantallas

4.2.3.1.1.- Camara_jugador

El código comienza incluyendo los espacios de nombres, “System.Collections”, “System.Collections.Generic” y UnityEngine. A continuación, creamos la clase “Camara_jugador” con herencia de “MonoBehaviour”, clase base de todos los scripts de Unity.

Dentro de la clase empezaremos creando una variable pública del tipo GameObject “player”, con esta variable podremos asignar el objeto jugador que se encuentra en la escena, desde el editor de Unity. Además, también incluirá dos métodos: “Void Start ()” y “Void LateUpdate ()”.

El método “Void Start ()” se llamará solo una vez al principio del juego, en este caso se encuentra vacío. El método “Void LateUpdate ()” se llamará una vez por frame y es donde se encuentras las funcionalidades. Primero se encuentra, “transform. position” que nos permitirá establecer la posición de la cámara igual a la cámara del jugador a continuación nos encontramos “transform.rotation” que nos permitirá establecer la rotación de la cámara igual a la cámara del jugador.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Camara_jugador : MonoBehaviour
{
    public GameObject player;

    void Start()
    {

    }

    // Update is called once per frame
    void LateUpdate()
    {

        transform.position = player.transform.position;
        transform.rotation = player.transform.rotation;

    }
}
```

Figura 4.33 Código Camara_jugador

Para resumir, el script se centra en permitir que la cámara pueda seguir al jugador, teniendo en cuenta tanto ajustes de posición y de rotación en cada frame para coincidir así con el jugador.

4.2.3.1.2.- Pantallas

El código comienza incluyendo los espacios de nombres, “System.Collections”, “System.Collections.Generic” y UnityEngine. A continuación, creamos la clase “Pantallas” con herencia de “MonoBehaviour”, clase base de todos los scripts de Unity.

Dentro de la clase incluiremos el método “Void Start () que será llamado solo una vez al inicio del juego y engloba la funcionalidad del script. En el método empezaremos sacando por consola el número de pantallas que tenemos conectadas. Con el bucle “for” recorreremos todas las pantallas y utilizando el código de su interior las iremos activando. El código se trata de una versión de un código recomendado en los manuales de Unity. [Manual Unity A]

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Pantallas : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
        Debug.Log("displays connected: " + Display.displays.Length);

        for (int i =1 ; i < Display.displays.Length; i++)
        {
            Display.displays[i].Activate();
        }


    }


}
```

Figura 4.34 Código Pantallas

El objetivo del script es activar todas pantallas disponibles. Esto será útil con configuraciones de pantallas extendidas en las que necesitaremos activar pantallas después del inicio del juego.

4.2.3.1.3.- Menú

El código comienza incluyendo los espacios de nombres, “System.Collections”, “System.Collections.Generic”,“UnityEngine”,“System.Security.AccessControl”,“System.Reflection”, “System.Threading”, y “UnityEngine.SceneManagement”. Este último espacio de nombres será esencial para trabajar con un sistema de escenas en Unity. A continuación, creamos la clase “Menu” con herencia de “MonoBehaviour”, clase base de todos los scripts de Unity.

El método “Void Start ()” que será llamado al inicio del juego, se encuentra vacío, mientras que el método “Void LateUpdate ()” , llamado cada frame, contamos una serie de bucles anidados. El bucle “if” principal comprobemos que sea pulsadas las teclas necesarias para acceder al interior. En su interior dependiendo de la variable booleana menú, comentado anteriormente, existen dos posibilidades. Si el valor es “false” se activará el menú de pausa liberando el cursor, se pausará el tiempo y por último se cambiará el valor de la variable a “true”. Por otra parte, se acede al bucle con un valor “true” se llamará al método continuar.

En las imágenes que se muestran a continuación encontraremos este código representado:

```
using System.Security.AccessControl;
using System.Reflection;
using System.Threading;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class Menu : MonoBehaviour
{

    public GameObject Pausa;
    public bool menu = false;
    // Start is called before the first frame update
    void Start()
    {

    }
```

Figura 4.35 Código Menu Fragmento 1

```
void Update()
{
    if (Input.GetButtonDown("Pausa")  || Input.GetKeyDown(KeyCode.Escape) )

    {
        if ( menu == false)
        {
            Pausa.SetActive(true);
            menu = true ;

            Time.timeScale = 0 ;
            Cursor.visible= true ;
            Cursor.lockState = CursorLockMode.None;

        }
        else if (menu == true)
        {
            continuar ();
        }

    }
}
```

Figura 4.36 Código Menu Fragmento 2

En la siguiente imagen encontramos el método “continuar” se trata del mimo código que, para la entrada en pausa, pero invirtiendo los valores para poder continuar la experiencia de juego, mientras que el método “escenajuego” realiza la función de saltar a la escena “TFG”, escena principal del juego, desde el lugar donde sea invocado.

```
public void continuar ()
{
    Pausa.SetActive(false);
    menu = false ;

    Time.timeScale = 1 ;
    Cursor.visible= false ;
     Cursor.lockState = CursorLockMode.Locked;
}

public void escenajuego ()
{
    Debug.Log("Clic detectado");
    SceneManager.LoadScene ("TFG");
}
```

Figura 4.37 Código Menu Fragmento 3

El método “cargar_nivel” realiza la misma función que el método comentado anteriormente con la diferencia que este valdría para cualquier escena indicando su nombre. Finalmente, el método “SalirDelJuego” se encarga de finalizar el proceso del videojuego cuando sea invocado. Como podemos observar en las imágenes estos métodos utilizan de forma sistemática el espacio de nombre UnityEngine.SceneManagement que está especialmente diseñado para la navegación entre escenas.

```
public void cargar_nivel ( string nombre_nivel )

{
    string nombreEscenaActual = SceneManager.GetActiveScene().name;

    // Descarga la escena actual
    SceneManager.UnloadSceneAsync(nombreEscenaActual);

    // Cargar la escena actual (esto reiniciará la escena)
    SceneManager.LoadScene(nombreEscenaActual, LoadSceneMode.Additive);
    SceneManager.LoadScene (nombre_nivel);

}
 public void SalirDelJuego()
{
  #if UNITY_EDITOR

  #else

    Application.Quit();
  #endif
}
```

Figura 4.38 Código Menu Fragmento 3

En resumen, este script se encarga de realizar las funciones relacionadas con la gestión de los menús y de la pantalla de pausa

4.2.3.2.- Movimiento

Carpeta que engloba los scripts encargados de las funciones motrices como, el movimiento del brazo o las funciones de movimiento típicas de cualquier vehículo. En esta carpeta se encuentra:

- Movimiento
- Gira_cabina
- BrazoControlador

4.2.3.2.1.- Movimiento

El código comienza incluyendo los espacios de nombres, "System.Collections", "System.Collections.Generic" y "UnityEngine. A continuación, creamos la clase "Movimiento" con herencia de "MonoBehaviour", clase base de todos los scripts de Unity.

