UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Montaje de Vehículos Arduino VR

Trabajo Fin de Grado

Grado en Tecnología Digital y Multimedia

AUTOR/A: Jiménez Lahoz, José Luis

Tutor/a: Cerdá Boluda, Joaquín

Cotutor/a: Rey Solaz, Beatriz

CURSO ACADÉMICO: 2023/2024

Resumen

Mi Trabajo de Fin de Grado se enfoca en el desarrollo de una aplicación de Realidad Virtual (VR) para optimizar la capacitación en el montaje de vehículos dentro de la industria automotriz. El objetivo principal del proyecto es proporcionar una herramienta inmersiva que permita a los empleados aprender y practicar los procedimientos de ensamblaje en un entorno seguro y sin los costos asociados al uso de equipos reales. La aplicación está diseñada para dispositivos como Meta Quest 3 y Apple Vision Pro, integrando tanto controladores como seguimiento de manos para interactuar con el entorno virtual. A través de la creación de un sistema modular, la aplicación facilita la adaptación a diferentes modelos de vehículos y futuros dispositivos de realidad virtual. También se implementan modos de práctica y examen, ofreciendo retroalimentación en tiempo real para mejorar la precisión y eficiencia del entrenamiento. En el futuro, se prevé la inclusión de nuevas funcionalidades y modelos de vehículos.

Resum

El meu Treball de Fi de Grau se centra en el desenvolupament d'una aplicació de Realitat Virtual (VR) per optimitzar la formació en el muntatge de vehicles dins de la indústria de l'automòbil. L'objectiu principal del projecte és proporcionar una eina immersiva que permeti als empleats aprendre i practicar els procediments d'assemblatge en un entorn segur i sense els costos associats a l'ús d'equips reals. L'aplicació està dissenyada per a dispositius com Meta Quest 3 i Apple Vision Pro, integrant tant controladors com seguiment de mans per interactuar amb l'entorn virtual. Mitjançant la creació d'un sistema modular, l'aplicació facilita l'adaptació a diferents models de vehicles i futurs dispositius de realitat virtual. També s'implementen modes de pràctica i examen, oferint retroalimentació en temps real per millorar la precisió i l'eficiència de la formació. En el futur, es preveu la inclusió de noves funcionalitats i models de vehicles.

Abstract

My Final Degree Project focuses on the development of a Virtual Reality (VR) application to optimize training in vehicle assembly within the automotive industry. The main objective of the project is to provide an immersive tool that allows employees to learn and practice assembly procedures in a safe environment, without the costs associated with using real equipment. The application is designed for devices such as Meta Quest 3 and Apple Vision Pro, integrating both controllers and hand tracking for interaction with the virtual environment. Through the creation of a modular system, the application facilitates adaptation to different vehicle models and future virtual reality devices. Practice and exam modes are also implemented, offering real-time feedback to improve training accuracy and efficiency. In the future, the inclusion of new functionalities and vehicle models is anticipated.

RESUMEN EJECUTIVO

La memoria del TFG del Grado en Tecnología Digital y Multimedia debe desarrollar en el texto los siguientes conceptos, debidamente justificados y discutidos, centrados en el ámbito de las tecnologías digitales y multimedia.

CONCEPT (ABET)	**CONCEPTO (traducción)**	**¿Cumple? (S/N)**	**¿Dónde? (páginas)**
1. IDENTIFY:	**1. IDENTIFICAR:**		
1.1. Problem statement and opportunity	1.1. Planteamiento del problema y oportunidad	**S**	5 y 9
1.2. Constraints (standards, codes, needs, requirements & specifications)	1.2. Toma en consideración de los condicionantes (normas técnicas y regulación, necesidades, requisitos y especificaciones)	**S**	13 - 19
1.3. Setting of goals	1.3. Establecimiento de objetivos	**S**	10
2. FORMULATE:	**2. FORMULAR:**		
2.1. Creative solution generation (analysis)	2.1. Generación de soluciones creativas (análisis)	**S**	27 - 41
2.2. Evaluation of multiple solutions and decision-making (synthesis)	2.2. Evaluación de múltiples soluciones y toma de decisiones (síntesis)	**S**	49 - 51
3. SOLVE:	**3. RESOLVER:**		
3.1. Fulfilment of goals	3.1. Evaluación del cumplimiento de objetivos	**S**	54
3.2. Overall impact and significance (contributions and practical recommendations)	3.2. Evaluación del impacto global y alcance (contribuciones y recomendaciones prácticas)	**S**	50 - 51 y 54 - 55

Índice

Capítulo 1. Introducción

Durante mis prácticas en la empresa Innoarea, he participado en varios proyectos que han ampliado mi comprensión en áreas clave para el desarrollo de la aplicación de montaje de vehículos Arduino VR. En particular, esta aplicación forma parte de un proyecto mayor de la empresa llamado *Roomote*, que aún no ha sido lanzado en la tienda de Quest, por lo que solo puedo centrarme en mi aportación específica. Dentro de *Roomote*, existe una sección dedicada a entrenamientos virtuales, donde se encuentra este proyecto de montaje de vehículos en VR.

La aplicación simula un entorno virtual en el que el usuario entra a una sala y se encuentra con una mesa que contiene las 18 piezas del coche. Además, hay un canvas flotante que muestra las instrucciones paso a paso para completar el montaje, organizadas en un total de 9 pasos.

El proceso de entrenamiento está dividido en dos modos: el modo práctica y el modo examen. En el modo práctica, se asiste al usuario especificando dónde se debe colocar cada pieza, mientras que en el modo examen se evalúan los errores. En ambos modos, solo se puede avanzar si el paso se realiza correctamente, lo cual se controla mediante un botón interactivo.

La aplicación también cuenta con una funcionalidad multijugador, lo que permite a varios usuarios colaborar en tiempo real dentro del mismo entorno virtual, añadiendo una dimensión social y colaborativa al proceso de aprendizaje. Esta versión está diseñada para ser compatible con los dispositivos Quest 2 y Quest 3, aprovechando su capacidad para experiencias inmersivas de alta calidad.

Adicionalmente, he desarrollado una versión compatible con las Apple Vision Pro, adaptando la aplicación a las características avanzadas de este dispositivo, como su enfoque en la realidad mixta y su capacidad de interacción precisa.

Esta aplicación busca proporcionar una experiencia inmersiva y colaborativa que mejore el aprendizaje y la retención de conocimientos en entornos de formación técnica, combinando la precisión de la programación en Unity con las ventajas educativas de la realidad virtual.

1.1. Experiencia Previa y Preparación

A lo largo de las prácticas, he trabajado en proyectos referentes a la programación en entornos de Unity, la integración de tecnologías de realidad virtual y la gestión de proyectos de software. Estos proyectos son los siguientes:

1.1.1. Catálogo VR

El primer proyecto realizado durante mis prácticas fue el desarrollo de una aplicación de realidad virtual para dispositivos Quest con el objetivo principal de familiarizarme con las interacciones y herramientas ofrecidas por el paquete Oculus Integration. Este proyecto tenía un propósito formativo, enfocándose en explorar todas las funcionalidades posibles dentro del entorno de desarrollo VR.

La aplicación consiste en un catálogo interactivo, donde el usuario puede invocar una tablet virtual al presionar un botón en un reloj. Esta tablet permite generar objetos de un catálogo, los cuales pueden ser trasladados y escalados mediante el Grab Interaction de Oculus Integration. Además, se incluyen herramientas avanzadas para la manipulación de estos objetos, como herramientas de dibujo, medición y seccionado.

Para complementar la formación y facilitar el aprendizaje de las funcionalidades del catálogo, desarrollé un entrenamiento interactivo dividido en 7 pasos. En este entrenamiento, el usuario debía completar tareas como hacer aparecer un objeto específico, colocarlo en una posición determinada, medir sus dimensiones y dibujar las medidas sobre el objeto. Este proceso guiado ayudó a crear una experiencia de aprendizaje estructurada y dinámica, permitiendo al usuario dominar las herramientas y operaciones del catálogo de manera progresiva. Además de tener un control de errores y de puntuación.

1.1.2. La Valentina AR

El segundo proyecto lo realicé para un cliente y consistió en el desarrollo de una web interactiva que permitía a los usuarios explorar diferentes proyectos urbanísticos de Valencia en 3D. La aplicación ofrecía tres opciones, cada una representando un diseño urbanístico de un barrio de Valencia. El objetivo era proporcionar una herramienta accesible para visualizar y comprender mejor los planes de desarrollo.

Una característica destacada de la web era la posibilidad de visualizar los proyectos en realidad aumentada (AR), lo que permitía a los usuarios proyectar los modelos 3D en su entorno real a través de sus dispositivos móviles Android e IOs. Esta funcionalidad hacía la experiencia más inmersiva y proporcionaba una visión más tangible de cómo se integrarían los proyectos en su contexto urbano.

Este proyecto requirió la integración de modelos 3D en un entorno web y la implementación de tecnologías AR, lo que supuso un reto técnico al trabajar con diferentes plataformas y formatos.

1.1.3. RECIA, Diseño de Recipientes Cerámicos mediante IA

El tercer proyecto, llamado *RECIA*, fue el más desafiante de mi período de prácticas. Este proyecto consistió en el desarrollo de una aplicación para PC en Unity destinada a mostrar en pantalla el perfil optimizado de un tarro cerámico, basado en los parámetros configurados por el usuario. La optimización del diseño se realiza mediante inteligencia artificial, integrando un proceso desarrollado en Python.

Dado que este proyecto está sujeto a un contrato de confidencialidad, no puedo revelar detalles específicos de la interfaz. Sin embargo, los clientes proporcionaron el código fuente del algoritmo de optimización en Python. La principal dificultad radicó en lograr una comunicación eficiente entre Unity y el servidor Python para gestionar los datos de entrada y salida, asegurando que la aplicación pudiera renderizar en tiempo real los resultados optimizados. Además, implementé una funcionalidad para guardar tanto el perfil generado como los parámetros de optimización en ficheros CSV. Estos archivos pueden cargarse posteriormente en la aplicación para visualizar los diseños guardados, facilitando la reutilización de configuraciones previas.

1.1.4. Cupa Pizarra

El cuarto proyecto lo desarrollé para un cliente y consistió en la creación de un tour virtual en 360 grados del interior de las minas de pizarra de *Cupa Pizarra* para dispositivos Quest pro. La aplicación permitía a los usuarios explorar diferentes áreas de las minas a través de videos inmersivos en 360 grados, proporcionando una experiencia realista y detallada del entorno.

El desarrollo presentó varios desafíos, especialmente relacionados con el tratamiento de los videos. Dado el gran tamaño de los archivos y la naturaleza oscura de las grabaciones en el interior de las minas, las Oculus Pro mostraban ciertas limitaciones en cuanto a la reproducción y calidad visual. Para optimizar los vídeos sin perder detalles importantes, recurrí a la herramienta FFmpeg, lo que me permitió ajustar los niveles de brillo y compresión, mejorando así la calidad visual y la fluidez de la experiencia.

1.1.5. VR Connect

El quinto proyecto no estuvo destinado a un cliente, sino que fue un desarrollo experimental enfocado en la creación de una aplicación de sincronización de videos 360 para Oculus Go, de hasta 9 dispositivos conectados simultáneamente, todas controladas por un administrador desde un dispositivo móvil o PC. El administrador tiene la capacidad

de seleccionar qué video 360 se reproduce en los dispositivos, así como ajustar parámetros como el volumen, la velocidad de reproducción, pausar y adelantar el video.

Este proyecto me sirvió como una valiosa experiencia para profundizar en la implementación de funcionalidades multijugador utilizando Photon. Sin embargo, me encontré con dificultades técnicas debido a la falta de soporte para Oculus Go desde 2021. La aplicación tuvo que ser desarrollada con Photon PUN v1, una versión bastante desactualizada y con recursos limitados, lo que añadió complejidad al proceso. A pesar de estas limitaciones, logré implementar un sistema funcional que cumplía con los requisitos de sincronización y control de contenido en múltiples dispositivos.

1.1.6. Silman Caídas

El sexto proyecto, realizado para un cliente, consistió en el desarrollo de una aplicación de realidad virtual destinada a concienciar sobre la peligrosidad de los trabajos en alturas. La aplicación simula un escenario en el que el usuario debe realizar tareas de construcción en el tejado de un edificio. El usuario comienza ascendiendo por unos andamios y debe colocar varias tablas utilizando un atornillador.

Una característica clave de la simulación es que, al colocar la tercera tabla, el techo se rompe y el usuario cae al vacío. La experiencia educativa se centra en la seguridad, ya que si el usuario está equipado con un arnés de seguridad, se salvará de una caída mortal. En caso contrario, la caída es de 20 metros, subrayando los riesgos de trabajar sin protección adecuada.

Para el desarrollo de esta aplicación, utilicé la plantilla de Innoarea, lo que facilitó el proceso al evitar la necesidad de configurar manualmente las interacciones de Oculus Integration. Esto permitió centrarse en la implementación de la simulación y en la experiencia del usuario, logrando una herramienta efectiva para la formación en seguridad laboral.

La combinación de estas experiencias ha sentado las bases para abordar con éxito el desarrollo de la aplicación de montaje de vehículos Arduino VR, proporcionando un conjunto diverso de habilidades técnicas y de gestión necesarias para llevar a cabo este proyecto de manera efectiva.

1.2. Contexto y Justificación

La industria automotriz actual enfrenta desafíos críticos relacionados con la eficiencia, la seguridad y la formación del personal en un entorno que evoluciona rápidamente. La producción de vehículos ha incrementado su complejidad debido a la integración de nuevas tecnologías y a la demanda de procesos cada vez más precisos y rápidos. En este contexto, la capacitación de los empleados debe adaptarse a la necesidad de adquirir habilidades técnicas especializadas de manera rápida, segura y efectiva.

Los métodos tradicionales de formación en el montaje de vehículos suelen ser costosos y conllevan riesgos físicos, tanto para el personal como para los equipos. La dependencia de componentes reales y la posible interrupción de las líneas de producción representan un problema considerable en términos de costos y eficiencia. Además, estos enfoques convencionales no siempre logran simular adecuadamente situaciones críticas o complejas, limitando la efectividad de la formación.

En respuesta a estos retos, la incorporación de tecnologías de Realidad Virtual (VR) ofrece una solución innovadora. La VR no solo elimina los riesgos asociados con el manejo de equipos y piezas físicas, sino que también permite recrear escenarios realistas y personalizables en un entorno seguro. Esta tecnología mejora la retención de conocimientos al proporcionar experiencias inmersivas, interactivas y repetitivas que permiten a los empleados practicar hasta dominar los procesos.

El proyecto se centra en el desarrollo de una aplicación VR para dispositivos Meta Quest 2 y 3, orientada a optimizar el entrenamiento en el montaje de vehículos. La aplicación utiliza herramientas como Unity, Oculus Integration [1] y Photon Fusion [2] para crear un entorno educativo que promueva la colaboración y la práctica segura en situaciones simuladas. Asimismo, se ha desarrollado una versión compatible con Apple Vision Pro, diseñada para aprovechar las capacidades avanzadas de este dispositivo y proporcionar una experiencia de formación más precisa y envolvente.

Adoptar estas tecnologías no solo permite a las empresas del sector automotriz mejorar la formación técnica de sus empleados, sino que también ofrece una solución eficiente y escalable para enfrentar los desafíos actuales en términos de seguridad, costos y adaptación a nuevas tecnologías.

1.2.1. Unity Engine [3]

Unity es uno de los motores de desarrollo más utilizados para crear experiencias interactivas en 2D, 3D, VR y AR. Su popularidad radica en su accesibilidad y flexibilidad, lo que permite a los desarrolladores tanto principiantes como avanzados crear desde pequeños proyectos hasta aplicaciones complejas y videojuegos. Unity ofrece una interfaz visual donde los desarrolladores pueden diseñar entornos, personajes, animaciones y otras dinámicas sin necesidad de ser expertos en programación. Además, es compatible con múltiples plataformas, incluyendo PC, consolas, móviles y dispositivos de realidad virtual y aumentada, lo que facilita el desarrollo multiplataforma. También permite la integración de diversas bibliotecas y plugins, como el sistema de física, inteligencia artificial, gráficos avanzados y motores de audio, lo que convierte a Unity en una herramienta integral para el desarrollo de proyectos de simulación y entretenimiento interactivo. Es ampliamente utilizado en campos como el desarrollo de videojuegos, simulaciones industriales, educación y aplicaciones de AR/VR.

