FACULTAD BUSINESS & DIGITAL SCHOOL

Trabajo de Titulación:

"Implementación de una Aula virtual inmersiva en VR como alternativa a plataformas de comunicación convencionales"

Realizado por:

Pedro José Bustamante Salazar

Director del proyecto:

Ing., Msc. Ikiam Santiago Zurita Londoño

Como requisito para la obtención del título de:

INGENIERO EN SISTEMAS DE INFORMACIÓN CON ÉNFASIS EN CONTENIDOS INTERACTIVOS

Quito, julio 2024

DECLARATORIA

El presente Trabajo de Titulación titulado :

“Implementación de una Aula virtual inmersiva en VR como alternativa a plataformas de comunicación convencionales”

Realizado por:

PEDRO JOSE BUSTAMANTE SALAZAR

Ha sido dirigido por el profesor :

Ing., MSc. IKIAM SANTIAGO ZURITA LONDOÑO

Quien considera que constituye un trabajo original de su autor

Como requisito para la obtención del título de:

INGENIERO EN SISTEMAS DE INFORMACIÓN CON ÉNFASIS EN CONTENIDOS INTERACTIVOS

Ing., MSc. IKIAM SANTIAGO ZURITA LONDOÑO

PROFESOR

DECLARACION JURAMENTADA

Yo , PEDRO JOSE BUSTAMANTE SALAZAR, con cedula de identidad No. 1754437356, declaro bajo juramento que el trabajo aquí desarrollado es de mi autoria, que no ha sido previamente presentado para ningun grado a calificación profesional; y que ha consultado las referencias bibliograficas que se incluyen en este documento.

A traves de la presente declaración, cedo mis derechos de propiedad intelectual correspondiente a este trabajo, a la UNIVERSIDAD INTERNACIONAL SEK, según lo establecido por la Ley de Propiedad Intelectual, por su reglamento y por la normativa institucional vigente.

Pedro Jose Bustamante Salazar

C.C: 1754437356

INDICE

Tabla de contenido

INDICE DE GRAFICOS

INDICE DE TABLAS

DEDICATORIA

Este trabajo está dedicado, con profunda gratitud y cariño, a mis padres, cuyo esfuerzo y apoyo incondicional han sido el pilar fundamental para alcanzar cada meta propuesta. A mi hermano, por su constante ánimo y compañía en cada paso del camino. Dedico también estas páginas a mis docentes, quienes con su guía han encendido en mí la pasión por la tecnología y la innovación, y a mis compañeros de carrera, con quienes compartí experiencias y aprendizajes que enriquecieron este proceso.

AGRADECIMIENTO

Deseo expresar mi más sincero agradecimiento a quienes hicieron posible la realización de este proyecto. En primer lugar, a mis padres, por su apoyo constante, motivación diaria y ejemplo de perseverancia; y a mi hermano, por su aliento y paciencia a lo largo de este camino.

Extiendo mi gratitud a la Universidad Internacional SEK (UISEK) por brindarme la formación académica, las herramientas y los espacios necesarios para desarrollar mis conocimientos en el área de la ingeniería y la tecnología.

Finalmente, agradezco a mis docentes y compañeros de carrera, quienes con sus aportes, críticas constructivas y colaboración constante, contribuyeron de manera significativa al desarrollo y culminación de este proyecto, que representa no solo un logro académico, sino también un crecimiento personal.

RESUMEN

Este proyecto presenta el diseño e implementación de un aula virtual inmersiva desarrollada en Unity para dispositivos Meta Quest, con el objetivo de reproducir la experiencia de una clase presencial en un entorno tridimensional. El sistema integra un ambiente VR realista, una pizarra virtual para escritura colaborativa en tiempo real, un visor de presentaciones compatible con imágenes, archivos PDF y PPTX, y herramientas interactivas como un marcador y un borrador virtual. La integración en red se realizó mediante Photon Fusion 2.0, permitiendo sesiones multiusuario sincronizadas con baja latencia. Asimismo, se desarrolló un sistema completo de avatares con seguimiento de cabeza y manos mediante Meta XR SDK (v76) y AutoHand para interacciones físicas, junto con un chat de voz implementado con Photon Voice para comunicación en tiempo real. Un sistema de gestión de permisos incorporado en la interfaz de los avatares permite al anfitrión otorgar o revocar permisos de presentación y silenciar participantes. Esta solución se plantea como una alternativa escalable, interactiva y realista para la educación a distancia, fomentando la participación activa y la sensación de presencia en entornos de aprendizaje virtual.

Palabras clave: Realidad Virtual, Aprendizaje Inmersivo, Unity, Photon Fusion, Meta Quest, Aula Interactiva

ABSTRACT

This project presents the design and implementation of an immersive virtual classroom developed in Unity for Meta Quest devices, integrating multiple interactive tools to replicate the experience of an in-person class in a three-dimensional environment. The system includes a realistic VR environment, a whiteboard for real-time collaborative writing, a presentation viewer capable of displaying images, PDF, and PPTX files, and fully interactive tools such as a virtual marker and eraser. Network integration was achieved using Photon Fusion 2.0, enabling synchronized multi-user sessions with low latency. Additional modules were implemented, including a complete avatar system with head and hand tracking via the Meta XR SDK (v76) and AutoHand for physically-based interactions, as well as a voice chat system using Photon Voice for real-time communication. A permissions management system allows the host to grant or revoke presentation rights and mute participants via in-world UI elements embedded in avatars. The proposed solution offers a scalable, interactive, and realistic alternative for remote education, aiming to enhance engagement and presence in virtual learning spaces.

Keywords: Virtual Reality, Immersive Learning, Unity, Photon Fusion, Meta Quest, Interactive Classroom

1. CAPITULO 1

INTRODUCCIÓN

La incorporación de tecnologías digitales en la educación ha transformado el panorama formativo, dando paso a nuevas tendencias pedagógicas acordes al siglo XXI. En los últimos años se ha observado una evolución hacia modelos de aprendizaje más flexibles, híbridos y centrados en el estudiante, impulsada por herramientas innovadoras. Entre estas, las tecnologías inmersivas –principalmente la Realidad Virtual (RV) y la Realidad Aumentada (RA)– destacan como una de las principales tendencias tecnológicas en el ámbito educativo actual, al demostrarse su valor para apoyar y mejorar los procesos de enseñanza-aprendizaje. Estas tecnologías permiten reducir las limitaciones de tiempo, espacio y costo de la educación tradicional, a la vez que incrementan la motivación del estudiante y potencian un aprendizaje más experiencial.

Los entornos virtuales inmersivos juegan un rol clave en la educación contemporánea al brindar experiencias interactivas que simulan o amplían situaciones del mundo real. La Realidad Virtual, por ejemplo, permite conectar las aulas con entornos y escenarios difícilmente accesibles en la vida real, promoviendo un aprendizaje basado en la exploración y el descubrimiento en primera persona. Al sumergir al estudiante en espacios tridimensionales diseñados para la interacción, se potencia la construcción activa de conocimiento en lugar de la recepción pasiva de información. De este modo, tecnologías inmersivas como RV y RA están redefiniendo el e-learning tradicional, ofreciendo nuevas formas de participación y experimentación que antes no eran posibles. En este contexto, el desarrollo de un aula virtual inmersiva se plantea como una solución

innovadora para enriquecer la experiencia educativa, alineada con las tendencias tecnológicas actuales y las necesidades de las generaciones digitales.

1.1 PLANTEAMIENTO DEL PROBLEMA

A pesar del auge de la educación en línea, persisten problemas significativos de baja participación y desmotivación estudiantil en los entornos virtuales convencionales. En el contexto académico actual, muchas instituciones observan que sus estudiantes muestran un involucramiento limitado en las plataformas de e-learning: la interacción suele reducirse a ver contenidos en pantalla y responder cuestionarios, con escasas oportunidades para la colaboración activa o la experimentación. Esta experiencia unidireccional y poco interactiva puede propiciar que el alumno se convierta en un receptor pasivo, generando desinterés, falta de motivación y altos índices de deserción en los cursos virtuales.

El problema se manifiesta, por ejemplo, en baja participación en foros y clases virtuales en vivo, cumplimiento superficial de tareas en plataformas digitales y una sensación general de aislamiento entre los estudiantes. Las limitaciones del e-learning tradicional –como la falta de retroalimentación inmediata, la escasa inmersión sensorial y la poca variedad de actividades prácticas– dificultan mantener la atención y el interés a lo largo del tiempo.

1.2 JUSTIFICACIÓN

El desarrollo de un aula virtual inmersiva se justifica por la necesidad de mejorar los procesos de enseñanza-aprendizaje en educación superior, superando limitaciones de accesibilidad, motivación y permanencia que presentan tanto la educación presencial como las plataformas digitales convencionales. La incorporación de tecnologías de realidad virtual (VR) y sistemas multijugador permite recrear la presencialidad en un

entorno digital, ampliando las posibilidades de interacción, colaboración y experimentación.

Pedagógicamente, los entornos inmersivos fortalecen la presencia social, cognitiva y docente, factores clave para el aprendizaje significativo. A diferencia de una clase virtual tradicional, donde el estudiante suele adoptar un rol pasivo, el aula inmersiva fomenta la participación, el trabajo en equipo y la resolución conjunta de problemas, generando mayor compromiso y motivación intrínseca. Asimismo, facilita la simulación de actividades prácticas —como laboratorios o dinámicas de discusión— que enriquecen la formación teórica con experiencias aplicadas.

La pertinencia de este proyecto también radica en su accesibilidad y adaptabilidad. Al implementarse en dispositivos de VR autónomos, no requiere infraestructura compleja, lo que favorece su adopción en instituciones con recursos limitados. Su diseño modular y escalable permite ajustarse a distintas asignaturas y metodologías, convirtiéndolo en una herramienta versátil y sostenible.

Por otra parte, la pandemia de COVID-19 evidenció las limitaciones de las plataformas de videoconferencia, en las que la interacción suele ser reducida y el sentido de comunidad, débil. Este proyecto responde a esa necesidad al ofrecer un entorno en el que los estudiantes interactúan mediante avatares, comparten recursos, utilizan pizarras colaborativas y presentan trabajos en tiempo real, replicando la dinámica de un aula presencial. Con ello, no solo se fortalece el aprendizaje académico, sino también el desarrollo de habilidades blandas, como la comunicación, la colaboración y la gestión del tiempo.

1.3 METODOLOGÍA

Para el desarrollo del aula virtual inmersiva se siguió un enfoque metodológico integral, abarcando desde la investigación teórica inicial hasta la evaluación del prototipo implementado. A continuación, se describen de forma general las fases y actividades principales de la metodología empleada:

I. Investigación documental

Se inició el proyecto con una exhaustiva revisión de literatura y documentación relacionada con entornos virtuales de aprendizaje inmersivo. La búsqueda bibliográfica se realizó en Scopus, Web of Science, IEEE Xplore, ACM Digital Library, ERIC, SciELO y Redalyc (2018–2025).

Palabras clave (ES/EN): *aula virtual inmersiva / immersive virtual classroom*, *educación en realidad virtual / VR education*, *student engagement*, *retention*, *Unity*, *Meta Quest*, *Meta XR SDK*, *Photon/Fusion*, *collaborative whiteboard*.

Criterios de inclusión: estudios con evaluación de compromiso/aprendizaje, descripciones técnicas replicables o reportes de usabilidad en contextos educativos.

Exclusión: reseñas sin métricas, reportes sin detalle técnico y literatura no académica sin validación. Se priorizaron artículos revisados por pares y experiencias con análisis de resultados cuantitativos o mixtos.

II. Identificación de herramientas

Con los requerimientos claros, se procedió a investigar y seleccionar las herramientas de desarrollo adecuadas para construir el aula

inmersiva. El desarrollo del aula virtual inmersiva se apoyó en un conjunto de herramientas tecnológicas específicas, cada una con un aporte diferenciado:

- **Unity 2022.3 LTS:** motor de desarrollo principal para la creación del entorno 3D, iluminación y lógica de interacción.
- **Meta XR SDK (Core y Interaction):** integración nativa con dispositivos Meta Quest para captura de movimiento de cabeza y manos, así como soporte a interacciones inmersivas.
- **Photon Fusion 2.0.6:** sistema de red utilizado para la sincronización multijugador en tiempo real con baja latencia y transferencia de autoridad sobre objetos.
- **AutoHand:** librería para implementar interacciones físicas precisas (agarre de objetos, manipulación de marcadores, colisiones naturales).
- **Flask (backend en Python):** servicio de soporte para la gestión y conversión de presentaciones (PDF/PPTX) en imágenes compartidas entre usuarios.

Este ecosistema de herramientas permitió garantizar robustez en la simulación, estabilidad en la interacción multijugador y escalabilidad del sistema.

III. Implementación (desarrollo del aula inmersiva):

La implementación se llevó a cabo mediante una metodología iterativa incremental (SCRUM), que permitió dividir el desarrollo en fases cortas y verificables. Cada sprint tuvo una duración de dos semanas y contempló las siguientes actividades principales:

- **Modelado del entorno:** diseño del aula 3D con estructura arquitectónica, mobiliario y condiciones de iluminación.
- **Integración XR:** vinculación del entorno con el SDK de Meta XR para habilitar movimiento natural del avatar, detección de manos y uso de controladores.
- **Networking multijugador:** incorporación de Photon Fusion 2.0.6 para la creación y gestión de salas, sincronización de objetos interactivos y definición de roles (profesor/estudiante).
- **Pruebas de interacción:** validación del uso de AutoHand para manipulación de objetos, escritura en la pizarra y visualización de presentaciones compartidas.
- **Optimización:** ajustes en latencia, estabilidad de conexión y usabilidad del menú de acceso para asegurar una experiencia fluida en realidad virtual.

Esta estructura metodológica permitió avanzar de manera controlada, reduciendo redundancias y garantizando entregables funcionales en cada iteración.

1.4 OBJETIVO DE LA IMPLEMENTACIÓN

Objetivo general: Desarrollar un aula virtual inmersiva que incremente la interacción significativa, la colaboración y la motivación de los estudiantes de educación

superior mediante la integración de tecnologías de realidad virtual y sistemas multijugador en tiempo real.

Objetivos específicos:

- Integrar un entorno 3D interactivo con funciones de pizarra, presentaciones y avatares personalizados.
- Implementar un sistema multijugador con control de roles (profesor/estudiantes) que regule permisos y uso de herramientas.
- Validar la funcionalidad del prototipo a través de pruebas de usabilidad y estabilidad en escenarios simulados de clase.

2 CAPITULO 2

2.1 Antecedentes Académicos en Entornos Inmersivos Educativos

En la última década, la educación superior ha mostrado un creciente interés en las tecnologías inmersivas por su capacidad de generar experiencias de aprendizaje innovadoras e interactivas (Cabero-Almenara & Marín-Díaz, 2018). Estas herramientas permiten al estudiantado acceder virtualmente a contextos históricos, culturales o científicos de difícil acceso, potenciando la comprensión de los contenidos. Diversos estudios evidencian que, cuando se implementa de forma adecuada, la realidad virtual (RV) supera a las metodologías tradicionales en términos de resultados educativos (Hamilton et al., 2021). En este sentido, la adopción institucional avanza rápidamente: en 2019, se reportó que más del 90% de las universidades británicas ya utilizaban RV o realidad aumentada (RA), consolidándolas como tecnologías transformadoras en la enseñanza (Calderón Zambrano et al., 2023).

El alcance de estas aplicaciones ha trascendido las ciencias experimentales, incorporándose también en áreas como las ciencias sociales, humanidades y formación docente (Calderón Zambrano et al., 2023). Este crecimiento se confirma en estudios bibliométricos, que muestran un aumento sostenido en publicaciones sobre RV y motivación educativa durante las dos últimas décadas (Campos, Navas & Moreno, 2019). No obstante, persisten vacíos de investigación: Angulo Mendoza et al. (2023) destacan la falta de integración de marcos pedagógicos sólidos y la escasa aplicación en campos como las neurociencias. Esto evidencia la necesidad de avanzar hacia experiencias inmersivas que trasciendan la simulación y se fundamenten en enfoques educativos capaces de mejorar de manera efectiva el aprendizaje.

2.2 Fundamentos Teóricos del Aprendizaje con Tecnologías Inmersivas

A continuación, se presentan las corrientes teóricas más relevantes que sustentan las experiencias de aprendizaje inmersivo, las cuales enfatizan el papel activo, significativo y experiencial del estudiante en la construcción del conocimiento.

2.2.1 Constructivismo

El constructivismo postula que el aprendiente construye activamente nuevos conocimientos a partir de sus experiencias previas, en lugar de absorber información de forma pasiva. Esta teoría —con raíces en los trabajos de Jean Piaget (1970) y ampliada por enfoques socio-constructivistas de Vygotsky— sostiene que el conocimiento se construye cuando el estudiante interactúa con el entorno, manipula objetos, resuelve problemas y reflexiona sobre esas experiencias. Las tecnologías inmersivas se alinean de forma natural con el constructivismo, ya que crean entornos virtuales ricos donde el estudiante puede aprender haciendo ("learning by doing", en la visión de John Dewey).

Esto sugiere que educadores e investigadores reconocen el valor de entornos virtuales que fomentan una educación centrada en el alumno y activa, coherente con los principios constructivistas (Marougkas et al., 2023). Por ejemplo, en aplicaciones de RV constructivista, se observa que los estudiantes pueden resolver problemas y vivir simulaciones realistas en primera persona, lo cual les permite asimilar conceptos abstractos de manera concreta e interactiva (Le et al., 2015). Una aplicación concreta es el uso de mundos virtuales 3D para proyectos de construcción colaborativa del conocimiento, donde los estudiantes co-crean objetos o escenarios virtuales y aprenden en el proceso, reforzando tanto el constructivismo individual como el social.

Además, replicar interacciones sociales genuinas dentro de la RV puede ser un desafío (por ejemplo, la riqueza de la comunicación cara a cara), lo cual impacta en

dimensiones del constructivismo social. Aun así, estas tecnologías siguen ofreciendo posibilidades para interacciones colaborativas a distancia mediante avatares y espacios virtuales comunes, facilitando discusiones y construcción colectiva de conocimientos cuando se diseñan adecuadamente las actividades (social constructivism).

2.2.2 Aprendizaje Significativo

El concepto de aprendizaje significativo, introducido por David Ausubel (1963), plantea que la adquisición de nuevos conocimientos es efectiva cuando el estudiante logra anclar los contenidos nuevos a sus conocimientos previos de manera no arbitraria y sustancial. En el contexto de tecnologías inmersivas, la RV y la RA ofrecen oportunidades valiosas para fomentar este tipo de aprendizaje, al presentar contenidos educativos en contextos ricos, cercanos a la realidad o simulados con alta fidelidad que facilitan la conexión con experiencias previas o situaciones concretas.

Por ejemplo, una experiencia de RA puede superponer modelos tridimensionales interactivos sobre el entorno real del estudiante (laboratorio, aula, campo), permitiéndole relacionar teoría y práctica de forma inmediata. Cabero-Almenara y Marín-Díaz (2018) han destacado que muchas aplicaciones educativas de RA/RV incorporan estrategias como el aprendizaje basado en problemas y el aprendizaje colaborativo, creando escenarios donde los estudiantes deben aplicar contenidos a situaciones simuladas. Esto conduce a un aprendizaje más significativo, pues los alumnos entienden la relevancia de lo que aprenden al verlo aplicado en contextos prácticos.

Asimismo, las experiencias inmersivas suelen involucrar múltiples canales sensoriales (visual, auditivo e incluso háptico), lo cual puede aumentar la codificación cognitiva de la información. Kounlaxay et al. (2021) subrayan que la RA ofrece experiencias multisensoriales e incluso emocionales que facilitan la comprensión de

conceptos y contenidos en estudiantes universitarios. Un ejemplo práctico es el uso de la RV para enseñar historia: permitir al estudiante "vivir" una escena del pasado (como presenciar virtualmente un acontecimiento histórico) genera una conexión emocional e intelectual que difícilmente se lograría leyendo un texto tradicional, promoviendo así un aprendizaje significativo de los hechos históricos.

2.2.3 Aprendizaje Experiencial

El aprendizaje experiencial se basa en la premisa de que la experiencia directa es la base fundamental para adquirir y consolidar conocimientos, tal como lo propuso Kolb (1984) en su modelo de ciclo de aprendizaje experiencial (experiencia concreta, reflexión, conceptualización y aplicación activa).

Las aulas virtuales inmersivas ofrecen espacios de prueba y error donde los alumnos pueden tomar decisiones, observar consecuencias inmediatas y reflexionar sobre ellas, cerrando así el ciclo experiencial; esto le brinda una experiencia concreta sin riesgo real, tras la cual puede analizar su desempeño, corregir errores y mejorar sus habilidades antes de enfrentarse a pacientes reales. De esta manera, la RV encarna el laboratorio perfecto para el aprendizaje experiencial: inmersivo, interactivo y seguro.

Marougkas et al. (2023) identificaron que el aprendizaje experiencial es considerado uno de los enfoques más apropiados para la educación con RV. Esto concuerda con otros hallazgos que resaltan cómo la inmersión aumenta la retención y la comprensión al involucrar activamente al alumno en situaciones de la materia de estudio.

Además, el aprendizaje experiencial en RV/RA se vincula con un mayor grado de motivación e implicación del estudiante. Al sentir que realmente vive la situación de aprendizaje (presencia), el estudiantado suele mostrar mayor interés y proactividad. Un caso ilustrativo es el uso de entornos multiusuario 3D para proyectos de ingeniería:

estudiantes de arquitectura pueden construir conjuntamente prototipos en un mundo virtual, lo que no solo les permite aplicar principios de diseño (experiencia concreta) sino también discutir y reflexionar en equipo sobre el proyecto (observación reflexiva), generando un ciclo de aprendizaje completo.

2.2.4 Otras Teorías Relevantes

Además de las anteriores, existen otras teorías y enfoques educativos que complementan el marco teórico de las tecnologías inmersivas. Por ejemplo, la Teoría Cognitiva de la Carga (Sweller) advierte sobre la necesidad de diseñar las experiencias VR/AR de modo que no sobrecarguen la memoria de trabajo del aprendiz. De igual forma, la Teoría del Flujo (Csikszentmihalyi) resulta pertinente: un buen diseño inmersivo busca que el estudiante entre en un estado de flujo, altamente concentrado y motivado, equilibrando los desafíos de la tarea con las habilidades del usuario. Esto se ha aplicado en la gamificación de entornos educativos VR, donde elementos de juego (retos, recompensas, narrativa) se integran para aumentar el engagement.

Otro marco relevante es el modelo TPACK (Conocimiento Tecnológico-Pedagógico del Contenido), que orienta a los docentes en la integración efectiva de nuevas tecnologías (como RV/RA) en sus estrategias didácticas, combinando saber disciplinar, pedagógico y técnico. Aplicado a aulas virtuales inmersivas, TPACK enfatiza que el simple hecho de usar RV no garantiza aprendizajes; es necesaria una convergencia entre el contenido curricular, la metodología apropiada (p. ej. colaboración, investigación guiada) y las características de la tecnología inmersiva para lograr resultados óptimos.

Finalmente, vale mencionar la presencia social y la telepresencia como constructos teóricos específicos de entornos virtuales. Según la teoría de presencia, el aprendizaje en RV es efectivo en la medida que el estudiante llega a sentir que “está

realmente ahí" en el entorno simulado, lo cual incrementa su involucramiento emocional y cognitivo. Esto sustenta el diseño de aulas virtuales multiusuario donde se promueve la interacción entre pares mediante avatares y comunicación en tiempo real, basándose en el entendimiento de que la dimensión social del aprendizaje sigue siendo crucial incluso en mundos virtuales.

2.3 Revisión de Tecnologías para Aulas Virtuales Inmersivas

A continuación, se presenta una revisión comparativa de las principales tecnologías empleadas en el desarrollo de aulas virtuales inmersivas, destacando sus características, ventajas y casos de uso en el ámbito educativo.

2.3.1 Motores de desarrollo 3D (Game Engines)

Dos de las herramientas más utilizadas para crear entornos virtuales interactivos son *Unity* y *Unreal Engine*. Ambos motores, originarios de la industria de los videojuegos, se han adaptado plenamente para aplicaciones de RV/RA y ofrecen potentes funcionalidades gráficas y de simulación.

2.3.1.1 Unity

Es reconocido por su versatilidad y facilidad de uso. Unity permite desarrollar aplicaciones multiplataforma, incluyendo PC, dispositivos móviles y cascos de RV, con relativa facilidad. Utiliza el lenguaje C# y cuenta con una interfaz intuitiva, lo que lo hace amigable para principiantes en desarrollo 3D. En el contexto educativo, Unity ha sido popular para prototipar laboratorios virtuales, simulaciones interactivas y experiencias de RA móvil debido a su amplia comunidad de desarrolladores y a la gran cantidad de recursos disponibles (Asset Store, plugins educativos, etc.). Entre sus ventajas están la enorme comunidad de soporte, abundantes tutoriales y la disponibilidad de versiones gratuitas o de bajo costo para instituciones académicas. Como posible desventaja, se

señala que aunque Unity ha mejorado su calidad gráfica, puede quedar atrás en realismo visual frente a otros motores en proyectos de máxima exigencia gráfica.

2.3.1.2 Unreal Engine

Desarrollado por Epic Games, Unreal Engine es célebre por lograr gráficos de altísima fidelidad y efectos visuales realistas. Emplea C++ como lenguaje principal, pero ofrece Blueprints, un sistema visual de scripting que facilita el desarrollo sin necesidad de programar en código, lo cual puede ser útil para educadores desarrolladores con menos experiencia en programación. Unreal es ideal para proyectos que requieran entornos virtuales muy detallados, con iluminación y físicas avanzadas – por ejemplo, simulaciones de ingeniería complejas o visualizaciones arquitectónicas inmersivas de gran realismo. En el ámbito educativo, aunque su curva de aprendizaje es más pronunciada que Unity (especialmente por su complejidad y requerimientos técnicos), Unreal ha sido utilizado para crear experiencias inmersivas ricas, como laboratorios de química virtual con reacciones visualmente impactantes o recorridos virtuales de patrimonio cultural con calidad casi cinematográfica. En comparación, Unreal tiende a exigir equipos de desarrollo y de ejecución más potentes, dado su motor gráfico avanzado. Sin embargo, soporta igualmente múltiples plataformas (PC, móviles de gama alta, dispositivos VR) y se ha vuelto más accesible en años recientes gracias a abundante documentación y a su licencia gratuita (solo cobra regalías en proyectos comerciales de gran envergadura).