Dentro de la clase creada empezaremos creando varias variables públicas que servirán para controlar aspectos como la fuerza del motor, el giro los colliders de las ruedas o la asignación de las wheelCollider a través de un vector. Además, también incluirá diferentes métodos: "Void Start ()", "Void LateUpdate ()" y "void AplicarFuerzaARuedas(WheelCollider [ ] ruedas, float velocidad)". En este archivo el método "Void Start ()" volverá a estar vacío como se aprecia en la imagen.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Movimiento : MonoBehaviour
{
    public WheelCollider[] ruedasIzquierdas;
    public WheelCollider[] ruedasDerechas;
    public float fuerzaMotorAdelante  = 50f;
    public float fuerzaMotorAtras = 25f;
    public float fuerzaGiro = 30f;
    public float velocidadFrenado = 2e+10f;
    public float velocidadMaxima = 5f;
    private float velocidadActual = 0f;
    private bool accionesHabilitadas = true;
    void Start()
    {
;
    }
```

Figura 4.39 Código Movimiento Fragmento 1

En el método “Void LateUpdate ()” se realizarán las funciones principales como obtener la entrada del acelerador y del giro del usuario o el cálculo de la fuerza que reciben las ruedas izquierdas y derechas. Se realizará el cálculo de la velocidad actual teniendo en cuenta la aceleración y la velocidad máxima permitida. También se hará la gestión de control de la posición de “Trabajo” y “Conducción” a través de los botones asignados en este caso hemos utilizado los input manager una mecánica explicada más adelante.

```
void Update()
{
    if (accionesHabilitadas == true)
    {
        float Acelerador = Input.GetAxis("Acelerador");
        float Giro = Input.GetAxis("Giro");


        float fuerzaMotor = Acelerador > 0 ? fuerzaMotorAdelante : fuerzaMotorAtras;

        velocidadActual = Mathf.Clamp(velocidadActual + Acelerador * fuerzaMotor * Time.deltaTime, -velocidadMaxima, velocidadMaxima);

        AplicarFuerzaARuedas(ruedasIzquierdas, velocidadActual);
        AplicarFuerzaARuedas(ruedasDerechas, velocidadActual);

        float giro = Giro * fuerzaGiro;

        transform.Rotate(Vector3.up, giro * Time.deltaTime);

        if ( Acelerador == 0 )
        {
            velocidadActual = 0f;
        }

        else
        {
            velocidadActual = Mathf.MoveTowards(velocidadActual, 0f, velocidadFrenado * Time.deltaTime);
        }
```

Figura 4.40 Código Movimiento Fragmento 2

Por ultimo , esta el metodo “void AplicarFuerzaARuedas(WheelCollider [ ] ruedas, float velocidad)”. Este método utilizara “motorTorque”para poder aplicar la fuerza a los colliders de las ruedas como muestra la imagen.

```
            if (Input.GetButtonDown("Trabajo"))
            {
                velocidadActual = 0f ;
                accionesHabilitadas = false;
            }

        }
             if (Input.GetButtonDown("Conducion"))
            {
                accionesHabilitadas = true;
            }
    }


    void AplicarFuerzaARuedas(WheelCollider[] ruedas, float velocidad)
    {
        foreach (WheelCollider rueda in ruedas)
        {
            rueda.motorTorque = velocidad;
        }
    }
}
```

Figura 4.41 Código Movimiento Fragmento 3

En resumen, el código de este script permite acelerar, frenar, girar y detenerse a un vehículo Unity. Además, no permitirá diferenciar entre dos estados, uno de conducción y otro destinado al trabajo.

4.2.3.2.2.- Gira_cabina

El código comienza incluyendo los espacios de nombres, “System.Collections”, “System.Collections.Generic” y UnityEngine. A continuación, creamos la clase “Giro_cabina” con herencia de “MonoBehaviour”, clase base de todos los scripts de Unity.

Dentro de la clase creada empezaremos creando varias variables públicas que servirán para controlar aspectos como la velocidad de rotación o una referencia al objeto “Base1” y otras variables privadas para controlar aspectos como, la declaración de un componente “HingeJoint” o la declaración de los ángulos de rotación. Además, también se establece una variable booleana para controlar si están habilitada la función de giro. Los métodos que también formaran la clase son: “Void Start ()” y “Void LateUpdate ()”.

El método “Void Start ()” se inicializa el ángulo de rotación y obtenemos el componente “HingeJoint” correspondientes a la base de la cabina y se establecerá una velocidad de rotación predeterminada.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Giro_cabina : MonoBehaviour
{
        public GameObject Base1;
        private HingeJoint  Base1Hinge;
        public float rotationSpeed;
        private float  rotateBase1;
        private bool accionesHabilitadas = false;

    void Start()
    {
        rotateBase1 = 0f;
        Base1Hinge = Base1.GetComponent<HingeJoint>();
        rotationSpeed = 0.25f;
    }
```

Figura 4.42 Código Giro_cabina Fragmento 1

El método “Void LateUpdate ()” se obtiene el resorte asociado al “HingeJoint” y se establece una configuración de la posición según el ángulo en el que se encuentre en ese momento. Realiza el control de la base según las entradas del usuario. Por último, habilita o deshabilita las funciones de giro según la entrada del usuario.

```
    void Update()
    {

        if ( accionesHabilitadas == true)
        {
            JointSpring Base1Spring = Base1Hinge.spring;
            Base1Spring.targetPosition = rotateBase1;
            Base1Hinge.spring = Base1Spring;


            if (Input.GetAxis("Giro") == -1 && rotateBase1 > Base1Hinge.limits.min)
            {
                rotateBase1 = rotateBase1 - rotationSpeed;
            }

            if (Input.GetAxis("Giro") == 1 && rotateBase1 < Base1Hinge.limits.max)
            {
                rotateBase1 = rotateBase1 + rotationSpeed;
            }

            if (Input.GetButtonDown("Conducion"))
            {
                accionesHabilitadas = false;
            }
        }
        if (Input.GetButtonDown("Trabajo"))
        {
            accionesHabilitadas = true;

        }
    }
}
```

Figura 4.43 Código Giro_cabina Fragmento2

En resumen, permite al usuario el control de la rotación de la cabina, así como la posibilidad de habilitar o la deshabilitar las acciones del script.

4.2.3.2.3.- BrazoControlador

El código comienza incluyendo los espacios de nombres, “System.Collections”, “System.Collections.Generic”, “System.Collections.Cryptography”, y “UnityEngine”. A continuación, creamos la clase “BrazoControlador” con herencia de “MonoBehaviour”, clase base de todos los scripts de Unity.

La clase comenzara declarando las variables encargadas de la rotación, la velocidad. Se crearán también referencias a los objetos del juego que forman el brazo y a los componentes “HingeJoint” asociados a las partes del brazo. Por último, se declara una variable booleana para poder diferencian si las acciones están permitidas o no y gestionar este cambio de estado.

```
using System.Security.Cryptography;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BrazoControlardor : MonoBehaviour
{

    public GameObject ArmA, ArmB, BucketMain;
    private HingeJoint ArmAHinge, ArmBHinge, BucketHinge;
    private float rotateArmA, rotateArmB, rotateBucketMain
    private float rotationSpeedB;
    private float rotationSpeedA;
    private float rotationSpeedC;
    private bool accionesHabilitadas = false;
```

Figura 2.44 Código BrazoControlardor Fragmento 1

La clase creada cotara a su vez con dos métodos: “Void Start ()” y “Void LateUpdate ()”. El método “Void Start ()” se inicializan variables creadas previamente y se realizan ajustes en la configuración del cursor para que no sea visible durante el proceso de simulación.

```
void Start()
{
    ArmAHinge = ArmA.GetComponent<HingeJoint>();
    ArmBHinge = ArmB.GetComponent<HingeJoint>();
    BucketHinge = BucketMain.GetComponent<HingeJoint>();
    rotateArmA = 0f;
    rotateArmB = 0f;
    rotateBucketMain = 0f;
    rotationSpeedA = 0.10f;
    rotationSpeedC = 0.20f;
    rotationSpeedB = 0.3f;
    Cursor.visible = false;
    Cursor.lockState = CursorLockMode.Locked;
}
```

Figura 4.45 Código BrazoControlardor Fragmento 2

El método “Void LateUpdate ()” comenzara actualizando la posición de los resortes “HingeJoint” para cada parte que forma nuestro brazo.