1.2.2. Realidad Virtual (VR) [4]

Tecnología que utiliza dispositivos especializados, como cascos o gafas VR, para transportar al usuario a un entorno completamente generado por computadora. Estos entornos pueden ser simulaciones de la vida real o mundos totalmente ficticios. En la VR, el usuario puede interactuar con objetos y elementos dentro de este entorno virtual a través de controladores o mediante tecnologías avanzadas como el seguimiento de manos. El propósito principal de la VR es proporcionar una experiencia inmersiva, lo que significa que el usuario se siente completamente "dentro" de ese entorno, aislado del mundo físico que lo rodea. Esta tecnología se utiliza en diversas áreas, como videojuegos, educación, formación profesional, medicina y simulación de situaciones peligrosas o costosas de replicar en la vida real. Por ejemplo, en la industria automotriz, la VR puede usarse para entrenar a los empleados en procesos de ensamblaje sin necesidad de manipular piezas reales.

1.2.3. Realidad Aumentada (AR) [5]

A diferencia de la VR, la Realidad Aumentada no reemplaza el entorno físico, sino que lo mejora añadiendo capas digitales de información sobre lo que el usuario ve. Esto se logra mediante el uso de dispositivos como smartphones, tabletas o gafas de AR que combinan la cámara del mundo real con gráficos generados por computadora. Los ejemplos más comunes de AR incluyen aplicaciones de navegación que muestran indicaciones en tiempo real o juegos como Pokémon Go, donde los personajes virtuales aparecen en el mundo real a través de la cámara del dispositivo. En la industria, la AR se utiliza para guiar a los trabajadores en la realización de tareas complejas, mostrando instrucciones visuales o señalando errores en tiempo real sin interrumpir la actividad física. Esto permite una integración perfecta entre la realidad física y la información digital para mejorar la eficiencia en áreas como la capacitación, la manufactura y el mantenimiento.

1.3. Objetivos del Proyecto

El objetivo principal de este Trabajo de Fin de Grado es desarrollar una aplicación de Realidad Virtual que permita a los usuarios realizar un entrenamiento en el montaje de vehículos de manera interactiva y efectiva. Los objetivos específicos incluyen:

- Diseñar e implementar una aplicación VR utilizando Unity, compatible con Meta Quest 2 y 3.

- Integrar Oculus Integration y Photon Fusion para proporcionar una experiencia de usuario inmersiva y multijugador.

- Desarrollar modos de práctica y examen, incluyendo un sistema de control de errores que facilite el aprendizaje y permita a los usuarios evaluar su progreso de manera efectiva.

- Estructurar de manera modular los scripts, permitiendo la futura inclusión de diferentes modelos de automóviles y la adaptabilidad a otros dispositivos VR como Apple Vision Pro.

- Adquirir habilidades técnicas avanzadas, mediante la aplicación práctica de conocimientos en desarrollo de entornos VR, programación en Unity, gestión de proyectos y adaptación a nuevas tecnologías, consolidando mi formación como desarrollador.

- Mejorar en la capacidad de gestión de proyectos y resolución de problemas, al enfrentar desafíos reales durante el desarrollo, integrando distintas tecnologías y adaptándome a requisitos específicos.

- Fomentar la documentación y transferencia de conocimientos, asegurando que las lecciones aprendidas durante el desarrollo puedan aplicarse en futuros proyectos y compartirse dentro del equipo.

Capítulo 2. Diseño y desarrollo de la aplicación

2.1. Planificación y Metodología

Durante el desarrollo del proyecto, trabajé alrededor de 6 horas diarias, y todas las horas quedaron registradas en Clockify. Desde mediados de marzo, acumulé un total de 150 horas, también reflejadas en la plataforma *Roomote*, donde se puede ver el tiempo dedicado al proyecto. Parte de esas horas, específicamente las fichadas en “demo”, corresponden al trabajo realizado en la versión para Apple Vision Pro.

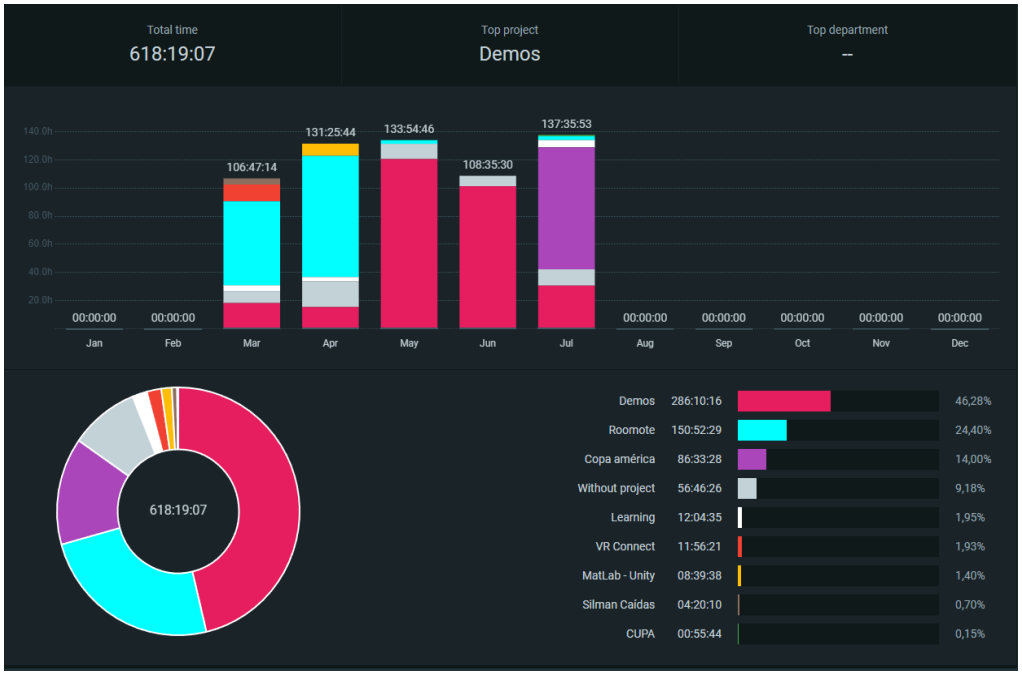


Figura 1: Gráfico de Horas Registradas en Clockify por Proyecto desde Marzo.

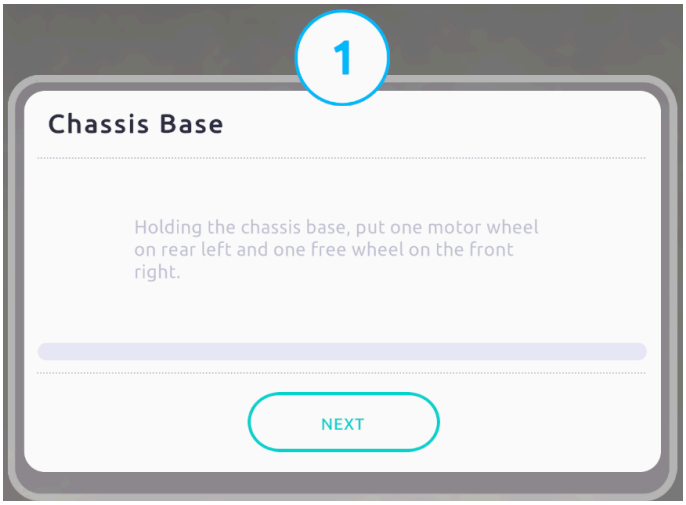
Para gestionar el desarrollo y coordinar los cambios entre el equipo, usamos Git Extensions como herramienta de control de versiones. Éramos un equipo de 6 personas, y cada cambio en el proyecto se registraba con commits. Esto nos permitía hacer un seguimiento detallado de las modificaciones y colaborar eficientemente. Dado que el proyecto no se limitaba solo a mi entrenamiento, cada vez que alguien hacía un cambio, era necesario hacer un push o pull para mantener el código actualizado según si el cambio lo hacía uno mismo o algún compañero.
Cabe destacar que tuve cierta libertad con respecto a la duración del desarrollo. No se me impuso un deadline estricto; más bien, me dieron una estimación de alrededor de 150 horas para completar todo el proyecto, lo que me permitió trabajar con flexibilidad y ajustar el ritmo según fuera necesario.

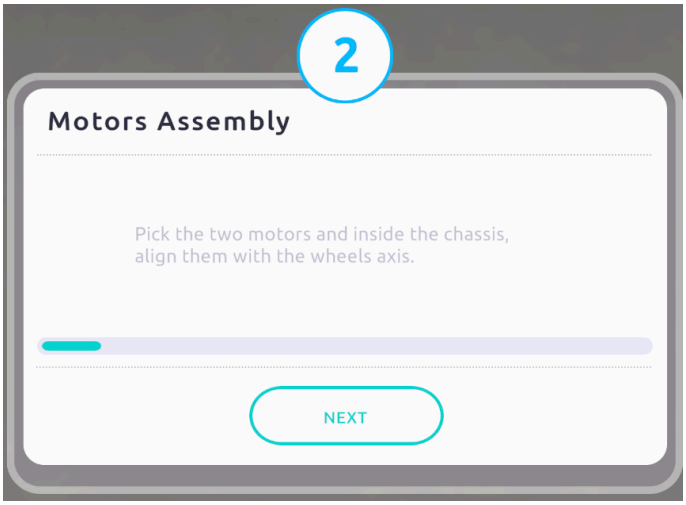
2.2. Diseño de la aplicación

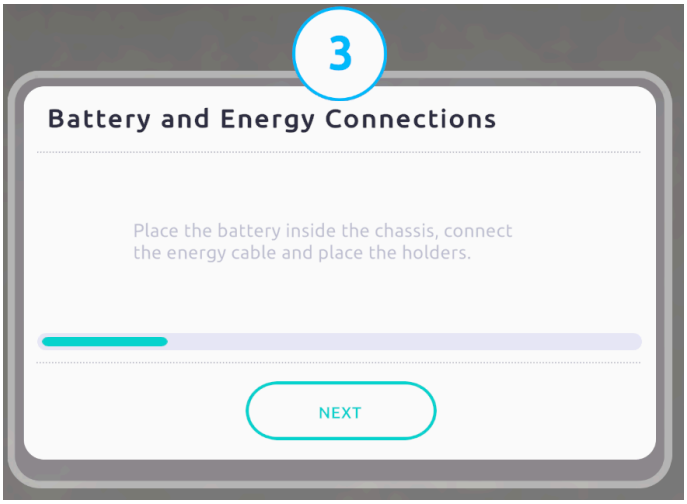
2.2.1. Interfaz de Usuario (UI)

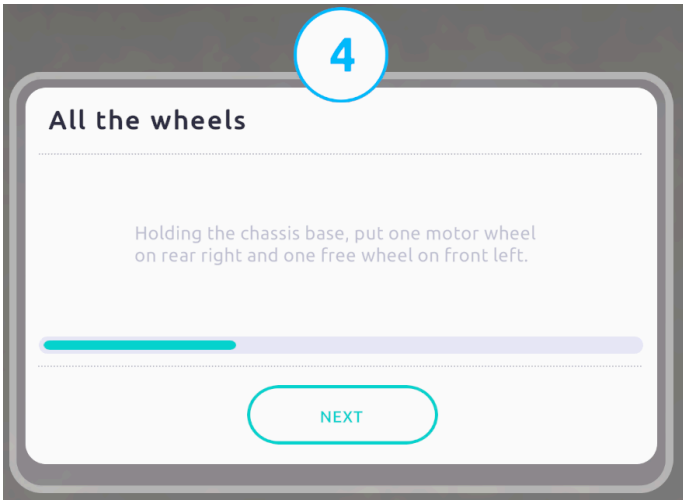
El diseño de la interfaz de usuario (UI) fue realizado por un compañero de la oficina, quien se encargó de crear los elementos visuales y la disposición de los menús, botones y demás componentes gráficos. Mi responsabilidad fue implementar este diseño en Unity, asegurando que cada elemento se comportara según lo planeado y se integrara correctamente en la aplicación. La estética es minimalista y centrada en la funcionalidad, facilitando la navegación y el acceso a las instrucciones.

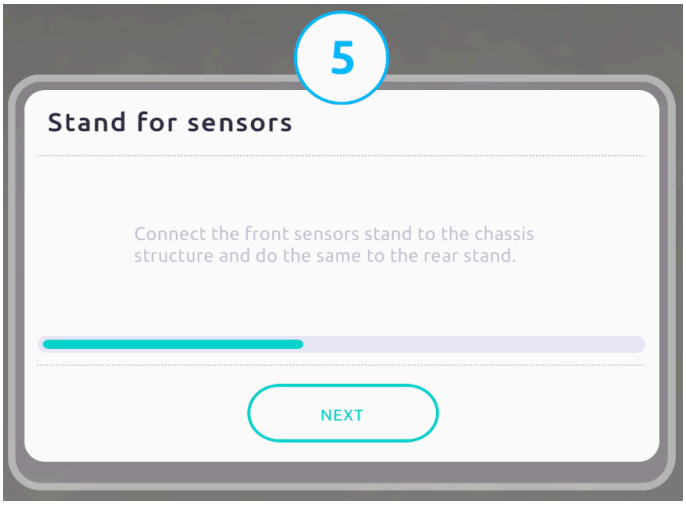
A continuación, presento de manera detallada el flujo del entrenamiento, incluyendo los títulos, textos y el diseño de cada paso:

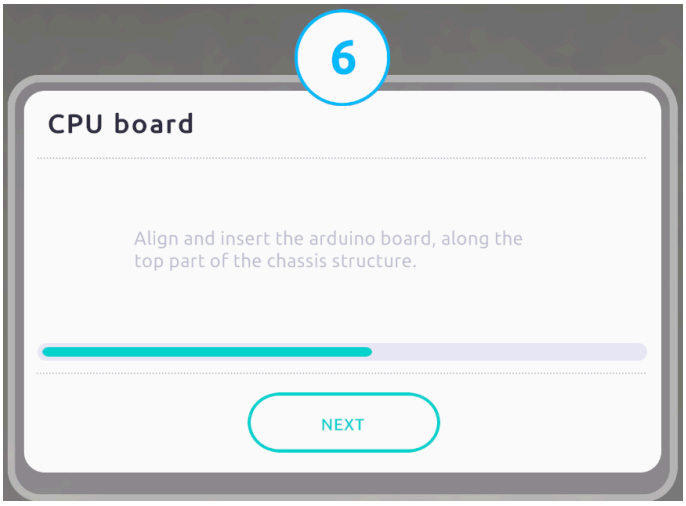


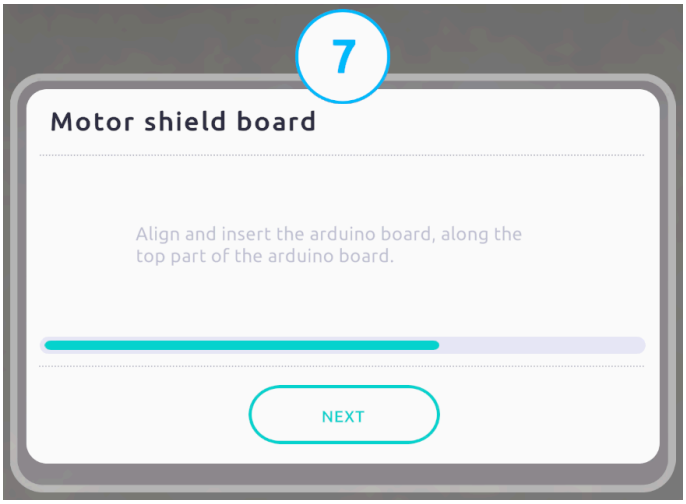


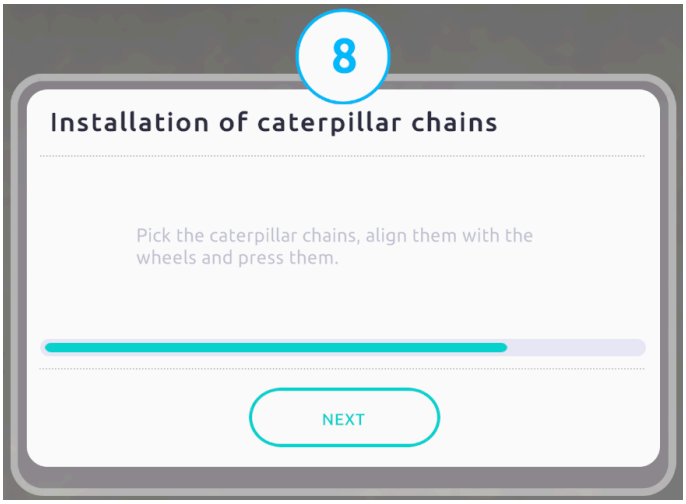


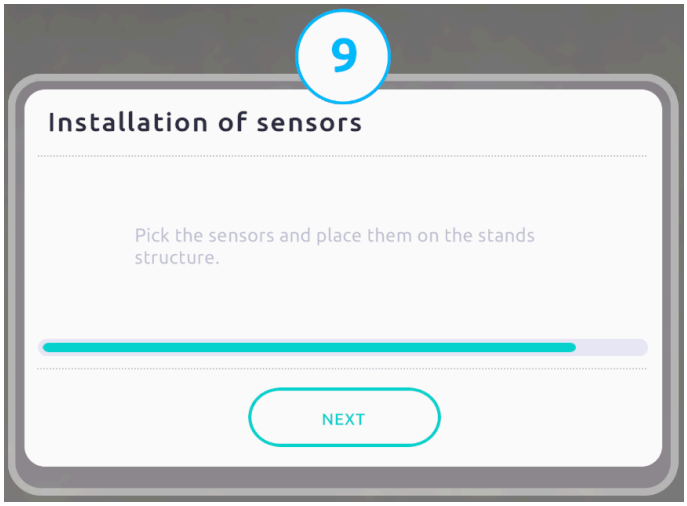






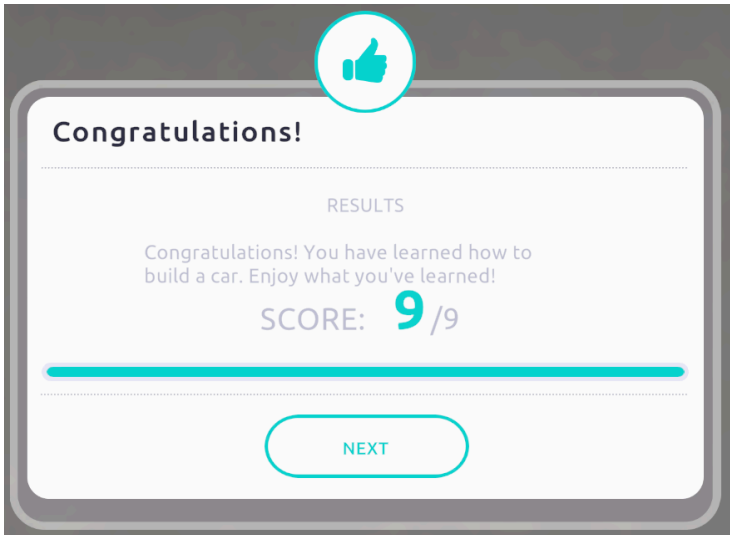






Figuras 2, 3, 4, 5, 6, 7, 8, 9 y 10: Pasos del Entrenamiento en Orden de Ejecución.

También muestro el diseño y los textos de la pantalla final, según si el entrenamiento se aprueba o se suspende. Además, incluyo el texto, título y diseño del canvas que aparece cuando se produce un error durante el proceso.



Figuras 11:
Diseño UI si el usuario ha sacado buena puntuación.

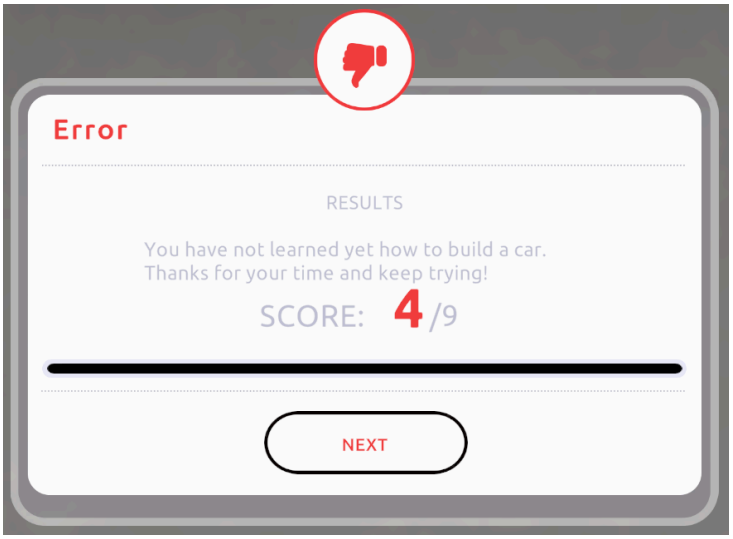


Figura 12:
Diseño UI si el usuario ha sacado mala puntuación.

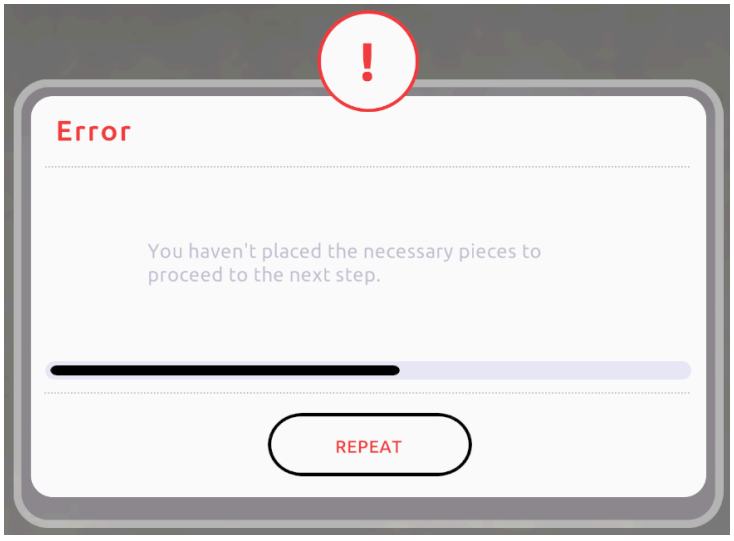


Figura 13: Diseño UI si el usuario se ha equivocado durante el entrenamiento.

2.2.2. Experiencia de Usuario (UX)

La experiencia de usuario (UX) está diseñada con un enfoque cómodo y de fácil uso, aprovechando tanto los mandos de las Meta Quest como la tecnología de hand tracking. Algunas características clave incluyen:

- Interacción con objetos (Grab Interaction): Los usuarios pueden agarrar las piezas del coche utilizando tanto los mandos como el hand tracking. Esta funcionalidad asegura una colocación intuitiva y exacta.

  Cada objeto interactivo en la aplicación tiene tres elementos hijos que permiten esta interacción. En las siguientes imágenes, se muestran los componentes hijos de un objeto agarrable, como es la placa Arduino.

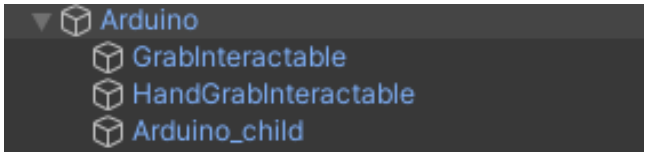


Figura 14: GameObject *Arduino*.

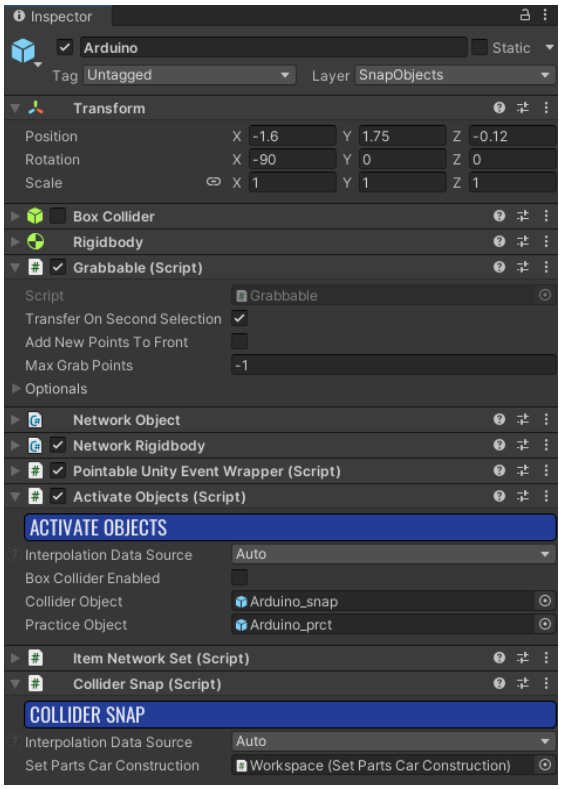


Figura 16: Componentes del GameObjects *Arduino*.

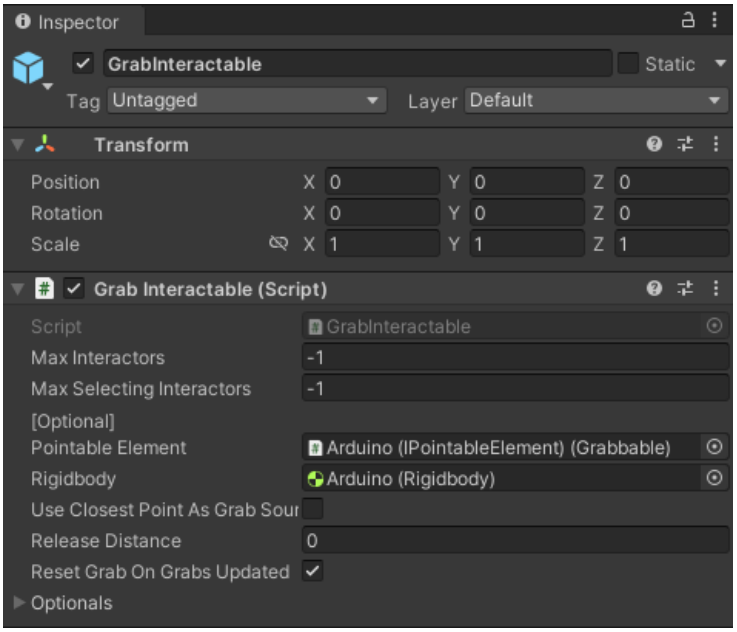
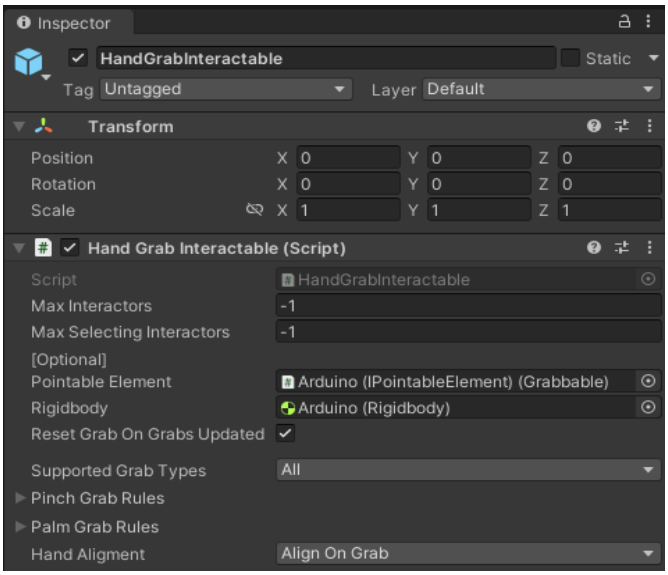


Figura 15: Componente *Grab Interactable*.



Figuras 17: Componente *Hand Grab Interactable.*

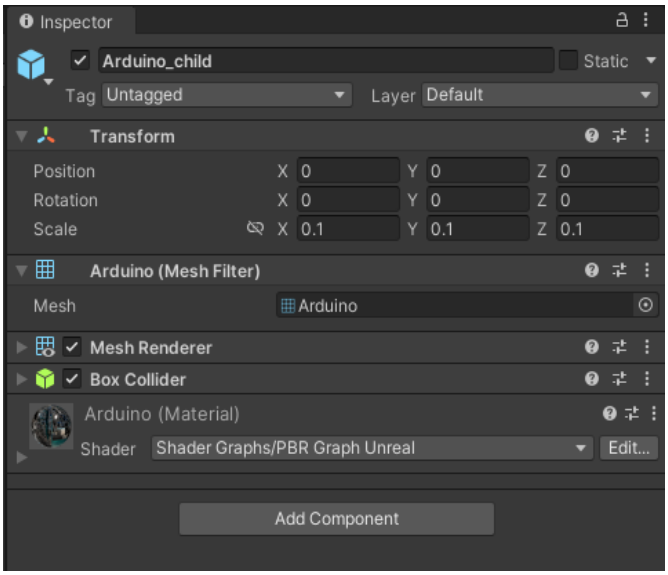


Figura 18: Componentes del GameObject *Arduino_child.*

- Movimiento en la Sala Virtual (Locomotion Interaction): El desplazamiento en el entorno virtual se realiza mediante teletransporte, ofreciendo una navegación rápida y cómoda que reduce la fatiga visual y los mareos asociados a otros métodos de locomoción. Este enfoque garantiza una experiencia fluida y agradable en el entorno VR, priorizando el confort del usuario. Para este proyecto, utilicé la plantilla de Innoarea, que ya incluye interacciones preconfiguradas y una configuración multijugador prediseñada, lo que facilitó el desarrollo. Además, hice uso de Oculus Integration para integrar las funcionalidades específicas de realidad virtual, como el control de mandos y el hand tracking, y de Photon Fusion para gestionar la sincronización y la interacción en tiempo real entre los usuarios en el entorno multijugador. Esto permitió una implementación eficiente y optimizada de las principales características de la aplicación.

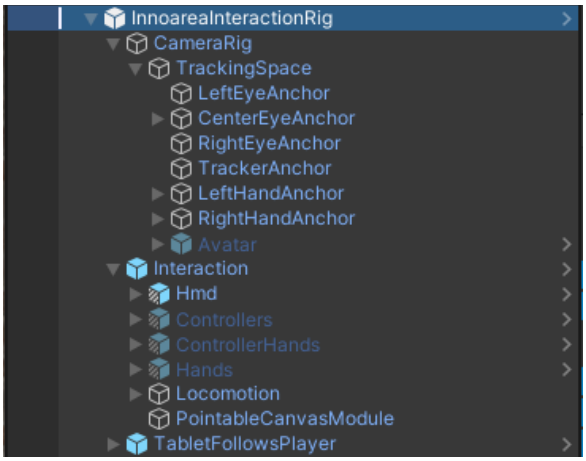
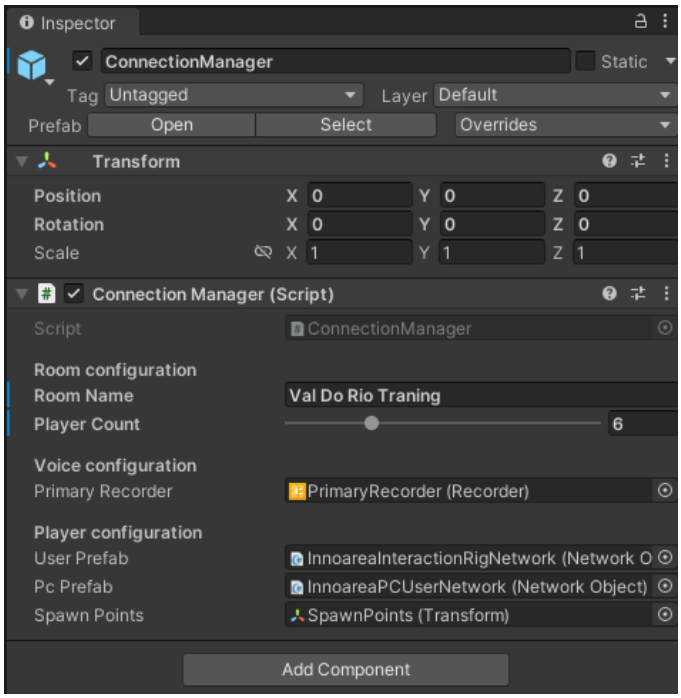


Figura 19: GameObject *InnoareaInteractionRig*.