2.3.2 Plataformas de aulas virtuales inmersivas

Además de desarrollar un entorno desde cero con motores, existen plataformas ya diseñadas que facilitan la creación de aulas virtuales y experiencias educativas inmersivas. Entre las más destacadas se encuentran:

2.3.2.1 Mozilla Hubs

Es una plataforma de realidad virtual social basada en la web, de código abierto, que permite a usuarios reunirse en salas virtuales 3D accesibles simplemente mediante un enlace URL. Mozilla Hubs se ha utilizado en educación por su facilidad de acceso, ya que los estudiantes pueden unirse desde distintos dispositivos (PC, móvil o visores VR) sin instalar software especializado, usando solo el navegador web (compatibilidad WebXR). Una ventaja clave es que Hubs es gratuito y permite albergar a un grupo completo de clase en un mismo espacio virtual, integrando voces y chat para la interacción en tiempo real. Docentes han empleado Mozilla Hubs para actividades como oficinas virtuales, presentaciones de proyectos en entornos inmersivos, e incluso laboratorios remotos donde los alumnos y el profesor interactúan con modelos 3D durante una sesión en línea. Estudios de caso (Huisinga, 2023) señalan que Hubs ofrece un ambiente colaborativo que puentea la brecha entre las videoconferencias tradicionales y los videojuegos, brindando una experiencia más participativa y lúdica para clases en línea.

2.3.2.2 Engage

Es una plataforma comercial de entornos virtuales multiusuario enfocada en educación y capacitación. Engage XR se destaca por proveer un ecosistema completo para clases inmersivas: permite conectar decenas de usuarios simultáneamente en espacios 3D variados (aulas, auditorios, entornos externos), con herramientas integradas como pizarras virtuales, capacidad de compartir presentaciones, videos 360°, modelos 3D y hasta un navegador web dentro del mundo virtual. Una de sus fortalezas es la seguridad y estabilidad a nivel empresarial; Engage es utilizada por universidades y corporaciones, ofrece control de usuarios, espacios privados y cumple con estándares de seguridad (ISO 27001, GDPR) para entornos educativos seguros. Desde el punto de vista pedagógico,

Engage ha sido el medio para tours educativos virtuales, simulaciones de ciencia (por ejemplo, experimentos en un laboratorio virtual) y clases magistrales donde estudiantes de diferentes lugares asisten con sus avatares a un aula común. El contenido en Engage puede crearse sin programación gracias a editores visuales (*no-code content editor*), facilitando a los docentes generar actividades interactivas. Además, la plataforma incorpora elementos modernos como avatares impulsados por IA y herramientas de creación con inteligencia artificial, lo que abre posibilidades para tutorías virtuales automatizadas u objetos de aprendizaje generados dinámicamente.

2.3.2.3 ClassVR

A diferencia de las anteriores, ClassVR no es solo una plataforma de software sino una solución integral de hardware y contenido diseñada especialmente para entornos escolares. Desarrollada por Avantis Education, ClassVR ofrece gafas de RV autónomas para el aula, junto con un portal web para el profesor que permite gestionar de forma sincronizada la experiencia de toda la clase. Esta solución "todo en uno" incluye una biblioteca de recursos educativos alineados con currículos (experiencias VR/AR en historia, ciencias, arte, etc.) y herramientas para que el docente controle lo que ven los alumnos, lance actividades y supervise el progreso en tiempo real. ClassVR se ha implementado sobre todo en educación primaria y secundaria, pero también existen experiencias en niveles superiores para introducir conceptos complejos de manera visual. Entre sus ventajas, la compañía destaca mejoras en la atención y retención: según su sitio, los estudiantes retienen hasta un 75% de lo que aprenden mediante experiencias inmersivas (frente a porcentajes mucho menores con métodos tradicionales), y se han observado aumentos de hasta un 20% en resultados de exámenes al integrar RV en las clases. Un estudio internacional reciente respaldó algunas de estas afirmaciones,

encontrando que el uso de ClassVR en escuelas de El Salvador aumentó la retención de conocimiento en 6.7% y la motivación en 23% comparado con grupos de control sin RV. Estas cifras sugieren que, implementada adecuadamente, la RV puede tener un impacto tangible en el desempeño académico. La fortaleza de ClassVR radica en su diseño específico para educadores: su interfaz sencilla de manejar en el aula y los contenidos ya preparados ahorran al docente la complejidad técnica. Como limitación, la adopción de esta solución requiere una inversión significativa en equipos (set de visores para la clase) y capacitación docente, además de un soporte continuo para actualización de contenidos.

2.3.3 Herramientas de desarrollo XR y otras tecnologías

Complementando a los motores y plataformas mencionados, existen herramientas y estándares que habilitan la creación de contenido inmersivo. Por ejemplo, WebXR es la API que permite experiencias de RV y RA directamente en navegadores web, fundamento de plataformas como Mozilla Hubs y otras (como FrameVR o AltspaceVR en su momento). Para RA en dispositivos móviles, kits de desarrollo como ARKit (Apple) y ARCore (Google) proporcionan funciones de rastreo de entorno, detección de planos y colocación de objetos virtuales en el mundo real, lo cual ha dado pie a numerosas aplicaciones educativas de RA (por ejemplo, visualizar moléculas en 3D sobre la mesa, o hacer aparecer elementos históricos en el campus). También destacan herramientas especializadas como Vuforia, AR.js o Unity Mars, que facilitan la creación de aplicaciones AR con reconocimiento de imágenes y objetos, útiles para materiales didácticos aumentados.

En cuanto a hardware, además de los cascos de RV (Oculus/Meta Quest, HTC Vive, etc.), han surgido dispositivos de Realidad Mixta como Microsoft Hololens que permiten integrar hologramas en entornos reales.

2.4 Ventajas y Limitaciones de las Tecnologías Inmersivas en Educación Superior

A continuación, se presentan de forma estructurada las principales ventajas y limitaciones del empleo de tecnologías inmersivas en la educación superior, respaldadas por hallazgos de investigaciones recientes:

2.4.1 Ventajas potenciales

- Aumento de la motivación y el compromiso: Los entornos inmersivos suelen ser altamente atractivos para los estudiantes, fomentando una participación activa. Un gran porcentaje de docentes (78%) ha observado mayor motivación estudiantil al usar soluciones de aprendizaje aumentadas/virtuales en el aula. Estudios bibliométricos señalan que el interés por RV en educación está ligado a su capacidad para incrementar la motivación intrínseca del alumno mediante actividades lúdicas e interactivas (Campos et al., 2019).

- Aprendizaje experiencial seguro: La RV permite ensayar tareas peligrosas o complejas en un ambiente seguro. Por ejemplo, estudiantes de medicina pueden practicar cirugías virtuales o procedimientos clínicos en pacientes simulados, sin riesgos reales para pacientes o aprendices. Esto mejora su destreza y confianza antes de enfrentarse al mundo real. De igual forma, en ingeniería o ciencias, los laboratorios virtuales ofrecen experimentación con reactivos o maquinaria virtual, evitando costos y peligros físicos.

- Comprensión de conceptos abstractos: Las visualizaciones 3D inmersivas ayudan a los estudiantes a entender mejor conceptos difíciles o abstractos al poder manipular y observar fenómenos de forma interactiva. Por ejemplo, en química los alumnos pueden visualizar moléculas en 3D y sus interacciones; en física, experimentar con leyes de movimiento en entornos virtuales. Esto facilita un aprendizaje significativo

al vincular la teoría con representaciones visuales y experiencias concretas (Kounlaxay et al., 2021).

- Mayor retención de conocimiento: Varias investigaciones sugieren que los estudiantes recuerdan más lo aprendido en contextos inmersivos en comparación con clases tradicionales. Una revisión meta-analítica halló efectos positivos de la RV en la retención y transferencia del aprendizaje (Moro et al., 2021). En un estudio controlado, el uso de ClassVR aumentó la retención en casi 7% frente al grupo tradicional. Esto se atribuye a que la inmersión involucra múltiples sentidos y emociones, creando memorias de aprendizaje más vívidas.

- Desarrollo de habilidades prácticas y colaborativas: Las aulas virtuales ofrecen oportunidades de aprendizaje colaborativo a distancia, donde estudiantes ubicados en diferentes lugares pueden trabajar juntos en tiempo real resolviendo problemas o completando proyectos (Dunleavy et al., 2009). Asimismo, las simulaciones permiten adquirir habilidades procedimentales (por ej., operar un equipo, realizar una práctica de laboratorio) que son directamente aplicables luego en entornos reales, mejorando la preparación profesional del alumnado.

2.4.2 Limitaciones y desafíos

- Requerimientos tecnológicos y brecha de acceso: La implementación de RV/RA a escala educativa puede verse limitada por los recursos. Los visores de RV de calidad y las computadoras capaces de ejecutarlos suponen una inversión elevada. Si bien hay opciones más económicas (cardboards, visores móviles), la experiencia puede ser inferior. Esto puede agravar la brecha digital entre instituciones o estudiantes que cuentan con la tecnología y los que no (Montenegro-Rueda & Fernández-Cerero, 2022). Además, incluso con dispositivos disponibles, factores como conexiones de internet lentas o

hardware obsoleto pueden dificultar el uso fluido de entornos inmersivos. En la práctica, se han observado problemas de acceso y usabilidad que afectan la equidad: no todos los estudiantes poseen un visor personal, y largos periodos frente a estas tecnologías en remoto pueden excluir a quienes no tienen las condiciones adecuadas en casa.

- Curva de aprendizaje y capacitación docente: Incorporar RV/RA en la enseñanza no solo implica dotar de dispositivos, sino también formar a los profesores en su uso pedagógico. Muchos docentes carecen de experiencia en estas tecnologías y pueden sentirse abrumados ante la idea de diseñar o guiar una clase inmersiva (Felkel & Dickmann, 2022). Se requieren capacitaciones específicas para que el profesorado desarrolle competencias digitales avanzadas y conozca las metodologías didácticas apropiadas para aprovechar la inmersión (por ejemplo, cómo moderar una discusión en VR, cómo evaluar aprendizajes en esos entornos, etc.). La resistencia al cambio o la falta de tiempo para entrenamiento son barreras frecuentes reportadas en la literatura.

- Sobrecarga cognitiva y distracciones: Un entorno virtual rico en estímulos puede, si no está bien diseñado, sobrecargar la atención del estudiante. Existe el riesgo de que el alumno se distraiga explorando elementos irrelevantes del mundo virtual o que la complejidad de la interfaz le impide centrarse en los objetivos de aprendizaje. Dunleavy, Dede y Mitchell (2009) señalaron que en simulaciones AR participativas, a veces los estudiantes se enfocan más en la tecnología o el juego que en los contenidos curriculares, diluyendo el propósito educativo. Por ello, es un desafío diseñar experiencias equilibradas que mantengan el foco pedagógico; de lo contrario, la tecnología inmersiva podría convertirse en un espectáculo superficial con poco beneficio educativo real.

- Problemas de comodidad física y salud: El uso prolongado de visores de RV puede provocar incomodidad, fatiga visual, e incluso ciber-cinetosis (mareo por

simulación). Algunos estudiantes experimentan náuseas, desorientación o dolor de cabeza tras sesiones extendidas en RV, especialmente si la frecuencia de cuadro o el diseño visual no son óptimos. Esto limita las sesiones recomendables de RV continua (suele sugerirse no más de 20-30 minutos sin descanso). También hay cuestiones ergonómicas: usar gafas voluminosas puede ser molesto y dificulta la toma de notas u otras tareas simultáneas.

En RA móvil, mantener dispositivos en alto mucho tiempo cansa los brazos y la vista. Además, se deben considerar medidas de seguridad física en entornos VR para evitar choques con objetos reales al moverse con el visor puesto. Todos estos factores representan desafíos para la comodidad y seguridad del estudiante, que los docentes deben tener en cuenta al planificar las actividades.

- Limitaciones en la interacción social real: Aunque las plataformas virtuales ofrecen interacciones mediante avatares, no reemplazan completamente la riqueza de la comunicación cara a cara. La ausencia de lenguaje corporal completo, de contacto visual directo o de espontaneidad en las conversaciones puede dificultar algunos aspectos del aprendizaje colaborativo o de la construcción de relaciones alumno-docente. Por ejemplo, en un aula VR los estudiantes podrían sentirse menos propensos a intervenir que en persona, o podrían ocurrir malentendidos por la falta de matices en la voz o expresiones faciales (según social presence theory).

3. CAPITULO 3

Especificación de Requerimientos y Planificación del Proyecto

3.1 Especificación de requerimientos

Este proyecto, orientado a plataformas Meta Quest, despliega una solución compacta y personalizable que combina avatares, pizarra y presentaciones en un mismo

espacio. A continuación se detallan los elementos que definen su estructura y comportamiento.

3.1.1 Requerimientos funcionales

Los requerimientos funcionales definen las capacidades y funciones específicas que el aula virtual inmersiva debe proporcionar al usuario. Estos requisitos se han identificado considerando la experiencia educativa a replicar y las características propias de la plataforma VR. Cada requerimiento funcional se describe con detalle técnico, indicando su estado de implementación actual (si procede) y decisiones de diseño relevantes.

RF01: Entorno de Aula Virtual Inmersiva. El sistema deberá proveer un entorno tridimensional que recree un aula de clase, incluyendo elementos habituales como una pizarra, un área frontal de exposición (por ejemplo, una pantalla o superficie para proyectar presentaciones) y un espacio para que los avatares de los estudiantes y docentes se ubiquen. Este entorno actuará como escenario principal de interacción, brindando a los usuarios la sensación de estar compartiendo un mismo espacio físico aunque se encuentren remotamente. El diseño del aula virtual debe ser coherente y realista, de manera que contribuya a la inmersión del usuario y facilite la orientación espacial dentro del entorno.

RF02: Soporte Multiusuario con Avatares en Tiempo Real. La aplicación deberá permitir la participación simultánea de múltiples usuarios en una misma sesión virtual. Cada usuario estará representado por un avatar en 3D que refleje su presencia dentro del aula virtual, mostrando al menos su posición, orientación y movimientos básicos (cabeza y manos) en tiempo real a los demás participantes. De este modo, estudiantes y docentes podrán verse e identificarse mutuamente en el espacio virtual, favoreciendo la comunicación no verbal (por ejemplo, el docente puede observar la

ubicación o gestos de un estudiante a través de su avatar). El sistema debe gestionar la sincronización de estado entre todos los clientes de manera eficiente, de forma que las acciones de cada usuario (movimiento, interacciones con objetos, etc.) se reproducen con mínima latencia en los visores del resto, garantizando una experiencia colaborativa coherente.

RF03: Locomoción Virtual Libre mediante Controladores. Se deberá implementar un mecanismo de locomoción virtual que permita a los usuarios desplazarse libremente por el aula inmersiva usando los controladores de movimiento VR. En particular, el usuario podrá mover su avatar usando el joystick del controlador (movimiento continuo en las direcciones deseadas), emulando el acto de caminar por el salón. Esta funcionalidad debe ofrecer movimientos suaves y controlados, con una velocidad apropiada, para que el usuario pueda recorrer distintos puntos del aula (por ejemplo, acercarse a la pizarra o desplazarse entre los asientos). Adicionalmente, aunque se priorizará el movimiento continuo por su naturalidad, el diseño contempla opciones de seguridad y confort (como límites de velocidad, zonas de navegación permitidas e incluso un modo de teletransporte opcional) para minimizar el riesgo de desorientación o mareo en usuarios sensibles. La locomoción virtual es fundamental para otorgar a los participantes libertad de exploración dentro del entorno educativo, similar a un aula real donde uno puede moverse para interactuar más de cerca con ciertos elementos.

RF04: Pizarra Virtual Interactiva. El sistema deberá contar con una pizarra virtual funcional sobre la cual los usuarios autorizados (principalmente el docente) puedan escribir o dibujar en tiempo real, utilizando un objeto puntero o marcador virtual. Esta pizarra servirá como medio para ilustrar conceptos durante la clase, tal como se hace con una pizarra física. La interacción consistirá en permitir que el docente trace líneas o texto sobre la superficie virtual, y que dichos trazos aparezcan de inmediato en la pizarra

de todos los participantes. Para ello, el módulo correspondiente detectará la posición y orientación del marcador virtual en la mano del usuario y renderiza trazos continuos allí donde haga contacto con la superficie de la pizarra, simulando la escritura con tiza o plumón. Es necesario que la pizarra sea visible para todos en el aula y que su contenido permanezca sincronizado en red, de modo que todos vean las anotaciones al unísono. Además, se podría permitir que los estudiantes también interactúen con la pizarra (por ejemplo, para resolver un ejercicio) siempre bajo control o permiso del docente, lo que implica mecanismos para la gestión de turnos o permisos sobre esta herramienta.

RF05: Presentación de Contenido Multimedia. El aula virtual deberá brindar la capacidad de proyectar presentaciones y otros materiales multimedia para complementar la clase, de forma análoga a un proyector o pantalla en un aula convencional. Es necesario que el docente pueda cargar o seleccionar un archivo de presentación (por ejemplo, diapositivas de PowerPoint, imágenes estáticas, o PDF) y que el sistema muestre dichas diapositivas dentro del entorno 3D para todos los participantes. El contenido podría desplegarse en una pantalla virtual o superficie dentro del aula, visible a todos los avatares desde sus posiciones. El sistema debe permitir al docente navegar entre las diapositivas (avanzar, retroceder, cambiar de página) con controles simples, y garantizar que el cambio se refleja inmediatamente para todos los estudiantes. Se contemplan diferentes enfoques técnicos para implementar esta característica, como la conversión de las diapositivas en texturas o imágenes secuenciales que el motor Unity pueda presentar, o incluso la integración de un navegador web interno (WebView) para mostrar contenido en vivo desde la web. En cualquier caso, la solución seleccionada deberá asegurar buena calidad visual y rendimiento, de modo que las letras y gráficos de las presentaciones sean legibles en VR y la carga de cada nuevo slide sea rápida. Este requerimiento también implica ofrecer soporte a formatos comunes de material educativo (al menos imágenes y quizás

PDFs o PPTX) o proporcionar herramientas para convertirlos previamente a un formato compatible. La funcionalidad de presentaciones en el entorno inmersivo es esencial para permitir a los docentes compartir material didáctico estructurado durante la sesión, emulando las clases magistrales apoyadas en diapositivas.

RF06: Comunicación de Voz en Tiempo Real. El sistema debe incorporar comunicación de audio en tiempo real entre los participantes. Esto significa que los usuarios (docente y estudiantes) podrán hablar usando los micrófonos de sus visores o dispositivos, y sus voces se transmitirán dentro del aula virtual de forma inmediata y bidireccional, permitiendo conversaciones naturales. Idealmente, la solución implementará audio posicional (spatial audio) de modo que la voz de cada avatar suene provenir de la dirección y distancia correctas en el entorno virtual, aumentando la inmersión (por ejemplo, si un estudiante-avatar habla desde el fondo de la sala, el docente lo escuchará más bajo y desde esa ubicación relativa). Este requerimiento es crítico para facilitar preguntas, discusiones y explicaciones orales durante la clase, tal como ocurriría en persona. Además, deberán considerarse controles básicos asociados a la voz, como la posibilidad de silenciar a los participantes (por el docente o por ellos mismos, para evitar interrupciones o ruido de fondo) y gestionar la calidad del audio para que la comunicación sea clara con la mínima latencia posible. La incorporación de voz en tiempo real implica integrar servicios o APIs de audio sobre la capa de red existente, garantizando seguridad (por ejemplo, que la voz solo sea oída dentro de la sesión correspondiente) y un consumo eficiente del ancho de banda.

RF07: Interacción y Manipulación de Objetos Virtuales. El sistema deberá permitir a los usuarios interactuar con objetos virtuales dentro del aula, de forma intuitiva mediante sus controladores de realidad virtual. Esto abarca funcionalidades como tomar y soltar objetos (por ejemplo, coger el marcador virtual para escribir en la pizarra, o

apuntar con un puntero láser virtual a elementos de una diapositiva), así como pulsar botones o controles de la interfaz virtual (menús flotantes, controles de navegación de diapositivas, etc.). La interacción estará basada en modelos de física e interfaz 3D: el usuario podrá estirar la mano virtual (representada por sus controladores o manos animadas) y tocar o agarrar elementos interactivos. El sistema debe detectar estas colisiones o proximidad y responder adecuadamente (por ejemplo, permitir agarrar el marcador y empezar a dibujar al entrar en contacto con la pizarra). Se asegurará que los objetos importantes (como el marcador, borrador, controles de presentación) sean destacados y accesibles en el entorno, y que solo los usuarios autorizados puedan usarlos cuando corresponda (ligado a RF09 de roles/permiso).

RF08: Gestión de Roles y Permisos (Docente, Estudiante, Administrador). La plataforma deberá distinguir entre diferentes roles de usuario, principalmente docente, estudiante y administrador, aplicando restricciones o privilegios específicos según el rol para asegurar un funcionamiento ordenado del aula virtual. El docente será el conductor de la sesión: tendrá permisos para utilizar la pizarra (RF04) y ser el principal en escribir en ella, controlar la presentación de diapositivas (RF05), hablar de forma preferente (por ejemplo, podría haber una función de push-to-talk prioritario o moderación de audio) y en general dirigir la clase. Los estudiantes, por su parte, tendrán un rol principalmente de participantes que pueden observar los contenidos (pizarra, presentaciones), desplazarse libremente (RF03) y comunicarse vía voz (RF06), pero con capacidades restringidas para no interrumpir inadecuadamente: por ejemplo, típicamente no podrán avanzar las diapositivas por sí mismos ni escribir en la pizarra sin autorización. El administrador representará un rol de gestión del sistema a nivel global: este usuario (o grupo de usuarios) podrá configurar ciertos aspectos técnicos y operativos de la plataforma, como la creación de salas o sesiones, la asignación de cuentas de docente, la supervisión de la

infraestructura de red, y potencialmente acciones de moderación más elevadas (expulsar usuarios indebidos, reiniciar la sala, etc.). El sistema deberá implementar controles que verifiquen el rol de cada usuario antes de ejecutar acciones sensibles (por ejemplo, solo permitir la carga de una nueva presentación si el comando proviene de un cliente con rol docente). La gestión de roles y permisos garantiza un orden estructurado en el entorno virtual, reflejando la jerarquía típica de un entorno educativo real (donde el profesor dirige y los estudiantes participan siguiendo ciertas normas), y previniendo usos indebidos o conflictos de control durante la interacción multiusuario.

RF09: Administración de la Plataforma Educativa Virtual. Además de las funciones dentro de la experiencia VR en sí, se deberá contemplar la existencia de una interfaz de administración de la plataforma, accesible para usuarios con rol administrador, para realizar tareas de mantenimiento y gestión. Esto incluye la capacidad de gestionar usuarios y credenciales (por ejemplo, registrar nuevos docentes o estudiantes, asignar roles, resetear contraseñas si las hubiera), crear o programar sesiones de clase virtual (generar "aulas" virtuales específicas a las que luego los usuarios se conectarán, posiblemente con un código de acceso), y monitorizar el sistema (ver estadísticas de uso, número de usuarios conectados, estado del servidor, etc.). Si bien muchas de estas tareas podrían realizarse fuera del casco de realidad virtual (por ejemplo, mediante una interfaz web o aplicación 2D para administradores), deben ser diseñadas como parte del sistema global. Un administrador podría, por ejemplo, iniciar o cerrar una sesión de aula virtual, asignar un docente a una clase, o expulsar a un usuario disruptivo. Cabe aclarar que la interfaz de administración debe ser segura y accesible solo para personal autorizado, dado que otorga poderes amplios sobre la plataforma.

3.1.2 Requerimientos no funcionales

A continuación, se listan los principales requerimientos no funcionales identificados:

RNF01: Compatibilidad Multiplataforma. El sistema debe ser ejecutable en visores Meta Quest 2/3 de forma nativa y, de manera opcional, en PC VR mediante Oculus Link, ALVR o SteamVR. Para garantizar portabilidad y flexibilidad, se empleará Unity con OpenXR, lo que también permite extender la experiencia a versiones de escritorio 2D como observador.

RNF02: Rendimiento y Fluidez Gráfica. La aplicación deberá mantener una tasa mínima de 72 FPS en Meta Quest, optimizando modelos, texturas y entornos con técnicas como LOD, occlusion culling y baked lighting. Además, se requiere que la lógica de red y físicas esté optimizada para evitar bloqueos, asegurando fluidez incluso con múltiples usuarios y elementos interactivos.

RNF03: Baja Latencia y Sincronización en Red. Todas las interacciones compartidas (movimientos, voz, escritura o presentaciones) deben transmitirse en tiempo real con latencias mínimas, garantizando que todos los clientes vean el mismo estado. El sistema debe tolerar variaciones de red y aplicar técnicas como interpolación, buffers mínimos de audio y envío en delta para mantener la consistencia.

RNF04: Escalabilidad de Usuarios Concurrentes. El diseño debe soportar entre 10 y 20 usuarios conectados por sesión, con capacidad de mantener la estabilidad en picos de carga. La arquitectura debe ser escalable mediante instancias adicionales o salas paralelas, evitando límites estrictos que comprometan la participación en clases de mayor tamaño.

RNF05: Usabilidad y Experiencia de Usuario. La interfaz debe ser intuitiva y de fácil aprendizaje, utilizando metáforas familiares como agarrar objetos o presionar botones virtuales. Los menús serán minimalistas y contextuales, con tutoriales in-VR y retroalimentación visual o auditiva. También se considerarán aspectos de comodidad y accesibilidad, como ajustes de altura, soporte para manos diestras o zurdas y opciones multilingües.