- ArmA
- ArmB
- BucketMain

Esta posición se actualizará basándose en las variables de rotación previamente creadas.

```
void Update()
{

        JointSpring ArmASpring = ArmAHinge.spring;
        ArmASpring.targetPosition = rotateArmA;
        ArmAHinge.spring = ArmASpring;

        JointSpring ArmBSpring = ArmBHinge.spring;
        ArmBSpring.targetPosition = rotateArmB;
        ArmBHinge.spring = ArmBSpring;

        JointSpring BucketMainSpring = BucketHinge.spring;
        BucketMainSpring.targetPosition = rotateBucketMain;
        BucketHinge.spring = BucketMainSpring;
```

Figura 4.46 Código BrazoControlardor Fragmento 3

A continuación, si las acciones están habilitas entraremos en un bucle donde se encontrará las lógicas para la rotación de las partes del brazo. Encontraremos un primer bucle que permite rotar los componentes ArmB y BucketMain si mantenemos pulsado un botón. A continuación, encontró códigos similares para realizar la rotación de los componentes ArmB y BucketMain por separado. También encontramos unas líneas de código utilizadas para realizar debug del

código que se han mantenido para presenta una opción de mostrar por consola el valor de una variable en tiempo real para conocer así mejor como usarla.

```
if ( accionesHabilitadas == true)
{
    if(Input.GetButton("Gatillo") )
    {
     if (Input.GetAxis("Brazo") > 0 && rotateArmB > ArmBHinge.limits.min )
     {
        rotateArmB = rotateArmB - rotationSpeedB;

     }

     if (Input.GetAxis("Brazo") < 0 && rotateArmB < ArmBHinge.limits.max   )
     {
        rotateArmB = rotateArmB + rotationSpeedB;
     }


     if  (Input.GetAxis("Cazo") > -0.2f && rotateBucketMain > BucketHinge.limits.min)
     {
        rotateBucketMain = rotateBucketMain - rotationSpeedC;

     }

     if (Input.GetAxis("Cazo") < 0.2f  && rotateBucketMain < BucketHinge.limits.max)
     {
        rotateBucketMain = rotateBucketMain + rotationSpeedC;
     }

    }
```

Figura 4.47 Código BrazoControlador Fragmento 4

```
if ( (Input.GetAxis("Brazo") > 0  && rotateArmA > ArmAHinge.limits.min) )
 {
        rotateArmA = rotateArmA - rotationSpeedA;

 }

 if ((Input.GetAxis("Brazo") < 0 && rotateArmA < ArmAHinge.limits.max)  )
 {
        rotateArmA = rotateArmA + rotationSpeedA;
 }

float cazoValor= Input.GetAxis("Cazo");
Debug.Log ("valor cazo = " + cazoValor);

if  (cazoValor > 0.0f && rotateBucketMain > BucketHinge.limits.min)
{
    rotateBucketMain = rotateBucketMain - rotationSpeedC;

}

if (cazoValor < 0.0f  && rotateBucketMain < BucketHinge.limits.max)
{
    rotateBucketMain = rotateBucketMain + rotationSpeedC;
}
```

Figura 4.48 Código BrazoControlardor Fragmento 5

Por último, encontramos el código necesario para controlar si las acciones están habilitadas, es decir, los bucles que nos permiten pasar entre los dos estados de mi videojuego, estado de conducción o estado de trabajo como se aprecia en el código de las imágenes de a continuación.

```
if (Input.GetButtonDown("Conducion"))
{
    accionesHabilitadas = false;
}



if (Input.GetButtonDown("Trabajo"))
{
    accionesHabilitadas = true;
}
```

Figura 4.49 Código BrazoControlardor Fragmento 6

En resumen, el código nos permite la interacción entre el usuario y el brazo mecánico de la excavadora, nos permite controlar partes en conjunto o algunos pates de forma independiente. Además, también podemos controlar en qué estado nos encontramos en el mundo virtual.

4.2.4.- Input Manager

Dentro de las herramientas con las que cuenta Unity, el "Input Manager" o Gestor de Entrada es un sistema que se emplea en el tratamiento de la entrada de dispositivos como, ratones, teclados, controles de videojuegos y otros dispositivos de entrada. Su cometido fundamental es mapear las acciones del usuario a entradas específicas del hardware, pudiendo así manejar las entradas de una manera más abstracta y adaptable. Algunas de las funciones más calves de los “Input Manager” son: [Manual InputManager]

- Soporte para Múltiples Dispositivos
- Reasignación de Teclas
- Mapeo de Botones y Ejes
- Entrada Cross-Plataforma

.Pero los componentes principales de la configuración son:

- Name: nombre con el que identificaremos el input mánager correspondiente al llamarlo en el Código.
- Negative Button: asociación con la dirección negativa.
- Positive Button: asociación con la dirección positiva.

- Gravity: relación con l velocidad con la que el valor se acerca a cero cuando el botón no está siendo presionado.
- Dead: aumenta el valor del rango en el que considerar cero, algunos controles con el uso el valor de cero se pierde y es útil aumentar este rango del cero.
- Sensitivity: ajusta la velocidad de reacción.
- Snap: si este habilitado cuando se deja de utilizar el botón el valor se ajusta a cero
- Invert: si se lesiona invierte los valores.
- Type: sirve para determinar el tipo de entrada que se asocian con un botón o eje
- Axis: hace referencia a los ejes en los que afecta.
- Joy Num: especificar el número de joystick al que se aplicará esa configuración. Esto es útil cuando tienes múltiples joysticks conectados a tu sistema y deseas asignar acciones específicas a un joystick en particular.

La configuración de los Input Manager empleados en el desarrollo será mostrada en las imágenes siguientes con una breve explicación de cada uno.

4.2.4.1.- Conducion

Este Input Manager se ha relacionado con el botón “joystick button 1” que corresponde al botón número dos de la botonera que se menciona como “Joystick 1” para Unity como se puede apreciar en la imagen introducida a continuación. Con su utilización haremos referencia a el botón comentario. Anteriormente esto deberá de ser tenido en cuenta a la hora de escribir el código. Aquí los valores de gravity, dead y sensitive no son relevantes y se ha utilizado una configuración recomendada Unity. En cambio, es importante indicar el tipo, este caso tipo botón, como se aprecia en la imagen.