Figuras 20: Componente *Connection Manager*.

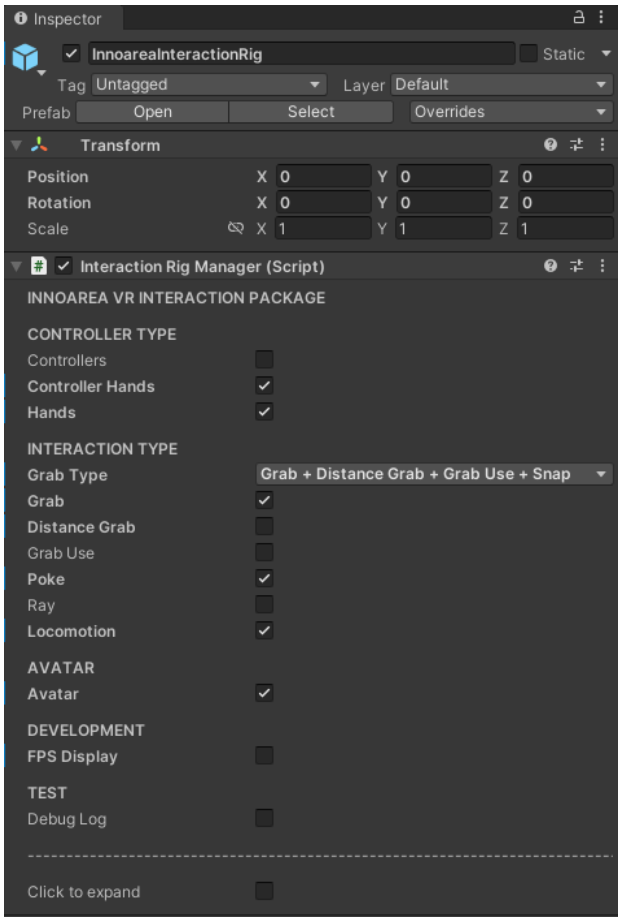


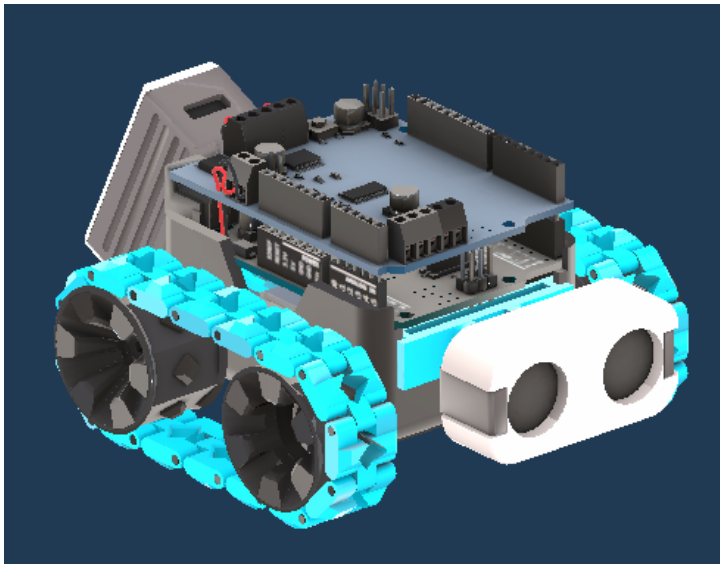
Figura 21: Componente *Interaction Rig Manager*.

- Progresión de Pasos (Poke Interaction): Para avanzar en el entrenamiento, el usuario debe presionar un botón virtual con el dedo índice, activando así el siguiente paso del montaje. Este mecanismo se repite en cada fase del proceso, guiando al usuario de manera clara y estructurada a través de los 9 pasos del entrenamiento.

2.2.3. Modelos 3D

Los modelos 3D utilizados en la aplicación fueron desarrollados por un compañero de la oficina, especializado en modelado y diseño. Estos modelos incluyen todas las piezas del coche que se utilizan durante el entrenamiento, cuidadosamente detalladas para garantizar una experiencia realista en el entorno de realidad virtual, además de la mesa de montaje donde se organizan las piezas. Mi responsabilidad fue integrarlos en el proyecto, asegurando que cada modelo estuviera correctamente configurado y listo para la interacción en Unity.

A continuación, os muestro imágenes de los modelos 3D utilizados en la aplicación:



Figuras 22: Vehículo Arduino montado.

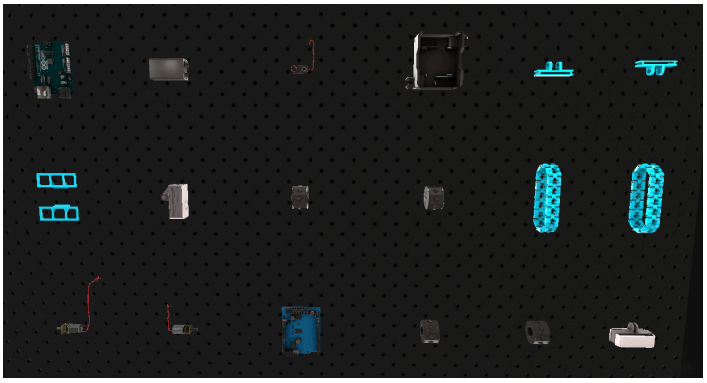


Figura 23: Piezas distribuidas para su montaje.

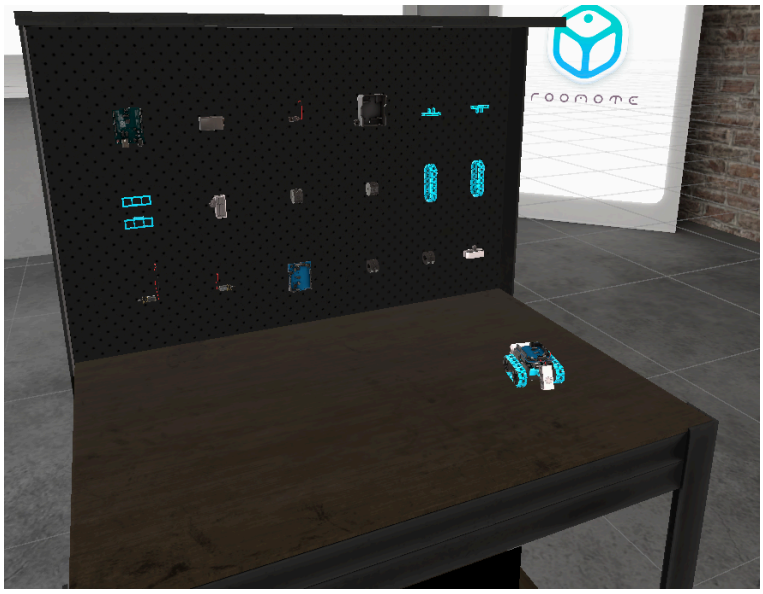


Figura 24: Mesa de trabajo con las figuras 22 y 23.

2.3. Configuración del Entorno de Desarrollo

En este punto, detallo cómo se configura el entorno de desarrollo en Unity para el proyecto. Muestro cómo se organizan los objetos en la escena y se asignan los componentes necesarios utilizando el editor de Unity.

Las siguientes imágenes ofrecen una visión clara de cómo se realiza esta configuración en el editor para asegurar el funcionamiento óptimo de la aplicación.

Figura 25: Estructura completa de la escena en Unity del entrenamiento.

La imagen muestra la jerarquía completa del proyecto en Unity. En la parte superior, se encuentran los objetos de Environment, que representan toda la sala del entorno virtual. A continuación, el *InteractionRig* (ver figura 21) incluye el personaje o avatar, junto con la cámara y las manos, así como todos los elementos necesarios para las interacciones en el entorno.

Otro componente clave es el *ConnectionManager* (ver figura 20), que se encarga de gestionar el funcionamiento del multijugador, asegurando que los usuarios puedan interactuar en tiempo real. Finalmente, el objeto *Training Val Do Rio* es crucial, ya que de este objeto heredan todos los demás elementos necesarios para el entrenamiento. Cabe destacar que "Val Do Rio" es el nombre clave que utilizamos en la oficina para referirnos a este entrenamiento específico.

Figura 26: GameObject *Training Val Do Rio*.

La imagen muestra la jerarquía del objeto *Training Val Do Rio* y sus objetos hijos. Este objeto se organiza en dos secciones principales:

- Workspace: Esta sección incluye los elementos interactivos del entorno de entrenamiento, como las piezas del coche que se pueden agarrar. Aquí también se encuentra el objeto *SMARS*, que sirve como modelo de coche de ejemplo, y la *MesaDeTrabajo*, donde se colocan todas las piezas necesarias para el montaje.
- Activities: Bajo este objeto se encuentra el Canvas, que gestiona los textos, imágenes y botones relacionados con cada paso del entrenamiento.

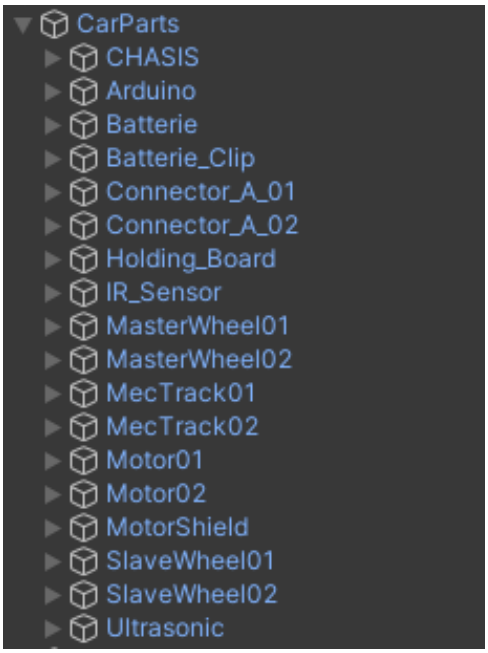
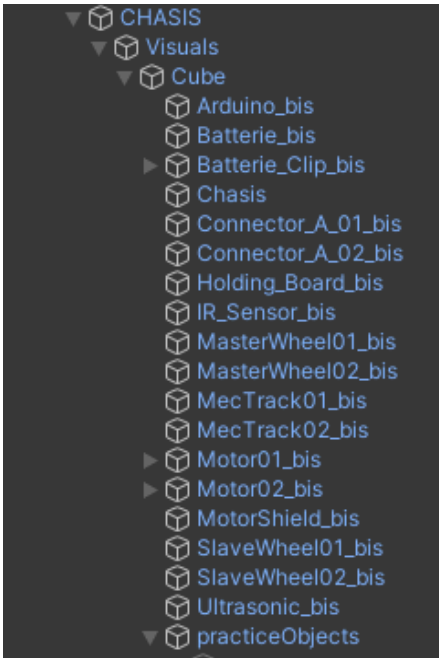
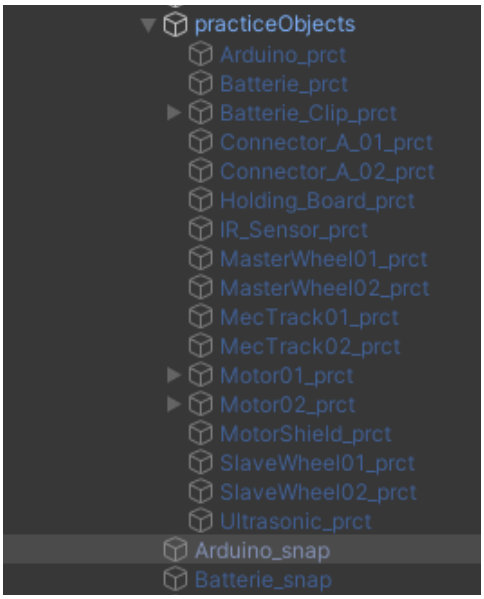
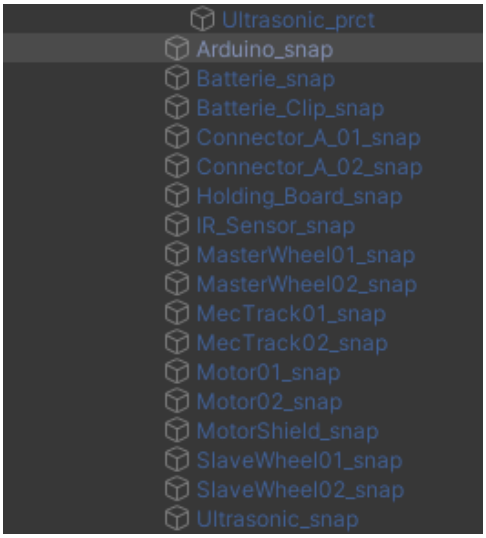


Figura 27: GameObject *CarParts*.

La imagen muestra la jerarquía dentro del objeto CarParts, que incluye las 24 piezas agarrables del entrenamiento. Un aspecto destacado es que el objeto CHASIS actúa como la base del coche, siendo el elemento del que heredan todas las piezas del coche final. Además, CHASIS contiene toda la lógica de funcionamiento del entrenamiento, coordinando la interacción entre las piezas y el proceso de montaje.







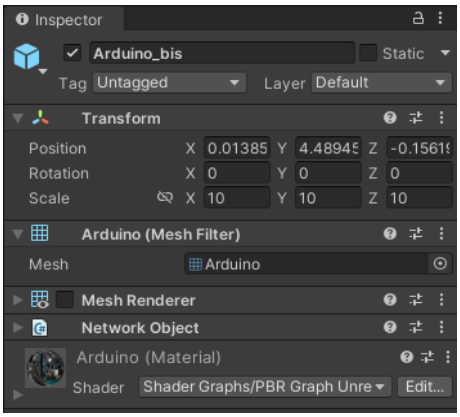
Figuras 28, 29 y 30: GameObject *CHASIS* y todos sus GameObjects hijos.

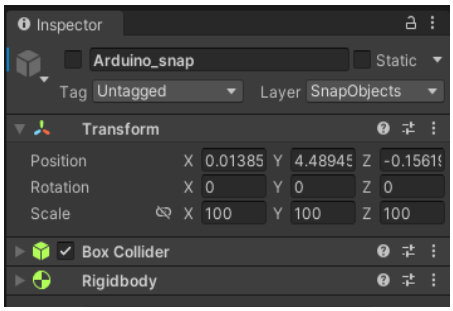
Las imágenes muestran el objeto *CHASIS*, del cual heredan todos los demás objetos del montaje. Dentro de *CHASIS*, se encuentran varios componentes clave:

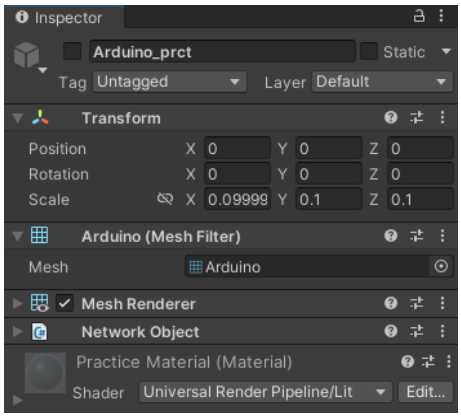
Objetos_bis: Estos elementos se activan cuando una pieza del coche, que lleva el mismo nombre pero sin el sufijo "*_bis*", colisiona con el box collider correspondiente que tiene el sufijo "*_snap*". En otras palabras, un objeto llamado "*pieza1_bis*" se activará cuando la pieza "*pieza1*" colisione con el box collider denominado "*pieza1_snap*". Esta activación ocurre cuando se detecta la colisión correcta entre la pieza y el área designada para su encaje.

Objetos_snap: Estos son box colliders que actúan como triggers para activar los *objetos_bis* cuando se produce la colisión correcta.

Objeto_prct: Este es un objeto sombreado que indica la posición correcta donde debe colocarse cada pieza. Este componente es visible únicamente en el modo de práctica.







Figuras 31, 32, 33: Componentes de los GameObjects *Arduino_bis*, *Arduino_snap* y *Arduino_prct*.