RNF06: Inmersión y Confort del Usuario. El sistema debe ofrecer una sensación de presencia realista mediante gráficos y audio adecuados, manteniendo la ergonomía de los objetos y escenarios. Para reducir la ciber-enfermedad, se implementarán opciones como viñeteado, aceleración suave y teletransporte alternativo. Asimismo, se cuidará la calidad visual para evitar la fatiga ocular durante sesiones prolongadas.

RNF07: Seguridad y Control de Acceso. La aplicación debe garantizar que solo usuarios autorizados ingresen a las sesiones mediante autenticación con credenciales o códigos de acceso. La comunicación de datos se manejará por canales seguros, validando la integridad de la información. Además, se incluirán herramientas de moderación que permitan al docente o administrador silenciar o expulsar participantes problemáticos.

3.2 Actores del sistema

Tabla 1

Actores del sistema

Actor	*Descripción / Objetivos en el sistema*
Docente	Profesor o instructor que dirige la clase. Inicia la sesión, carga y muestra presentaciones, escribe en la pizarra virtual, se desplaza por el aula, supervisa/interactúa con los estudiantes y controla funciones especiales como cambiar diapositivas o borrar la pizarra.
Estudiante	Alumno participante. Se desplaza por el aula, observa la presentación y lo escrito en la pizarra. Actúa principalmente como observador, pero puede interactuar escribiendo si el docente le concede permisos (ejercicios, participación activa).
Sistema/Plataforma	Infraestructura de VR y red. Gestiona conexiones de múltiples usuarios, sincroniza pizarra y presentaciones, y asegura que las acciones del docente se propaguen correctamente a todos los estudiantes. No es un usuario humano, pero actúa de soporte automático.

3.3 Casos de uso

Tabla 2

Casos de uso

CU	**Actor(es)**	**Descripción / Objetivo**
CU1: Iniciar Clase Virtual	Docente	El docente inicia la aplicación y crea una sesión. Configura el entorno (presentación, pizarra limpia). En versión multiusuario, implica crear un "room" de red.
CU2: Unirse a Clase Virtual	Estudiante	El estudiante ejecuta la aplicación y se conecta a la sesión del docente. Controla su avatar desde el punto de spawn. En prototipo single-player, se simula iniciando la escena VR.
CU3: Desplazarse por el Aula	Docente / Estudiante	Los usuarios se mueven por el aula mediante joystick. El sistema actualiza posiciones, evita atravesar paredes/pisos y permite cambiar de ubicación para observar mejor la pizarra o presentaciones.
CU4: Escribir en la Pizarra	Docente (y Estudiante con permiso)	El actor toma un marcador virtual y escribe/dibuja en la pizarra, con trazos en tiempo real. Puede borrar contenido con borrador virtual o botón. En multiusuario, la acción se replica a todos.

CU5: Presentar Diapositivas	Docente	El docente carga un archivo (PPTX/PDF/imagenes) o URL de presentación. Avanza/retrocede diapositivas mediante controles. Todos los usuarios ven la diapositiva actual. Se contemplan errores de carga.
CU6: Interacción y Participación	Estudiante	Los estudiantes pueden levantar mano virtual, hablar (si hay chat de voz) o resolver ejercicios en la pizarra cuando el docente lo autoriza. Actualmente limitado a moverse y observar.
CU7: Finalizar la Sesión	Docente	El docente termina la clase. En multiusuario, cierra la sala y desconecta participantes. Opcionalmente, puede guardar estado de la pizarra o registro de la sesión antes de volver al menú.

3.4 Comparativa de tecnologías y enfoques considerados

La selección final de cada tecnología respondió a criterios de compatibilidad con dispositivos Meta Quest, estabilidad, rendimiento, facilidad de mantenimiento y alineación con los objetivos educativos del proyecto. A continuación se describen tres decisiones clave donde se evaluaron alternativas específicas, incluyendo aquellas que fueron descartadas, junto con su justificación técnica.

Manejo de presentaciones: Se compararon dos enfoques. La primera opción contemplaba la integración de un WebView, es decir, incrustar un navegador web dentro

de Unity que pudiera mostrar contenidos en línea como documentos de Google Slides o visores web de PDF. En segundo lugar, depender de una conexión externa para cargar las presentaciones introduce riesgos de latencia y fallos si la conectividad no es estable, lo cual contradice el requisito de funcionamiento fluido en entornos educativos. Por estas razones, se optó por desarrollar un sistema nativo en Unity que permita al docente cargar archivos locales (PDF, PPTX o imágenes), convertirlos en recursos gráficos optimizados para realidad virtual y desplegarlos como texturas sobre superficies virtuales.

Arquitectura de red multiusuario: Se opto por utilizar una solución de red en la nube ya probada, como Photon (Photon Fusion y Photon Voice), la cual provee sincronización de objetos, gestión de salas y transmisión de voz optimizada para VR. Esta opción fue la elegida por su estabilidad, facilidad de integración y escalabilidad inmediata, lo que permite reducir los tiempos de desarrollo y concentrarse en las funciones pedagógicas del entorno. Aunque el uso de un servicio de terceros implica cierto grado de dependencia y costos de licenciamiento, sus ventajas en robustez y soporte justifican la decisión. Cabe destacar que la arquitectura modular del sistema permitirá en el futuro una eventual migración hacia una infraestructura propia, si así lo requiere la institución.

Método de locomoción del usuario: Se optó por el movimiento continuo con joystick, que permite una navegación fluida, coherente con el entorno tridimensional, y fomenta la interacción social espacial. Para minimizar el riesgo de malestar en usuarios sensibles, se incluirán medidas de confort como el viñeteado dinámico o la limitación de velocidad de desplazamiento. Aunque el teletransporte se descarta como método principal, no se descarta su inclusión futura como opción configurable para usuarios que así lo prefieran.

3.5 Arquitectura Técnica del Sistema Propuesto

La solución sigue una arquitectura cliente-servidor: cada usuario ejecuta una aplicación cliente en Unity (en su visor VR o PC), la cual se conecta a un servicio de backend (por ejemplo, los servidores de Photon) que gestiona la comunicación en red y sincronización entre clientes. Sobre esta base, en el cliente se estructuran varios módulos o subsistemas encargados de aspectos particulares: locomoción, interacción, renderizado de presentaciones, red, gestión de usuarios y permisos, entre otros. A continuación, se describen los principales módulos de la arquitectura técnica y cómo se integran entre sí:

3.5.1 Módulo de Locomoción

El módulo de locomoción es el encargado de gestionar el movimiento del usuario dentro del entorno virtual. Está implementado aprovechando las herramientas provistas por la plataforma VR de Unity: se emplea un XR Rig (una configuración estándar que representa la cámara del usuario y sus controladores en el mundo 3D) equipado con un componente de movimiento continuo. En concreto, se utilizará el Continuous Move Provider ofrecido por el SDK de Meta o el XR Interaction Toolkit, configurado para tomar la entrada del joystick del controlador y traducirla en desplazamientos del avatar/cámara. Este módulo se ocupa de actualizar la posición del usuario en cada frame conforme a la dirección del input y la velocidad definida, aplicando también colisiones y restricciones para evitar atravesar objetos o salir del área virtual. Además, el módulo de locomoción puede incluir funcionalidades de salto suave de altura (por si hay escalones o tarima en el aula, aunque es probable que no se requiera en un salón plano) y alineación con el piso (garantizando que el usuario siempre se desplace sobre el plano del suelo virtual, evitando fluctuaciones verticales). Un aspecto técnico importante es mantener la experiencia cómoda: se calibrará la velocidad de movimiento y se dará la opción de activar mitigaciones de mareo (como un ligero oscurecimiento de los bordes de la vista

al moverse rápido). Desde la perspectiva de red, el módulo de locomoción genera cambios en la posición del usuario que deben ser comunicados a los demás: por tanto, cada vez que un cliente mueve su avatar, estas nuevas coordenadas se transmiten mediante el módulo de red (por ejemplo, Photon transform views) para que los demás clientes actualicen la posición de ese avatar en sus escenas.

3.5.2 Módulo de Interacción y Manipulación

El módulo de interacción abarca todas las formas en que el usuario puede manipular objetos virtuales o interfaces dentro del aula. Se basa en el XR Interaction Toolkit de Unity, que provee un marco para reconocer cuándo un controlador VR toca o toma un objeto. En la escena, los objetos interactivos (como el marcador, el borrador de la pizarra, los botones de avanzar diapositiva, etc.) estarán equipados con componentes como XR Grab Interactable (que permite que puedan agarrarse) o XR Ray Interactable (para apuntar a distancia con un rayo, útil para botones lejanos o para interactuar con la pantalla de presentaciones). El módulo de interacción configura los interactors asociados a las manos del usuario: típicamente, se habilitará tanto la interacción directa (tocar algo con la mano) como la interacción a distancia mediante un rayo proyectado desde el controlador (simulando un puntero láser). Esto último es importante especialmente para el docente, que puede necesitar hacer clic en un botón de “siguiente diapositiva” situado en la pantalla virtual sin tener que caminar hasta ella; con un rayo interactivo puede pulsar el botón a distancia. Dentro de este módulo se implementan las lógicas específicas de cada objeto interactivo: por ejemplo, para el marcador virtual, se programa que al detectar que un usuario con permiso (docente) lo ha agarrado y lo está presionando contra la superficie de la pizarra, se inicie el trazo (esto requiere integración con el módulo de pizarra descrito más adelante, para dibujar). Para un botón de interfaz (como “pasar diapositiva”), el módulo de interacción se suscribe al evento de selección de ese botón y

ejecuta la acción correspondiente en el módulo de presentaciones. Asimismo, maneja aspectos como la retroalimentación háptica: cuando un usuario toca o pulsa algo, se puede activar una ligera vibración en su controlador para dar realismo. Desde el punto de vista técnico, la interacción local del usuario se procesa en su cliente, pero muchas interacciones deben reflejarse en todos los usuarios. Se puede utilizar un patrón maestro-cliente: por ejemplo, el primer usuario que agarre el objeto podría volverse el "owner" de ese objeto en la red, enviando actualizaciones de su movimiento. El sistema Photon (u otro) ayuda a resolver conflictos, de manera que un objeto no pueda ser agarrado. Su diseño modular permite añadir fácilmente nuevos objetos interactivos en el futuro (por ejemplo, si se incluye un libro virtual que los estudiantes puedan hojear, bastaría con agregar un script interactuable con su lógica).

3.5.3 Módulo de Renderizado de Presentaciones

El módulo de renderizado de presentaciones se dedica específicamente a la funcionalidad de mostrar contenido multimedia (principalmente diapositivas) dentro del entorno VR. Este módulo consta de dos subcomponentes: por un lado, la parte de procesamiento/entrada de contenido, y por otro, la parte de visualización en la escena 3D. El sistema permite al docente o administrador cargar archivos de presentación (PPTX, PDF o imágenes) para su uso en el aula virtual. Este módulo incluye utilidades para importar esos archivos: El sistema convierte automáticamente los archivos de presentación en diapositivas de imagen (PNG/JPG) de alta resolución cuando se trata de PPTX o PDF, y acepta imágenes sueltas en su formato original. Una vez convertidos o preparados los slides, el módulo mantiene un repositorio local (cache) de las texturas o materiales listos para usar en Unity. En la parte de visualización, dentro de la escena del aula virtual hay un objeto que actúa como pantalla o proyector. El módulo de renderizado de presentaciones controla el material asignado a esa superficie: cuando se debe mostrar

una determinada diapositiva, asigna la textura correspondiente a la pantalla. Así, los usuarios verán la imagen de la diapositiva proyectada en el mundo virtual. Este módulo se integra con la interacción: El sistema incluye un controlador que permite al docente avanzar o retroceder las diapositivas mediante botones virtuales o controles en el dispositivo. Un desafío técnico es asegurar que la calidad de imagen en VR sea suficiente: a diferencia de un monitor tradicional, en VR la resolución efectiva por ojo es limitada, así que este módulo implementa técnicas como zoom o acercamiento para el usuario (quizá permitiendo que si un estudiante se acerca a la pantalla vea la imagen con mayor detalle), o asegurarse de que las imágenes tengan el tamaño óptimo para que el texto sea legible a la distancia promedio de los asientos virtuales. En cuanto a la sincronización en red, este módulo opera mayoritariamente en modo maestro-esclavo: solo el docente (o quien tenga control) emite comandos de cambio de diapositiva, y el módulo de red los distribuye a todos los clientes, de forma que todos los clientes cambian a la misma diapositiva al mismo tiempo. También se envía el estado inicial al usuario que se une posteriormente a la inicialización de la presentación (por ejemplo, “estamos en la diapositiva 5 actualmente” para que el nuevo participante cargue directo esa imagen). El contenido de las diapositivas en sí puede enviarse de antemano o asumirse que cada cliente tiene acceso (si la presentación fue pre-cargada con la sesión, cada cliente la descarga del servidor al unirse para no sobrecargar la red enviando imágenes grandes en tiempo real).

3.5.4 Módulo de Comunicación en Red y Sincronización

El módulo de red es el núcleo que permite la experiencia multiusuario, ocupándose de conectar a los distintos clientes y mantener la consistencia del estado compartido. Se basa en la utilización del SDK de Photon (Photon Unity Networking, PUN, complementado con Photon Voice para audio). Al iniciar la aplicación, el módulo

de red conecta el cliente a los servidores Photon en la nube, autentica al usuario (posiblemente pasando un token o ID definido por el sistema de login) y lo une a una sala virtual correspondiente a la clase. Photon se encarga de ubicar a los usuarios en el mismo "room" para que puedan verse. Dentro del módulo de red se definen qué elementos y eventos deben sincronizarse:

Posición y rotación de avatares: Cada cliente, a intervalos muy frecuentes, envía la posición y rotación de las partes relevantes de su avatar (cabeza y manos básicamente) al servidor, que las retransmite a los demás. Para esto se usan componentes de Photon que sincronizan transforms o se envían actualizaciones manualmente optimizadas (p. ej., solo enviar cambios significativos). Los clientes remotos al recibir estos datos actualizan la representación del avatar correspondiente en su escena.

Interacciones con objetos: Cuando un usuario agarra o suelta un objeto, o presiona un botón virtual, se envían mensajes de evento vía Photon usando Llamadas a Procedimiento Remoto (RPCs) o eventos indicando esa acción, de modo que todos los clientes apliquen el mismo resultado (por ejemplo, "el objeto X ahora está en manos del usuario Y" o "cambiar diapositiva a la 3").

Contenido de la pizarra: Dado que la escritura en la pizarra genera datos continuos (puntos o líneas trazadas), este módulo deberá transmitir esas trazadas de manera eficiente. El sistema enviará parámetros de la línea (coordenadas inicial y final de segmentos, color, grosor, etc.) de forma parcelada: el cliente docente al dibujar continuamente manda una secuencia de puntos que los clientes estudiantes reconstruyen para dibujar la misma línea. También cabe la opción de implementar la pizarra como una textura compartida, pero sincronizar texturas en tiempo real es pesado; es más viable enviar vectores y redibujar.

Voz: La transmisión de voz se implementa mediante Photon Voice, complemento de la plataforma Photon que permite transmisión de audio en tiempo real dentro de aplicaciones multiusuario. Cada cliente captura el sonido de su micrófono, lo comprime y lo envía al servidor Photon; desde allí se redistribuye a los demás clientes conectados al mismo canal de voz. El sistema gestiona automáticamente la mezcla, la calidad y la latencia del audio, ofreciendo comunicación de voz sincronizada en entornos VR/AR y juegos en línea. Este subcomponente maneja con la calidad de audio, comprimiendo y descomprimiendo flujo de voz con codecs adecuados para que suene en tiempo real.

El módulo de red también implementa lógica de reconciliación y autoridad: por ejemplo, si dos eventos confluyen (dos usuarios intentan agarrar el mismo objeto casi al mismo tiempo), define reglas de prioridad o deja que Photon resuelva por timestamp de llegada. Igualmente, maneja que pasa cuando un usuario se desconecta abruptamente: por diseño, debe informar a todos para que, por ejemplo, se elimine el avatar de ese usuario de la escena y se liberen los objetos que tenía. En caso de desconexiones temporales (pérdida de conexión), Photon intenta reconectar al usuario a la sala. La arquitectura considera la seguridad en red: aunque Photon provee cierto nivel, se podrían añadir validaciones en cliente (por ejemplo, si llega un evento de que un estudiante cambió la diapositiva, ignorarlo a menos que venga marcado como originado por el docente).

3.5.5 Módulo de Gestión de Usuarios, Roles y Permisos

El módulo de gestión de usuarios y roles se encarga de todo lo relativo a la identidad y privilegios de cada participante dentro del sistema. Este módulo actúa tanto en la fase de inicio de sesión/entrada al sistema, como en tiempo de ejecución para autorizar acciones sensibles. En la fase de autenticación, este módulo verificará las credenciales del usuario que intenta ingresar. Dado que en nuestro diseño se contempla

un registro previo de usuarios (por la administración), se implementa una ventana de login donde el usuario introduzca usuario/contraseña, o simplemente un código de sala junto a su nombre (si se opta por un acceso más simple estilo “join with a code and nickname”). Una vez validado, el módulo asigna el rol correspondiente: por ejemplo, si en la base de datos el usuario X está registrado como docente de la clase Y, al unirse a esa sala se le marca con rol docente; a otros se les marca como estudiantes, etc. Esta información de rol viaja con el usuario al unirse a la sala (podría adjuntarse en los custom properties del usuario en Photon, de modo que todos los clientes puedan saber el rol de cada participante). Dentro de la experiencia VR, el módulo de permisos consulta esa información para habilitar o deshabilitar funcionalidades. Por ejemplo:

Si un usuario con rol estudiante intenta agarrar el marcador virtual, el módulo de interacción preguntará al módulo de permisos si eso está permitido; probablemente responderá que no, y entonces ese objeto no reacciona al agarre por parte del estudiante (podría mostrarse un feedback de “No tienes permiso para usar esto” si se estima necesario).

Si un estudiante pulsa el botón de avanzar diapositiva, el módulo de presentaciones consultará permisos y lo ignorará, dado que solo el docente debe hacerlo.

Contrariamente, cuando el docente realiza estas acciones, el sistema lo permite y propaga en red.

El módulo de gestión de roles también controla aspectos como visibilidad o acceso a UI: quizás ciertos menús (por ejemplo, el menú para cargar nueva presentación) solo se muestran en la interfaz del docente y están ocultos para estudiantes. O el administrador podría tener un panel especial de opciones de moderación visible solo para él. En cuanto al administrador, este módulo se integra con sistemas externos o herramientas fuera del VR. Por ejemplo, podría haber un archivo de configuración o base de datos donde se

listan los usuarios administradores; el módulo de login verifica si el usuario es administrador y entonces podría permitirle lanzar la aplicación en un modo especial o acceder a comandos admin. Dado que en la ejecución típica el administrador no participa activamente dentro de la clase VR (salvo que se integre de forma especial), muchas de sus funciones residen fuera, pero es importante que este módulo permite su existencia. Otro subcomponente de este módulo es el log de auditoría: opcionalmente, podría registrar acciones importantes, especialmente las administrativas o de control, para fines de seguimiento.

3.5.6 Otros Componentes Técnicos

Subsistema de Avatares: Encargado de la representación gráfica de los usuarios. Incluye los modelos 3D de cuerpo/manos, animaciones (por ejemplo, animar la mano cuando se cierra el puño al agarrar algo, o mover una boca simulada cuando el usuario está hablando, etc.) y la asignación de identificadores (quizá una etiqueta con el nombre del estudiante sobre su avatar para fácil identificación). Este subsistema trabaja de la mano con los módulos de locomoción (para posicionar el avatar local según los movimientos) y red (para actualizar avatares remotos).

Módulo de Interfaz de Usuario (UI) y Menús: Este módulo gestiona todos los elementos de interfaz que apoyan la interacción en el aula virtual, tanto en 2D como en paneles 3D dentro del entorno. Sus funciones principales son:

1. **Login y autenticación:** interfaz inicial para que los usuarios ingresen credenciales o códigos de acceso a sala.
2. **Configuración personal:** menú accesible para ajustar volumen, silenciarse o editar preferencias de control.
3. **Gestión de la sesión:** opciones de pausa o salir de la clase, visibles para todos los usuarios.

4. **Carga de presentaciones:** menú para que el docente seleccione y cargue archivos (PPTX, PDF, imágenes).
5. **Controles de navegación:** paneles o botones virtuales que permiten al docente avanzar/retroceder diapositivas.
6. **Feedback e indicaciones:** notificaciones visuales o auditivas que confirman acciones (archivo cargado, cambio de diapositiva, usuario conectado).

Este módulo se responsabiliza de dibujar esas UIs (ya sea en pantalla o en el mundo como paneles interactivos) y de gestionar su lógica (conexión con módulo de red para login, etc.).

Gestión de Persistencia de Datos: Si se guardan algunos datos (lista de usuarios, contraseñas cifradas, archivos de presentaciones subidos, configuraciones de aulas), habrá un pequeño módulo o servicio de persistencia. Se implementa un servidor web/API externo programado en Python/Flask que el cliente Unity contacte para la obtención de archivos.

Módulo de Sonido y Voz: Complementario al de red, encargado de reproducir adecuadamente los sonidos en VR. Incluye no solo la voz (que viene del módulo de red) sino también efectos de sonido locales, para añadir realismo. Se integra con el sistema de audio 3D de Unity ajustando volúmenes y espacialización.

Todos estos componentes juntos conforman una arquitectura integral. La comunicación entre módulos dentro del cliente se realiza principalmente a través de eventos y llamadas bien definidas: por ejemplo, cuando el módulo de interacción detecta un botón de siguiente diapositiva presionado, llama a una función del módulo de presentaciones; cuando el módulo de presentaciones carga una nueva slide, notifica al módulo de red para enviar el cambio; cuando el módulo de red recibe una actualización de avatar remoto, envía esos datos al subsistema de avatares para actualizar el modelo en

la escena, etc. Esta separación en módulos facilita la mantenibilidad (RNF07) y posibles futuras extensiones, ya que cada componente tiene una responsabilidad clara.

3.6 Planificación del proyecto

Desarrollar un entorno virtual de estas características es un proceso complejo, por lo que es fundamental establecer una planificación por fases y tareas que guíe la implementación de manera organizada. A continuación, se describe el plan técnico, dividido en etapas secuenciales, indicando las tareas principales de cada fase y sus dependencias. Esta planificación busca asegurar que los componentes básicos estén operativos antes de añadir los más complejos, permitiendo iterar con pruebas frecuentes:

Tabla 3

Cronograma de planificación por fases

Fase	Duración Estimada	Hitos y Tareas Principales
Fase 0: Investigación Inicial	2 semanas	– Estudio de viabilidad: revisión de Unity vs Unreal, SDKs VR disponibles.– Aprendizaje de base: tutoriales de Meta Quest, ejemplos de XR Interaction Toolkit.– Definición inicial de requisitos (borrador de RFs y RNFs).
Fase 1: Configuración del Entorno	2 semanas	– Creación del proyecto Unity 2022.3 LTS con URP/OpenXR.– Integración del Meta XR AIO SDK y XR Interaction Toolkit en el proyecto.– Prueba básica en Quest: escena de

		ejemplo con XR Rig para verificar deployment.
Fase 2: Locomoción y Base VR	3 semanas	– Implementar movimiento del usuario con joystick (Continuous Move).– Agregar y configurar colisiones de entorno (paredes, suelo) y del jugador (CharacterController).– Probar variantes de movimiento (velocidad, giro suave vs por pasos) y ajustar según comodidad.
Fase 3: Interacción Pizarra	4 semanas	– Modelar o colocar objeto Pizarra en la escena con collider adecuado.– Desarrollar script de Marcador y lógica de dibujo (raycast a pizarra, pintar textura).– Iterar optimizaciones: calibrar sensibilidad de dibujo, grosor de línea, etc.– Pruebas en Quest de escritura, identificar problemas de precisión y latencia.
Fase 4: Presentaciones VR	4 semanas	– Investigar métodos para mostrar PPT/PDF (ver Tabla 3.1 de alternativas).– Implementar primera versión: carga de imágenes como diapositivas (crear varios sprites/texturas y un mecanismo de cambio).– Integrar

		plugin FileBrowser para seleccionar archivo local, y parsear contenido (simple: seleccionar un directorio de imágenes por ejemplo).– UI de control de presentaciones (botones siguiente/anterior o asignar a controles físicos).– Pruebas con distintos tamaños de imágenes, ajuste de calidad legibilidad.
Fase 5: Integración y Entorno Final	3 semanas	– Integrar todos los subsistemas en una escena cohesiva (movimiento + pizarra + presentaciones).– Añadir detalles del aula: por ejemplo, algunos objetos estáticos (mesas, sillas) para ambientar y probar colisiones físicas.– Implementar manos virtuales básicas (usar modelos prefabricados o primitivas para representar manos).– Ensayo general: simular una clase uno mismo, detectar fallos de flujo.
Fase 6: Pulido y Optimización	2 semanas	– Perfilar rendimiento (usar Unity Profiler, OVR Metrics) durante sesiones prolongadas.– Optimizar draw calls (combinar mallas estáticas, reducir

		Canvas overlay si hubiera).– Corregir bugs conocidos: ej. marcador traspasa pizarra (mejorar colisión), rarezas en movimiento (clipping en esquinas).– Documentar limitaciones actuales que no se pueden resolver por tiempo.
Fase 7: Diseño de Red (Investigación)	2 semanas (en paralelo parcial)	– Explorar SDKs de networking (Photon, Mirror): crear proyecto de prueba multiusuario con objetos básicos sincronizados.– Diseñar arquitectura de red para futura implementación (qué sincronizar, frecuencia, etc.).– *Nota:* Esta fase es exploratoria; no se integra al producto final pero sienta las bases teóricas para desarrollo posterior (futuro trabajo).
Fase 8: Elaboración de Tesis	4 semanas (tras desarrollo)	– Redacción de capítulos finales de la tesis, incluyendo el presente Capítulo 3 con la descripción técnica detallada.– Preparación de figuras, capturas de pantalla del aula virtual funcionando, diagramas de arquitectura y casos de uso.– Revisión general,

		correcciones y ajuste del documento a formato requerido.