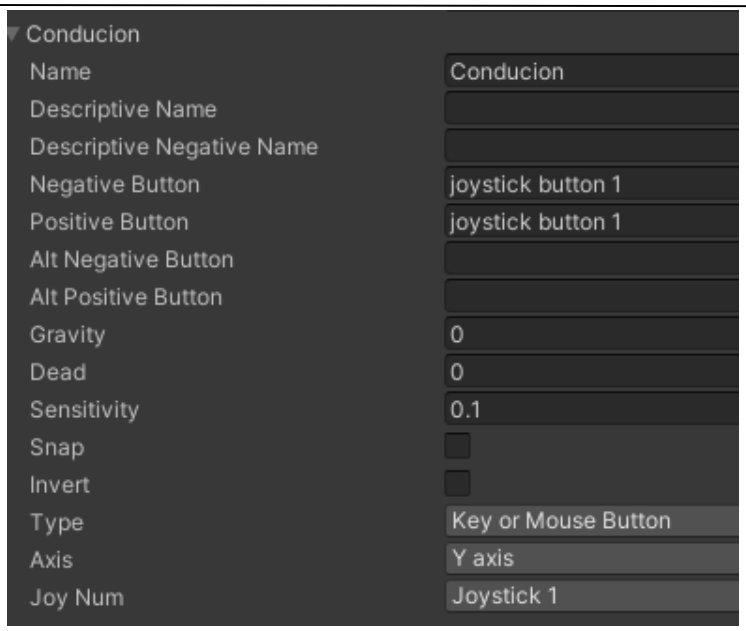

Figura 4.50 Configuración Input Manager Conducion

4.2.4.2.- Trabajo

Se trata de un Input Manager relacionado con el botón “joystick button 0” que corresponde al botón número uno de la botonera indicado como “Joystick 1” para Unity como se puede observar en la imagen inferior.

Hace referencia al botón comentario, esto deberá de ser tenido en cuenta a la hora de escribir el código. En este caso nuevamente los valores de gravity, dead y Sensitive no son relevantes y se ha usado una configuración recomendada Unity.

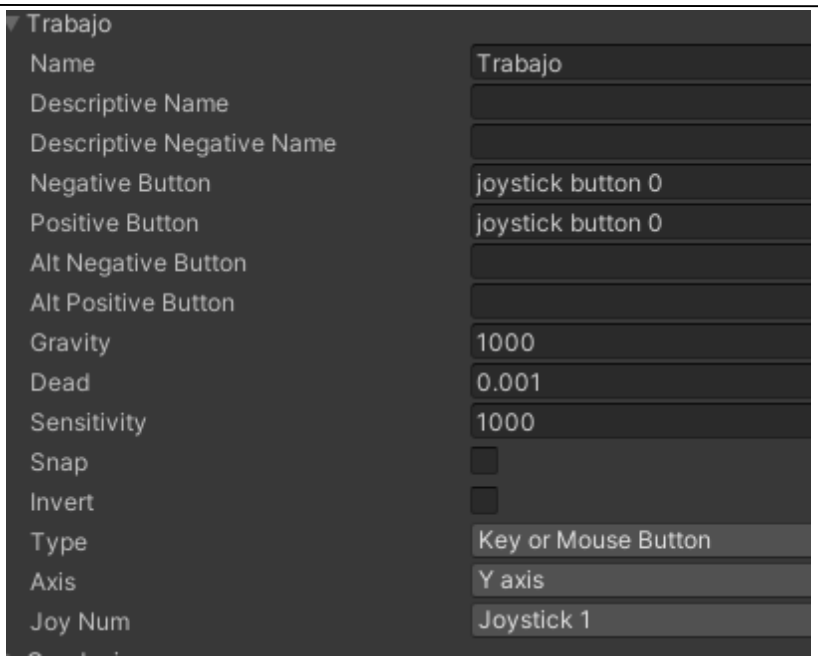

Figura 4.51 Configuración Input Manager Trabajo

4.2.4.3.-Giro

El Input Manager Giro se relaciona con las posiciones “left” y ”right” que corresponden con la rotación del volante indicado como “Joystick 2” para Unity como se puede apreciar en la imagen.

Con él hacemos referencia a la rotación del volante esto deberá de ser tenido en cuenta a la hora de escribir el código. En este caso los valores de gravity, y Sensitive se trata de valores genéricos.

El campo dead ha sido calculado a través de un proceso de debug del código para averiguar la zona muerta del volante y que esta no repercutirá en la experiencia de juego además de este cálculo también se ha seleccionado el campo snap para asegurar que vuelva al valor cero cuando no esté en uso. Además, se ha elegido el tipo axis para que el giro sea de una forma gradual según la rotación del volante.

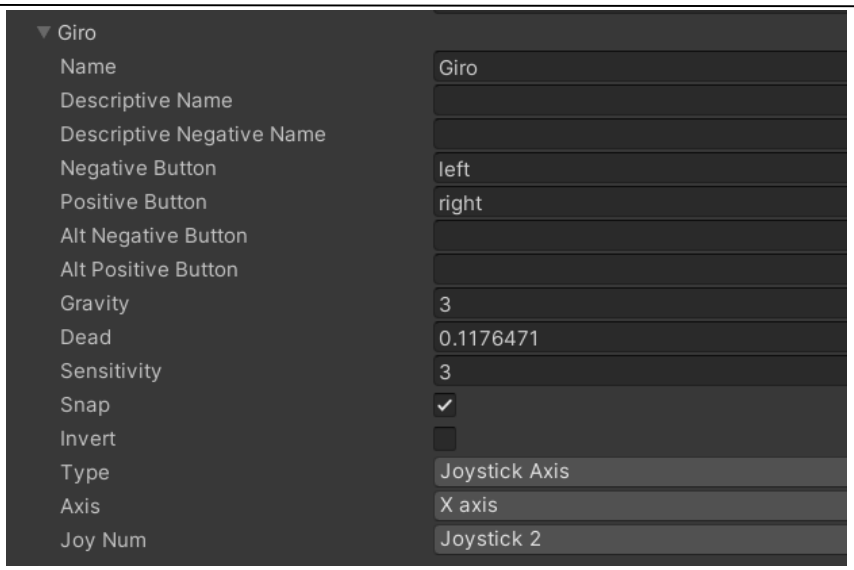

Figura 4.52 Configuración Input Manager Giro

4.2.4.4.- Acelerador

Acelerador hace referencia a los botones “joystick button 1” y ” joystick button 0” vinculados a los pedales de la pedalera del volante, nombrados como “Joystick 2” para Unity como se observa en la imagen inferior.

Su utilización hace referencia a los pedales, uno para movernos para adelante y otro para atrás, esto deberá de ser tenido en cuenta a la hora de escribir el código. En este caso los valores de gravity, y Sensitive son genérico.

Calculamos el campo dead a través de un proceso de debug del código para averiguar la zona muerta del volante y que esta no afectará a la experiencia de juego. Además de este cálculo también se ha seleccionado el campo snap para asegurar que vuelva al valor cero cuando no esté en uso y la selección de invert porque necesitamos invertir los valores como apreciamos en la imagen. Se ha optado por el tipo axis para que la detección de presión en el pedal sea de una forma gradual logrando un movimiento natural.

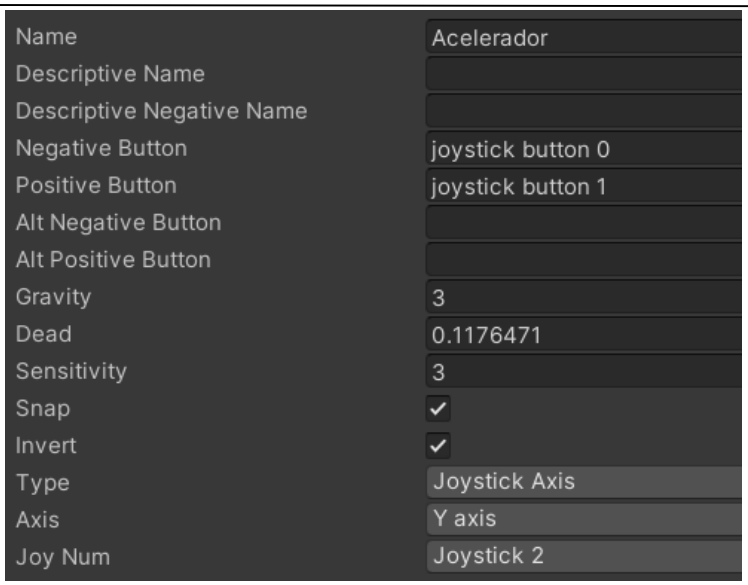

Figura 4.53 Configuración Input Manager Acelerador

4.2.4.5.- Gatillo

Este Input Manager se ha relacionado con el botón “joystick button 6” que encontraremos en el botón posterior de lado izquierdo del volante que se indica como “Joystick 2” para Unity como se puede apreciar en la imagen.