Las imágenes muestran los objetos *_bis*, *_snap* y *_prct*:

_bis: El Mesh Renderer de este objeto está desactivado por defecto y solo se activa cuando la pieza correspondiente ha sido colocada correctamente. Esto asegura que el objeto *_bis* sólo se visualice en el momento adecuado, y el objeto agarrable de la mesa se desactiva para indicar que la pieza ha sido correctamente colocada.

_snap: Este objeto tiene un *Box Collider* que actúa como trigger para detectar la colisión con la pieza correcta. Está desactivado en la vista general y solo se activa durante el paso en el que se necesita realizar la colisión. Esto garantiza que el sistema solo detecte las piezas en el momento específico requerido.

_prct: El objeto _prct es un simple *Mesh Renderer* con un material semitransparente de color azulado. Está desactivado por defecto y solo se activa en el modo de práctica y en el paso específico en el que se requiere mostrar la ubicación de la pieza.

Los objetos *_bis* y *_prct* tienen el componente *Network Objects* para permitir la sincronización de la propiedad de transformación entre usuarios en un entorno multijugador, asegurando que la posición y el estado de cada objeto se mantengan consistentes para todos los usuarios.

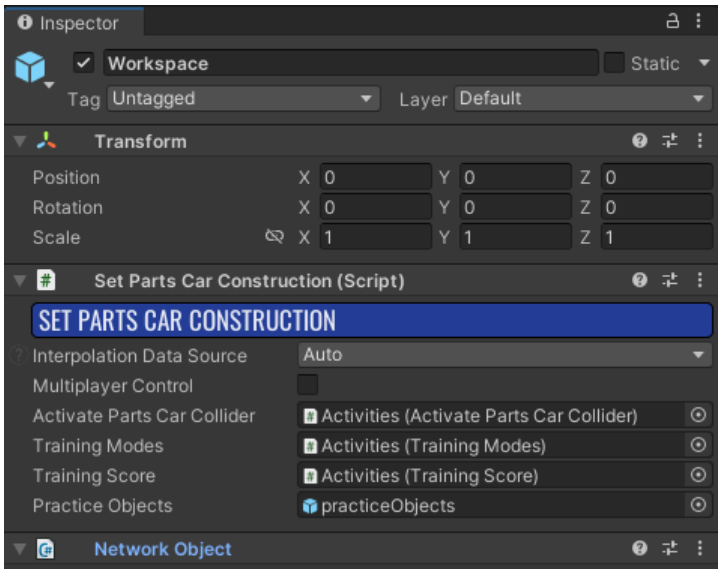


Figura 34: Componente *Set Parts Car Construction*.

El objeto Workspace contiene el script principal *SetPartsCarConstruction*, al ser *NetworkBehaviour*, es necesario añadir el componente *Network Object* para permitir la sincronización en entornos multijugador.
El script *SetPartsCarConstruction* tiene asignados otros tres scripts cruciales para el correcto funcionamiento del entrenamiento. *ActivatePartsCarCollider*, *TrainingModes* y *TrainingScore*.

Estos scripts son esenciales para la ejecución fluida del proceso de montaje y se explican en detalle en el siguiente punto.

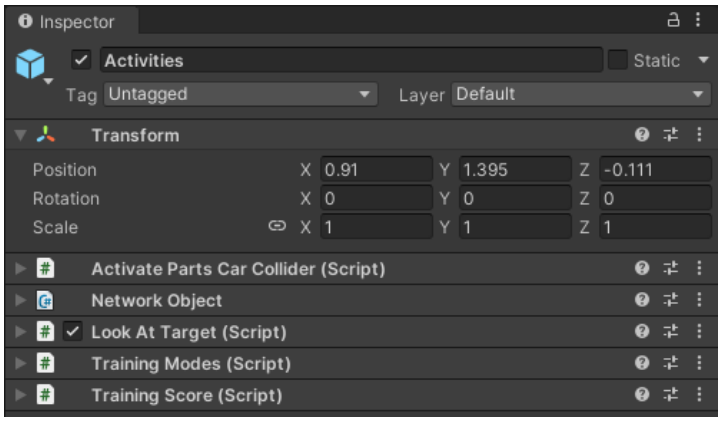


Figura 35: Componentes del GameObject *Activities*.

La imagen muestra los componentes del *GameObject Activities*, que son *ActivatePartsCarCollider*, *NetworkObject*, *LookAtTarget*, *TrainingModes* y *TrainingScore*.

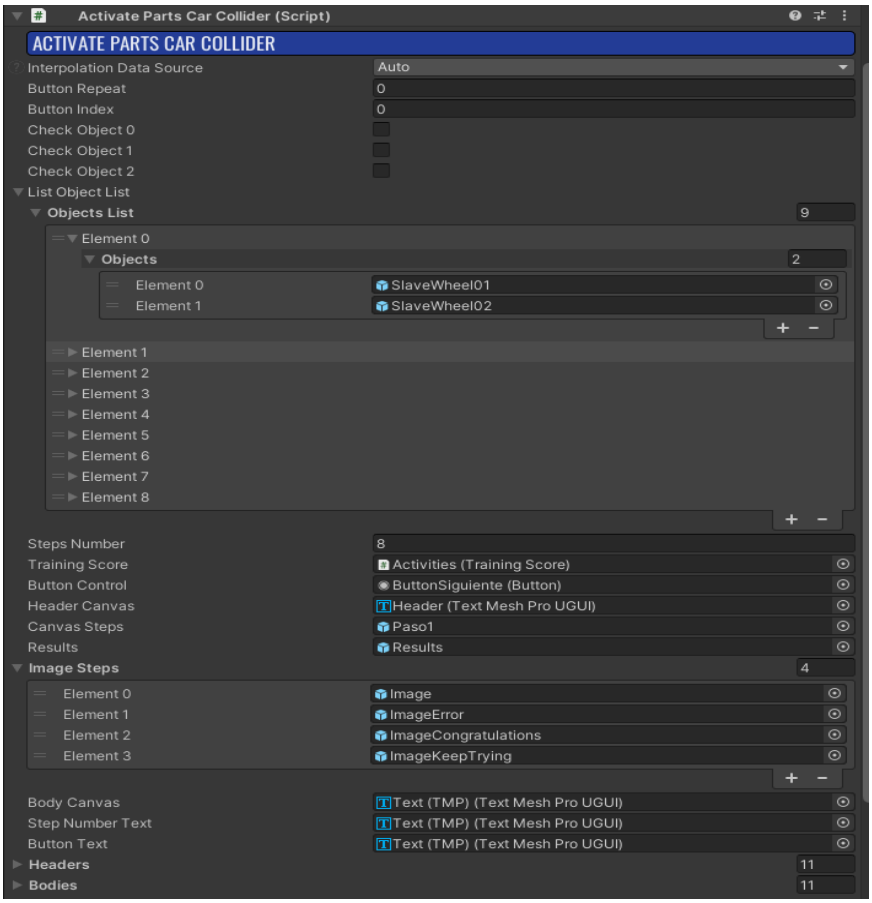


Figura 36: Componente *Activate Parts Car Collider*.

La imagen muestra el script *ActivatePartsCarCollider* en el editor de Unity. En este script destaca la estructura de listas de listas de *GameObjects*, que organiza los objetos de cada paso, permitiendo una gestión secuencial y ordenada del proceso de montaje.

Además, he asignado otros objetos y variables esenciales para garantizar el correcto funcionamiento del script y su integración en el entorno multijugador, también el script *TrainingScore*.

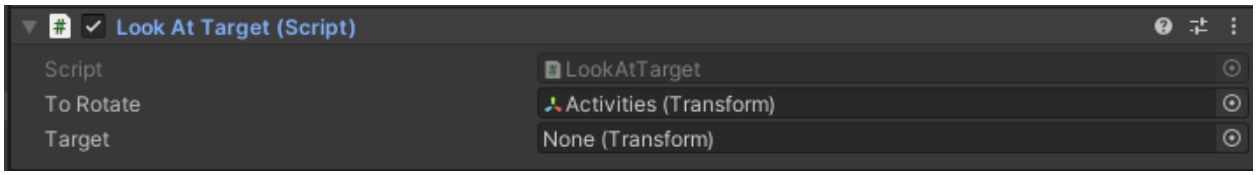


Figura 37: Componente *Look At Target*.

Esta imagen muestra un script diseñado para asegurar que el canvas siga constantemente la mirada del usuario, manteniéndose siempre orientado hacia él durante el entrenamiento.

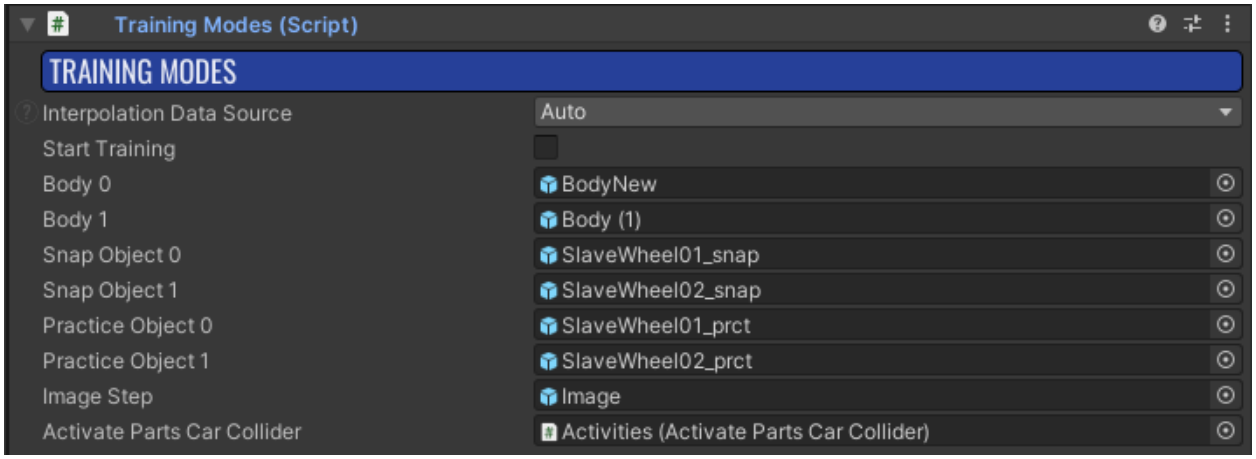


Figura 38: Componente *Training Modes*.

La imagen muestra el script *TrainingModes*. Sirve para cambiar entre los modos de práctica y examen. Dependiendo del modo seleccionado por el usuario, se activan o desactivan los objetos *_prct* en cada paso. Además, este script lleva un conteo de errores y gestiona la evaluación final del entrenamiento.

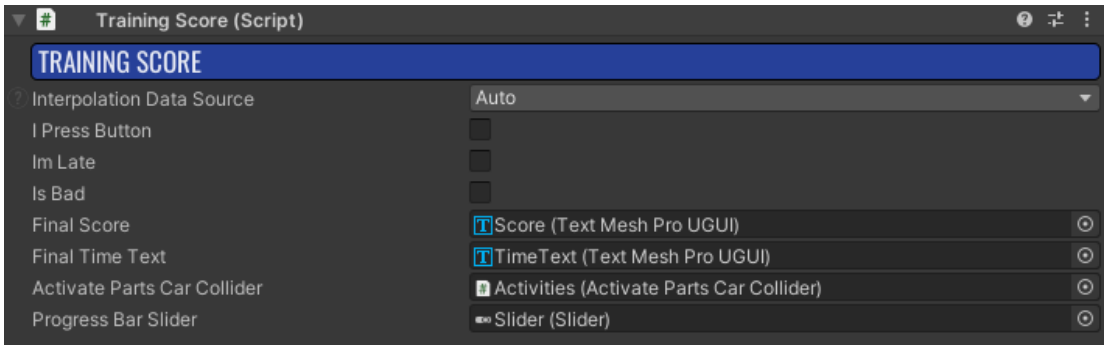


Figura 39: Componente *Training Score*.

La imagen muestra el script *TrainingScore* que se encarga de llevar el conteo de errores y modificar el mensaje y color de la pantalla final.

2.4. Scripts

2.4.1. SetPartsCarConstruction

Este script es el núcleo del sistema, encargado de coordinar la activación y desactivación de las diferentes partes del coche durante la simulación. Implementa lógica asincrónica para manejar modos de entrenamiento y examen, sincronizando acciones entre múltiples jugadores mediante RPCs. Controla qué piezas deben activarse, cuál es el siguiente paso en el proceso y cuándo desactivar componentes en función del progreso.

Variables:

```
public bool multiplayerControl;

[SerializeField] private ActivatePartsCarCollider activatePartsCarCollider;
[SerializeField] private TrainingModes trainingModes;
[SerializeField] private TrainingScore trainingScore;
[SerializeField] private GameObject practiceObjects;

[Networked] private bool ExamMode { get; set; }
private string NameObject { get; set; }
private string _nameObject0;
private string _nameObject1;
private string _nameObject2;
private ObjectsList _localObjectsList;
private bool _control;
private GameObject _mainObject;
private GameObject _practiceObject;
private GameObject _bisObject;
private int _localIndex;
private bool _checkObject0;
private bool _checkObject1;
private bool _checkObject2;
```

Funciones principales:

Spawned: Es llamado cuando el objeto se ha instanciado en la red e inicializa la lista de objetos y llama a *LocalControl*.

```
public override async void Spawned()
{
   _localObjectsList = activatePartsCarCollider.listObjectList;
   await LocalControl();
}
```

LocalControl: Esta función se encarga de sincronizar al nuevo usuario con el estado actual del entrenamiento. Si está en el modo de examen (*ExamMode*), actualiza el canvas para la selección local y destruye los objetos de práctica. Esto asegura que el nuevo usuario vea la interfaz correcta y se sincronice con el resto del grupo. Si no está en modo de examen, solo actualiza el canvas para la selección local. Establece los índices y estados de los botones para el nuevo usuario y sincroniza los objetos iterando a través de los objetos del entrenamiento para desactivar los componentes y sincronizar el estado con el resto de los usuarios.

```
private async Task LocalControl()
{
   if (trainingModes.StartTraining)
   {
       trainingScore.imLate = true;
       if (ExamMode)
       {
           trainingModes.CanvasLocalSelection();
           DestroyObjects();
       }


       else
       {
           trainingModes.CanvasLocalSelection();
       }
   }
   _localIndex = activatePartsCarCollider.ButtonIndex;
   _checkObject0 = activatePartsCarCollider.CheckObject0;
   _checkObject1 = activatePartsCarCollider.CheckObject1;
   _checkObject2 = activatePartsCarCollider.CheckObject2;
   _control = true;
   if (_localIndex != 0)
   {
       activatePartsCarCollider.ActivateObjects_MultiplayerControl(_localIndex);
   }
   var index = _localIndex != 10
       ? activatePartsCarCollider.ButtonIndex
       : activatePartsCarCollider.ButtonRepeat;

   while (true)
   {
       if (index is 0 or >= 9)
       {
```

```
            var lastIndex = _localIndex != 10
                ? activatePartsCarCollider.ButtonIndex
                : activatePartsCarCollider.ButtonRepeat;
            var middleStepList = _localObjectsList.objectsList[lastIndex].objects;
            if (_checkObject0)
            {
                NameObject = middleStepList[0].name;
                await DeactivateComponents();
            }
            if (_checkObject1)
            {
                NameObject = middleStepList[1].name;
                await DeactivateComponents();
            }
            if (_checkObject2)
            {
                NameObject = middleStepList[2].name;
                await DeactivateComponents();
            }
            break;
        }
        var currentList = _localObjectsList.objectsList[index - 1].objects;
        foreach (var obj in currentList)
        {
            NameObject = obj.name;
            await DeactivateComponents();
        }
        index--;
    }
    _control = false;
}
```

ActivateObjects_MultiplayerControl en **ActivatePartsCarCollider.cs:** Activa los objetos para el control multijugador. Necesario para la sincronización.

```
public void ActivateObjects_MultiplayerControl(int activateListIndex)
{
    var index = activateListIndex != 10 ? ButtonIndex : ButtonRepeat;
    foreach (var obj in listObjectList.objectsList[index].objects)
    {
        obj.GetComponent<ActivateObjects>().Local_StartStep();
    }
    ActiveTextCanvas(activateListIndex);
}
```