3.7 Riesgos del proyecto y estrategias de mitigación

La creación del aula virtual inmersiva conlleva una serie de riesgos técnicos y de proyecto que deben ser tenidos en cuenta. A continuación, se enumeran los principales riesgos identificados, junto con las estrategias planificadas para prevenirlos o mitigar su impacto en caso de que se materialicen:

Riesgo 1 – Latencia de red elevada o inestabilidad en la comunicación: Existe la posibilidad de que, debido a conexiones de internet lentas o a la latencia inherente de la comunicación con servidores en la nube, se produzcan demoras perceptibles en la sincronización).

Mitigación: Se utiliza un servicio de networking confiable (Photon) que ofrece centros de datos globales reduciendo la distancia a los clientes. Además, se utiliza técnicas de interpolación y buffering: por ejemplo, los movimientos de avatares recibirán una ligera predicción para moverse suavemente incluso si llegan datos con retraso; en voz, Photon Voice aplica codecs con baja latencia. Se define límites de usuarios por sesión (RNF04) para no saturar el ancho de banda disponible. Igualmente, en las pruebas (Fase 5) se mide la latencia y, si es necesario, se ajustarán parámetros (como la tasa de envío de actualizaciones) para garantizar que las acciones críticas se prioricen. Contar con mensajes más pequeños y eficientes (por ejemplo, enviar solo diferencias en la pizarra en lugar de toda la textura) también ayudará a que la red se use óptimamente. En caso de inestabilidad (pérdida de conexión temporal), el sistema incluye mecanismos de reconexión automática: si un cliente se desconecta, se intenta volver a unirse a la sala sin requerir reiniciar la aplicación, de modo transparente para el usuario.

Riesgo 2 – Carga gráfica y bajo rendimiento en el visor Meta Quest: Dado el hardware limitado del Quest, hay riesgo de que la aplicación experimente baja tasa de frames o sobrecalentamiento si la escena es muy pesada o si la lógica no está optimizada. Esto podría manifestarse como movimientos entrecortados, retardos en la respuesta a las acciones, e incluso mareo en los usuarios.

Mitigación: Desde el diseño se ha priorizado la optimización (RNF02): se utilizarán modelos 3D sencillos para el aula y los avatares (evitando geometrías innecesariamente complejas), texturas comprimidas adecuadamente, y se aprovecharán técnicas como baked lighting para que la iluminación no ejecute cálculo en tiempo real. Durante el desarrollo, se usará el Unity Profiler y la herramienta OVR Metrics en Quest para identificar cuellos de botella. Se implementa, de ser necesario, Fixed Foveated Rendering (FFR) en Quest, que reduce la resolución de render en la periferia de la visión para ahorrar rendimiento sin pérdida de calidad notable. Asimismo, si se detecta que alguna funcionalidad es muy costosa (por ejemplo, dibujar en la pizarra en alta resolución), se pueden ajustar parámetros (reduciendo resolución de la textura de la pizarra ligeramente, o implementando un sistema de capas donde los dibujos antiguos se "flatten" para no recalcular siempre). El plan de pruebas finales incluye ensayos prolongados con la aplicación funcionando para observar si el dispositivo mantiene rendimiento o si hay caídas debido a termal throttling, abordando estos hallazgos puntualmente (p.ej., reduciendo efectos gráficos que no sean esenciales).

Riesgo 3 – Conectividad inestable o pérdida de usuarios durante la sesión: En un entorno real, es posible que algunos participantes sufran desconexiones de internet o caídas de la aplicación (crashes) en medio de la clase, lo que podría interrumpir la experiencia educativa (por ejemplo, el docente se desconecta accidentalmente dejando a los estudiantes sin guía).

Mitigación: Para este riesgo se plantean varias acciones: primero, como mencionado, integrar una funcionalidad de reconexión que intente automáticamente volver a conectar al usuario a la sala si su conexión se corta brevemente. Photon provee algunas facilidades para reconexión si es rápido; sino, el sistema al menos detectará la desconexión y mostrará al usuario un mensaje con la opción de reintentar. En caso de que el docente sea el que se desconecta, se pueden tomar medidas como pausar temporalmente las actividades (los estudiantes recibirán un aviso en VR de "El instructor se ha desconectado, esperando reconexión...") o, en escenarios más avanzados, designar a un sustituto si hubiera co-docente. El cliente podría mostrar indicadores de calidad de conexión (barras de red) para que el usuario se percate si su señal es débil y quizás pueda moverse a mejor cobertura. A nivel técnico, se mantiene la lógica de la aplicación robusta ante desconexiones: por ejemplo, si un estudiante se desconecta, su avatar simplemente desaparecerá de los otros clientes sin causar bloqueos; si vuelve a entrar, se le permite reingresar a la clase. El administrador del sistema también puede contar con registros para ver si hay problemas recurrentes de conectividad y tomar medidas externas (como mejorar la red WiFi del lugar, etc.).

Riesgo 4 – Problemas de comodidad del usuario (mareo por movimiento o fatiga): La locomoción continua y el uso prolongado de un visor VR pueden provocar ciber-cinetosis (mareos) en algunos usuarios, así como fatiga física (por ejemplo, brazos cansados por sostenerlos). Esto representaría un riesgo de adopción: si los usuarios se sienten incómodos o enfermos al usar la plataforma, no querrán utilizarla por mucho tiempo.

Mitigación: Para abordar este riesgo, se implementan varias opciones de confort (RNF06). Por un lado, como se discutió, aunque el movimiento por defecto es continuo, se incluye la opción de teletransporte en la configuración personal del usuario, de modo

que aquel que sea susceptible al mareo pueda elegir saltos instantáneos en vez de desplazamiento suave. Se podrá activar/desactivar el viñeteado durante el movimiento: algunos usuarios prefieren que, al moverse rápidamente, la periferia de la visión se oscurezca ligeramente para reducir la sensación de movimiento, mientras que otros lo desactivarán; se dará esa opción en un menú. En cuanto a la fatiga, el diseño del entorno y las interacciones buscará minimizar esfuerzos: por ejemplo, ubicar la pizarra a una altura cómoda para escribir sin tener que levantar los brazos demasiado alto; permitir que el usuario se teletransporte a la posición de la pizarra (incluso si está en modo continuo) con un atajo, para que no deba caminar cada vez; proveer asientos virtuales o la posibilidad de usar el sistema sentado (muchos entornos VR permiten tanto estar de pie como sentado recalibrando la altura del avatar). Se incluye descansos si la sesión es muy larga – esto ya depende de la dinámica de la clase, pero el sistema podría recomendar al usuario quitarse el visor unos minutos cada cierto tiempo. Se pide a varias personas que usen la plataforma y den feedback específico sobre cómo se sienten, haciendo ajustes en consecuencia (por ejemplo, reduciendo la velocidad de movimiento predeterminada si muchos reportan inestabilidad).

Riesgo 5 – Dificultades técnicas no previstas o retrasos en el desarrollo: Por último, existe un riesgo general de que surjan obstáculos técnicos imprevistos (bugs complejos, limitaciones de alguna herramienta) o que alguna tarea tome más tiempo de lo planificado, amenazando el cronograma del proyecto. Por ejemplo, podría haber complicaciones integrando Photon Voice con el sistema de roles, o se podría detectar que la sincronización de la pizarra requiere un enfoque distinto después de mucha prueba y error.

Mitigación: La planificación propuesta ya incorpora cierto orden lógico y buffer para manejar riesgos: al desarrollar primero las funcionalidades localmente (Fase 2) se

puede validar su viabilidad antes de sumarle la capa de complejidad de red; esto ayuda a aislar problemas. Además, se ha priorizado el uso de tecnologías y SDKs bien soportados (Unity XR, Photon) para reducir riesgo de fallos low-level, contando con documentación y foros de soporte abundantes que pueden ayudar a resolver inconvenientes rápidamente. En caso de retrasos, se aplicaría planificación adaptativa: por ejemplo, si cierta característica extra (digamos, animaciones faciales en avatares) no es esencial y está demorando mucho, podría recortarse o posponerse, enfocando los recursos en completar los requisitos mínimos primero. El seguimiento del proyecto incluye hitos de verificación al final de cada fase, donde se evalúa el estado y se ajusta la siguiente fase si es necesario (metodología iterativa). También se consideran pruebas continuas para identificar bugs pronto; cualquier problema crítico descubierto será atendido tan pronto como sea posible antes de que se acumule con otros.

4. CAPITULO 4

Implementación local de las funcionalidades básicas del aula virtual

Antes de incorporar funcionalidades multijugador, fue necesario establecer un conjunto de herramientas básicas que definirán la experiencia inmersiva del aula desde una perspectiva individual. Esta etapa inicial permitió consolidar los elementos fundamentales del entorno virtual, asegurando una base estable para su futura expansión.

4.1 Diseño del entorno virtual tridimensional del aula

El primer paso en la creación del aula virtual fue diseñar el entorno 3D que recrea un salón de clases de manera realista y funcional. Para lograr la sensación de inmersión, se trabajó con unidades físicas reales (1 unidad de Unity ≈ 1 metro) al dimensionar la sala, de modo que las proporciones de la habitación y la altura de los objetos correspondan con las expectativas de un usuario en realidad virtual. El entorno desarrollado incluye elementos básicos de un aula (ver figura 1): suelo, techo y paredes que delimitan el espacio, una pizarra amplia colocada en la pared frontal, y una ambientación de iluminación adecuada.

En cuanto a la iluminación, se utilizaron luces direccionales y ambientales para simular la iluminación típica interior de una sala de clases, evitando zonas demasiado oscuras que pudieran afectar la comodidad visual en VR. Las paredes y el piso llevan texturas simples y colores neutros para evitar distracciones y realzar la visibilidad de la pizarra y las diapositivas proyectadas. Para el cielo (skybox) u horizonte visible, se configuró un color uniforme o textura apropiada a través del componente Skybox de Unity, de forma que al usuario le resulte natural mirar alrededor sin encontrar vacíos

gráficos.

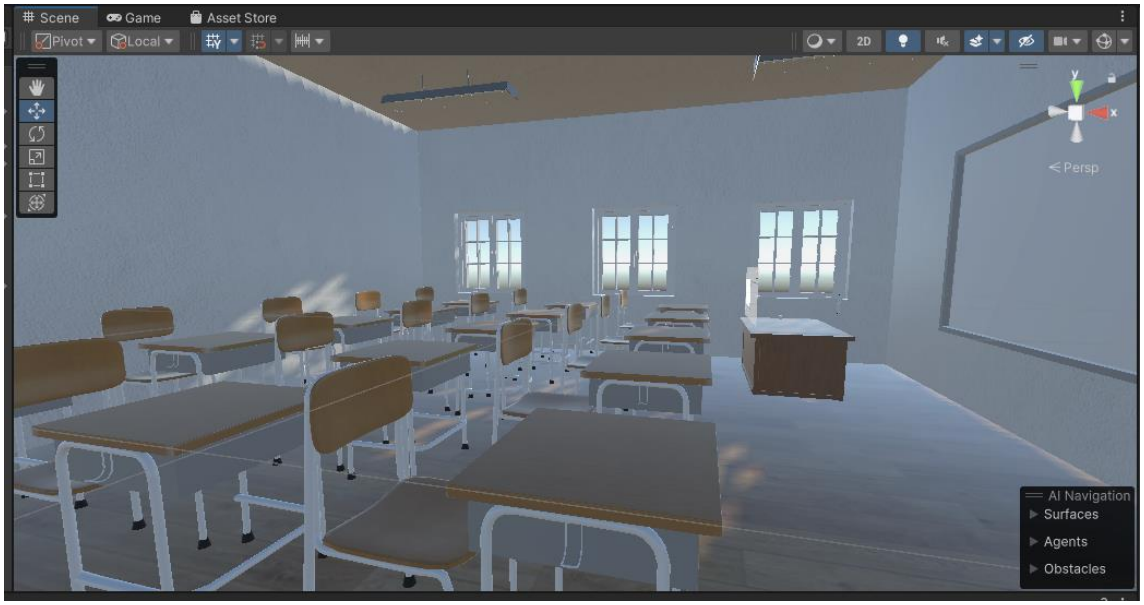

Fig. 1 Vista del entorno del aula

Un componente central del entorno es la pizarra blanca (whiteboard) colocada en la parte frontal de la sala, la cual será interactiva. La pizarra se modeló como un plano rectangular grande con una textura blanca inicial, similar a un tablero acrílico real. Se le añadieron componentes de colisión (Collider plano) para detectar interacciones físicas (toques del marcador virtual) y se organizó dentro de la escena a una altura cómoda (aproximadamente 1.2 m desde el suelo) y de dimensiones amplias para escribir.

Se diseñó un sistema de representación del usuario dentro del espacio virtual mediante un XR Rig adaptado para visores Meta Quest (Ver Figura 2). Se optó por una configuración híbrida que combina el Meta XR SDK (v76) y su sistema de locomoción con el paquete Auto Hand, encargado de gestionar la manipulación física de objetos dentro del entorno. Para el movimiento del usuario, se utilizó el componente *Continuous Move Provider* del XR Interaction Toolkit (v3.0.8), el cual permite una locomoción continua controlada por el joystick análogo del visor, simulando desplazamientos naturales por el aula. Este tipo de locomoción mejora la sensación de presencia frente a

técnicas de teletransporte, y se acompaña de ajustes como velocidad moderada y compatibilidad con efectos de confort (como viñeteado) para mitigar posibles mareos. Por otro lado, las interacciones físicas —como tomar el marcador o tocar la pizarra— se gestionan mediante Auto Hand, un sistema que detecta colisiones y simula agarres realistas, permitiendo que las manos virtuales respeten la física del motor de Unity y reaccionen de forma coherente al contacto con objetos tridimensionales. Esta combinación permite integrar de manera fluida tanto el desplazamiento libre como la interacción tangible dentro del aula inmersiva.

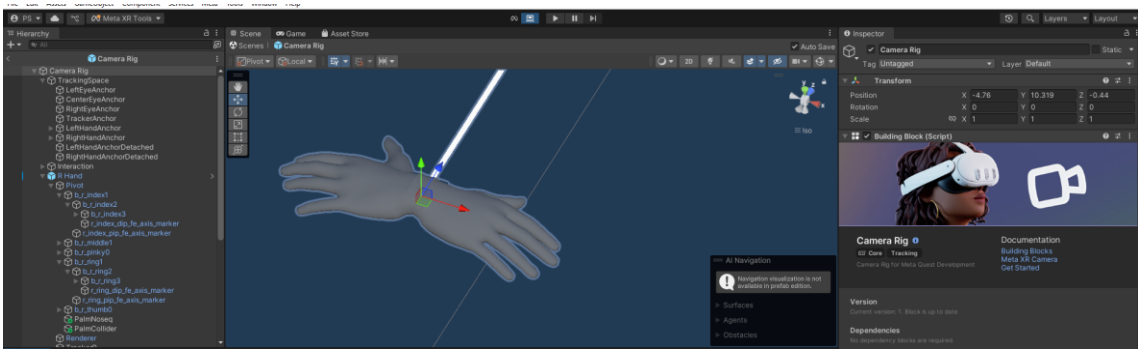

Fig. 2 Vista de prefab de Camera Rig

Se añadió un CharacterController al XR Rig del usuario, configurado con un radio y altura adecuados, como se muestra en la Figura 3, para que al moverse por el entorno colisione con las paredes y el mobiliario virtual en lugar de pasar a través de ellos. Asimismo, se verificó que todos los elementos estáticos importantes del aula (suelo, paredes, pizarra) tuviesen componentes Collider en Unity, de modo que el CharacterController del usuario los detectara. Se estableció la capa de colisiones para el jugador y objetos interactivos (por ejemplo, asignando capas "Player" e "Interactable" en Unity) y se ajustó la matriz de colisión en los Project Settings para evitar colisiones no deseadas (por ejemplo, se deshabilitan colisiones entre el propio jugador y ciertos objetos que porta, como el marcador, para impedir interferencias. El CharacterController detiene

el movimiento en seco al chocar con el collider de la pared; similarmente, la pizarra actúa como una barrera física que detiene al jugador.

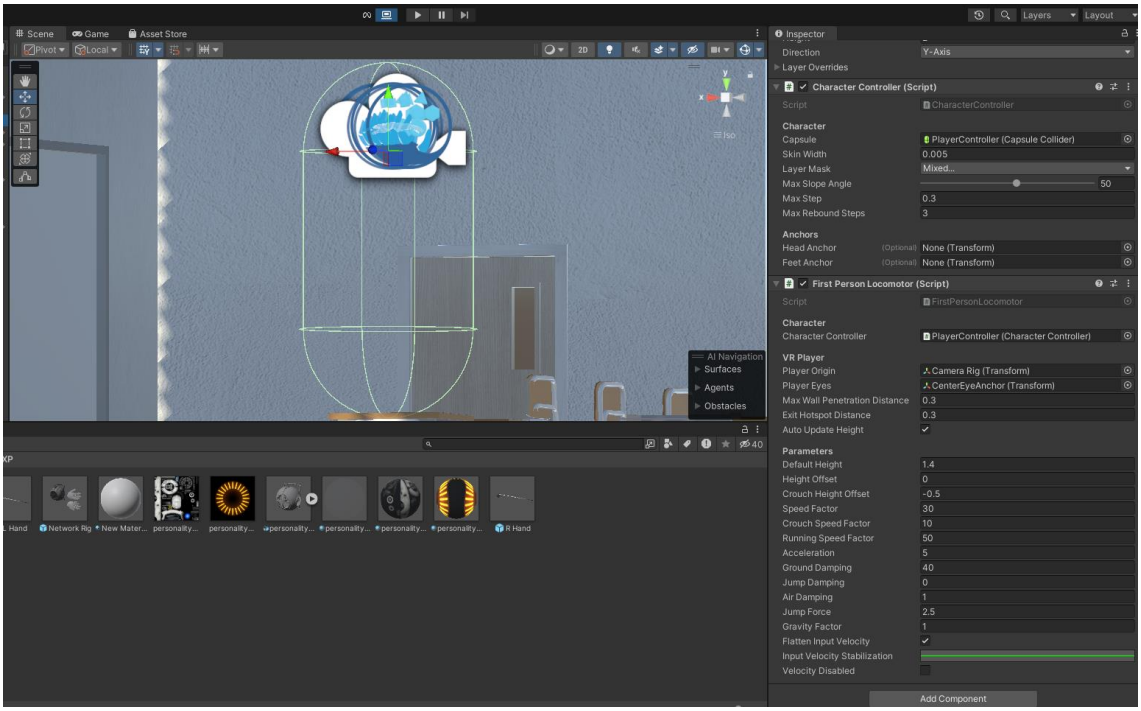

Fig. 3 Vista de rig + Locomotor con collider activo

4.2 Implementación de la pizarra de escritura interactiva

La pizarra interactiva es uno de los componentes fundamentales del aula virtual, permitiendo al usuario escribir o dibujar de forma natural en un espacio virtual como lo haría en un pizarrón físico. Su implementación se basó en manipular una textura 2D en tiempo real mediante las entradas del usuario con el marcador virtual.

Configuración básica de la pizarra: Como se mencionó, la pizarra se modeló como un objeto plano (un GameObject de tipo Plane o Quad) colocado verticalmente. A este objeto se le asignó un material inicial de color blanco (simulando una pizarra blanca acrílica). Para hacerlo *escribible*, se creó mediante script en tiempo de ejecución una Textura 2D (Texture2D) que cubre la superficie de la pizarra. En el método de inicialización (Start) del script de la pizarra (Whiteboard.cs), se instancia una nueva textura en blanco de resolución definida (por ejemplo, 2048×2048 píxeles para garantizar

suficiente detalle) y esta textura se asigna al material del objeto pizarra para sustituir su textura por defecto. De esta manera, la pizarra comienza con una superficie blanca limpia representada por una textura sobre la cual se puede dibujar. El tamaño de 2048×2048 se escogió para tener una alta resolución de dibujo que permite trazos relativamente finos sin pixelación evidente (ver Figura 4).

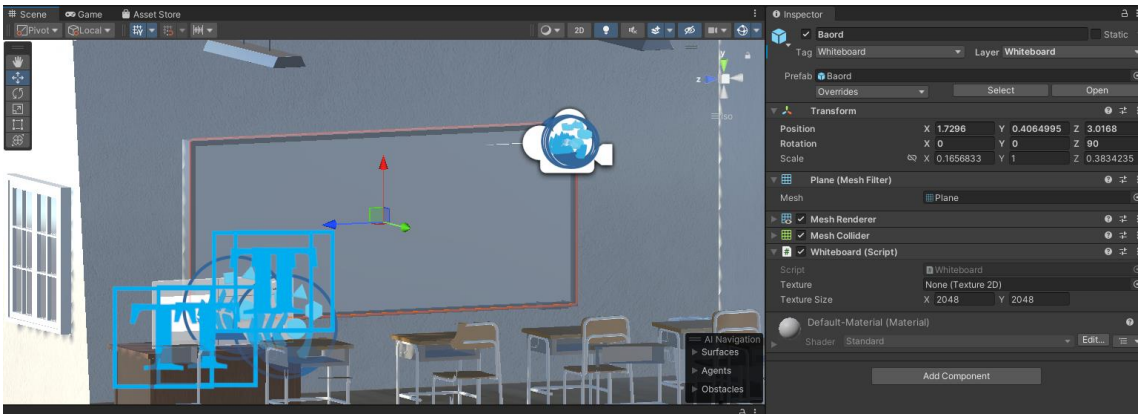

Fig. 4 Vista de pizarra con configuración en inspector

Una vez preparada la superficie de la pizarra, el siguiente desafío fue detectar de forma precisa dónde y cuándo el usuario escribe sobre ella. En modo local, donde se asume que solo hay un usuario interactuando, cualquier trazo se refleja directamente sobre la textura local asociada a la pizarra. Para capturar la interacción, se utilizó un enfoque basado en *raycasting*, descartando el uso de colisiones físicas convencionales para evitar interferencias no deseadas.

4.3 Desarrollo del marcador virtual 3D

Para permitir la interacción del usuario con la pizarra, se desarrolló un marcador virtual en 3D, análogo a un rotulador o plumón de pizarra en el mundo real (ver Figura 5). Este marcador es el instrumento con el que el usuario puede escribir en la superficie asignada, y su implementación abarcó tanto el aspecto visual y físico como el comportamiento interactivo.

4.3.1 Modelo y propiedades físicas

El marcador virtual se representó mediante un modelo 3D de forma alargada (cilindro delgado), diseñado para ser sostenido con la mano del usuario en el entorno de realidad virtual. En la punta del marcador se colocó un objeto hijo que representa visualmente el extremo de escritura. A este se le asignó un transform vacío, utilizado como punto de origen para el raycast de detección de contacto (ver sección anterior). Aunque inicialmente se evaluó el uso de un collider diminuto en la punta para detectar colisiones, en la versión final este se mantuvo únicamente con propósitos de agarre, no para el dibujo. El marcador completo cuenta con un componente Rigidbody, configurado como kinematic mientras está en uso, lo cual asegura que se mueva junto con la mano del usuario sin verse afectado por la gravedad o colisiones mientras está agarrado.

4.3.2 Interactividad y agarre

La interacción física del usuario con el marcador se gestionó mediante el sistema Auto Hand, un paquete especializado en interacciones de manos realistas dentro de Unity. Se asignó al marcador el componente Grabbable de Auto Hand, lo cual permite que cualquier mano equipada con el componente Hand pueda detectar su presencia y agarrarlo de forma dinámica. Para lograrlo, se definieron correctamente los colliders del marcador (por ejemplo, un BoxCollider alrededor del cuerpo), lo que permite que Auto Hand detecte el volumen del objeto y calcule una postura de agarre coherente. Cuando el usuario acerca la mano y presiona el botón correspondiente en el controlador, el sistema acopla automáticamente el marcador a la mano virtual, generando una transición natural y física. Durante el agarre, el marcador se convierte en hijo del Transform correspondiente a la mano, asegurando que sus movimientos se mantengan sincronizados hasta que se libere. Al soltarlo, el Rigidbody vuelve a estar libre y sujeto a las leyes físicas, lo que permite que el marcador caiga o rebote con realismo.

4.3.3 Detección de escritura

El comportamiento central del marcador es, naturalmente, escribir en la pizarra. Para esto, se implementó un sistema basado en raycasting, ejecutado desde la punta del marcador en cada cuadro (frame). Se lanza un rayo corto en la dirección de la punta, que corresponde al eje local transform.up, con una distancia limitada (e.g., 0.02 m), suficiente para detectar el contacto cercano sin requerir colisión física directa. Este raycast se restringe específicamente a objetos asignados al *layer* "Whiteboard", evitando así que elementos como las manos o superficies cercanas interrumpan los trazos.

Cuando el raycast impacta la superficie de la pizarra, Unity devuelve un objeto RaycastHit que incluye las coordenadas UV del punto de contacto (RaycastHit.textureCoord). Estas coordenadas normalizadas en el rango [0,1] se escalan al tamaño real de la textura (Texture2D) para determinar con precisión los píxeles que deben modificarse al dibujar.

Una vez determinada la posición de impacto en la textura, se procede a "pintar" sobre la textura. Para lograrlo en Unity, la clase Texture2D ofrece métodos para manipular pixeles. Se usó Texture2D.SetPixels(x, y, width, height, color[]) para colorear un bloque de pixeles de un determinado color. El script del marcador (Marker) mantiene un buffer de color (un arreglo de Color) con dimensiones equivalentes al grosor del trazo deseado. Por ejemplo, si definimos el tamaño del "bolígrafo" (pen size) a 5, se construye un arreglo de 5×5 pixeles llenos con el color del marcador (por defecto negro, pero podría cambiar si hubiera marcadores de distintos colores). Cada vez que se detecta contacto con la pizarra, se colorea en la textura un cuadrado de 5×5 píxeles centrado en la posición de impacto, aplicando ese arreglo de color. Este cuadrado representa la marca del plumón sobre la pizarra, con un grosor equivalente a 5 píxeles de la textura.