El uso de este Input Manager hace referencia al botón comentado anteriormente esto deberá de ser tenido en cuenta a la hora de escribir el código. En este caso los valores de gravity, dead y Sensitive vuelven a ser valores genéricos. Si resulta importante indicar el tipo, este caso tipo botón.

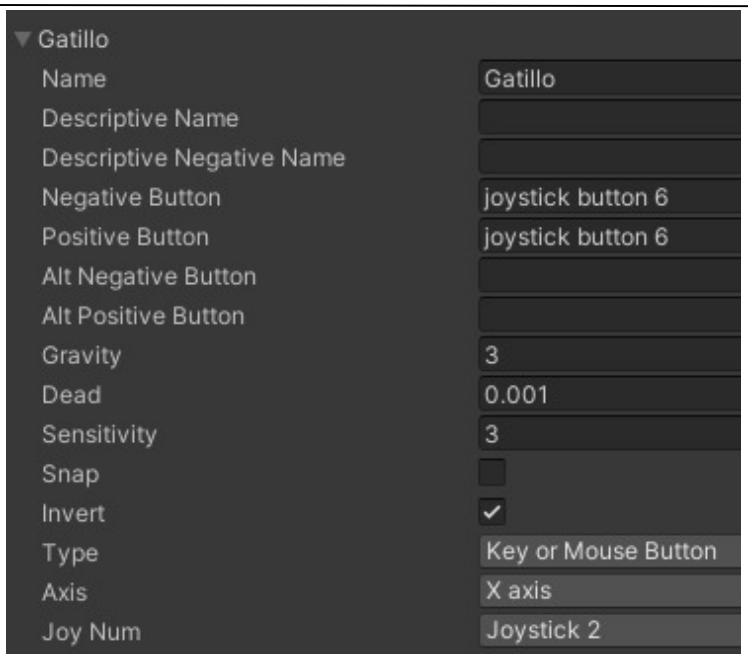

Figura 4.54 Configuración Input Manager Gatillo

4.2.4.6.- Pausa

Este Input Manager se localiza en el botón “joystick button 15” que corresponde al botón número dieciséis de la botonera que se manifiesta como “Joystick 1” para Unity, según se observa en la imagen.

Con la llamada de este Input Manager siempre haremos referencia a el botón comentario anteriormente, lo que deberá de ser tenido en cuenta a la hora de escribir el código. Los valores de gravity, dead y Sensitive no son relevantes y se ha utilizado una configuración recomendada Unity. En cambio, es relevante indicar el tipo, este caso tipo botón.

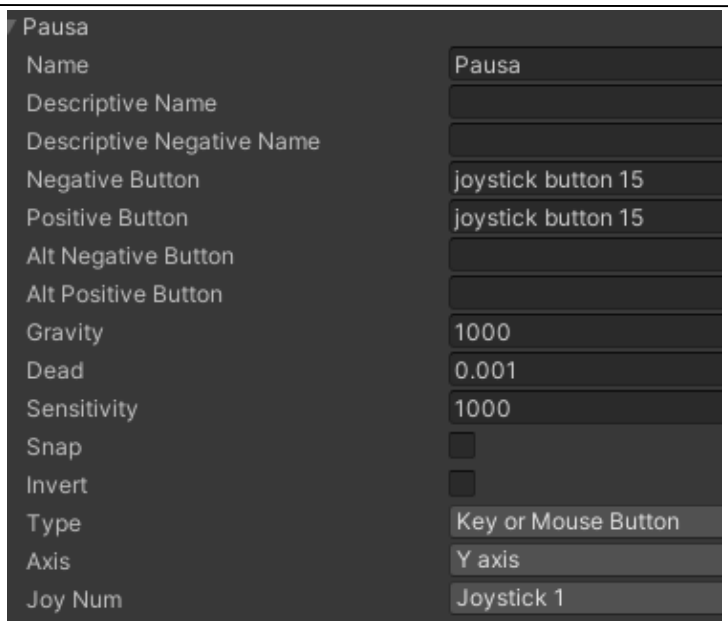

Figura 4.55 Configuración Input Manager Pausa

4.2.4.6.- Brazo

Este Input Manager se ha ubicado en el eje Y que corresponden con Joystick de la botonera que Unity, indicado como “Joystick 1” para Unity como se puede apreciar en la imagen inferior.

Con este Input Manager siempre haremos referencia a este eje para su movimiento, esto deberá de ser tenido en cuenta a la hora de escribir el código. En este caso los valores de gravity, y Sensitive se trata de valores genéricos nuevamente.

El campo dead se ha calculado tras un proceso de debug del código para averiguar la zona muerta del volante y que esta no afectará al juego. La selección de invert se ha realizado por la necesidad de invertir los valores como apreciamos en la imagen. Se ha optado por el tipo axis para que la detección de presión de Joystick sea de una forma gradual y así el movimiento sea más natural.

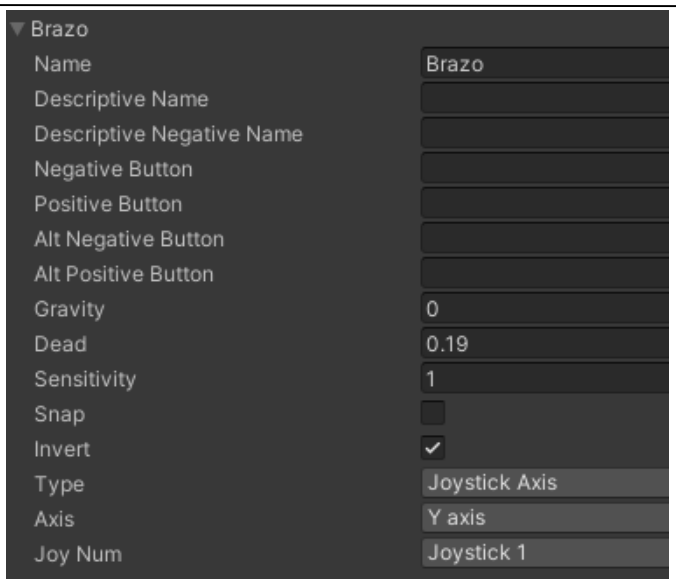

Figura 4.56 Configuración Input Manager Brazo

4.2.4.7.- Cazo

Este Input Manager se ha relacionado con al eje X que corresponden con Joystick de la botonera que Unity referencia como “Joystick 1” para Unity como se puede apreciar en la imagen.

La utilización de este Input Manager nos lleva a este eje para su movimiento, lo que deberá de ser tenido en cuenta a la hora de escribir el código. En este caso los valores de gravity, y Sensitive se trata de valores genéricos.