DeactivateComponents: Desactiva varios componentes y objetos en función del nombre recibido. Espera a que el índice de los botones se verifique si está en el rango permitido.

```
public async Task DeactivateComponents()
{
    _bisObject = GameObject.Find(NameObject + "_bis");
    _mainObject = GameObject.Find(NameObject);
    _practiceObject = GameObject.Find(NameObject + "_prct");

    if (_practiceObject != null && !ExamMode)
    {
        _practiceObject.gameObject.SetActive(false);
    }
    _mainObject.gameObject.SetActive(false);
    if (_bisObject != null)
    {
        var meshRenderer = _bisObject.GetComponent<MeshRenderer>();
        if (meshRenderer != null)
        {
            meshRenderer.enabled = true;
        }
        ActivateRecursive(_bisObject.transform);
    }


    if (activatePartsCarCollider.ButtonIndex < 9 )
    {
        await activatePartsCarCollider.Check_MultiplayerObjectIndex();
    }
    if (_control)
    {
        return;
    }
    RPC_DeactivateComponents();
}
```

Check_MultiplayerObjectIndex en **ActivatePartsCarCollider.cs:** Verifica el estado de los objetos para el control multijugador. Esto sirve por si entra otro usuario a mitad entrenamiento.

```
public Task Check_MultiplayerObjectIndex()
{
   for (var i = 0; i < listObjectList.objectsList[ButtonIndex].objects.Count; i++)
   {
       if (listObjectList.objectsList[ButtonIndex].objects[i].activeSelf == false)
       {
             switch (i)
             {
                 case 0:
                     CheckObject0 = true;
                     break;
                 case 1:
                     CheckObject1 = true;
                     break;
                 case 2:
                     CheckObject2 = true;
                     break;
             }
         }
    }
    return Task.CompletedTask;
}
```

RPC_ChooseMode: Cambia el modo a *ExamMode* y destruye los objetos.

```
[Rpc]
public void RPC_ChooseMode()
{
   ExamMode = true;
   DestroyObjects();
}
private void DestroyObjects()
{
   foreach (Transform child in practiceObjects.transform)
   {
       Destroy(child.gameObject);
   }
}
```

RPC_SendString: Asigna el nombre del objeto local de un usuario a una variable *Networked* compartida por todos los usuarios de la sala.

```
[Rpc]
public void RPC_SendString(string newName)
{
   NameObject = newName;
}
```

RPC_DeactivateComponents: Finaliza la desactivación de componentes para todos los jugadores.

```
[Rpc]
private void RPC_DeactivateComponents()
{
   if (!multiplayerControl)
   {
      Igual que DeactivateComponents()
   }
   else
   {
       multiplayerControl = false;
   }
}
```

2.4.2. ActivatePartsCarCollider

Este script controla la secuencia de pasos en la simulación, activando los objetos correctos (*BoxColliders* que permiten el *snap*) en cada etapa y gestionando la interfaz de usuario que guía el entrenamiento. Detecta si todas las piezas en un paso específico han sido colocadas correctamente antes de avanzar.

Variables:

```
[System.Serializable]
public class Objects
{
   public List<GameObject> objects;
}
[System.Serializable]
public class ObjectsList
{
   public List<Objects> objectsList;
}
public class ActivatePartsCarCollider : NetworkBehaviour
{
   [Networked] public int ButtonRepeat { get; private set; }
   [Networked] public int ButtonIndex { get; private set; }
   [Networked] public bool CheckObject0 { get; private set; }
   [Networked] public bool CheckObject1 { get; private set; }
   [Networked] public bool CheckObject2 { get; private set; }
   public ObjectsList listObjectList;
   public int stepsNumber;
   [SerializeField] private TrainingScore trainingScore;
   [SerializeField] private Button buttonControl;
   [SerializeField] private TMP_Text headerCanvas;
   [SerializeField] private GameObject canvasSteps;
   [SerializeField] private GameObject results;
   [SerializeField] private GameObject[] imageSteps;
   [SerializeField] private TMP_Text bodyCanvas;
   [SerializeField] public TMP_Text stepNumberText;
   [SerializeField] private TMP_Text buttonText;
   [SerializeField] private List<string> headers;
   [SerializeField] private List<string> bodies;
   private bool _control;
   private bool _flag;
   private bool _allObjectsInactive;
```

Funciones principales:

PressButton: Maneja la lógica al pulsar un botón, activando los objetos correspondientes y gestionando la interfaz.

```
private async void PressButton()
{
   trainingScore.iPressButton = true;
   if (ButtonIndex == stepsNumber + 1 || ButtonIndex == stepsNumber + 2)
   {
       if (ButtonIndex == stepsNumber + 1)
       {
           RPC_CanvasControl(ButtonIndex);
       }
       else
       {
           RPC_Repeat(ButtonRepeat);
       }
   }
   else
   {
       await CheckObjects(ButtonIndex);
       if (_flag)
       {
           ActivateObjects(ButtonIndex);
       }
   }
}
```

CheckObjects: Verifica si todos los objetos en la lista están inactivos y actualiza el estado de la bandera *_flag*. De esta manera, al ser una *Task,* el orden de ejecución vuelve a la función *PressButton,* justo en el *await* donde se había llamado a *CheckObjects,* si el valor de *_flag* es *true* se llamará a *ActivateObjects.*

```
private Task CheckObjects(int checkListIndex)
{
   var toCheckList = listObjectList.objectsList[checkListIndex].objects;
   var allObjectsInactive = toCheckList.All(obj => obj.activeSelf == false);
   if (allObjectsInactive)
   {
       _flag = true;
   }
```

```
    else
    {
        ButtonRepeat = ButtonIndex;
        ButtonIndex = 10;
        RPC_CanvasControl(ButtonIndex);
    }
    return Task.CompletedTask;
}
```

ActivateObjects: Activa objetos en la lista y controla el canvas de entrenamiento.

```
private void ActivateObjects(int activateListIndex)
{
    _flag = false;
    if (ButtonIndex < stepsNumber)
    {
        activateListIndex++;
listObjectList.objectList[activateListIndex].objects)
        {
            obj.GetComponent<ActivateObjects>().RPC_StartStep();
        }
    }
    RPC_CanvasControl(ButtonIndex);
}
```

RPC_CanvasControl: Actualiza el control del canvas basado en el índice del objeto. Si se quisiera implementar un entrenamiento de diferente número de pasos habría que cambiar los índices del *switch*. Una manera cómoda sería crear variables públicas y asignar el número de pasos desde el editor.

```
[Rpc]
private void RPC_CanvasControl(int objIndex)
{
    switch (objIndex)
    {

        case < 8:
            ButtonIndex++;
            CheckObject0 = false;
            CheckObject1 = false;
            CheckObject2 = false;
            ButtonRepeat = ButtonIndex;
            objIndex++;
            stepNumberText.text = (ButtonIndex+1).ToString();
            trainingScore.RPC_ProgressSlider(objIndex);
            break;



        case 8:
```

```
            trainingScore.finalScore.enabled = true;
            trainingScore.finalTimeText.enabled = true;
            trainingScore.RPC_FinalScore();
            ButtonIndex++;
            objIndex++;
            trainingScore.RPC_ProgressSlider(objIndex);
            break;
        case 9:
            canvasSteps.SetActive(false);
            break;
        case 10:
            trainingScore.RPC_TotalScore();
            break;
    }
    ActiveTextCanvas(objIndex);
}
```

ActiveTextCanvas: Actualiza el texto y los elementos visuales del canvas en función del índice.

```
private void ActiveTextCanvas(int indexCanvas)
{
    if (indexCanvas == 9)
    {
        imageSteps[0].SetActive(false);
        results.SetActive(true);
        headerCanvas.text = trainingScore.isBad ? "Keep trying" : "Congratulations!";
        var imageIndex = trainingScore.isBad ? 3 : 2;
        imageSteps[imageIndex].SetActive(true);
        bodyCanvas.text = trainingScore.isBad
          ? "You have not learned yet how to build a car.\nThanks for your time and keep trying!"
          : "You have learned how to build a car. Thanks for your time and enjoy you've learned!";
        headerCanvas.color = trainingScore.isBad
            ? new Color(0.95f, 0.24f, 0.24f, 1f)
            : new Color(9f / 255f, 212f / 255f, 209f / 255f);
    }
    else
    {
        trainingScore.progressBarSlider.gameObject.SetActive(indexCanvas != 10);
        stepNumberText.text = (indexCanvas + 1).ToString();
        imageSteps[0].SetActive(false);
        imageSteps[1].SetActive(false);
        headerCanvas.text = headers[indexCanvas];
        headerCanvas.color = indexCanvas == 10



            ? new Color(0.95f, 0.24f, 0.24f, 1f)
            : new Color(0.2f, 0.2f, 0.25f, 1f);
```

```
        var imageIndex = indexCanvas == 10 ? 1 : 0;
        imageSteps[imageIndex].SetActive(true);
        bodyCanvas.text = bodies[indexCanvas];
        buttonText.text = indexCanvas == 10 ? "REPEAT" : "NEXT";
    }
}
```

RPC_Repeat: Controla la interfaz del canvas cuando el usuario ha producido un error.

```
[Rpc]
private void RPC_Repeat(int objIndex)
{
    imageSteps[0].SetActive(true);
    imageSteps[1].SetActive(false);
    ButtonIndex = objIndex;
    ActiveTextCanvas(ButtonIndex);
}
```

2.4.3. ActivateObjects

Este script se encarga de activar colisionadores y objetos durante un paso específico del entrenamiento. Es utilizado tanto de manera local como en red para sincronizar el estado de los objetos.

Variables:

```
public bool boxColliderEnabled;
[SerializeField] private GameObject colliderObject;
[SerializeField] private GameObject practiceObject;
private BoxCollider _boxColliderComponent;
```

Funciones principales:

RPC_StartStep: Activa los *BoxColliders* y objetos *snap* durante un paso en un entorno multijugador. En modo Práctica activa los *objetos_prct.*

```
[Rpc]
public void RPC_StartStep()
{
    _boxColliderComponent.enabled = true;
    colliderObject.SetActive(true);
    if (practiceObject != null)
    {
```

```
        practiceObject.SetActive(true);
    }
}
```

Local_StartStep: Activa los objetos localmente si no se requiere sincronización en red.

```
public void Local_StartStep()
{
    _boxColliderComponent.enabled = true;
    colliderObject.SetActive(true);
    if (practiceObject != null)
    {
        practiceObject.SetActive(true);
    }
}
```

2.4.4. Collider Snap

Este script gestiona la detección de colisiones para verificar cuándo una pieza ha sido colocada correctamente en su posición. Al detectar una colisión válida, comunica el estado al script *SetPartsCarConstruction* para desactivar componentes y continuar con la simulación.

Variables:

```
[SerializeField] private SetPartsCarConstruction setPartsCarConstruction;
private GameObject _collidedObject;
private bool _chassisGrab;
```

Funciones principales:

OnTriggerEnter: Detecta cuándo una pieza "encaja" en su lugar y envía el nombre del objeto colisionado a *DeactivateComponents* para desactivar la pieza y avanzar en el entrenamiento.

```
private async void OnTriggerEnter(Collider other)
{
   if (other.gameObject.name != gameObject.name + "_snap" || !_chassisGrab) return;
   _collidedObject = gameObject;
   var collidedObjectName = _collidedObject.name;
   setPartsCarConstruction.RPC_SendString(collidedObjectName);
   setPartsCarConstruction.multiplayerControl = true;
   await setPartsCarConstruction.DeactivateComponents();
}
```

ChassisGrabControl_Select y **ChassisGrabControl_UnSelect:** Activan o desactivan la capacidad de "ajustar" una pieza cuando el usuario agarra el chasis. Esto permite que se pueda ajustar una pieza al chasis y no al revés. Estas funciones son llamadas desde el editor, desde el componente *Pointable Unity Event Wrapper,* asignando el *GameObject* Chasis en el *pointable* y llamando a cada una de las funciones desde los eventos *When Select* y *When Unselect*

```
public void ChassisGrabControl_Select()
{
   _chassisGrab = true;
}
public void ChassisGrabControl_UnSelect()
{
   _chassisGrab = false;
}
```

2.4.5. TrainingModes

Este script controla los diferentes modos de entrenamiento.

Variables:

```
[Networked] public bool StartTraining { get; private set; }
[SerializeField] private GameObject body0;
[SerializeField] private GameObject body1;
[SerializeField] private GameObject snapObject0;
[SerializeField] private GameObject snapObject1;
[SerializeField] private GameObject practiceObject0;
[SerializeField] private GameObject practiceObject1;
[SerializeField] private GameObject imageStep;
[SerializeField] private ActivatePartsCarCollider activatePartsCarCollider;
```

RPC_ModeSelection: Configura los objetos y el estado de entrenamiento basado en el modo de práctica.

```
[Rpc]
public void RPC_ModeSelection(bool practiceMode)
{
   if (practiceMode)
   {
       practiceObject0.SetActive(true);
       practiceObject1.SetActive(true);
   }
   imageStep.SetActive(true);
   body0.SetActive(true);
   body1.SetActive(false);
   snapObject0.SetActive(true);
   snapObject1.SetActive(true);
   StartTraining = true;
}
```

CanvasLocalSelection: Actualiza la selección de canvas local en función del índice del botón.

```
public void CanvasLocalSelection()
{
   if (activatePartsCarCollider.ButtonIndex == 0)
   {
       practiceObject0.SetActive(true);
       practiceObject1.SetActive(true);
   }
   body0.SetActive(true);
   body1.SetActive(false);
}
```

2.4.6. TrainingScore

Este script maneja el seguimiento de puntuaciones y errores durante el entrenamiento.

Variables:

```
public bool iPressButton;
public bool imLate;
public bool isBad;
public TMP_Text finalScore;
public TMP_Text finalTimeText;
public ActivatePartsCarCollider activatePartsCarCollider;
public Slider progressBarSlider;
[Networked] private float Score { get; set; }
[Networked] private int NumberErrors { get; set; }
[Networked] private float CurrentValue { get; set; }
private bool _imFirst;
private NetworkRunner _networkRunner;
```

Spawned: Inicializa el valor del slider de progreso. *CurrentValue* es una variable *Networked*, por lo que se sincroniza con todos los usuarios una vez entren al entrenamiento.

```
public override void Spawned()
{
   progressBarSlider.value = CurrentValue;
}
```

RPC_ProgessSlider: Actualiza el valor del slider de progreso.

```
[Rpc]
public void RPC_ProgressSlider(float newStep)
{
   CurrentValue = newStep / (activatePartsCarCollider.stepsNumber + 1);
   progressBarSlider.value = CurrentValue;
}
```

RPC_TotalScore: Calcula la puntuación total y actualiza la variable *isBad* en función del número de errores. Si el número de errores supera la mitad del número de pasos la UI de la pantalla final del entrenamiento tendrá un diseño en rojo indicando que se necesita mejorar.

```
[Rpc]
public void RPC_TotalScore()
{
   if (!iPressButton) return;
   NumberErrors++;
   if (NumberErrors > activatePartsCarCollider.stepsNumber / 2)
   {
       isBad = true;
   }
   iPressButton = false;
}
```

RPC_FinalScore: Calcula y actualiza la puntuación final. Llama a *RPC_LocalFinalScore* que se encarga de actualizar el texto.

```
[Rpc]
public void RPC_FinalScore()
{
   if (imLate) return;
   var totalSteps = activatePartsCarCollider.stepsNumber + 1f;
   var corrects = totalSteps - NumberErrors;
   Score = corrects;
   iPressButton = false;
   RPC_LocalFinalScore();
}

[Rpc]
private void RPC_LocalFinalScore()
{
   finalScore.color =
       !isBad ? new Color(9f / 255f, 212f / 255f, 209f / 255f) : new Color(0.95f, 0.24f, 0.24f, 1f);
   finalScore.text = Score.ToString(CultureInfo.InvariantCulture);
}
```

El entrenamiento comienza con la selección del modo en *TrainingModes*, configurando el entorno para práctica o examen. A medida que el usuario interactúa con las piezas del coche (gestionadas por *ActivatePartsCarCollider*), los pasos avanzan activando o desactivando objetos según sea necesario. Si una pieza se coloca correctamente (detectado por *ColliderSnap*), el entrenamiento continúa. El sistema garantiza la sincronización entre varios usuarios y realiza un seguimiento del progreso y los errores, mostrando finalmente los resultados en la interfaz de usuario.
Este enfoque asegura una experiencia de entrenamiento sincronizada e interactiva, con la lógica separada de manera ordenada en distintos scripts que gestionan tareas específicas como la activación de objetos, el control del modo, la detección de colisiones y la puntuación.