Para lograr trazos continuos (líneas suaves en lugar de puntos desconectados), el algoritmo conserva la última posición dibujada en la pizarra y, si el marcador continúa en contacto de un frame a otro, se traza una línea interpolando entre la posición anterior y la nueva. Concretamente, si en el frame anterior se dibujó en la posición (x_{ant}, y_{ant}) de la textura y en el frame actual el raycast dio una nueva posición (x_{nuevo}, y_{nuevo}), el código interpola múltiples puntos intermedios entre esos dos puntos y en cada uno dibuja un pequeño bloque de color. De este modo, aunque el movimiento del controlador en VR pueda ser rápido, se van rellenando los segmentos intermedios en la textura para no dejar huecos en el trazo. Finalmente, tras dibujar los píxeles correspondientes, se llama a Texture2D.Apply() para aplicar los cambios de la textura en la GPU y que el usuario vea inmediatamente el trazo apareciendo en la pizarra. Este proceso se repite continuamente mientras el usuario mantenga el marcador sobre la pizarra y mueva su mano, produciendo la sensación de escritura en tiempo real.

Es importante destacar que este procedimiento se ejecuta en cada frame durante la escritura, pero dada la resolución de la textura y el tamaño relativamente pequeño del área modificada por frame, el impacto en rendimiento es manejable en el modo local. Se optimizó el código para evitar cálculos innecesarios: por ejemplo, si el marcador se aleja de la pizarra (el raycast deja de golpearla), el sistema deja de intentar dibujar hasta que haya un nuevo contacto. Asimismo, el uso de un penSize fijo facilita que con un solo arreglo de píxeles (creado una vez) se realicen las operaciones de pintado repetidas, en lugar de generar nuevas estructuras en cada contacto.

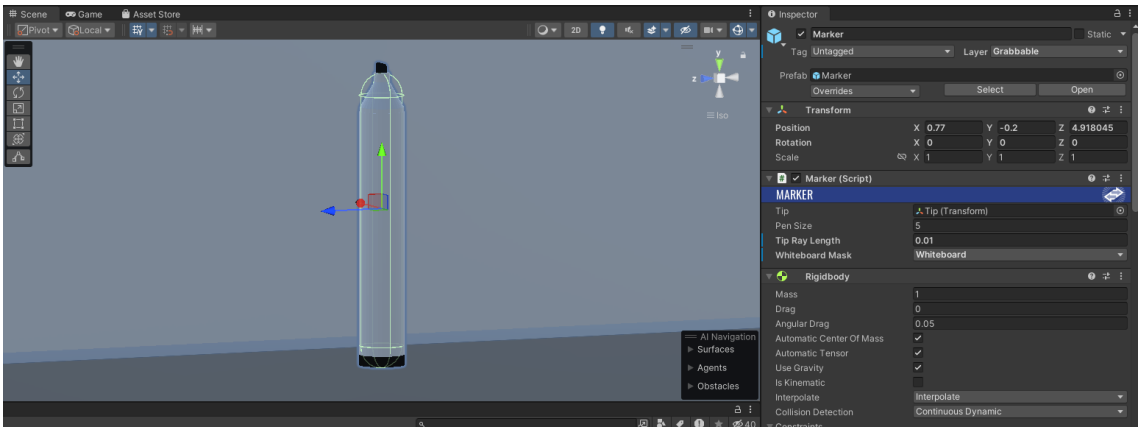

Fig. 5 Vista del marcador

Junto al marcador virtual, se añadió una herramienta de borrador para la pizarra, concebida de manera similar al rotulador pero con la función inversa de eliminar trazos. El borrador virtual tiene dimensiones proporcionales a la mano del usuario, lo que le permite cubrir un área de borrado más amplia en cada pasada. En la práctica, al colocar el borrador sobre la superficie de la pizarra, este borra los dibujos en un área rectangular (aproximadamente de 90 x 30 píxeles, tomando como referencia un grosor de línea de ~5 px del marcador). Su funcionamiento imita al de un borrador físico: el usuario lo desliza contra la pizarra y cualquier línea o trazo dentro de la zona de contacto desaparece gradualmente. Esta herramienta agiliza la limpieza del contenido de la pizarra, permitiendo corregir o borrar secciones completas de forma natural, y complementa al marcador virtual al brindar la posibilidad de enmendar o reiniciar la escritura en la pizarra cuando sea necesario.

4.4 Sistema de presentaciones y proyección de contenidos

Otra funcionalidad esencial del aula virtual es la capacidad de mostrar presentaciones (diapositivas, imágenes, documentos) dentro del entorno, tal como se haría con un proyector en una clase real. En el modo local, esto permite al usuario cargar

archivos de presentación y visualizarlos en el mundo virtual, facilitando la exposición de material educativo en VR.

Interfaz de usuario para carga de presentaciones: Se diseñó un pequeño menú de usuario (UI en 2D) al que el usuario puede acceder dentro de la aplicación para gestionar las presentaciones (ver Figura 6). Este menú incluye:

- Un botón “Cargar Archivos” que permite seleccionar uno o varios archivos desde el sistema de archivos (imágenes sueltas o un documento PDF/PPTX).
- Botones “Anterior” y “Siguiente” para navegar entre las diapositivas cargadas (retroceder o avanzar).
- Un texto o etiqueta mostrando el nombre del archivo actualmente abierto y quizás el número de diapositiva actual.
- Un toggle (interruptor) para mostrar u ocultar la proyección dentro del mundo virtual (es decir, encender o apagar la visualización en el “proyector” de la clase).

Dado que en VR la interacción con elementos de archivo puede ser complicada, durante el desarrollo en Unity Editor se integró un plugin llamado SimpleFileBrowser que provee un cuadro de diálogo de selección de archivos amigable. En modo local (especialmente si se prueba desde PC con el visor en Link), este navegador de archivos se abre permitiendo al usuario escoger archivos .png, .jpg, .jpeg (imágenes individuales) o bien un archivo .pdf o .pptx (presentación multipágina). Se configuraron filtros para estos formatos mencionados, de modo que el usuario sepa qué tipos de archivo puede abrir. Una vez el usuario selecciona el archivo(s) y confirma, el sistema cierra el diálogo de archivos y procede a procesar la selección.

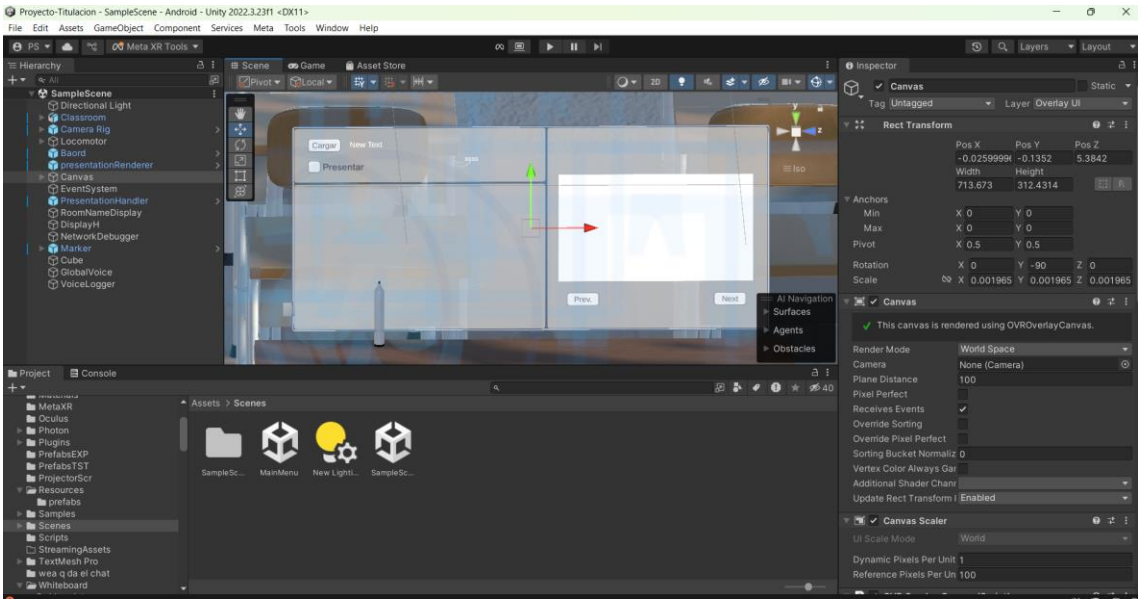

Fig. 6 Interfaz de usuario para carga de presentaciones

Procesamiento de archivos de presentación: El sistema distingue dos casos:

1. **Colección de imágenes sueltas:** Si el usuario seleccionó múltiples imágenes (o incluso una sola imagen estática), se asume que cada imagen corresponde a una diapositiva. Internamente, el sistema recorre cada archivo de imagen, lee sus bytes y los carga en una textura Unity (Texture2D.LoadImage), obteniendo así una lista de Texturas en memoria. Estas texturas luego serán usadas directamente para mostrar las "diapositivas". En esta modalidad, el orden de las imágenes podría seguir el orden de selección o alfabético por nombre de archivo.
2. **Archivo PDF o PPTX:** Si el usuario seleccionó un documento multi-página (como un PDF o una presentación de PowerPoint), Unity por sí solo no tiene soporte nativo para extraer su contenido. Para solventar esto, se implementó una solución basada en un servicio externo de conversión. En concreto, se preparó una integración con un *servicio web* de desarrollo personal hospedado en Azure (Actualmente disponible gratuitamente con el plan *Free* con un límite de uso de 1h de uso de procesador al día) que recibe un archivo PDF/PPTX y devuelve una

serie de imágenes (por ejemplo, PNG) representando cada página o diapositiva del documento. El flujo es el siguiente:

- Cuando el usuario elige un PDF/PPTX, la aplicación lee el archivo binario y lo envía mediante una petición HTTP POST a una API de conversión (esta API, previamente desarrollada, procesa el documento con herramientas de servidor para extraer cada página como imagen PNG).
- La API devuelve, en caso de éxito, un JSON con un identificador de sesión único (session_id) y la cantidad de diapositivas procesadas, o bien las imágenes mismas en base64. En la implementación realizada, se optó por que el servidor devuelva la lista de imágenes codificadas en base64, o un ID para luego recuperarlas por partes.
- La aplicación entonces decodifica esas cadenas base64 a texturas (similar a cargar imágenes) o utiliza el ID de sesión para descargar las imágenes una a una. En nuestro caso particular, el servidor Azure provee dos endpoints: uno para subir (upload_base64) y recibir de vuelta session_id y slide_count, y otro para descargar (getSlideBaseUrl/session_id/slide_X.png) cada imagen mediante su índice. El flujo implementado fue: enviar imágenes (ya sea directamente si eran imágenes locales o convertidas por el servidor en caso de PDF/PPTX) al endpoint de upload, obtener session_id y slide_count, luego almacenar esos valores y cargar la primera diapositiva.
- Todas estas operaciones (envío de archivo, espera de respuesta, descarga de imágenes) se realizan de forma asíncrona utilizando corutinas de Unity (StartCoroutine). Así, la aplicación no se congela durante la conversión,

sino que muestra un breve estado de carga hasta que las diapositivas estén listas.

Una vez se tienen las diapositivas en formato de texturas, el sistema actualiza la interfaz: la variable interna totalSlides se establece al número total de imágenes obtenidas y se inicializa un índice imagenActual = 0. El nombre del archivo cargado se muestra en la UI (archivoActualText). Automáticamente se muestra la primera diapositiva en una vista previa.

Visualización de las diapositivas: Para mostrar las diapositivas al usuario, se emplearon dos métodos simultáneamente:

- **Vista previa en 2D (canvas):** En el menú de usuario (que puede estar anclado a la mano o fijado en la vista), se colocó un componente RawImage de Unity. La textura de este RawImage se actualiza para mostrar la diapositiva actual que el usuario tiene seleccionada. Esto sirve como referencia cercana para el usuario, permitiendo leer el contenido en alta resolución desde su HUD sin tener que mirar hacia la pantalla grande todo el tiempo.
- **Proyección en la pantalla virtual:** En el aula, se dispuso un **plano de proyección** (por ejemplo, una gran superficie en la pared frontal, al lado o sobre la pizarra, simulando una pantalla de proyector desplegable). Este plano tiene un material cuyo mainTexture se cambia dinámicamente para mostrar la diapositiva actual, de forma que todos en la escena (en este caso solo el usuario local) puedan verla en tamaño grande. En modo local, esto es redundante con la vista previa, pero se establece pensando en el modo multijugador donde otros usuarios verán esta pantalla.

El toggle de presentación que mencionamos en la interfaz permite activar o desactivar la visibilidad del plano de proyección en el mundo. Internamente, esto se

implementó modificando la transparencia del material del plano: cuando el toggle está apagado (no se desea mostrar la pantalla en el mundo), se ajusta el material para que sea completamente transparente (alpha = 0) haciendo el plano prácticamente invisible. Cuando se activa, se restaura la opacidad (alpha = 1) para que la pantalla aparezca mostrando la diapositiva. Esta característica es útil, por ejemplo, si el usuario quiere ocultar la pantalla grande momentáneamente o si en un futuro se desea alternar entre distintos modos de visualización.

Navegación entre diapositivas: Los botones "Anterior" y "Siguiente" permiten cambiar la diapositiva mostrada. Al presionar uno de ellos, el sistema calcula el nuevo índice imagenActual sumando o restando 1 (con límites entre 0 y totalSlides-1 para no salir de rango). Luego llama a una función que actualiza la visualización (MostrarImagenActual() en el código). Esta función básicamente toma el índice imagenActual y busca la textura correspondiente: dependiendo de la implementación, puede que las texturas ya estén todas cargadas en memoria en una lista, o en nuestro caso, para ahorrar memoria en archivos grandes, se implementó de forma que carga bajo demanda cada diapositiva cuando se navega. Dado que tenemos el sessionId y sabemos el índice, MostrarImagenActual() construye la URL al recurso (.../slide_X.png) y lanza una coroutine para descargar esa imagen si no estaba ya cargada. Cuando la descarga completa, la textura obtenida se asigna tanto al RawImage de preview como al material del plano de proyección. En caso de que las imágenes ya estuvieran pre-cargadas (por ejemplo, si eran pocas o pequeñas, podríamos haber almacenado en memoria la lista), la función simplemente asignaría la textura correspondiente del array prellenado.

En el modo local, la navegación ocurre instantáneamente para el usuario: al darle a siguiente/anterior, en una fracción de segundo se actualiza la imagen mostrada. Si hay un ligero retardo (por ejemplo, si requiere descargar la siguiente imagen), se puede

manejar mostrando un indicador de carga momentáneo, pero en nuestras pruebas con presentaciones de tamaño moderado la transición es lo suficientemente rápida.

5. CAPITULO 5

Integración multijugador con Photon Fusion 2.0.6

Tras haber desarrollado las funcionalidades básicas en modo local, el siguiente paso consistió en integrar un sistema multijugador que permitiera a varios usuarios conectarse al aula virtual inmersiva y compartir la experiencia en tiempo real. En este capítulo se detalla cómo se incorporó la red utilizando la tecnología Photon Fusion 2.0.6, logrando sincronizar avatares, escenas, presentaciones y demás interacciones entre múltiples participantes. Se explican los mecanismos empleados para la sincronización de estado (posición de usuarios, contenido de la pizarra, diapositivas mostradas, etc.), la gestión de permisos entre usuarios (por ejemplo, quién puede escribir o controlar la presentación) y cómo se mantuvo la cohesión del entorno a medida que cada usuario interactúa.

5.1 Arquitectura multijugador

Para dotar al proyecto de capacidades multijugador, se seleccionó Photon Fusion 2.0.6, una librería de red de alta performance especializada en la sincronización de estados en Unity. Photon Fusion es la evolución de las soluciones de Photon para Unity y se destaca por ofrecer soporte para diversos modelos de red (arquitectura cliente-servidor dedicada, host-cliente, autoridad compartida, etc.) bajo una misma API, facilitando el desarrollo.

Modelo de red adoptado: Dado el caso de uso de un aula virtual, se optó por una topología cliente-host manejada mediante Photon Fusion (ver Figura 7). Esto significa que uno de los clientes actúa también como host (anfitrión) o autoridad principal del

estado, y los demás clientes se conectan a su sesión. En la práctica, al iniciar la aplicación multijugador, el primer usuario que crea o entra a la sala asume el rol de host (State Authority en términos de Fusion), y los subsiguientes usuarios se unen como clientes normales. Photon Fusion se conecta a los servidores de Photon en la nube para facilitar el matchmaking: todos los clientes se registran bajo un mismo ID de sesión o sala, de forma que comparten el mismo espacio virtual. Este esquema fue elegido por ser suficiente para un entorno de aula con decenas de usuarios como máximo, evitando la complejidad de desplegar un servidor dedicado separado.

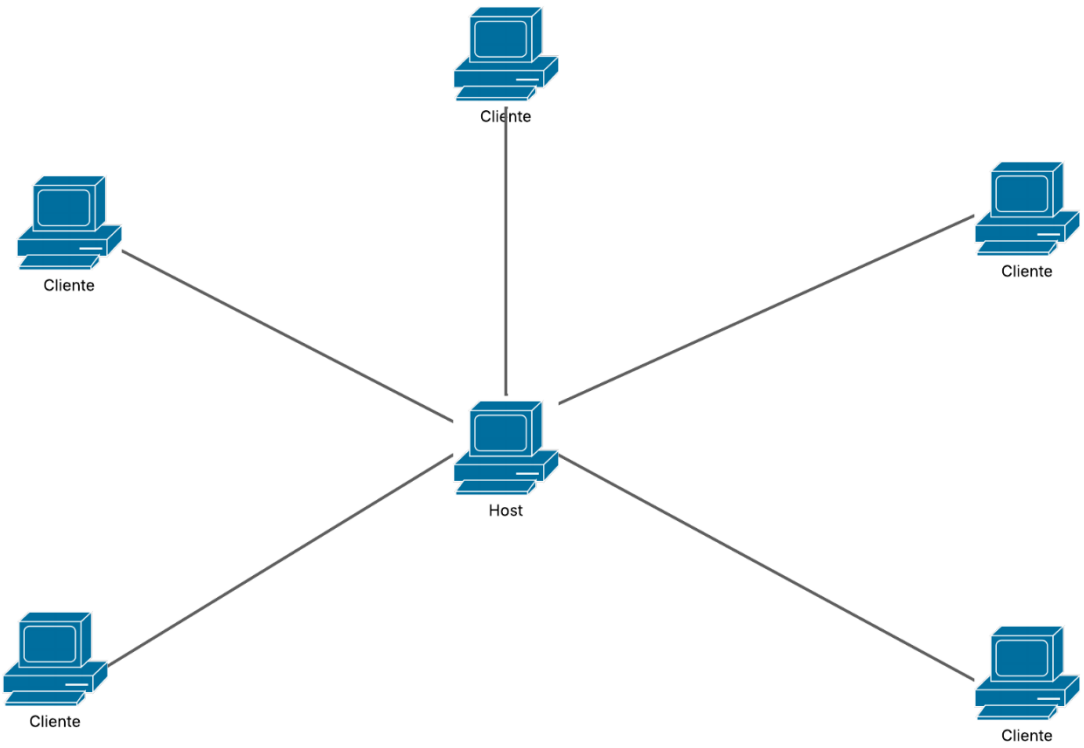

Fig. 7 Topologia de red Cliente-Host

La inicialización de la red se maneja mediante un NetworkRunner, componente central de Fusion que orquesta la sesión. Al lanzar el modo multijugador, el NetworkRunner se configura y ejecuta su método StartGame en modo *Host* o *Client* según corresponda. Por ejemplo, si el usuario decide "crear aula", el Runner inicia en modo Host

(creando una sala nueva), y si otro decide "unirse a aula existente", el Runner busca unirse como Client a la sesión designada. Una vez conectados, el NetworkRunner se encarga de sincronizar los *ticks* de juego entre los participantes y replicar los estados de los objetos marcados como *Networked*.

Prefabs en red y objetos de escena: En Unity, un *prefab* es una plantilla de objeto que almacena componentes y configuraciones predefinidas, permitiendo instanciar múltiples copias idénticas en la escena o en tiempo de ejecución. Photon Fusion opera instanciando objetos de red basados en prefabs predefinidos. Para nuestro proyecto, se prepararon varios prefabs de red:

- **Avatar del jugador**: un prefab que representa a un usuario dentro del aula. Este prefab incluye el modelo visual (cuerpo, cabeza, manos del avatar) y scripts para sincronizar su movimiento.
- **Marcador (plumón) en red**: el marcador virtual se convirtió en un objeto de red para que su posición y estado de agarre se sincronicen entre usuarios.
- **Gestor de Presentación**: un objeto que maneja la lógica de presentaciones (carga de archivos, cambio de diapositivas) en contexto multijugador, para coordinar que todos vean lo mismo.
- **Otros objetos interactivos**: en caso de haber otros elementos manipulables (p. ej., apuntador láser u objetos de la escena que se muevan), también se convirtieron en objetos de red.

Estos prefabs llevan el componente NetworkObject (requerido por Fusion) y scripts derivados de NetworkBehaviour que contienen la lógica sincronizada.

Al iniciar la sesión, Photon Fusion instancia automáticamente un avatar de jugador por cada cliente conectado. Esto se logró registrando el prefab del avatar en la configuración del NetworkRunner (ya sea en los *Assets* Resources o mediante la función de *Spawn* manual en código cuando un nuevo jugador se conecta). El resultado es que, cuando un usuario entra, todos los demás ven aparecer un nuevo avatar que lo representa, y viceversa.

5.2 Sincronización de avatares y movimiento multiusuario

Una vez que múltiples usuarios pueden unirse a la misma aula virtual, es fundamental que cada uno vea a los demás en la posición correcta y con los movimientos correctos. Esto implica sincronizar en red la posición y orientación de la cabeza y manos de cada usuario (su avatar), así como su desplazamiento por la sala.

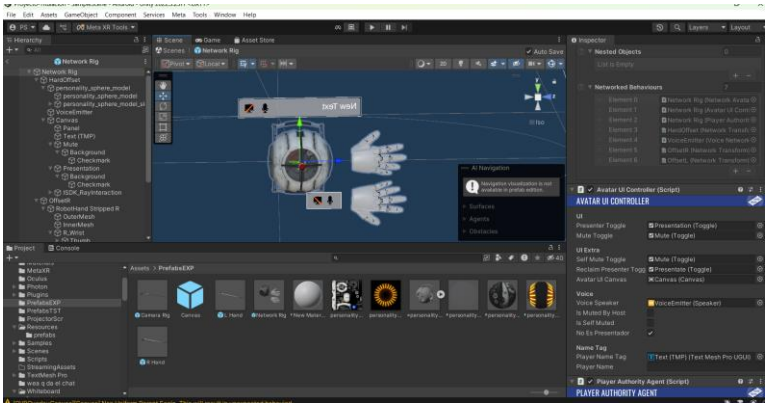

Fig. 8 Prefab de avatar

Instanciación del avatar de jugador: Cuando el NetworkRunner agrega un nuevo jugador (identificado por un PlayerRef único), el sistema invoca la creación de un objeto avatar asociado. Este avatar es una instancia de nuestro prefab de avatar de red (ver Figura 8), posicionado inicialmente en un punto de spawn predefinido (por ejemplo, la entrada del aula). Cada avatar contiene:

- Un modelo (puede ser un cuerpo genérico o solamente cabeza y manos).

- El script NetworkAvatarSync (sincronización de avatar en red), que hereda de NetworkBehaviour.
- Posiblemente, un componente de sincronización de transformaciones (como NetworkTransform o NetworkCharacterController) para la raíz del avatar.

Movimiento del jugador (locomoción) en red: El desplazamiento con el joystick (movimiento continuo) se integró con Photon Fusion de la siguiente manera: dado que en modo local ya usábamos el XR Rig con un CharacterController para mover al usuario, en multijugador mantuvimos ese esquema para el usuario local, pero la posición resultante se propagó a su avatar de red para que los demás la vean. Photon Fusion provee la clase NetworkCharacterController que extiende de NetworkBehaviour, la cual puede controlar y sincronizar a un CharacterController en red (incluyendo gravedad, velocidad, etc.). Cada avatar de jugador posee un NetworkCharacterController atado a un CharacterController. El usuario local controla su desplazamiento leyendo el input del joystick (en Update, por ejemplo, del XR Rig) y aplicando un movimiento a su CharacterController; simultáneamente, en el contexto de Fusion, eso se envía como Input al host y se reproduce para todos. De cualquier modo, el resultado es que cuando un usuario camina en su espacio VR, su avatar se mueve en el mundo virtual y los otros clientes ven dicho movimiento casi en tiempo real, con la latencia mínima. Fusion emplea interpolación para que los movimientos se vean suaves en los clientes receptores, incluso si hay pequeños retrasos, evitando saltos bruscos.

Sincronización de cabeza y manos (postura del avatar): Es necesario replicar la orientación de la cabeza y las manos para que las miradas y gestos de cada usuario sean visibles a los demás. Para esto se utilizó el script NetworkAvatarSync en cada avatar, que funciona así:

- Cada avatar de red tiene dentro de su jerarquía unos objetos vacíos que representan la cabeza, mano izquierda y mano derecha del modelo (estas son referencias que deben asignarse en el prefab a headAvatar, leftHandAvatar, rightHandAvatar).

- Cuando la instancia local de un avatar es la del propio usuario (Photon Fusion distingue cuál avatar corresponde al jugador local mediante *Input Authority*: el avatar cuyo Object.InputAuthority coincide con el jugador local), el script activa la sincronización. Es decir, solo se toman datos de la posición real para el avatar propio.

- El script busca en la escena las referencias reales del XR Rig local: la cámara principal (cabeza real) y los objetos de mano izquierda y derecha reales. El script busca GameObjects llamados "L Hand" y "R Hand" (nombres asignados a las manos del jugador local). Si las encuentra, establecerá un enlace entre los huesos de la mano real y los huesos del avatar.

- En cada frame (LateUpdate, idealmente después de que el XR Rig actualizó la posición de la cámara y manos), el script toma las transformaciones de la cabeza y manos reales y las copia a la cabeza y manos del avatar. Es decir, la cabeza del modelo avatar se mueve exactamente a donde está la cabeza del usuario, y lo mismo para las manos. Adicionalmente, el script mapea la rotación de las articulaciones de los dedos si están disponibles: mediante la integración con Autohand, se obtienen las rotaciones de los joints de cada dedo real (por ejemplo, articulaciones del pulgar, índice, etc.) y se aplican a los huesos correspondientes del modelo avatar. De este modo, si un usuario hace un gesto (como cerrar parcialmente la mano), su avatar reproducirá ese gesto en la animación de la mano. El código está preparado para mapear dedos en caso de disponer de esa

información (por ejemplo, usando los sensores de dedo en controladores Touch Pro, o con tracking óptico de manos).