En cambio, la campo dead ha sido calculado a través de un proceso de debug del código para averiguar la zona muerta del volante, sin repercutir en el juego. La selección de invert se ha realizado por la necesidad de invertir los valores como se plasma en la imagen. De nuevo nos hemos decantado por el tipo axis para que la detección de presión de Joystick sea de una forma gradual, consiguiendo un movimiento mucho más natural.

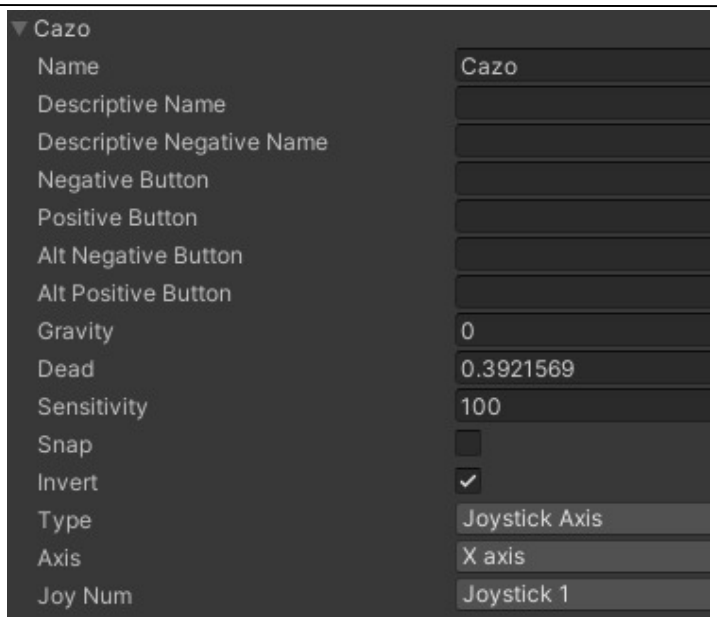

Figura 4.57 Configuración Input Manager Cazo

5. Requisitos

En esta sección, se describirán los requisitos de la aplicación, los cuales se dividen en tres categorías: requisitos operativos, requisitos de calidad y mínimos tecnológicos.

5.1.- REQUISITOS OPERATIVOS

Los requisitos operativos consisten en declaraciones concretas y detalladas que delinean los atributos y comportamientos esenciales que un sistema o software debe exhibir para alcanzar los objetivos y aún más importante, para satisfacer las necesidades de los usuarios finales. Estos requisitos se enfocan en contestar a cuestiones como:

- Acciones y tareas para realizar
- Características fundamentales.
- Capacidades que cubrir.

Estos requisitos escriben las acciones y comportamientos particulares que se esperan del sistema, tales como funciones específicas, operaciones, interacciones con los usuarios, cálculos, procesamientos de datos y cualquier otra actividad necesaria para el funcionamiento del sistema. Todos definen las capacidades y características esenciales que el sistema debe poseer para satisfacer de manera integral las demandas y expectativas del usuario.

ID	DEFINICION	PRIORIDAD
RO-1	Capacidad de movimiento en diferentes direcciones	ALTA
RO-2	Capacidad de giro en diferentes direcciones	ALTA
RO-3	Capacidad de rotación de la cabina	MEDIA
RO-4	Capacidad de movimiento del brazo de la excavadora	ALTA
RO-5	Capacidad de movimiento del cucharon	MEDIA
RO-6	Capacidad de posición de trabajo	BAJA

Tabla 5.1 Requisitos operativos

5.2.- REQUISITOS DE CALIDAD

Los requisitos de calidad hacen referencia a las características y criterios que establecen los estándares de calidad del sistema, con más profundidad que las funcionalidades principales de nuestro simulador. Con estos requisitos tratamos de establecer los criterios de confiabilidad, seguridad, rendimiento, escalabilidad y otros aspectos de afecten a la calidad de nuestro simulador. Además, también se centran en cómo debe de ser el sistema en lugar de que acciones debe hacer el sistema como en caso de los requisitos operativos.

Los requisitos de calidad tratan características y limitaciones del sistema como:

- Capacidad de manejar carga de usuarios
- Compatibilidad de plataformas
- Tolerancia a fallos
- Velocidad de respuesta
- Escalabilidad
- Mantenibilidad

ID	DEFINICION
RC-1	Garantizar la seguro para el usuario y sus cercanías
RC-2	Proporcionar una respuesta en tiempo real de las acciones de los mandos del simulador para una correcta experiencia.
RC-3	Dar una interfaz de usuario fácil y comprensible hasta para usuarios nuevos en el uso de simuladores.
RC-4	Proporcionar de una experiencia realista al usuario
RC-5	Proporcionar compatibilidad con el sistema operativo para el cual se ha realizado el diseño
RC-6	Posibilidad de añadir nuevas funcionalidades al sistema en un futuro
RC-7	Asegurar cierto nivel de precisión y control en los movimientos de la excavadora para la correcta experiencia de juego.

Tabla 5.2 Requisitos de calidad

5.3.- MINIMOS TECNOLÓGICOS

Los mínimos tecnológicos se refieran a las tecnologías que cuentan con las especificaciones y capacidades mínimas, tanto de hardware como software, que son necesarias para un funcionamiento correcto y eficiente del simular para poder satisfacer las necesidades del usuario.

5.3.1.- Requisitos de hardware

ID	DEFINICION
RH-1	Escenario con simulación compuesto por asiento pantalla o pantallas y los periféricos necesarios para los controles
RH-2	Memoria RAM de 8GB.
RH-3	Procesador AMD Ryzen 7 48000H o similares.
RC-4	Tarjeta Grafica NVIDEA GeFore GTX 1650
RC-5	Puertos USB: tantos como sean necesarios para permitir la conexión de los periféricos que se van a utilizar.
RC-6	Puertos HDMI: tantos como sean necesarios para permitir la conexión de los periféricos que se van a utilizar.

Tabla 5.3 Requisitos de hardware

5.3.2.- Requisitos de software

ID	DEFINICION
RH-1	Sistema Operativo Windows o versiones compatibles.
RH-2	Motor de desarrollo Unity

Tabla 4.4 Requisitos de software

6. Manuales

6.1.- MANUAL DE USO

El manual de uso se trata de un documento culla finalidad es proporcionar instrucciones y las nociones básicas de cómo se debe utilizar un producto, en este caso los controles de un videojuego.

Suele contener la información necesaria para una correcta instalación, desarrollo y mantenimiento para dar una solución rápida de problemas. Su finalidad es servir de guía al usuario para poder acceder al producto de una forma correcta y satisfactoria.

En este caso daremos una breve explicación de los controles que son necesarios para el correcto funcionamiento de nuestro videojuego. Los periféricos utilizados será una combinación de pedales, volante y una botonera. La combinación de este periférico se suficiente para el disfrute de la experiencia de juego.

Figura 6.1 Periféricos Necesarios

Para el funcionamiento típico de un vehículo utilizaremos el volante y la pedalera de la forma:

Movimiento	Tecla
Acelerar	Pedal derecho
Retroceder	Pedal izquierdo
Giro	Circunferencia volante

Tabla 6.1 Controles Movimiento vehículo

La botonera será utilizada en conjunto al volante para dotar de movimiento algunas funcionalidades más complejas como el giro de la cabina o el movimiento del brazo. Los controles serán:

Movimiento	Tecla	
	Volante	Botonera
Modo Trabajo		botón 1
Modo Conducción		botón 2
Modo Pausa		botón 16
Brazo A		Joystick posición Eje Y
Brazo B	Gatillo posterior izquierdo	Joystick posición Eje Y
Cazo		Joystick posición Eje X
Cabina	Circunferencia volante	

Tabla 6.2 Controles Movimiento industriales

Cabe a destacar que lo movimientos industriales principales de bazas, cazo y cabina solo se podrán realizar si previamente se ha pasado a modo de trabajo, mientras no se indique el cambio estaremos en modo de conducción, modo definido al inicio del videojuego, y solo serán posibles los movimientos de conducción del vehículo, los cuales quedaran desactivados a la hora de presionar el botón para cambiar a modo de conducción.