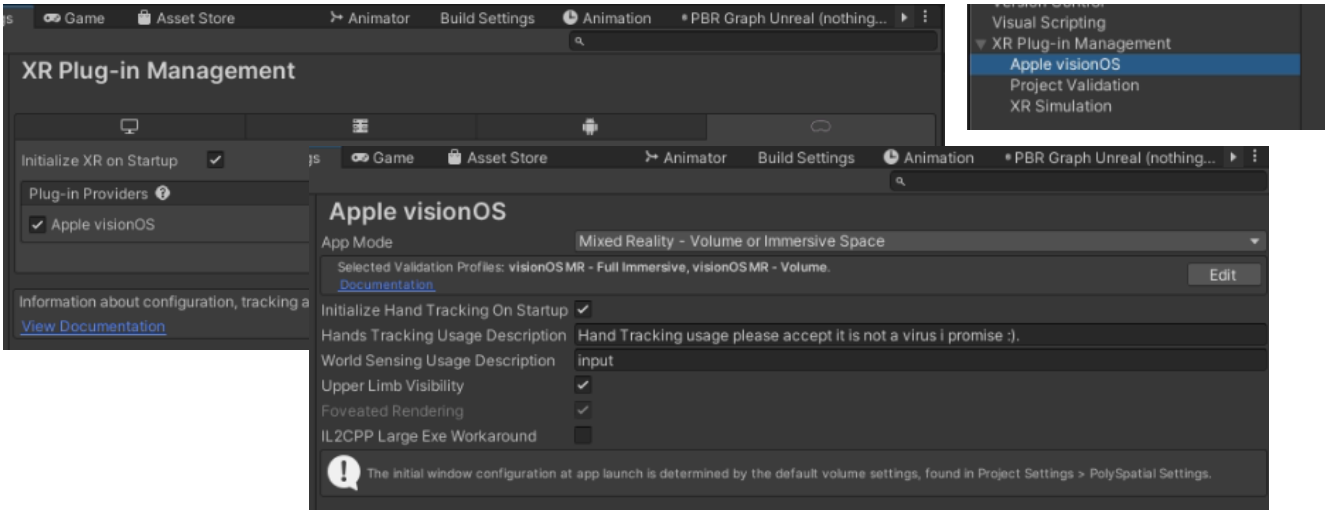
2.5. Versión en Apple Vision Pro

Adicionalmente, he desarrollado una versión compatible con las Apple Vision Pro, adaptando la aplicación a las características avanzadas de este dispositivo, como su enfoque en la realidad mixta y su capacidad de interacción precisa.

El desarrollo de la versión para Apple Vision Pro implicó algunos requisitos específicos debido a la naturaleza de la plataforma. Para empezar, es necesario trabajar en un entorno macOS, ya que Vision Pro solo es compatible con sistemas Apple. Además, se requiere la licencia Pro de Unity, ya que sin ella ni siquiera es posible cambiar la plataforma a *VisionOS* y, por lo tanto, no se puede hacer nada en este entorno. Otro aspecto importante fue la necesidad de utilizar *Xcode* en su versión 15.2 o superior, que es indispensable para trabajar con Apple Vision Pro. Como Oculus Integration no es compatible con este dispositivo, tuve que usar el SDK de *PolySpatial* [6], que se adapta a las características de Vision Pro.

Durante el desarrollo, pude testear la aplicación directamente en las Apple Vision Pro que tenemos en la oficina. Sin embargo, también utilicé el simulador de Vision Pro disponible en *Xcode*, lo cual fue de gran ayuda para no tener que generar una build e instalarla en las gafas cada vez que quería realizar una prueba. Este simulador me permitió acelerar considerablemente el proceso de desarrollo y optimización.

A continuación, presento algunas imágenes que muestran la estructura de desarrollo de la aplicación en Apple Vision Pro. Estas imágenes destacan la organización de los prefabs de *PolySpatial*, que están preconfigurados para permitir las interacciones pertinentes dentro de este entorno.



Figuras 40, 41 y 42: Implementación XR plug-in Apple *visionOS*.

En las siguientes imágenes se presenta la configuración de la pestaña *XR Plug-in Management*, donde se observa la activación de Apple *VisionOS*, un paso imprescindible para desarrollar esta versión de la aplicación. Cabe destacar que esta opción solo se puede activar si se cuenta con una licencia Pro de Unity, lo cual es un requisito fundamental para trabajar en el entorno de *VisionOS*. Además, se incluye la configuración específica para Apple *VisionOS* en la pestaña de *Mixed Reality*, que permite ajustar las opciones avanzadas de realidad mixta, necesarias para aprovechar al máximo las capacidades del Apple Vision Pro.

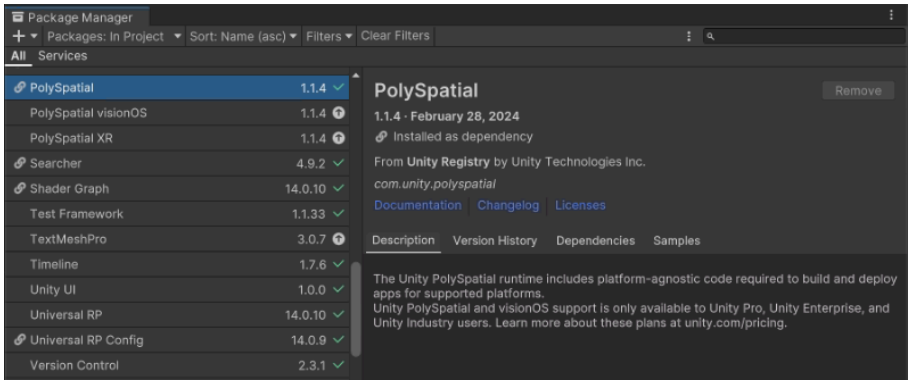
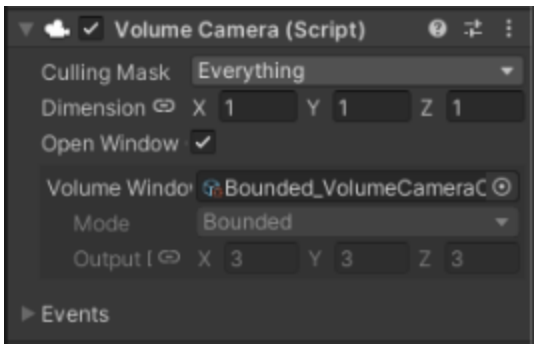


Figura 43: *PolySpatial SDK* en la pestaña de *Package Manager*.

La siguiente imagen muestra el *SDK PolySpatial* versión 1.1.4 en el *Package Manager* de Unity, que es fundamental para desarrollar la versión de la aplicación para Apple Vision Pro. Este *SDK*, que se actualiza constantemente, es esencial para habilitar las interacciones avanzadas y optimizar la experiencia en *VisionOS*, ya que está diseñado específicamente para aprovechar las características únicas del dispositivo, como la realidad mixta y las capacidades de interacción precisa.



Figuras 44: Componente *Volume Camera*.

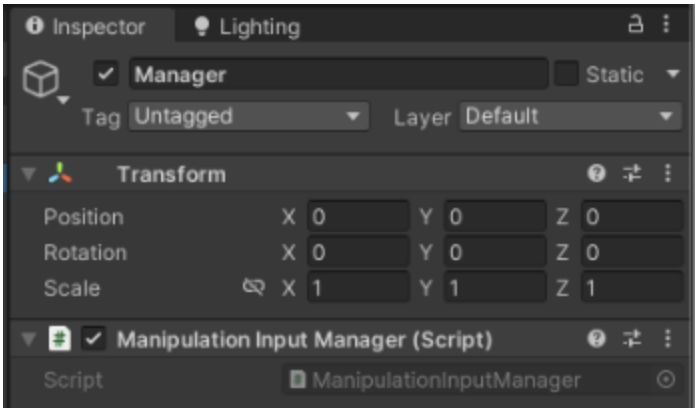


Figura 45: Componente *Manipulation Input Manager*.

La imagen muestra la configuración de la *VolumeCamera*, que define un área de 1 m³ donde se renderizan los objetos virtuales y se habilitan las interacciones en el entorno de realidad mixta. En este volumen, se encuentran activos componentes esenciales como el script *ManipulationInputManager* de *PolySpatial*, que permite a los usuarios agarrar objetos a distancia utilizando eye tracking y el gesto de *pinch* (juntar el dedo índice con el pulgar) con cualquiera de las manos. Esta configuración es clave para aprovechar al máximo las capacidades interactivas de Apple Vision Pro.

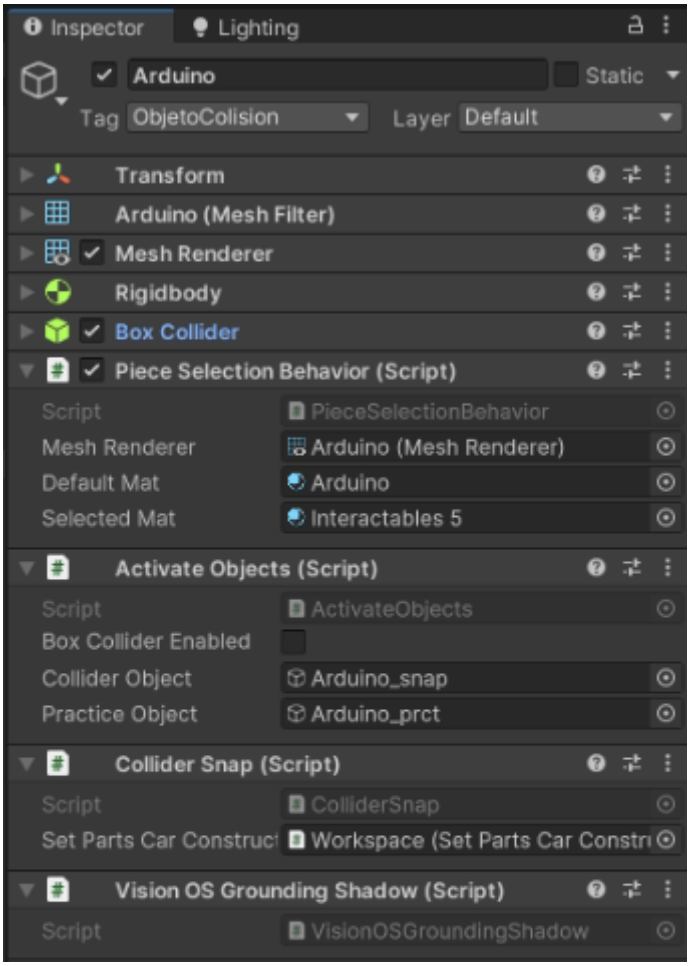


Figura 46: Componentes del GameObject *Arduino*.

La imagen muestra una de las piezas del entrenamiento que es agarrable. Esta pieza tiene asignado el script *PieceSelectionBehaviour*, un componente del *SDK PolySpatial*. Este script es fundamental para habilitar la interacción en Apple Vision Pro, permitiendo que la pieza sea seleccionada y manipulada por el usuario. Además del script, es necesario que el objeto tenga un *BoxCollider* y un *RigidBody* para asegurar la detección de colisiones y la física adecuada durante la interacción. Estos elementos en conjunto permiten que la pieza sea agarrada y movida en el entorno virtual.

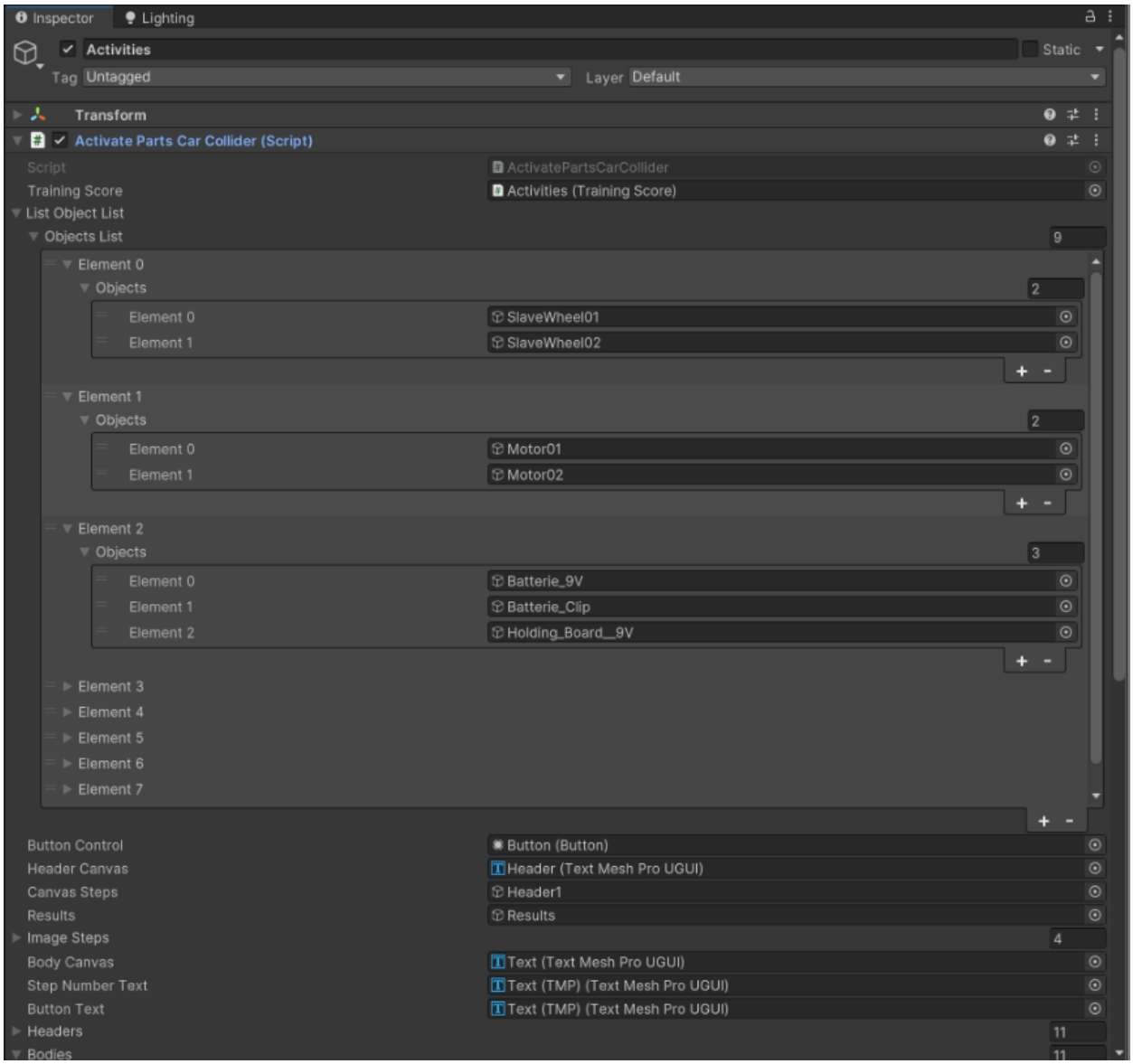


Figura 47: Componente *Activate Parts Car Collider* en la versión de Apple Vision Pro.

La imagen muestra el script ActivatePartsCarCollider.cs, un ejemplo de cómo el diseño modular del código facilita la implementación del entrenamiento. Gracias a su estructura, configurar los pasos del montaje es sencillo y eficiente. El script permite organizar los objetos involucrados en cada etapa del entrenamiento simplemente añadiéndolos a la lista correspondiente. Esta modularidad no solo agiliza el proceso de desarrollo, sino que también permite escalar y adaptar la aplicación para otros escenarios de montaje con facilidad.