- Estas actualizaciones de postura solo las realiza el usuario local sobre su propio avatar (ya que solo él conoce la posición real de su cuerpo). Photon Fusion entonces se encarga de replicar el estado de ese objeto avatar a los demás. El avatar es un NetworkObject y está siendo modificado por su dueño (input authority), Fusion transmite los cambios de transformaciones a todos los clientes. Para optimizar, se marcó el avatar para que use *State Authority* en el host; esto quiere decir que las posiciones en última instancia son validadas/autorizadas por el host, pero Fusion facilita que el propietario las actualice cada tick. En caso de retrasos, Fusion puede aplicar técnicas de prediction para la posición del avatar.

Al aplicar esta estrategia, cada participante ve en la sala:

- Un avatar para sí mismo. Generalmente no es visible para el propio usuario en primera persona, o solo se muestra parcialmente el cuerpo según la configuración. En nuestro caso, se ocultó la representación de las manos/cuerpo local para no estorbar la visión, dado que el usuario ya ve sus manos reales.

- Avatares para los demás usuarios, los cuales se mueven y animan en tiempo real, reflejando las acciones de cada participante. Por ejemplo, si el usuario A camina hacia la pizarra, el usuario B verá el avatar de A desplazarse hasta la pizarra; si A levanta su mano para escribir, B verá la mano virtual de A levantarse; y si A habla (si hubiese chat de voz) o indica algo, su avatar estaría en la posición correspondiente mirándola.

La precisión de la sincronización es alta: Photon Fusion, con su tasa de tick (configurable, por ejemplo 60Hz), permite actualizaciones frecuentes. Se configura la

tasa de envío de estado de los avatares de forma que fuera suficiente para no percibir retardo notable en VR (típicamente 30 o 60 veces por segundo). De esta manera se logra una experiencia compartida coherente, donde todos los usuarios ven prácticamente la misma escena desde sus respectivas perspectivas.

5.3 Consistencia de la escena y de los objetos interactivos

Con múltiples usuarios presentes, es crítico mantener la consistencia de la escena: todos deben ver el mismo entorno y los mismos cambios que ocurran en él. Esto abarca tanto objetos estáticos (la disposición de la sala, la posición de la pizarra y la pantalla, etc., que de por sí son iguales para todos al cargar la escena base) como objetos dinámicos o interactivos. Los objetos interactivos principales son el marcador y el contenido de la pizarra (los dibujos) y de la presentación proyectada.

Instanciación de la escena compartida: Utilizando Photon Fusion, al iniciar la partida en red, se puede optar por cargar una escena en todos los clientes. En nuestro caso, definimos la escena del aula como la escena de juego principal. Cuando el host crea la sesión, esa escena ya está cargada localmente; Fusion ofrece la opción de sincronizar la escena en los clientes al unirse (usando NetworkSceneManager que carga la misma escena en modo sincronizado). Así, cuando un cliente se une a la sala, se asegura automáticamente de cargar la escena "AulaVirtual" localmente. Al hacer esto, todos los objetos marcados como *Networked* en la escena (por ejemplo, si el marcador o la pizarra ya estuvieran colocados en la escena con NetworkObject) se sincronizan. También es común instanciar programáticamente algunos objetos de red. En nuestro diseño:

- La pizarra se mantuvo como parte fija de la escena (no necesita instankjciarse dinámicamente para cada cliente, simplemente existe en la escena y todos la tikm,enen).

- El marcador inicialmente es único y existe en la escena. Lo marcamos como NetworkObject para que Photon Fusion lo rastree. Como es un objeto que puede moverse, lo gestionamos con una lógica de red especial (ver más adelante).
- La pantalla de proyección también es un objeto en la escena activable/desactivable (su material cambia transparencia).

Sincronización del marcador (objeto físico compartido): En un aula real solo hay un cierto número de rotuladores; aquí conceptualizamos un único marcador compartido que los usuarios pueden pasarse. Para esto, el marcador virtual se convirtió en un objeto de red interactivo:

- Se aplicó el script NetworkedGrabbableState (como se muestra en Código) al objeto marcador. Este script asegura que el objeto solo sea simulado físicamente (Rigidbody no-kinemático) en el cliente que tenga la *State Authority*. En Photon Fusion, por defecto, el host podría ser la autoridad de estado de la mayoría de objetos. Sin embargo, para un objeto agarrable, conviene transferir la autoridad al usuario que lo toma, para que ese cliente pueda moverlo directamente y luego sincronizar la posición a otros (reduciendo latencia en la percepción de quien lo mueve). Fusion permite cambiar la autoridad de un NetworkObject dinámicamente.
- Al implementar el agarre del marcador en multijugador, se hizo que al momento en que un usuario coge el marcador, se invoque un RPC o función de Fusion para asignarle a ese cliente la autoridad del objeto. En la práctica, como usamos Fusion, una opción fue: el host, al detectar que el marcador fue agarrado por X jugador, llama markerObject.RequestStateAuthority(playerRef) para transferir la autoridad. Alternativamente, se pudo diseñar que el host siempre sea autoridad y

simplemente reciba la posición de la mano del jugador y mueva el objeto (pero eso agrega latencia en la vista del que agarra). Se da autoridad al cliente que agarra para máxima responsividad local.

- Mientras un jugador sostiene el marcador, su movimiento (posicional y rotacional) se envía a los demás automáticamente, porque ese jugador está moviendo un objeto networked del cual tiene control. Otros clientes ven el marcador en la mano de esa persona moviéndose congruentemente.

- Al soltar el marcador, la autoridad puede revertir al host o quedarse donde cayó. Implementamos que el host recupera la autoridad al soltar (para que el objeto quede bajo física controlada centralmente si cae al suelo, por ejemplo). De esta manera, la gravedad y colisiones del marcador suelto se simulan en el host (garantizando una única versión de la "verdad" de dónde quedó) y luego su posición final se sincroniza a todos.

Sincronización del dibujo en la pizarra: Los trazos dibujados en la pizarra por un usuario deben aparecer en las pizarras de todos los demás en tiempo real. La textura de la pizarra es local a cada cliente, así que necesitamos difundir los eventos de dibujo. Para ello, se utilizó un enfoque basado en RPCs de Fusion: cada vez que un usuario dibuja un trazo nuevo con el marcador, en lugar de tratar de enviar la textura entera (lo cual sería inviable por ancho de banda), se envían los puntos o segmentos dibujados.

- Se modificó el script del marcador/pizarra para detectar cuándo iniciar y cuándo finalizar un trazo (por ejemplo, OnCollisionEnter con la pizarra marca "comenzó a dibujar", OnCollisionExit marca "terminó"). Durante el dibujo (mientras el marcador está en contacto y moviéndose), se recolectan las posiciones UV de los puntos dibujados.

- Periódicamente (o al terminar cada segmento corto), se envía una RPC a *todos los clientes* que contiene la información necesaria: puede ser la coordenada inicial y final del segmento, el color y grosor. Por ejemplo, una RPC RPC_DibujarSegmento(float u1, float v1, float u2, float v2, Color color) podría ser invocada desde el cliente que está dibujando (que tiene input authority de su avatar, pero quizás el host tenga state authority de la pizarra; aún así, Fusion nos permite enviar RPCs from All to All).

- Al recibir la RPC, cada cliente toma esos parámetros y ejecuta localmente la misma rutina de dibujo en su propia textura de pizarra. Es decir, llaman a Texture2D.SetPixels en los píxeles correspondientes del punto/segmento recibido y hacen Apply(). De este modo, replican el trazo. Para que el resultado sea consistente, todos los clientes deben partir de la misma textura base y aplicar los mismos cambios en el mismo orden. Fusion garantiza el orden de RPCs tal como fueron enviadas si se usa apropiadamente (puede configurarse como *Reliable Sequenced*).

- En nuestro caso, como optimización y simplicidad, designamos que solo un usuario a la **vez** dibuja (en la práctica, el presentador o quien tenga el marcador). Así no hay dos flujos de dibujo simultáneos que podrían intercalar. Si se diera el caso, Fusion aún así podría manejarlo. El sistema de permisos, descrito más adelante, ayuda a garantizar esto.

Gracias a este mecanismo, cuando el profesor escribe en la pizarra, los estudiantes ven aparecer los mismos trazos casi instantáneamente en su copia de la pizarra virtual, como si estuvieran todos mirando la misma pizarra física. La latencia es muy baja (Photon

Fusion en redes locales o buenas conexiones puede estar en el orden de decenas de milisegundos), por lo que el dibujo es fluido.

Sincronización de la presentación compartida: El sistema de presentaciones, implementado en modo local, se extendió para que todos los usuarios vean las mismas diapositivas al mismo tiempo. Para lograr esto, se centralizó el control de la presentación en un rol de presentador y se utilizó el script PresentacionHandler adaptado con funciones de red:

- La variable presentadorActual fue marcada con el atributo [Networked] en el script. Esto significa que su valor (un identificador de jugador, PlayerRef) es replicado a todos los clientes y mantenido por Fusion. El presentadorActual inicial se establece en el host al iniciar (o específicamente en el usuario que cargó la presentación).
- Cuando el presentador (profesor) utiliza el menú para cargar un archivo de presentación, su instancia del PresentacionHandler realiza todo el proceso de conversión de archivos descrito en el Capítulo 4.4 (Sistema de presentaciones y proyección de contenidos). Una vez obtenidas las diapositivas (ya sea cargadas o tras recibir respuesta del servidor), en lugar de simplemente mostrarlas localmente, ahora se debe notificar a todos los demás de esta nueva presentación. Para ello se implementó una llamada RPC RPC_BroadcastPresentacion(sessionId, slideCount, fileName) que se invoca desde el presentador hacia *All*. Esta RPC distribuye los metadatos necesarios:

Tabla 4

Metadatos enviados via RPC

Elemento	*Descripción*
sessionId	Identificador único de la sesión de conversión o identificador compartido de las imágenes. Se utiliza para que todos los clientes puedan descargar la misma presentación desde el servicio externo.
slideCount	Número total de diapositivas. Permite a los clientes sincronizarse desde la diapositiva 0 y avanzar de forma coordinada en toda la sesión.
fileName	Nombre del archivo de la presentación (opcional). Puede mostrarse en la interfaz de usuario como referencia, y se emplea junto con sessionId para descargar las imágenes desde la nube.

Cuando los clientes reciben esta RPC, su código asigna esos valores localmente (actualiza su sessionId y totalSlides) y automáticamente llama a MostrarImagenActual() para cargar y mostrar la primera diapositiva. En esencia, esto inicializa la presentación en todos los clientes simultáneamente con la misma diapositiva 0. Cada cliente realiza la descarga de la imagen de la diapositiva 0 desde el servidor (usando el sessionId compartido). Todos los clientes deben tener acceso al mismo

servicio externo para obtener las imágenes; en nuestro caso, sí, todos pueden solicitar a la URL del Azure Function las imágenes dado el mismo sessionId. Como las imágenes están almacenadas en la nube, cada cliente las descarga independientemente. Esto descentraliza la carga (no hace falta que el presentador envíe las imágenes pixel a pixel, solo comparte la referencia).

- Para navegar entre diapositivas, solo el presentadorActual tiene habilitados sus botones. Cuando él presiona "Siguiente" o "Anterior", su script ejecuta la función CambiarImagen(delta) la cual actualiza el índice local y luego invoca un RPC RPC_SetSlide(nuevoIndice) para *All*. Este RPC es simple: al llegar a cada cliente, asigna el imagenActual = nuevoIndice y llama a MostrarImagenActual(), provocando que cada uno cargue/visualice esa diapositiva. Así, si el profesor avanza a la diapositiva 5, instantáneamente todos saltan a la diapositiva 5.
- La sincronización incluyó también el toggle de visibilidad de la pantalla: si el presentador decide mostrar/ocultar la pantalla grande, esa acción se envía mediante RPC_TogglePlanoPresentacion(bool activo) a todos, de forma que en todos los visores la pantalla virtual se encienda o apague al unísono.

Con este sistema, se garantiza que todos los participantes ven la misma diapositiva en todo momento, manteniendo la coherencia de la exposición. Incluso si un usuario se conecta tarde, Photon Fusion puede tener configurado que reciba el estado actual de las variables de red: por ejemplo, el valor actual de imagenActual y de sessionId al momento de unirse. Así, un recién llegado podría automáticamente cargarse en la diapositiva en curso. (Esto requiere que dichas variables estén sincronizadas correctamente como [Networked] properties; en el código, sessionId y totalSlides se mantienen vía RPC

broadcast, pero podríamos mejorarlo guardándolos también como [Networked] para hot-join).

En cuanto a la transferencia de los archivos en sí, dado que utilizamos un servicio externo, cada cliente que recibe la notificación va a obtener las imágenes por sí mismo. La solución híbrida adoptada (Photon para sincronizar comandos y URLs, HTTP externo para los datos pesados) resultó eficiente y escalable.

5.4 Sistema de Permisos

En el entorno virtual educativo se implementó un sistema de permisos que otorga al usuario anfitrión (host) control sobre las interacciones de los participantes (ver Figura 9). Para ello se incorporaron dos canvases de interfaz de usuario (UI) en cada avatar, cumpliendo roles diferenciados:

Canvas de etiqueta sobre la cabeza: Ubicado flotando encima del avatar, actúa como identificador visual con el nombre del participante (un "tag" con texto) e integra dos toggles (interruptores) que solo el host puede manipular. El primer toggle corresponde a la función de silenciar al usuario; cuando el anfitrión lo activa, el participante queda mudo para todos los demás, inhibiendo la transmisión de su voz. El segundo toggle permite asignar o revocar el rol de presentador al usuario. Este interruptor indica si el avatar tiene permisos de presentación (por defecto aparece activo con una "X" indicando no presentador, y al desactivarse por el host concede el permiso). Ambos controles son visibles sobre cada avatar para facilitar que el docente identifique el estado de cada participante (silenciado o presentador) de un vistazo. Sin embargo, por diseño solo el host posee permiso de interacción: en la implementación, estos toggles están marcados como no-interactuables para los clientes normales, de modo que, aunque todos puedan ver los iconos o su estado, únicamente el host puede accionarlos. Al activarse un

toggle por el host, se ejecuta la lógica de red correspondiente. De igual manera, al otorgar permisos de presentación a un participante, el sistema actualiza el estado de presentador actual en el gestor de presentación, asegurando que solo un avatar tenga el rol de presentador a la vez (el cambio desactiva la etiqueta de presentador en los demás). Esta sincronización se maneja mediante variables de estado en red (Networked Booleans) que se replican a todos los clientes, garantizando consistencia en la interfaz: todos los usuarios ven inmediatamente reflejado quién está silenciado o es el presentador vigente.

Canvas de menú personal en la muñeca: Cada avatar posee un segundo canvas fijado a la muñeca, visible solo para el propio usuario dueño de ese avatar. Este menú personal proporciona controles de auto-gestión al participante. Incluye un toggle de auto-silencio que el usuario puede activar para mutearse a sí mismo voluntariamente (por ejemplo, para evitar transmitir ruido de fondo). Al togglear esta opción, localmente se desactiva la transmisión de voz de su micrófono, simulando un "mute" personal sin afectar a los demás. Adicionalmente, el menú muestra un toggle de permiso de presentación que en principio aparece bloqueado para usuarios comunes y que únicamente el host puede habilitar remotamente. En el caso del anfitrión, este mismo control funciona como un botón de "reclamar presentación": el profesor puede activarlo en su propio menú de muñeca para reasignarse el rol de presentador en cualquier momento, revocándolo de quien lo tuviera. Para los estudiantes, este toggle sirve más como indicador de estatus – se actualiza su estado visual según si el host les ha concedido o no permiso de presentar, aunque ellos no puedan pulsarlo manualmente (ver Figura 10). La visibilidad y actualización de este canvas personal está rigurosamente controlada: cuando un usuario se conecta, el sistema activa solo el canvas de muñeca de su propio avatar (usando la autoridad de entrada del objeto en red), manteniendo ocultos o deshabilitados los menús personales de los demás. De esta forma, cada participante ve en

VR únicamente su propio panel de control en la muñeca, mientras que los toggles sobre las cabezas de otros avatares sirven de referencia global pero no invaden la vista personal.

Interacción en VR: La interacción con estos elementos UI se diseñó acorde a la realidad virtual, utilizando canvases en espacio mundial y las herramientas de interacción XR. El host, para accionar los toggles sobre otro avatar, puede apuntarlos con un puntero láser VR o acercar la mano controladora y pulsar el interruptor como si fuese un botón físico, aprovechando el raycast y las capacidades táctiles del XR Interaction Toolkit. Los canvases están configurados en modo World-Space con componentes de Graphic Raycaster que detectan la colisión del haz interactivo del controlador VR. De igual modo, el usuario puede levantar su mano y ver el menú de su muñeca para auto-silenciarse con un gesto natural, tocando el toggle correspondiente. Se tuvo cuidado en el posicionamiento y escala de estos UI para que fueran cómodamente accesibles: la etiqueta sobre la cabeza está a una altura visible, pero sin estorbar la vista, y el menú de muñeca aparece al voltear la mano hacia arriba, emulando un reloj inteligente. La lógica de permisos integrada impide interacciones no autorizadas: si un participante intenta pulsar el toggle de presentador de su muñeca (que no le corresponde activar), este estará deshabilitado y no producirá ningún evento. Solo cuando el host modifica los permisos, el cambio se refleja visualmente en el menú del usuario (por ejemplo, el toggle de presentador podría cambiar de estado o color indicando que ahora tiene privilegios de presentación).

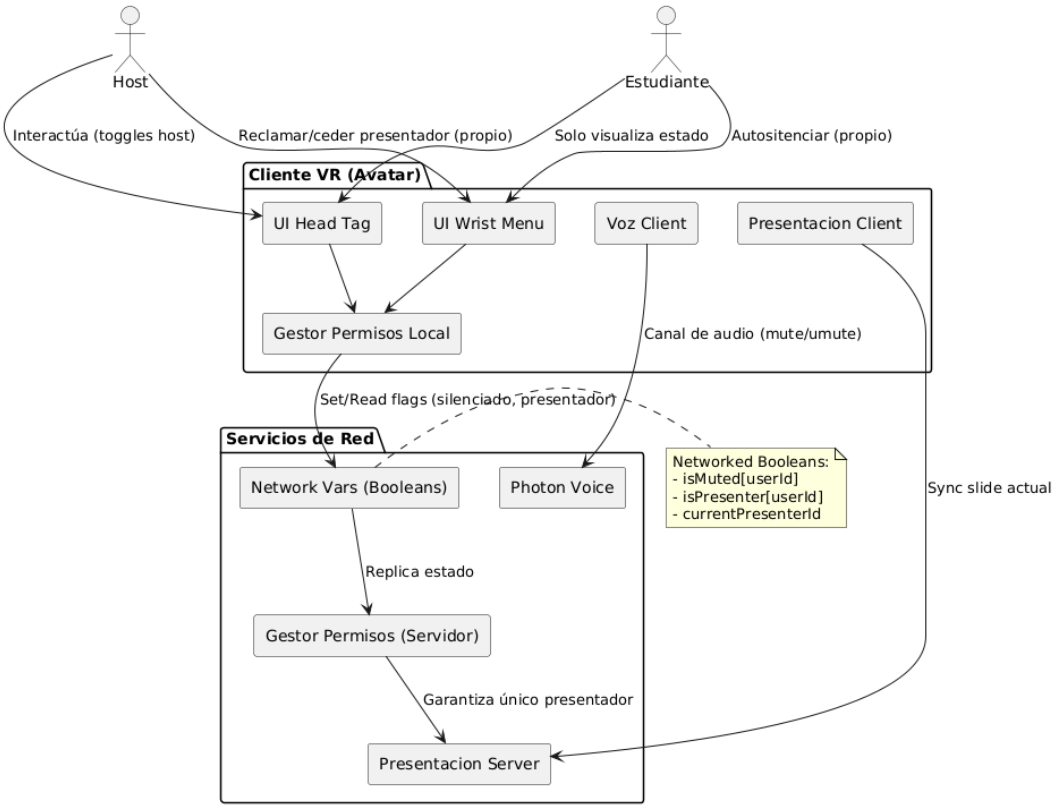

Fig. 9 Arquitectura general del sistema de permisos

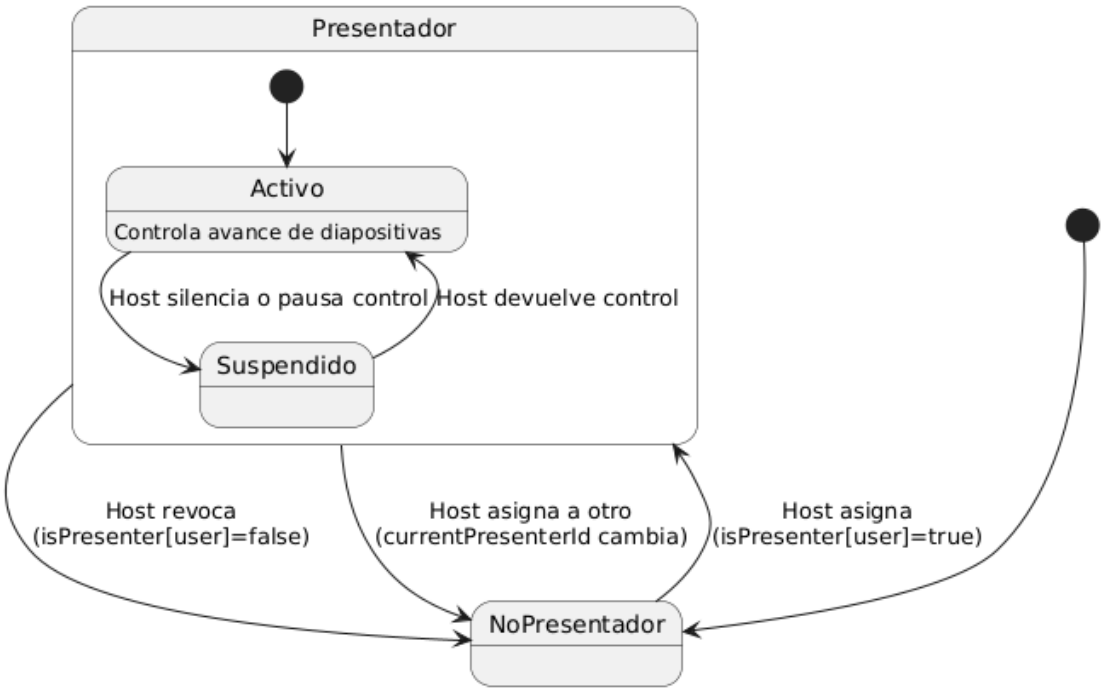

Fig. 10 Diagrama de estados del rol de presentador

5.5 Chat de Voz

Para reproducir la comunicación oral propia de una clase, el proyecto integró un sistema de chat de voz en tiempo real basado en Photon Voice, complementando la experiencia inmersiva. Photon Voice provee una infraestructura cliente-servidor especializada para transmisión de audio de baja latencia, la cual se aprovechó configurando canales de audio y componentes de voz enlazados a los avatares en la escena VR. En la práctica, todos los participantes que se unen a la sesión entran en un canal de voz común correspondiente al aula virtual (es decir, una misma sala de audio compartida). Cada cliente dispone de un componente Recorder (grabador/micrófono) que captura el audio de entrada de su dispositivo, y se le asigna un stream de voz identificado dentro de ese canal. Paralelamente, cada avatar en la escena incluye un componente Speaker (altavoz) ubicado en la cabeza del modelo 3D, encargado de reproducir el audio proveniente de la persona que dicho avatar representa. Photon Voice enlaza automáticamente cada flujo de voz entrante con el Speaker del avatar correcto usando la identidad de usuario en la sala: así, cuando un participante habla por su micrófono, los demás escuchan su voz emitida espacialmente desde la posición de su avatar. No se requirió segmentar múltiples sub-canales de audio dado que el escenario es una sola aula; todos comparten el mismo espacio auditivo. En la configuración actual, la prioridad fue la simplicidad y sincronía grupal, con un único canal unificado donde la voz de cada usuario es diferenciada y entregada a los correspondientes altavoces 3D en la escena. La integración al avatar es directa: en el prefab del jugador se añadió el componente Speaker de Photon Voice acoplado a un AudioSource Unity situado en la cabeza del avatar, con propiedades de audio espacial habilitadas (espacialización 3D, atenuación por distancia, etc.). Esto permite que la voz remota suene con procedencia direccional: si un estudiante habla, su voz se oirá más fuerte para quienes estén cerca virtualmente y atenuada para los

que estén lejos, recreando la dinámica sonora de un salón de clases real. Por otro lado, el Recorder de Photon Voice (asociado al micrófono local de cada usuario) se gestionó a través de un objeto de sistema. En los visores Meta Quest (dispositivos standalone Android), el sistema detecta y utiliza automáticamente el micrófono integrado del casco. Al iniciar la aplicación en Quest, Photon Voice emplea la API de Microphone de Unity para identificar el dispositivo de entrada predeterminado (el micrófono del visor) y solicita al usuario los permisos de acceso al micrófono la primera vez (esto se maneja mediante utilidades de Photon que invocan la ventana de permiso de Android para grabación de audio). Una vez concedido el permiso, el Recorder comienza a capturar sonido ambiente/voz del usuario. No hizo falta una selección manual de dispositivo dado que en Quest solo existe un canal de micrófono, pero el sistema está preparado para manejar múltiples dispositivos de entrada en plataformas donde aplique. Además, se incorporaron comprobaciones de estado para garantizar que el Recorder esté activo únicamente cuando debe: por defecto el Recorder transmite audio mientras el participante no esté silenciado y tenga permisos para hablar, caso contrario se desactiva su emisión. El control del micrófono según permisos se implementó vinculando el sistema de voz con el sistema de permisos descrito anteriormente. En concreto, cuando el host activa el toggle de “silenciar” sobre un avatar, inmediatamente en ese cliente silenciado se ejecuta una orden para deshabilitar su micrófono (el Recorder deja de transmitir) y, en paralelo, los Speakers de los demás desactivan el audio de ese usuario (garantizando que aunque hubiera audio en cola, nadie lo escuche). De forma similar, si el propio usuario decide mutearse mediante el toggle de su menú personal, el Recorder local se pone en modo mudo (TransmitEnabled = false) deteniendo el envío de voz al canal. En todo momento, solo el anfitrión tiene la autoridad para silenciar a otros o conceder/quitar el rol de presentador, lo cual se respeta en el código de interacción: cualquier cambio de estos

estados pasa por verificaciones de que quien lo ordena es el servidor/host antes de propagarse. El rol de presentador en sí no silencia automáticamente a los demás (es decir, varios pueden hablar a la vez si no están silenciados manualmente), pero en escenarios prácticos el moderador puede combinar ambas funciones. La gestión de estas reglas recae en el servidor Photon Fusion (host): este coordina que solo haya un presentador activo y aplica los mutes a nivel global, mientras que Photon Voice se encarga de rutear el audio conforme a esas configuraciones. Se puso especial atención en la retroalimentación visual para el usuario sobre el estado de su voz. En la interfaz de muñeca, el toggle de micrófono actúa como indicador: si el usuario ve el icono de micrófono cruzado (toggle activado en "mute"), sabe que está silenciado y que su voz no se transmite; si está en estado normal, su micrófono está abierto. Este indicador está directamente ligado al estado del Recorder: al apagarse la transmisión, el toggle cambia de estado (y podría cambiar de color o mostrar un símbolo de silencio, según las convenciones de la UI). De igual forma, cuando el host silencia a un participante de forma remota, el sistema podría notificarlo cambiando algún elemento visual – por ejemplo, manteniendo su toggle de micrófono forzado en "off" mientras dure el silencio impuesto – de modo que el usuario vea que ha sido silenciado por la moderación. En la etiqueta sobre la cabeza del avatar, el icono del toggle de mute sirve también de señalización para los demás: un compañero silenciado por el host podría mostrar un símbolo (como un micrófono tachado) visible para que todos sepan que ese usuario no puede hablar en ese momento. Estas pistas visuales refuerzan la conciencia del estado de audio de cada quien en el espacio virtual. En términos de latencia y rendimiento, Photon Voice está optimizado para comunicaciones en tiempo real, empleando codecs de audio (como Opus) que comprimen la voz manteniendo buena fidelidad con un retardo muy bajo (generalmente del orden de 100-200 ms dependiendo de la calidad de la conexión). Durante las pruebas, el audio de voz mostró una

sincronización aceptable con las acciones en VR; la ligera latencia inherente no impactó la interacción natural entre usuarios. La espacialización de audio contribuye a la inmersión sin añadir carga significativa: Unity maneja el posicionamiento 3D del sonido por hardware del Quest, por lo que múltiples fuentes de voz pueden sonar simultáneamente con cálculos de atenuación eficientes. No obstante, se vigiló el consumo de ancho de banda y CPU: cada flujo de voz supone tráfico de red, así que el sistema de permisos al permitir solo las voces necesarias (silenciando a participantes inactivos o ruidosos) ayuda también a reducir el uso global de datos y procesamiento de audio.