Para hacernos una idea más concreta de los controles contamos con la siguiente imagen:

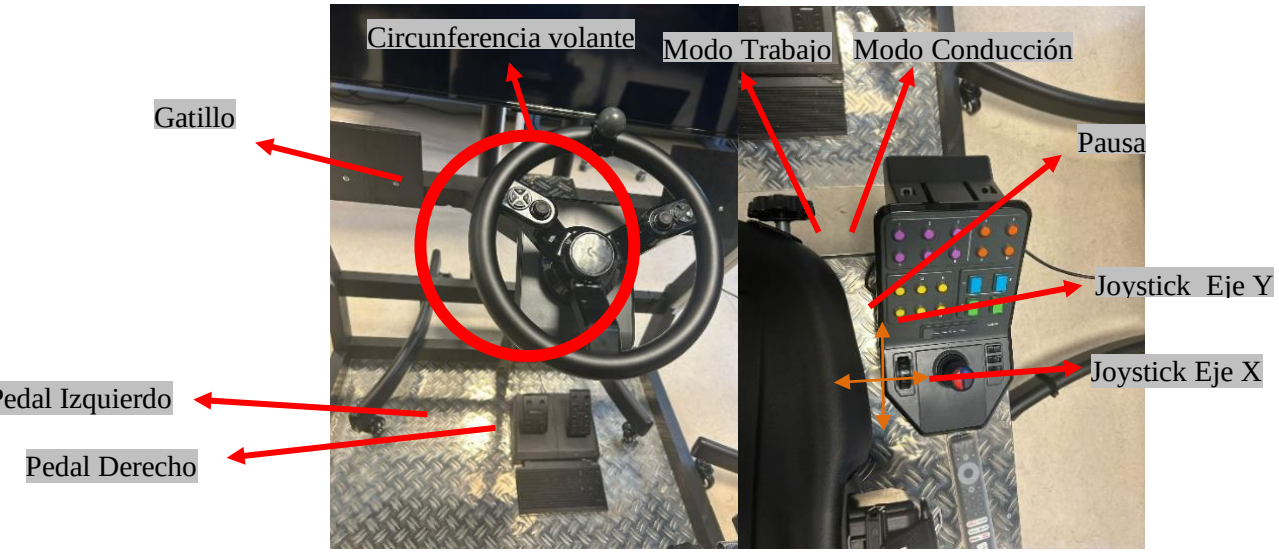

Figura 6.2 Controles del simulador

Además de todos estos controles descritos anteriormente también será necesario la utilización de un ratón para hacer las funciones básicas como la de acceder al juego o desplazarse por las diferentes escenas y clicar para interactuar con algunos elementos como los botones o las barras deslizadoras del menú de opciones.

6.2.- MANUAL DE INSTALACION

En la deberemos tener instalado Unity para así tener acceso a la visión de desarrollador para realizar mejoras si nuestra intención es continuar o ayudar en el desarrollo de la versión beta que presentamos.

En cambio, sí nuestra intención es solo disfrutar del juego debemos conectar correctamente los periféricos para que sean detectado por el PC que vayamos a utilizar para controlar el vehículo. Para esto será necesario contar con una instalación previa que permita la conexión de dos periféricos uno para la botonera y otro para el volante y la pedalera a través de conexiones USB.

Por último, colocar las pantallas en el orden correcto a los display para la formación de la imagen en 360º para una correcta experiencia de juego este orden será el representado en la

siguiente imagen. La conexión de las pantallas será a través del método que más convenga al usuario, pero desde desarrollo recomendamos la conexión directa mediante HDMI ya que creemos que es las más sencilla e intuitiva.

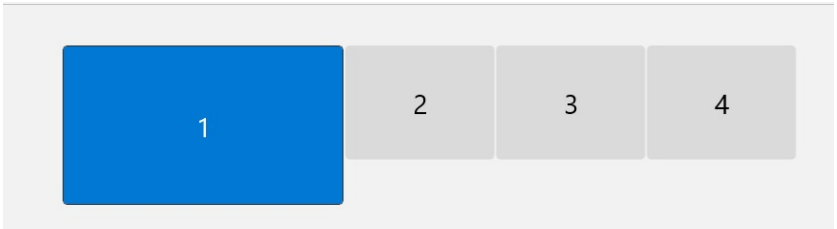

Figura 6.3 Ejemplo de colocación de Pantallas

7. Mejoras a futuro

A lo largo del desarrollo del proyecto, han surgido diversas ideas respecto a posibles mejoras, modificaciones o adiciones que podrían implementarse en el futuro. No obstante, debido a restricciones de tiempo o por estar fuera del alcance principal del proyecto, no se han podido llevar a cabo durante el desarrollo del proyecto.

Estas propuestas de mejoras podrían incluir la optimización de ciertas funcionalidades, la expansión de características para una experiencia más inversiva, o la incorporación de tecnologías que puedan beneficiar la aplicación.

Una clave del diseño de este proyecto es el diseñado como un sistema escalable, con la capacidad de ser ampliado con nuevas funcionalidades en el futuro de una forma sencilla. Aunque algunas ideas no se hayan materializado en esta versión, es importante tenerlas en cuenta, ya que podrían ser añadidas en futuras actualizaciones o versiones del simulador.

Esta flexibilidad junto a la escalabilidad, otro pilar base del proyecto, proporcionan una base sólida para el desarrollo continuo del simulador. La capacidad de incorporar nuevas características o mejoras a lo largo del tiempo garantiza que el proyecto pueda adaptarse a las necesidades continuamente en cambio de la sociedad actual. Este enfoque proactivo hace posible la evolución del simulador a medida que avanza en su ciclo de vida.

Una mejora futura de consideración podría ser la implementación de una funcionalidad de personalización de controles en tiempo real. Esta mejora proporcionaría a los jugadores la libertad de ajustar los controles según sus preferencias individuales, mejorando así su experiencia del simulador.

También, se ha planteado la incorporación de sonidos tanto propios de la excavadora como puede ser el sonido del motor y el claxon o incluso sonidos ambientales del entorno donde se desarrolla la experiencia del juego. A parte del sonido también se ha planteado la incorporación de animaciones para el volante o las orugas de la excavadora, pero a causa del apretado margen de tiempo y la complejidad que añadía al diseño se tuvo que desechar la idea.

Es fundamental destacar que estas mejoras representan solo algunas de las ideas que han surgido durante el desarrollo del simulador. A medida que el proyecto evolucione y se reciba el feedback de los usuarios, se podrán explorar más opciones y considerar otras mejoras adicionales para enriquecer aún más la experiencia de juego.

La escalabilidad con la que se ha enfocado el proyecto permitirá que estas ideas y mejoras puedan ser implementadas en futuros DLCs, asegurando que el simulador continúe evolucionando y que los jugadores experimenten cada vez una experiencia más realista e inversiva. La retroalimentación continua y la disposición para incorporar nuevas funcionalidades garantizan que el simulador se mantenga actualizado y atractivo a medida que las tecnologías cambien con el tiempo.