Para concluir el tema de la estructura de desarrollo en Apple Vision Pro, es importante destacar que no se mostrarán más detalles específicos, ya que la configuración es muy

similar a la de los dispositivos Quest. La principal diferencia radica en la eliminación de las funcionalidades multijugador, lo que simplifica el desarrollo para esta plataforma. Este enfoque resalta la eficacia de utilizar una estructura modular y sencilla, que no solo facilita la adaptación a diferentes dispositivos, sino que también permite una implementación más eficiente de entrenamientos virtuales.

Capítulo 3. Aplicación en la industria automotriz

3.1. Videos de la Aplicación en Acción

3.1.1. Video en Versión Quest

Para ver el resultado final de la aplicación, consulta este vídeo.

El video muestra una demostración completa de la aplicación en un dispositivo Quest 3 en modo cooperativo. La primera parte del video presenta el modo práctica, en el que se utilizan los mandos para interactuar con la aplicación. Se puede observar cómo los usuarios siguen las instrucciones y montan las piezas del vehículo en un entorno virtual.

En la segunda parte, el video muestra el modo examen, donde se utiliza el hand tracking para realizar el montaje. Este segmento destaca cómo la aplicación evalúa el desempeño del usuario en tiempo real y proporciona feedback durante el proceso. La transición entre ambos modos ilustra la versatilidad y la capacidad de la aplicación para adaptarse a diferentes métodos de interacción.

3.1.2. Video en Version Apple Vision Pro

Para ver la demostración de la aplicación en el dispositivo Apple Vision Pro, consulta este vídeo.

En el video se presenta la aplicación ejecutada en modo local, sin la funcionalidad multijugador. La primera parte muestra el modo práctica, en el que se utiliza la realidad mixta para interactuar con los objetos y montar las piezas del vehículo. La segunda parte ilustra cómo se lleva a cabo el montaje en un entorno de realidad mixta, destacando la capacidad de la aplicación para integrar elementos virtuales con el entorno físico. Este video resalta cómo la aplicación se adapta a las características únicas del Apple Vision Pro y proporciona una experiencia inmersiva en el modo local.

3.2. Herramienta de Capacitación y Formación Continua

La aplicación desarrollada representa una solución innovadora para la capacitación de nuevos empleados en la industria automotriz. Mediante un entorno virtual inmersivo, los trabajadores pueden aprender y dominar los procesos de montaje de vehículos sin enfrentar los riesgos y costos que implicaría la formación en un entorno físico tradicional. La Realidad Virtual permite que los empleados practiquen y perfeccionen sus habilidades antes de trabajar con componentes reales, lo que reduce significativamente la probabilidad de errores y optimiza la eficiencia del proceso de aprendizaje. Además, la aplicación ofrece retroalimentación en tiempo real, ayudando a los usuarios a corregir errores de inmediato y a mejorar sus técnicas de manera más efectiva.

Esta herramienta no solo es valiosa para los nuevos empleados, sino que también desempeña un papel clave en la formación continua del personal existente. En una industria tan dinámica como la automotriz, es crucial que los trabajadores se mantengan al día con las últimas tecnologías y métodos de montaje. La aplicación VR facilita la actualización de conocimientos y habilidades de una manera flexible y accesible. Los empleados pueden realizar sesiones de entrenamiento periódicas para aprender sobre nuevos modelos de vehículos, técnicas de montaje avanzadas y mejores prácticas sin necesidad de interrumpir su trabajo diario. Esto no solo mejora la competencia técnica del personal, sino que también aumenta la adaptabilidad de la empresa ante los cambios tecnológicos constantes.

Además, la flexibilidad de la aplicación permite ajustarla a distintos entornos y necesidades. Ya sea integrando nuevos modelos de vehículos o adaptándola a diferentes dispositivos de Realidad Virtual, la solución desarrollada puede evolucionar junto con las demandas de la industria. De esta forma, se asegura que la capacitación no solo sea efectiva, sino también sostenible y relevante en el tiempo.

3.3. Beneficios y Retos

La implementación de una aplicación de Realidad Virtual en la industria automotriz ofrece una serie de beneficios clave que pueden transformar tanto la formación como la operación diaria de la empresa. En primer lugar, se destaca la reducción significativa en los costos de capacitación. Al trasladar la formación a un entorno virtual, se eliminan los gastos asociados con el uso de componentes físicos y equipos costosos, que pueden desgastarse o necesitar reposición. Además, la mejora en la seguridad laboral es notable; al practicar en un entorno simulado, los empleados pueden familiarizarse con procedimientos complejos sin exponerse a los riesgos reales de un taller o una línea de ensamblaje, lo que reduce la probabilidad de accidentes.

Otro beneficio importante es la eficiencia en el aprendizaje. La Realidad Virtual permite una práctica continua y repetitiva en un ambiente interactivo y atractivo, lo que potencia la retención del conocimiento y refuerza la motivación de los empleados. Al integrar feedback en tiempo real, los usuarios pueden corregir errores sobre la marcha y perfeccionar sus habilidades de forma más rápida y precisa, logrando una curva de aprendizaje más pronunciada en menos tiempo.

No obstante, la adopción de esta tecnología conlleva ciertos retos. La inversión inicial para adquirir el equipo necesario y desarrollar el software adecuado puede ser considerable. Además, garantizar que todos los empleados tengan acceso a esta tecnología y que reciban la capacitación adecuada para usarla eficientemente es un desafío importante. La resistencia al cambio, una constante en cualquier proceso de modernización tecnológica, también es un factor a tener en cuenta. Algunos trabajadores podrían mostrar reticencia a adoptar estas nuevas herramientas debido a la curva de aprendizaje que implican, lo que requiere una estrategia de implementación bien planificada y un apoyo continuo durante la transición.

En resumen, aunque la introducción de VR en la industria automotriz puede presentar obstáculos, los beneficios a largo plazo en términos de seguridad, eficiencia y reducción de costos son claros. Con una implementación adecuada, esta tecnología tiene el potencial de redefinir los estándares de formación en el sector, convirtiéndose en una herramienta indispensable para la capacitación técnica y continua.

Capítulo 4. Futuras ampliaciones y mejoras

4.1. Futuras mejoras

Una de las principales áreas de expansión para la aplicación es la inclusión de nuevos modelos de automóviles. Esta ampliación permitirá que la herramienta se mantenga relevante a medida que las empresas automotrices lancen nuevos vehículos al mercado. Debido a que la aplicación permite cierta modularidad es sencillo implementar estos modelos sin hacer ningún cambio en el código. Con este sistema, la aplicación podría convertirse en una plataforma versátil para entrenamientos de montaje, abarcando desde vehículos hasta otros productos industriales o incluso dispositivos electrónicos.

Además, se considera la posibilidad de integrar un sistema de medición del tiempo transcurrido durante el entrenamiento, proporcionando métricas valiosas tanto para la evaluación del rendimiento individual como para la optimización del proceso de formación. Esta función añadida permitirá un análisis detallado del progreso de los usuarios, ayudando a identificar áreas de mejora y a establecer estándares de eficiencia.

Por último, se prevé la expansión hacia un entorno multijugador multiplataforma, en el que usuarios de dispositivos Quest 3 y Apple Vision Pro puedan colaborar simultáneamente en un espacio de realidad mixta. Gracias a la tecnología Crossplatform Mixed Reality de Photon, la aplicación podría ofrecer una experiencia de formación colaborativa a través de distintas plataformas, lo que no solo enriquecería la interacción entre los participantes, sino que también permitiría a las empresas realizar sesiones de capacitación globales sin importar la ubicación o el dispositivo utilizado.

4.2. Actualización Tecnológica Continua (vídeo)

Para asegurar que la aplicación se mantenga en la vanguardia de la tecnología, se prevé una actualización constante que incorpore las innovaciones más recientes en realidad virtual y mixta. Esto incluirá mejoras en la resolución gráfica, técnicas avanzadas de seguimiento de movimiento, y el desarrollo de nuevas interfaces de usuario más intuitivas y eficientes. El objetivo es no solo optimizar la experiencia actual, sino también expandir las funcionalidades educativas hacia otras industrias. De hecho, ya se han realizado varias demos que exploran estas posibilidades.

Un ejemplo es la visualización interactiva del cuerpo humano, que se planea evolucionar hacia un modo de examen con un modelo 3D más detallado y mayor rigor científico. También se ha desarrollado una demo de construcción modular con piezas de construcción, donde el usuario tiene libertad total para construir lo que desee. Esta función podría adaptarse para enseñar conceptos de ingeniería o diseño creativo en un entorno completamente flexible.

Además, se ha trabajado en una aplicación para la empresa Porcelanosa, que presenta un catálogo de azulejos en realidad mixta. Este proyecto destaca por su capacidad para alternar entre un entorno de realidad mixta y una experiencia de realidad completamente inmersiva, permitiendo a los usuarios visualizar y combinar diferentes materiales en un espacio virtual.

Estas iniciativas demuestran el potencial de la aplicación para expandirse más allá de la industria automotriz, ofreciendo soluciones adaptadas a sectores tan diversos como la medicina, la educación creativa y el comercio. En conjunto con el feedback de los usuarios, se busca que la aplicación evolucione de manera constante, integrando nuevas tecnologías y funcionalidades que respondan a las necesidades cambiantes del mercado.

Capítulo 5. Conclusiones

5.1. Logros del Proyecto

El proyecto ha logrado avances significativos, demostrando la viabilidad y efectividad de una aplicación de Realidad Virtual (VR) para el entrenamiento en el montaje de vehículos. Se ha desarrollado una aplicación funcional que incluye modos de práctica y examen, proporcionando una experiencia de aprendizaje interactiva y segura. La integración de tecnologías como Unity, Oculus Integration y Photon Fusion ha permitido la creación de un entorno multijugador que no solo mejora la colaboración entre usuarios, sino que también explora nuevas formas de aprendizaje colectivo en un entorno inmersivo. Además, la estructura modular de la aplicación facilita futuras expansiones, permitiendo la incorporación de nuevos modelos de automóviles y la adaptabilidad a otros dispositivos de VR como Apple Vision Pro. Estos logros representan un avance importante en el uso de VR para la formación técnica en la industria automotriz, abriendo la puerta a la creación de soluciones educativas personalizadas para otros sectores.

5.2. Impacto Educativo y Profesional

La aplicación ha tenido un impacto notable tanto en el ámbito educativo como en el profesional. Desde una perspectiva educativa, se ha demostrado que es una herramienta efectiva para el aprendizaje práctico, al permitir que los usuarios adquieran y perfeccionen sus habilidades en un entorno controlado y seguro. Esta metodología inmersiva no solo mejora la retención de conocimientos, sino que también fomenta la confianza y precisión de los usuarios al eliminar los riesgos y costos inherentes a los métodos tradicionales de formación. La posibilidad de realizar entrenamientos en modo cooperativo añade una dimensión social al aprendizaje, potenciando la colaboración y el intercambio de conocimientos. A nivel profesional, la aplicación no solo es una solución innovadora para la capacitación continua, sino que también permite a las empresas mantenerse al día con las nuevas tecnologías, optimizar los procesos de formación y, en última instancia, mejorar la seguridad y eficiencia en el lugar de trabajo. Además, la capacidad de actualizar el contenido de la aplicación en función de las necesidades del mercado asegura su relevancia a largo plazo.

5.3. Reflexión Personal y Profesional

Desde una perspectiva personal y profesional, la realización de este proyecto ha sido una experiencia enriquecedora y formativa en muchos niveles. A nivel técnico, ha permitido el dominio de herramientas avanzadas para el desarrollo en VR y la implementación de soluciones tecnológicas complejas. La adopción de una metodología ágil y la colaboración con un equipo multidisciplinario han sido fundamentales para lograr los objetivos propuestos. La experiencia adquirida en la integración de tecnologías emergentes como la realidad mixta para Apple Vision Pro y en la adaptación de la aplicación a diferentes plataformas ha ampliado considerablemente mi capacidad de resolver problemas y adaptarme a nuevos desafíos. Profesionalmente, este proyecto ha abierto nuevas oportunidades en la intersección entre la tecnología educativa y la industria automotriz, destacando la importancia de la innovación continua, la flexibilidad y la cooperación en el desarrollo de aplicaciones tecnológicas avanzadas. Además, esta experiencia ha sido clave para entender mejor el impacto de la VR en la educación y la formación, no solo en la industria automotriz, sino también en otros sectores que podrían beneficiarse de este enfoque..

5.4. Presupuesto

Licencia Unity Pro: **1850,32€.** (Precio anual).

2 dispositivos Quest 3: 549,99€ x 2 = **1099,98€.**

Dispositivo Apple Vision Pro: **3499€**.

Transporte AVP a España: **1500€.**

Licencia Apple developer enterprise program: **271,26€.** (Precio anual).

Contrato diseñador modelos 3D (40 horas): Siendo la hora a 8€. 40 x 8 = **320€**.

Contrato diseñador UI (20 horas): Siendo la hora a 8€. 20 x 8 = **160€**.

Contrato desarrollador plantilla interacciones Oculus Integration y multijugador en Unity (150 horas): Siendo la hora a 8€. 150 x 8 = **1200€**.

Contrato desarrollador de la aplicación (250 horas): Siendo la hora a 4,6€. 250 x 4,6 = **1150€.** Mi contrato ha sido de 4,6€ la hora, con los conocimientos que tengo ahora reduciría las horas a más de la mitad cobrando 8€ la hora por lo que el precio no varía mucho.

Total: 1850,32 + 1099,98 + 3499 + 1500 + 271,26+ 320 + 160 + 1200+ 1150 = **11050,56€**

Capítulo 6. Bibliografía

[1] Documentación de Oculus Integration:

https://developer.oculus.com/documentation/unity/unity-isdk-interaction-sdk-overview/

[2] Documentación Photon Fusion:

https://doc.photonengine.com/fusion/current/fusion-intro
https://doc.photonengine.com/arvr/current/apple-vision-pro/crossplatform-mixed-reality

[3] Unity Engine:

https://unity.com/es/learn

[4] Realidad Virtual (VR):

https://www.brainsigns.com/es/science/s2/technologies/virtual-reality

[5] Realidad Aumentada (AR):

https://www.techtarget.com/whatis/definition/augmented-reality-AR

[6] Documentación PolySpatial SDK:

https://docs.unity3d.com/Packages/com.unity.polyspatial.visionos@1.1/manual/index.html

Capítulo 7. Anexos

Todos los scripts utilizados:
https://drive.google.com/drive/folders/10zZxk6I-o6vB-JlhSm-CYhHuMYW48nSz?usp=sharing

Vídeos Apple Developer:
https://developer.apple.com/videos/play/wwdc2023/10088/
https://developer.apple.com/videos/play/wwdc2023/10089/
https://developer.apple.com/videos/play/wwdc2023/10096/
https://developer.apple.com/videos/play/wwdc2023/10093

Vídeos tutoriales de Dilmer Valecillos, desarrollador XR:
https://www.youtube.com/watch?v=EeGgrff5zoc
https://www.youtube.com/watch?v=-gmteyMP9w0

Vídeo Black Whale Studio, desarrollador XR:
https://www.youtube.com/watch?v=k9cJfn4JcTA

Power Ups Daniel Parra, desarrollador español CSharp: Enseña principios SOLID y CLEAN code, necesario para desarrollar aplicaciones de proyectos grandes:
https://thepowerups-learning.com/como-clean-code-puede-mejorar-la-vida-util-de-tu-juego/
https://thepowerups-learning.com/como-los-principios-solid-y-clean-code-pueden-acelerar-la-produccion-de-tu-juego-consejos-y-ejemplos/

Unity Forum:
https://discussions.unity.com/t/fully-immersive-tutorial/357184/4
https://discussions.unity.com/t/stencil-support-in-polyspatial/298205/6
https://discussions.unity.com/t/upcoming-plans-to-support-visionos-2/366047
https://discussions.unity.com/t/vr-sample-vision-pro-build/339628
https://discussions.unity.com/t/casting-ray-from-pinch-point-to-gaze-collision-with-xri/336961
https://discussions.unity.com/t/spatialpointerdevice-not-listed-in-input-action-binding-properties/345437
https://docs.unity3d.com/Packages/com.unity.polyspatial@2.0/changelog/CHANGELOG.html
https://docs.unity3d.com/Packages/com.unity.polyspatial.visionos@2.0/manual/TutorialCreateFromTemplate.html