6. CAPITULO 6

Desafíos encontrados durante el desarrollo

En este capítulo se describen y analizan críticamente los principales desafíos enfrentados durante el desarrollo del entorno de aula virtual inmersiva, así como las decisiones tomadas para superarlos y los aprendizajes obtenidos en cada caso. La exposición se estructura en cinco áreas temáticas clave: la interacción física en realidad virtual, el procesamiento de presentaciones dentro del entorno, el manejo de la locomoción del usuario, la sincronización en red mediante Photon Fusion y el diseño modular del sistema. Cada sección a continuación aborda uno de estos ámbitos, detallando las dificultades encontradas, las soluciones adoptadas y las lecciones derivadas del proceso de desarrollo, todo ello en un tono técnico y objetivo.

6.1 Interacción física en VR

Implementar interacciones físicas naturales en realidad virtual fue uno de los primeros retos del proyecto. En particular, se requería que los usuarios pudieran agarrar objetos virtuales (como un marcador) y manipularlos con sus manos virtuales de forma realista. Para lograr esto, se optó por utilizar el paquete Auto Hand, un sistema de interacción VR basado en física que facilita el agarre de objetos con manos articuladas. La decisión de incorporar Auto Hand respondió a la necesidad de una solución robusta y probada para el manejo de manos físicas en VR, evitando reinventar mecánicas complejas de agarre. No obstante, su integración implicó comprender y ajustar ciertos parámetros (como fuerzas de articulación y fricción) para adecuarlos a nuestra aplicación educativa. En general, el uso de Auto Hand permitió que hacer un objeto grappable fuera tan sencillo como añadirle un componente, aprovechando un solver que previene atravesar objetos al agarrarlos y simula peso y colisiones de manera realista.

Uno de los retos más significativos consistió en el desarrollo de la escritura sobre la pizarra utilizando un marcador virtual. Para este propósito, se lanzó un raycast desde la punta del marcador hacia la superficie, proyectando el trazo sobre una textura determinada. Las primeras evaluaciones revelaron inexactitudes que provocaban líneas entrecortadas o pérdida de colisión. Estos inconvenientes se solucionaron mediante la interpolación entre las posiciones muestreadas en fotogramas contiguos y la modificación de la detección de colisiones. Además, se implementaron modificaciones en el grosor de la línea y la resolución de la textura con el objetivo de hallar un equilibrio entre la fidelidad gráfica y el rendimiento. Estas modificaciones resultaron en una escritura consistente y fluida, idónea para el ritmo de clase en línea. Por otro lado, para solventar el problema del marcador atravesando la pizarra, fue necesario afinar la detección de colisiones físicas. Se añadieron colliders apropiados tanto en la punta del marcador como en la superficie de la pizarra, y se ajustaron las capas de colisión de modo que el marcador interactuara con la pizarra pero no colisionara indebidamente con otros elementos del entorno o con la propia mano del usuario. En particular, se configuró el sistema de físicas de Unity para que la capa asignada al jugador (cuerpo y controladores VR) ignorara colisiones con la capa de objetos interactuables. También se incorporó un CharacterController al rig de la cámara para restringir el movimiento del usuario y evitar que atravesara paredes o introdujera el controlador más allá de los límites físicos esperados. Estas medidas mitigaron sustancialmente los comportamientos no deseados, como el traspaso de la punta del marcador a través de la pizarra, mejorando la confiabilidad de la interacción de escritura.

6.2 Procesamiento de presentaciones virtuales

Otro pilar del entorno educativo desarrollado fue la capacidad de mostrar presentaciones (diapositivas) dentro del aula virtual. En las primeras etapas del diseño se

consideraron distintas opciones para implementar esta funcionalidad, incluyendo la idea de integrar un navegador web embebido (WebView) que permitiera cargar directamente contenido HTML o visores en línea de archivos PDF/PPT. Sin embargo, tras investigar la viabilidad técnica, se decidió no usar un WebView en la aplicación debido a varios factores. Adicionalmente, interactuar con un contenido web dentro de VR introduce desafíos de usabilidad (mover un puntero como ratón, scroll de páginas, etc.) que se consideraron poco prácticos en el contexto de una clase virtual. En base a estas consideraciones, el equipo optó por un enfoque alternativo: convertir las presentaciones a imágenes estáticas y mostrarlas en una superficie dentro del mundo virtual.

De esta manera, todas las conversiones pesadas (interpretar PDF/PPTX, renderizar contenido) se realizan fuera del visor VR, evitando cargar al dispositivo con esa tarea. En la aplicación Unity, simplemente se maneja la descarga de las imágenes ya renderizadas y su mapeo a un material de Unity, lo cual es mucho más ligero.

Un punto crítico de esta implementación fue garantizar que todos los usuarios vean la misma diapositiva al unísono en un contexto multiusuario. Para ello, se estableció la noción de un "presentador" actual: solo el usuario designado como presentador puede avanzar o retroceder las diapositivas, y dichas acciones se replican en los demás clientes mediante llamadas de procedimiento remoto (RPC).

Entre los desafíos enfrentados en esta área estuvo el manejo de múltiples imágenes de alta resolución sin agotar la memoria ni provocar latencia perceptible al cambiar de diapositiva. La estrategia adoptada fue cargar bajo demanda: inicialmente se muestra la primera diapositiva y las siguientes solo se cargan cuando el usuario las solicita (al presionar "siguiente"), liberando la anterior si ya no es necesaria. También se limitó el tamaño de las texturas para que, aunque legibles, no excedieran una resolución óptima para el Quest. Durante el desarrollo se comprobó que esta solución de imágenes, si bien

supone un pre-procesamiento, ofreció un rendimiento adecuado en el visor, con transiciones de diapositiva en unos pocos segundos dependiendo de la conexión de red. En suma, la decisión de descartar el WebView y convertir las presentaciones a imágenes resultó acertada para los objetivos del proyecto, permitiendo integrar de manera efectiva los materiales visuales de clase en el mundo virtual de forma sincronizada y con un control total sobre cómo se muestran.

6.3 Manejo de la locomoción del usuario

La manera en que los usuarios se desplazan dentro del aula virtual fue otro tema de diseño importante. En experiencias de realidad virtual existen principalmente dos enfoques de locomoción: la *teletransportación* (donde el usuario apunta a un destino y aparece instantáneamente allí) y el *movimiento continuo* usando el joystick o pad, similar al desplazamiento en videojuegos tradicionales. Cada método tiene ventajas y desventajas bien documentadas. La teletransportación sobresale por reducir el mareo de movimiento, ya que evita el flujo visual continuo que genera conflicto vestibular –de hecho, es conocida por minimizar la cinetosis al eliminar las aceleraciones visuales–. Sin embargo, este método segmenta la experiencia, ya que el usuario "salta" de un lugar a otro sin recorrer el espacio intermedio, lo cual rompe la sensación de continuidad e incluso puede impactar la conciencia espacial del participante. Por el contrario, el movimiento suave con joystick mantiene la inmersión al permitir traslados fluidos y precisos, a costa de potencial incomodidad física si no se implementan las salvaguardas necesarias. Dado el contexto de una aula virtual multiusuario, en el proyecto se tomó la decisión de utilizar locomoción continua y rechazar la teletransportación. La razón principal fue preservar la coherencia visual y social: con movimiento continuo, todos los participantes pueden ver a los avatares de los demás desplazarse de forma natural por la sala, lo que imita mejor la

experiencia de un aula real (donde uno ve a otros caminar, no aparecer súbitamente en otro lugar).

En la implementación del movimiento con joystick, el mayor desafío fue lograr un desplazamiento cómodo y estable. A pesar de que el Continuous Move Provider del XR Toolkit solucionaba la locomoción básica, se requería modificar los parámetros de aceleración, velocidad y colisiones a través de un CharacterController para impedir que el usuario atravesara obstáculos. Además, se ajustó la suavidad del movimiento para minimizar las molestias para los usuarios sensibles a la realidad virtual. Estas modificaciones hicieron posible conservar una experiencia realista y responsiva, dejando para fases posteriores la incorporación de opciones avanzadas de confort como teletransporte o reducción de campo visual.

De esta decisión de locomoción se extrajeron varias enseñanzas. En primer lugar, el equilibrio entre inmersión y comodidad es delicado: elegir movimiento continuo aportó una mayor sensación de presencia y realismo grupal, confirmando que para una aplicación colaborativa como un aula virtual, vale la pena mitigar el mareo de otras formas antes que comprometer la cohesión espacial entre usuarios. Al mismo tiempo, se reafirmó la importancia de proporcionar cierto grado de adaptación al usuario; aunque no se implementó aún, el simple hecho de planificar opciones de comodidad mostró ser necesario para atender la diversidad de los participantes. Desde el punto de vista técnico, la experiencia subrayó que los componentes estándar de Unity para locomoción VR son un buen punto de partida pero requieren integración con la lógica de colisiones del entorno específico (piso, muebles, límites de aula). Finalmente, cabe mencionar que la elección de locomoción tuvo también implicaciones en la sincronización de red: el movimiento continuo genera un flujo constante de actualizaciones de posición para cada avatar, lo que supuso un mayor tráfico en comparación con teletransportes esporádicos.

Esto fue tenido en cuenta en la configuración de Photon Fusion para garantizar que el movimiento de los jugadores se replicara suavemente en todos los clientes sin saturar la red. En suma, el manejo de la locomoción mediante joystick cumplió el objetivo de permitir desplazamiento libre e inmersivo en el aula virtual, a la vez que sentó las bases para abordar de manera informada el desafío de la comodidad en VR.

6.4 Sincronización en red con Photon Fusion

Implementar la interacción multiusuario en tiempo real añadió otra capa de complejidad al desarrollo. Se seleccionó Photon Fusion como motor de red para sincronizar a múltiples participantes en la misma aula virtual, lo que introdujo consideraciones sobre autoridad de objetos, latencia y consistencia del estado compartido. Uno de los primeros conceptos clave manejados fue el de *ownership* (propiedad o autoridad) sobre los objetos de la escena. Para ello se aprovechó la separación de *Input Authority*: cuando un usuario necesita manipular un objeto (por ejemplo, tomar el marcador para escribir), el host le asigna la autoridad de entrada para ese objeto, permitiéndole enviar sus comandos de movimiento al servidor. En la práctica, esto se implementó mediante un mecanismo de transferencia de autoridad al agarrar: al producirse el evento local de agarre de un objeto (detectado por Auto Hand), se invoca una función que solicita asignar el Input Authority del objeto al jugador correspondiente. Solo el host (como dueño del estado) puede conceder dicha asignación, y Photon Fusion expone métodos seguros para ello, de forma que no haya dos jugadores con control simultáneo sobre un mismo objeto. Esta decisión de diseño –mantener la autoridad de estado en el host y otorgar autoridad de entrada temporal a clientes según necesidad– ayudó a evitar conflictos de concurrencia, garantizando que siempre hubiera un árbitro (el host) resolviendo el estado final de cada interacción crítica.

A pesar de este modelo bien definido, surgieron desafíos importantes de sincronización durante el desarrollo. Uno de ellos fue el manejo de la latencia y el efecto de posesión percibido por el usuario que agarra un objeto. Cuando un cliente tomaba el marcador, se esperaba que lo sintiera responsivo en su mano; sin embargo, bajo el esquema autoritativo, sus movimientos debían primero viajar al host y luego replicarse a todos, incluido de regreso a sí mismo. Esto introdujo un ligero retraso que podía hacer que el usuario percibiera el marcador con retardo o temblor (jitter) al moverlo rápidamente. Para minimizar este efecto, se aplicaron varias medidas. En primer lugar, se configuró el objeto marcador con interpolación de movimiento en los clientes, suavizando la trayectoria recibida desde el servidor. En segundo lugar, se estableció el objeto como cinemático (Kinematic, ignora simulaciones de fisicas) en todos los nodos menos en el que poseía autoridad. Así, únicamente el host simulaba la gravedad y las colisiones, mientras los demás recibían actualizaciones. Además, se investigó la implementación de la predicción de movimiento en el cliente con control de entrada; sin embargo, se utilizó con precaución para evitar inestabilidades visuales. Con estos ajustes se consiguió un desempeño aceptable, permitiendo que la escritura colaborativa se mantuviera estable pese al leve retraso inevitable en usuarios remotos.

Otro aspecto crítico fue la sincronización de la pizarra y sus dibujos en un entorno multiusuario. A diferencia de un objeto físico como el marcador (que tiene transformaciones claras a sincronizar, como posición y rotación), la pizarra contiene un estado más complejo: la textura con los trazos dibujados. Transmitir directamente la textura de la pizarra por la red cada vez que alguien dibuja habría sido inviable por el gran ancho de banda requerido. Por tanto, se adoptó una solución ingeniosa: aprovechar la propia sincronización del marcador para replicar el acto de dibujar *sin* enviar la textura. En concreto, el marcador ya está sincronizado en posición mientras un usuario escribe;

cada cliente local ejecuta el algoritmo de dibujo (raycast y pintado de textura) cuando detecta que el marcador toca su pizarra local. En esencia, cada participante dibuja en su copia local de la pizarra siguiendo el movimiento sincronizado del mismo marcador compartido. Este enfoque supuso que todos los usuarios ven aparecer los trazos casi simultáneamente mientras alguien escribe, sin que el dibujo en sí viaje por la red, sino solo la posición del marcador (que ya se transmite continuamente). Si bien esta técnica funcionó de manera eficiente, introdujo una limitación: un usuario que se una tarde a la sesión no tendrá los trazos previos en su pizarra, ya que no estuvo presente cuando se dibujaron. En la versión actual, la pizarra no conserva un estado histórico que se pueda enviar a nuevos clientes (por ejemplo, no se reenvía la textura completa al conectarse alguien). Se consideró que, dado el alcance del prototipo, esta desventaja era asumible –los participantes que ingresan desde el inicio comparten plenamente la experiencia, y en un contexto de clase real generalmente todos empezarían juntos–. No obstante, el hecho destacó un aprendizaje: para lograr persistencia completa en elementos compartidos (como una pizarra llena de anotaciones) sería necesario implementar mecanismos adicionales, ya sea sincronizando la textura de vez en cuando o guardando vectores de dibujo que se repliquen a nuevos usuarios. Esta mejora quedó identificada para trabajos futuros, entendiendo que implica un trade-off entre consumo de red y consistencia total del estado.

Durante la implementación de Photon Fusion, también hubo que enfrentar y resolver diversos problemas de consistencia entre host y clientes. En pruebas iniciales se detectaron casos en los que ciertas acciones solo se reflejaban correctamente en el host y no en los clientes, o viceversa. Por ejemplo, en un principio, si un cliente intentaba activar una funcionalidad privilegiada (como cargar una nueva presentación), su interfaz podía mostrar el cambio mientras que otros no lo veían porque la lógica no había sido validada

por el host. Este tipo de situación se abordó reforzando la regla de oro de las arquitecturas cliente-servidor: *solo el host orquesta los cambios globales*. Se refactorizó la interacción de carga de presentaciones para que el cliente solicitara al host realizar la acción, en lugar de ejecutarla localmente. De igual manera, se revisaron todas las RPC y funciones de sincronización para asegurarse de que estaban siendo llamadas con las fuentes y objetivos correctos (por ejemplo, usar RpcTargets.All cuando debía llegar a todos, o RpcTargets.StateAuthority cuando solo debía ejecutarla el host). Una vez corregido esto, la sincronización entre host y clientes fue mucho más confiable. Esta experiencia dejó como lección la importancia de diseñar cuidadosamente la lógica de red, definiendo qué tipo de acciones pertenecen al lado servidor y cuáles pueden permitirse en el lado cliente. Además, subrayó la necesidad de probar el sistema con múltiples dispositivos en diferentes roles para detectar discrepancias: muchas veces, un desarrollador probando solo como host o solo como cliente en el editor no percibe ciertos errores hasta simular la interacción real entre ambos.

6.5 Diseño modular del sistema

A medida que el proyecto fue integrando todas las funcionalidades anteriores – interacción VR, presentaciones, locomoción, red, etc.– se volvió crucial adoptar un diseño modular que permitiera mantener la claridad y manejabilidad del sistema. Inicialmente, el desarrollo comenzó con un prototipo *offline* (para un solo usuario, sin red) donde se implementaron las interacciones básicas. Al incorporar la capa de red, se identificó el riesgo de mezclar lógica de multiplayer con la lógica base de la aplicación, lo cual podía generar confusión y dificultades de depuración. Para evitar esto, se tomó la decisión de separar en la medida de lo posible los componentes offline de los online, siguiendo principios de arquitectura modular y de separación de preocupaciones. En la práctica, esto

significó crear scripts y objetos distintos para funcionalidades locales versus las sincronizadas. Por ejemplo, la pizarra y el marcador tienen comportamientos locales (el dibujo de la textura, la detección de colisiones con la punta del lápiz) que residen en scripts independientes de aquellos que gestionan la sincronización en red (como el script que maneja la transferencia de ownership del marcador o los RPC de la presentación). De este modo, el sistema podía ejecutarse en modo unijugador sin inicializar ningún componente de red –útil para pruebas rápidas del entorno VR o para casos de uso en solitario–, mientras que para habilitar el modo multiusuario simplemente se instanciaban o activaban los módulos de networking pertinentes.

Un acierto particular fue la creación de puentes de comunicación entre módulos en lugar de dependencia directa. Un caso ilustrativo es la integración de Auto Hand con Photon Fusion: en vez de modificar el código del sistema de manos para que invocara funciones de red (lo cual habría acoplado demasiado ambas partes), se desarrolló un componente *bridge* separado, HandFusionBridge, que se adjunta a las manos virtuales. Este componente escucha los eventos genéricos de Auto Hand (por ejemplo, el evento de "objeto agarrado") y desencadena las acciones de red necesarias, como asignar la autoridad de entrada del objeto agarrado al jugador local mediante Fusion. De esta forma, la lógica de interacción (agarre en sí mismo) permanece independiente y limpia, y la lógica de red se mantiene encapsulada en el puente, pudiendo modificarse o desactivarse sin alterar el funcionamiento base de agarrar objetos en VR. Se aplicó un enfoque similar para el manejo de presentaciones: el módulo de UI/archivo opera localmente cargando imágenes, y por separado un módulo de red se encarga de sincronizar el cambio de diapositiva entre usuarios (vía RPC).

El diseño modular también favoreció la extensibilidad y mantenibilidad del código. Al delimitar componentes con responsabilidades únicas, el equipo pudo trabajar

en paralelo en distintas facetas del proyecto sin pisar la lógica unos de otros. Por ejemplo, mientras una parte del equipo afinaba la física del marcador y la calidad del trazo en la pizarra, otra podía encargarse de los scripts de Photon Fusion, definiendo cómo se instancian los objetos de red y se reparten las autoridades. La interfaz entre estos subsistemas estaba bien definida (eventos, llamadas RPC, etc.), de modo que cambios internos en uno no rompían al otro siempre que respetaran el contrato. Un beneficio evidente se manifestó en la corrección de errores: si surgía un problema en la sincronización de un objeto, se podía aislar rápidamente si el origen estaba en el código de red o en la lógica local, gracias a que no estaban entremezclados en un mismo monolito. Además, esta estructura prepara el camino para futuras expansiones o refactorizaciones. Por ejemplo, si en un futuro se decidiera cambiar de framework de red (suponiendo migrar de Photon Fusion a otra solución), la afectación se limitaría principalmente a los módulos de networking, mientras que los sistemas centrales de interacción (movimiento, agarre, dibujo, UI de presentaciones) permanecerían prácticamente intactos. De forma similar, si se quisiera reutilizar este entorno en un modo estrictamente unijugador o con un distinto modelo de red, la modularidad ofrece la flexibilidad de habilitar o deshabilitar componentes según la necesidad.

7. CONCLUSIONES

El desarrollo del *Sistema de Aula Virtual Inmersiva* ha logrado replicar de manera satisfactoria la experiencia de un aula presencial dentro de un entorno de realidad virtual. Utilizando la plataforma Unity orientada a dispositivos Meta Quest, se construyó un espacio tridimensional interactivo en el cual docentes y estudiantes pueden interactuar en tiempo real a través de avatares y objetos virtuales. Este resultado cumple el objetivo principal del proyecto al proporcionar un ambiente inmersivo que simula la dinámica de una clase tradicional con soporte tecnológico avanzado.

A lo largo del proyecto se implementaron diversas funcionalidades clave que sustentan la experiencia educativa virtual. En primer lugar, se desarrolló un entorno tridimensional completo que incluye elementos como mesas, pizarras y proyección de contenidos, creando un espacio familiar y navegable para los usuarios. Destaca la incorporación de una pizarra virtual interactiva, en la cual es posible realizar escritura en tiempo real empleando un marcador virtual; esta herramienta permite que profesores o estudiantes escriban y dibujen en el mundo virtual de forma natural, facilitando la ilustración de conceptos tal como se haría en una pizarra física. Asimismo, se integró un sistema de presentaciones virtuales, capaz de mostrar secuencias de diapositivas o imágenes dentro del aula inmersiva, de modo que el docente puede proyectar material didáctico a todos los participantes. Complementando estas funciones, el sistema soporta el desplazamiento libre mediante joystick, permitiendo a los usuarios moverse por el entorno virtual usando el stick de los controladores Meta Quest; esta locomoción continua mejora la exploración del espacio y la sensación de presencia, al dar libertad de movimiento similar a caminar por un aula real.

Además de las funciones individuales, el proyecto logró incorporar capacidades multijugador utilizando la tecnología Photon Fusion, lo que habilita la interacción simultánea de múltiples usuarios dentro del mismo entorno virtual. La integración de este framework de red en Unity permitió sincronizar en tiempo real las acciones y posiciones de cada participante, de forma que todos los usuarios compartan una experiencia común coherente. Gracias a esta funcionalidad de red, los avatares de estudiantes y docentes se actualizan al unísono para todos los participantes, y características como la escritura en la pizarra o los cambios de diapositiva en las presentaciones se reflejan inmediatamente en cada visor de realidad virtual. Con ello, el aula inmersiva trasciende la experiencia de un solo usuario y se convierte en una plataforma colaborativa, sentando las bases para sesiones educativas virtuales grupales con interacción social auténtica.