8. Conclusion

Durante la elaboración de este Trabajo de Fin de Grado, he tenido la privilegiada oportunidad de explorar en profundidad el cautivador universo de los videojuegos. Mi proyecto se ha enfocado en la creación de un simulador de una excavadora. En este contexto, he abordado la convergencia entre el mundo virtual y el mundo físico, buscando integrar de manera efectiva y envolvente elementos digitales con situaciones o entornos del mundo real.

A medida que avanza en el desarrollo del proyecto, he podido descubrir el abanico de posibilidades que nos presenta la combinación de Unity y C# para la creación de simuladores y videojuegos. He descubierto que Unity se trata de una herramienta increíble y con una gran comunidad, perfecta para la creación de entornos virtuales inversivos en un ámbito profesional o amateur.

Por otro lado, la elección de C# como lenguaje de programación fue perfecta para implementar la lógica y las funcionalidades esenciales del simulador. C# nos ha permitido tener un escenario de desarrollo donde organizar el código de una manera sencilla. Además, presenta una perfecta relación con Unity que nos ha permitido la implementación de características complejas, facilitando así la creación de un simulador con una experiencia de usuario envolvente.

Otra aprendizaje de este proyecto fue, la necesidad de priorizar al usuario en todo momento, pues se trata de la pieza final de nuestro desarrollo. Se priorizado al usuario tanto en las funcionalidades como en los aspectos visuales. También, he intentado proporcionar una accesibilidad sencilla y de fácil compresión. Esto se pretende conseguir gracias a la elección de mecánicas de diseño intuitivas.

El desarrollo de este Trabajo de Fin de Grado se a convertidos en una gran oportunidad para aplicar los conocimientos adquiridos a lo largo de mi carrera universitaria en un proyecto práctico y real. Los resultados me han dejado satisfecho.

En mi opinión, el proyecto ha sido una experiencia enriquecedora que me ha ayudado proporcionándome una primera experiencia para enfrentar futuros retos profesionales en el ámbito de la ingeniería de telecomunicaciones y especialmente el desarrollo de videojuegos, un

mundo que espero no dejar de lado ya que ha sido un gran descubrimiento para mí.

Aunque siempre existirá margen de mejora o aspectos adicionales que podrían mejorar el simulador, valoro mi aprendizaje continuo y los metas alcanzados durante su desarrollo. Este proyecto no solo me ha ayudado a consolidar mis habilidades técnicas, sino que también ha puesto a prueba mi capacidad para abordar problemas complejos y encontrar soluciones innovadoras, saliendo fortalecida tras el proyectó.

9. Bibliografía

- «Wikipedia- La enciclopedia viva» [En línea] (Accedido Enero 2024) https://es.wikipedia.org/wiki/Pala_excavadora

- «Imagen genérica de excavadora» [En línea] (Accedido Enero 2024) https://pbs.twimg.com/media/EMBfJizX0AE7DNY.jpg

- «Comunidad github- Pateman16» [En línea]. (Accedido Enero 2024) https://github.com/Pateman16/excavSimulator

- «Niantic- Pokémon Go» [En línea] (Accedido Enero 2024) https://pokemongolive.com/

- «Racing simulator- Carx Rally» [En línea] (Accedido Enero 2024) https://carx-online.com/

- «Giants Softwarw- Farming Simulator» [En línea] (Accedido Enero 2024) https://www.farming-simulator.com/

- «Micrisift - Microsoft Flight Simulator X:» [En línea] (Accedido Enero 2024) https://www.flightsimulator.com/

- «Medtronic – Touch Surgery» [En línea] (Accedido Enero 2024) https://www.medtronic.com/covidien/en-us/products/digital-surgery.html

- «PC Componente A»[En linea] (Accedido Enero 2024) https://www.pccomponentes.com/buscar/?query=ROG%20Strix%20G513IH%20&or-relevance

- «PC Componente B»[En linea] (Accedido Enero 2024): https://www.pccomponentes.com/tcl-50c649-50-qled-ultrahd-4k-hdr10

- «PC Componente C»[En linea] (Accedido Enero 2024) https://www.pccomponentes.com/logitech-g-saitek-paquete-de-equipo-pesado-volante-900-grados-pedales-panel-lateral.

- «Steam,» [En línea]. (Accedido Enero 2024) https://store.steampowered.com/?l=spanish

- «Unity Asset Store A,» [En línea]. (Accedido Enero 2024) https://assetstore.unity.com/

- «Unity Asset Store B» [En línea]. (Accedido Enero 2024) https://assetstore.unity.com/packages/3d/prototyping-pack-free-94277

- «Manual TextMeshPro». [En linea]. (Accedido Enero 2024) https://docs.unity3d.com/Manual/com.unity.textmeshpro.html

- «Manual InputManager». [En linea]. (Accedido Enero 2024) https://docs.unity3d.com/Manual/class-InputManager.html

- «Comunidad GitHub A». [En linea]. (Accedido Enero 2024) https://github.com/

- «Manual BuildSettings». [En linea]. (Accedido Enero 2024) https://docs.unity3d.com/Manual/BuildSettings.html

- «Comunidad GitHub B». [En linea]. (Accedido Enero 2024) https://github.com/matiassarzosa490/ManuInicioUnity/tree/main/Assets

- «Unity Asset Store C» [En línea]. (Accedido Enero 2024) https://assetstore.unity.com/packages/2d/textures-materials/floors/outdoor-ground-textures-12555

- «Unity Asset Store D» [En línea]. (Accedido Enero 2024) https://assetstore.unity.com/packages/2d/textures-materials/nature/terrain-textures-pack-free-139542

- «Unity Asset Store D» [En línea]. (Accedido Enero 2024) https://assetstore.unity.com/packages/2d/textures-materials/nature/grass-flowers-pack- free-138810

- «Manual Unity A». [En linea]. (Accedido Enero 2024) https://docs.unity3d.com/es/530/Manual/MultiDisplay.html

- «Manual Unity B». [En linea]. (Accedido Enero 2024) file:///C:/Program%20Files/Unity/Hub/Editor/2021.3.17f1/Editor/Data/Documentation/en/Manual/class-WheelCollider.html

- «Manual Unity C». [En linea]. (Accedido Enero 2024) file:///C:/Program%20Files/Unity/Hub/Editor/2021.3.17f1/Editor/Data/Documentation/en/Manual/class-HingeJoint.html

- «Manual Unity D». [En linea]. (Accedido Enero 2024) file:///C:/Program%20Files/Unity/Hub/Editor/2021.3.17f1/Editor/Data/Documentation/en/Manual/class-Rigidbody.html

- «Manual Unity E». [En linea]. (Accedido Enero 2024) file:///C:/Program%20Files/Unity/Hub/Editor/2021.3.17f1/Editor/Data/Documentation/en/Manual/class-BoxCollider.html

- «Manual Unity F». [En linea]. (Accedido Enero 2024) file:///C:/Program%20Files/Unity/Hub/Editor/2021.3.17f1/Editor/Data/Documentation/en/Manual/class-Material.html

- «Manual Unity G». [En linea]. (Accedido Enero 2024) file:///C:/Program%20Files/Unity/Hub/Editor/2021.3.17f1/Editor/Data/Documentation/en /Manual/script-Terrain.html

- «Canva». [En linea]. (Accedido Enero 2024) https://www.canva.com/templates/?query=diagramas-de-gantt