Se implementaron mejoras técnicas significativas en el sistema. Uno de los aprendizajes fue la necesidad de lograr interacciones físicas más realistas: para ello se incorporó la librería AutoHand, un sistema especializado en interacción física en VR que permitió mejorar la sensación de manipulación de objetos virtuales con las manos. Gracias a AutoHand, las manos virtuales de los usuarios pueden agarrar y mover objetos con comportamientos físicos naturales (peso, colisión, inercia), elevando el realismo de la experiencia más allá de las interacciones básicas. Otro aspecto técnico refinado fue el manejo de capas y colisiones en Unity para evitar conflictos entre los *raycasts* de interacción y la detección física de objetos. Se definieron capas específicas (por ejemplo, distinguir entre objetos interactuables y el cuerpo del jugador) y se ajustaron las matrices de colisión de forma que ciertas interacciones (como el trazo del marcador sobre la pizarra) no se vieran interrumpidas por colisiones no deseadas. Este

enfoque de capas, complementado por utilidades como scripts de ignorar colisiones entre el usuario y determinados objetos, solucionó problemas iniciales donde el marcador podía atravesar la superficie de la pizarra o interferir con la geometría del entorno. Por último, se mejoró la sincronización eficiente de recursos y estados en el contexto multijugador. Esto implicó optimizar la transmisión de datos por la red para que la experiencia compartida sea fluida, minimizando la latencia y el uso de ancho de banda. Por ejemplo, se ajustó qué elementos del entorno debían sincronizarse (posiciones de usuarios, trazos de la pizarra, diapositivas actuales) y con qué frecuencia, asegurando que todos los participantes vean los cambios prácticamente en tiempo real sin sobrecargar el sistema. Estas mejoras técnicas no solo resolvieron obstáculos encontrados durante el desarrollo, sino que también establecieron una base más sólida y escalable para futuras extensiones del aula virtual inmersiva.

8. RECOMENDACIONES

- *Establecer un sistema robusto de asignación de permisos* para docentes y estudiantes, que permita controlar el uso de herramientas compartidas en el aula virtual. Por ejemplo, el docente debería poder otorgar o revocar privilegios para escribir en la pizarra, manipular objetos o avanzar las diapositivas de una presentación. Este control de permisos garantizará un orden en la sesión virtual, evitando que múltiples usuarios editen simultáneamente un recurso de forma caótica y replicando la estructura de autoridad de un aula real (donde el profesor modera la participación). Implementar roles diferenciados con niveles de acceso (profesor, estudiante, asistente, etc.) contribuirá a una gestión más segura y organizada del entorno inmersivo.
- *Mejorar la interfaz de navegación de presentaciones*, permitiendo a los docentes controlar las diapositivas con mayor precisión y ofreciendo a los estudiantes cierta flexibilidad para revisar contenido previo. En la implementación actual, el profesor puede avanzar las diapositivas de forma secuencial; se recomienda extender esta funcionalidad para que el docente pueda saltar a diapositivas específicas, retroceder con facilidad o incluso resaltar partes de una presentación durante la exposición. Adicionalmente, sería beneficioso habilitar un modo de consulta de diapositivas previas para los alumnos, de manera que, con autorización del profesor, cada estudiante pueda revisar por su cuenta diapositivas ya expuestas (por ejemplo, para tomar notas o aclarar dudas) sin retrasar el ritmo general de la clase. Una navegación más robusta y flexible incrementará la eficacia del módulo de presentaciones y mejorará la experiencia tanto del instructor como de los participantes.

- *Incorporar opciones de personalización del entorno virtual* para adaptarlo a diferentes contextos educativos y preferencias de los usuarios. Se propone permitir al administrador o docente modificar la disposición del aula (por ejemplo, reubicación de mesas, sillas, pizarra), así como elegir entre distintos estilos visuales o configuraciones de iluminación según la asignatura o el ambiente deseado. Esta personalización podría incluir cambiar el tema o decoración del entorno (una aula tradicional, un laboratorio científico, un auditorio, etc.) e incluso ajustar parámetros como la luminosidad o la paleta de colores predominante. Al brindar opciones de configuración, el sistema se volverá más versátil y atractivo, favoreciendo que cada sesión educativa inmersiva se adapte mejor al contenido impartido y haciendo que docentes y estudiantes se sientan más cómodos e involucrados en el espacio virtual.
- *Explorar la integración de contenido web enriquecido mediante sistemas seguros de incrustación (WebView u otra técnica similar)*, para complementar los materiales presentados en clase con recursos externos. Esta recomendación apunta a permitir que dentro del aula virtual se puedan desplegar elementos como documentos PDF, videos, páginas web interactivas o aplicaciones web controladas, ampliando las posibilidades más allá de las diapositivas estáticas. Por ejemplo, un docente podría abrir un PDF académico o un video de YouTube dentro del mundo virtual para que todos los alumnos lo vean sincronizadamente, o mostrar una página web con simulaciones interactivas sin que los estudiantes tengan que quitarse el visor. Es importante que dicha integración se realice de forma segura, restringiendo la navegación solo a contenidos autorizados y evitando riesgos de seguridad o distracciones. Al incorporar contenido web embebido de forma transparente en la experiencia VR, se enriquece la

plataforma educativa con múltiples fuentes de información y se acerca aún más la funcionalidad del aula virtual a las herramientas utilizadas en entornos de aprendizaje tradicionales y modernos.

9. BIBLIOGRAFIA

Alcalá, F. (2024). Las tecnologías inmersivas aplicadas a la educación y formación. Espiral, El Diario de la Educación.

Vergara García, M. Á., et al. (2025). Impacto del e-learning en la retención de conocimientos y el rendimiento académico en la educación superior. e-Revista Multidisciplinaria del Saber, 3(Dic), 222-236.

Evolmind (2023). Aprendizaje inmersivo: qué es y por qué está revolucionando la formación. Blog Evolmind.

Evolmind (2023). Modalidades de aprendizaje virtual: tecnología, innovación y desafíos. Blog Evolmind.

Angulo Mendoza, G. A., Lewis, F., Plante, P., & Brassard, C. (2023). Estado del arte sobre el uso de la realidad virtual, la realidad aumentada y el video 360° en educación superior. EDUTEC. Revista Electrónica de Tecnología Educativa, (84), 35-51. https://doi.org/10.21556/edutec.2023.84.2769

Cabero-Almenara, J., & Marín-Díaz, V. (2018). Blended learning y realidad aumentada: Experiencias de diseño docente. RIED. Revista Iberoamericana de Educación a Distancia, 21(1), 57-74. https://doi.org/10.5944/ried.21.1.18719

Cabero-Almenara, J., Fernández-Batanero, J. M., & Barroso-Osuna, J. (2019). Adoption of augmented reality technology by university students. Heliyon, 5(5), e01597. https://doi.org/10.1016/j.heliyon.2019.e01597

Calderón Zambrano, R. L., Yánez Romero, M. E., Dávila Dávila, K. E., & Beltrán Balarezo, C. E. (2023). Realidad virtual y aumentada en la educación superior:

experiencias inmersivas para el aprendizaje profundo. Religación. Revista de Ciencias Sociales y Humanidades, 8(37), e2301088. https://doi.org/10.46652/rgn.v8i37.1088

Dalgarno, B., & Lee, M. J. W. (2010). What are the learning affordances of 3-D virtual environments? British Journal of Educational Technology, 41(1), 10-32. https://doi.org/10.1111/j.1467-8535.2009.01038.x

Dunleavy, M., Dede, C., & Mitchell, R. (2009). Affordances and limitations of immersive participatory augmented reality simulations for teaching and learning. Journal of Science Education and Technology, 18(1), 7-22. https://doi.org/10.1007/s10956-008-9119-1

Felkel, I., & Dickmann, I. (2022). Realidad virtual y formación docente: aportes, desafíos y límites. Educación Temática Digital (ETD), 24(2), 296-315. https://doi.org/10.20396/etd.v24i2.8659798

Hamilton, D., McKechnie, J., Edgerton, E., & Wilson, C. (2021). Immersive virtual reality as a pedagogical tool in education: A systematic literature review of quantitative learning outcomes and experimental design. Journal of Computers in Education, 8(1), 1-32. https://doi.org/10.1007/s40692-020-00169-2

Campos Soto, N., Navas-Parejo, M. R., & Moreno Guerrero, A. J. (2020). Virtual Reality and Motivation in the Educational Context: Bibliometric Study of the Last Twenty Years of Scopus. Alterity Education Magazine, 15(1), 47–60. https://www.redalyc.org/journal/4677/467761669004/467761669004.pdf

Kavanagh, S., Luxton-Reilly, A., Wuensche, B., & Plimmer, B. (2017). A systematic review of virtual reality in education. Themes in Science and Technology Education, 10(2), 85-119. Recuperado de LearnTechLib.

Kounlaxay, K., Shim, Y., Kang, S.-J., Kwak, H.-Y., & Kim, S. K. (2021). Learning media on mathematical education based on augmented reality. KSII Transactions on Internet and Information Systems, 15(3), 1016-1029. https://doi.org/10.3837/tiis.2021.03.011

Moro, C., Phelps, C., Redmond, P., & Stromberga, Z. (2021). HoloLens and Mobile Augmented Reality in Medical and Health Science Education: A Randomised Controlled Trial. British Journal of Educational Technology, 52(2), 680–694. https://doi.org/10.1111/bjet.13049

Marougkas, A., Troussas, C., Krouska, A., & Sgouropoulou, C. (2023). Virtual Reality in Education: A Review of Learning Theories, Approaches and Methodologies for the Last Decade. Electronics, 12(13), 2832. https://doi.org/10.3390/electronics12132832

Mikropoulos, T. A., & Natsis, A. (2011). Educational virtual environments: A ten-year review of empirical research (1999-2009). Computers & Education, 56(3), 769-780. https://doi.org/10.1016/j.compedu.2010.10.020

Montenegro-Rueda, M., & Fernández-Cerero, J. (2022). Realidad aumentada en la educación superior: Posibilidades y desafíos. Tecnología, Ciencia y Educación, (23), 95-114. https://doi.org/10.51302/tce.2022.858

Radianti, J., Majchrzak, T. A., Fromm, J., & Wohlgenannt, I. (2020). A systematic review of immersive virtual reality applications for higher education: Design elements, lessons learned, and research agenda. Computers & Education, 147, 103778. https://doi.org/10.1016/j.compedu.2019.103778

Johnson, L., Adams Becker, S., Estrada, V., & Freeman, A. (2019). NMC Horizon Report: 2019 Higher Education Edition. EDUCAUSE. (Informe sobre tendencias emergentes en educación superior).

10. Anexo A – Scripts de funcionalidades básicas

A.1 – Whiteboard.cs

Gestiona la pizarra virtual en modo local, permitiendo dibujar sobre una textura 2D en tiempo real mediante raycasting y coordenadas UV.

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

public class Whiteboard : MonoBehaviour
{
    public Texture2D texture;
    public Vector2 textureSize = new Vector2(2048,2048);

    // Start is called before the first frame update
    void Start()
    {
        var r = GetComponent<Renderer>();
        texture = new Texture2D((int)textureSize.x, (int)textureSize.y);
        r.material.mainTexture = texture;
    }

}
```

A.2 – Marker.cs

Controla el marcador virtual, incluyendo detección de contacto con la pizarra mediante raycast corto, asignación de color y grosor de trazo, y manipulación física con AutoHand.

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

public class Marker : NetworkBehaviour
{
    [SerializeField] private Transform _tip;
    [SerializeField] private int _penSize = 5;
```

```
    [SerializeField] private float tipRayLength = 3.1f; // 3 mm
    [SerializeField] private LayerMask whiteboardMask;

    private Renderer _renderer;
    private Color[] _colors;
    private float _tipHeight;
    private RaycastHit _touch;
    private Whiteboard _whiteboard;
    private Vector2 _touchPos;
    private bool _touchedLastFrame;
    private Vector2 _lastTouchPos;

    void Start()
    {
        var netTransform = GetComponent<NetworkTransform>();
        _renderer = _tip.GetComponent<Renderer>();
        _colors = Enumerable.Repeat(_renderer.material.color, _penSize *
_penSize).ToArray();
        _tipHeight = _tip.localScale.y + 0.2f;
    }

    void Update()
    {
        //if (!HasStateAuthority) return;
        Debug.DrawRay(_tip.position, _tip.up * tipRayLength, Color.red);
        if (Physics.Raycast(_tip.position, _tip.up, out _touch,
tipRayLength, whiteboardMask))
        {
            if (_touch.transform.CompareTag("Whiteboard"))
            {
                if (_whiteboard == null)
                {
                    _whiteboard =
_touch.transform.GetComponent<Whiteboard>();
                }
                _touchPos = new Vector2(_touch.textureCoord.x,
_touch.textureCoord.y);
                var x = (int)(_touchPos.x * _whiteboard.textureSize.x -
(_penSize / 2));
                var y = (int)(_touchPos.y * _whiteboard.textureSize.y -
(_penSize / 2));

                if (x < 0 || x > _whiteboard.textureSize.x || y < 0 || y
> _whiteboard.textureSize.y) return;


                if (_touchedLastFrame)
                {
```

```
                    for (float f = 0.01f; f < 1.00f; f += 0.05f)
                    {
                        int lerpx = (int)Mathf.Lerp(_lastTouchPos.x, x,
f);
                        int lerpy = (int)Mathf.Lerp(_lastTouchPos.y, y,
f);
                        _whiteboard.texture.SetPixels(lerpx, lerpy,
_penSize, _penSize, _colors);
                    }
                }

                // Pinta el punto actual tambi◆n
                _whiteboard.texture.SetPixels(x, y, _penSize, _penSize,
_colors);
                _whiteboard.texture.Apply();

                Color c = _renderer.material.color;
                RPC_DrawPoint(x, y, c.r, c.g, c.b);

                _lastTouchPos = new Vector2(_touchPos.x *
_whiteboard.textureSize.x - (_penSize / 2),
                            _touchPos.y * _whiteboard.textureSize.y -
(_penSize / 2));
                _touchedLastFrame = true;
                return;
            }
        }

        _whiteboard = null;
        _touchedLastFrame = false;
    }

    [Rpc(RpcSources.StateAuthority, RpcTargets.Proxies)]
    private void RPC_DrawPoint(int x, int y, float r, float g, float b)
    {
        if (_whiteboard == null)
        {
            var obj = GameObject.FindWithTag("Whiteboard");
            if (obj != null)
                _whiteboard = obj.GetComponent<Whiteboard>();
        }

        if (_whiteboard == null) return;

        Color[] remoteColors = Enumerable.Repeat(new Color(r, g, b),
_penSize * _penSize).ToArray();
        _whiteboard.texture.SetPixels(x, y, _penSize, _penSize,
remoteColors);
```

```
            _whiteboard.texture.Apply();
        }
}
```

A.3 – PresentationHandler.cs

Permite cargar y visualizar presentaciones (imágenes, PDF, PPTX) en el entorno VR de forma local, renderizando cada diapositiva como textura en una malla dentro de la escena.

```
using System.Collections;
using System.Collections.Generic;
using System.IO;
using UnityEngine;
using UnityEngine.UI;
using SimpleFileBrowser;
using TMPro;
using UnityEngine.Networking;
using Fusion;
using System.Text;

public class PresentacionHandler : NetworkBehaviour
{
    [Header("UI")]
    public Button btnCargarArchivos;
    public Button btnAnterior;
    public Button btnSiguiente;
    public Toggle togglePresentacion;
    public TextMeshProUGUI archivoActualText;
    public GameObject fileBrowserPrefab;
    public Transform contenedorTransform;
    public TextMeshProUGUI presentadorLabel;

    [Header("Preview en Canvas")]
    public RawImage previewRawImage;

    [Header("Plano de presentación real (opcional)")]
    public Renderer planoPresentacionRenderer;

    [Header("Debug Toast")]
    public TextMeshProUGUI debugToastText; // Asigna en el inspector

    private int imagenActual = 0;
    private string sessionId;
    private int totalSlides;
```

```
    [Header("Configuración del servidor")]
    private string uploadBase64Url = "https://pdftoimgservice-
aebag6ceeagtf8b9.canadacentral-01.azurewebsites.net/upload_base64";
    private string getSlideBaseUrl = "https://pdftoimgservice-
aebag6ceeagtf8b9.canadacentral-01.azurewebsites.net/slides";
    private string convertFileUrl = "https://pdftoimgservice-
aebag6ceeagtf8b9.canadacentral-01.azurewebsites.net/convert_file";

    [Networked]
    private PlayerRef presentadorActual { get; set; }

    private Coroutine toastCoroutine;
    private FileBrowser fileBrowserInstance;

    public override void Spawned()
    {
        if (Object.HasStateAuthority)
        {
            Object.AssignInputAuthority(Runner.LocalPlayer);
            presentadorActual = Runner.LocalPlayer;
        }

        btnCargarArchivos.onClick.AddListener(AbrirExplorador);
        btnAnterior.onClick.AddListener(() => CambiarImagen(-1));
        btnSiguiente.onClick.AddListener(() => CambiarImagen(+1));
        togglePresentacion.onValueChanged.AddListener((val) => {
            RPC_TogglePlanoPresentacion(val);
        });

        RPC_TogglePlanoPresentacion(false);
    }

    void Update()
    {
        bool soyPresentador = Runner.LocalPlayer == presentadorActual;

        btnCargarArchivos.interactable = soyPresentador;
        btnAnterior.interactable = soyPresentador;
        btnSiguiente.interactable = soyPresentador;
        togglePresentacion.interactable = soyPresentador;
    }

    void AbrirExplorador()
    {
        if (Runner.LocalPlayer != presentadorActual) return;

        if (fileBrowserInstance == null)
```

```
        {
            GameObject obj = Instantiate(fileBrowserPrefab,
contenedorTransform);
            fileBrowserInstance = obj.GetComponent<FileBrowser>();
            DontDestroyOnLoad(fileBrowserInstance.gameObject);
        }

        FileBrowser.SetFilters(false,
            new FileBrowser.Filter("Imágenes", ".png", ".jpg", ".jpeg"),
            new FileBrowser.Filter("Presentaciones", ".pdf", ".pptx"));

        fileBrowserInstance.gameObject.SetActive(true);
        fileBrowserInstance.onSuccess = OnArchivosSeleccionados;
        fileBrowserInstance.onCancel = OnCancel;
        fileBrowserInstance.PickerMode = FileBrowser.PickMode.Files;
        fileBrowserInstance.AllowMultiSelection = true;
        fileBrowserInstance.Title = "Seleccionar archivos";
        fileBrowserInstance.SubmitButtonText = "Cargar";

        fileBrowserInstance.Show(null, null);
    }

    void OnArchivosSeleccionados(string[] paths)
    {
        ShowToast("Archivos seleccionados: " + string.Join(", ", paths));

        if (paths.Length == 1)
        {
            string ext = Path.GetExtension(paths[0]).ToLower();
            if (ext == ".pdf" || ext == ".pptx")
            {
                StartCoroutine(ProcesarArchivo(paths[0]));
                return;
            }
        }

        List<string> imagenesBase64 = new();

        foreach (var path in paths)
        {
            try
            {
                string fixedPath = ResolveContentUriToPath(path); // ✅
Nuevo
                byte[] data = File.ReadAllBytes(fixedPath);

                Texture2D tex = new Texture2D(2, 2);
                tex.LoadImage(data);
```

```
                imagenesBase64.Add(System.Convert.ToBase64String(tex.Enco
deToPNG()));
            }
            catch (System.Exception ex)
            {
                ShowToast("Error leyendo archivo:\n" + path + "\n" +
ex.Message, 6f);
                return;
            }
        }

        StartCoroutine(EnviarBase64AlServidor(imagenesBase64.ToArray(),
Path.GetFileName(paths[0])));

        if (fileBrowserInstance != null)
            fileBrowserInstance.gameObject.SetActive(false);
    }

    IEnumerator ProcesarArchivo(string path)
    {
        ShowToast("Procesando archivo: " + path);

        string fixedPath = ResolveContentUriToPath(path); // ✅ Nuevo

        byte[] fileData;
        try
        {
            fileData = File.ReadAllBytes(fixedPath);
        }
        catch (System.Exception ex)
        {
            ShowToast("Error leyendo archivo:\n" + path + "\n" +
ex.Message, 6f);
            yield break;
        }

        string fileName = Path.GetFileName(fixedPath);
        string ext = Path.GetExtension(fileName).ToLower();
        ShowToast("Archivo seleccionado: " + fileName + " (" + ext +
")"+"Path:" + fixedPath,10f);
        /*
        // 🔷 Verificación básica de cabecera
        bool cabeceraValida = false;
        if (ext == ".pdf")
        {
            for (int i = 0; i < Mathf.Min(fileData.Length - 4, 2048);
i++)
            {
```

```
                if (fileData[i] == '%' && fileData[i + 1] == 'P' &&
                    fileData[i + 2] == 'D' && fileData[i + 3] == 'F')
                {
                    cabeceraValida = true;
                    break;
                }
            }
        }
        else if (ext == ".pptx")
        {
            cabeceraValida = fileData.Length > 2 && fileData[0] == 0x50
&& fileData[1] == 0x4B;
        }

        if (!cabeceraValida)
        {
            ShowToast("El archivo no parece ser un " + ext.ToUpper() + "
válido", 6f);
            yield break;
        }*/

        // 🔼 Subir al servidor
        List<IMultipartFormSection> form = new
List<IMultipartFormSection>
        {
            new MultipartFormFileSection("file", fileData, fileName,
"application/octet-stream")
        };

        UnityWebRequest req = UnityWebRequest.Post(convertFileUrl, form);
        yield return req.SendWebRequest();

        if (req.result != UnityWebRequest.Result.Success)
        {
            //ShowToast("Error procesando archivo: " + req.error, 4f);
            yield break;
        }

        string json = req.downloadHandler.text;
        PDFResponse response =
JsonUtility.FromJson<PDFResponse>(FixJson(json));

        List<string> imagenesBase64 = new();
        foreach (string b64 in response.images)
        {
            imagenesBase64.Add(b64);
        }
```

```
        StartCoroutine(EnviarBase64AlServidor(imagenesBase64.ToArray(),
fileName));
    }

    void OnCancel()
    {
        ShowToast("Selección cancelada");
        if (fileBrowserInstance != null)
            fileBrowserInstance.gameObject.SetActive(false);
    }

    public void CambiarImagen(int delta)
    {
        if (Runner.LocalPlayer != presentadorActual) return;
        if (totalSlides == 0) return;

        imagenActual = Mathf.Clamp(imagenActual + delta, 0, totalSlides -
1);
        MostrarImagenActual();
        RPC_SetSlide(imagenActual);
    }

    void MostrarImagenActual()
    {
        if (string.IsNullOrEmpty(sessionId)) return;

        string url =
$"{getSlideBaseUrl}/{sessionId}/slide_{imagenActual}.png";
        ShowToast("Cargando imagen: " + url);
        StartCoroutine(CargarImagenDesdeUrl(url));
    }

    IEnumerator CargarImagenDesdeUrl(string url)
    {
        UnityWebRequest req = UnityWebRequestTexture.GetTexture(url);
        yield return req.SendWebRequest();

        if (req.result == UnityWebRequest.Result.Success)
        {
            Texture2D tex = DownloadHandlerTexture.GetContent(req);
            if (previewRawImage != null)
                previewRawImage.texture = tex;

            if (planoPresentacionRenderer != null &&
planoPresentacionRenderer.gameObject.activeSelf)
                planoPresentacionRenderer.material.mainTexture = tex;
            ShowToast("Imagen cargada correctamente");
        }
```

```
        else
        {
            ShowToast("Error cargando imagen: " + req.error, 4f);
        }
    }

    IEnumerator EnviarBase64AlServidor(string[] imagenesB64, string
nombreArchivo)
    {
        ShowToast("Enviando al servidor: " + nombreArchivo + " (" +
imagenesB64.Length + " imágenes)");
        string json = JsonUtility.ToJson(new UploadRequest { images =
imagenesB64 });

        UnityWebRequest req = new UnityWebRequest(uploadBase64Url,
"POST");
        byte[] bodyRaw = Encoding.UTF8.GetBytes(json);
        req.uploadHandler = new UploadHandlerRaw(bodyRaw);
        req.downloadHandler = new DownloadHandlerBuffer();
        req.SetRequestHeader("Content-Type", "application/json");

        yield return req.SendWebRequest();

        if (req.result != UnityWebRequest.Result.Success)
        {
            ShowToast("Error subiendo imágenes: " + req.error, 4f);
            yield break;
        }

        UploadResponse response =
JsonUtility.FromJson<UploadResponse>(req.downloadHandler.text);
        sessionId = response.session_id;
        totalSlides = response.slide_count;
        archivoActualText.text = nombreArchivo;
        MostrarImagenActual();
        ShowToast("Presentación subida correctamente");
        RPC_BroadcastPresentacion(sessionId, totalSlides, nombreArchivo);
    }

    [Rpc(RpcSources.All, RpcTargets.All)]
    void RPC_BroadcastPresentacion(string session, int count, string
nombre)
    {
        sessionId = session;
        totalSlides = count;
        archivoActualText.text = nombre;
        imagenActual = 0;
        MostrarImagenActual();
```

```
        ShowToast("Presentación sincronizada");
    }

    [Rpc(RpcSources.All, RpcTargets.All)]
    void RPC_SetSlide(int nuevoIndice)
    {
        imagenActual = nuevoIndice;
        MostrarImagenActual();
        ShowToast("Slide cambiado: " + nuevoIndice);
    }

    [Rpc(RpcSources.All, RpcTargets.All)]
    void RPC_TogglePlanoPresentacion(bool activar)
    {
        if (planoPresentacionRenderer != null)
        {
            Material mat = planoPresentacionRenderer.material;
            Color color = mat.color;
            color.a = activar ? 1f : 0f;
            mat.color = color;
        }
        ShowToast("Plano presentación: " + (activar ? "Activado" :
"Desactivado"));
    }

    [System.Serializable]
    public class UploadRequest { public string[] images; }

    [System.Serializable]
    public class UploadResponse { public string session_id; public int
slide_count; }

    [System.Serializable]
    public class PDFResponse { public List<string> images; }

    public PlayerRef GetPresentadorActual() => presentadorActual;

    string FixJson(string value)
    {
        if (!value.Trim().StartsWith("{"))
            return "{\"images\":" + value + "}";
        return value;
    }

    public void AsignarPresentador(PlayerRef nuevoPresentador)
    {
        if (!Runner.IsServer) return;
        presentadorActual = nuevoPresentador;
```

```
        Object.AssignInputAuthority(nuevoPresentador);
        RPC_ActualizarPresentador(presentadorActual);
        ShowToast("Nuevo presentador: " + nuevoPresentador.PlayerId);
    }

    [Rpc(RpcSources.StateAuthority, RpcTargets.All)]
    void RPC_ActualizarPresentador(PlayerRef nuevoPresentador)
    {
        presentadorActual = nuevoPresentador;
        if (presentadorLabel != null)
            presentadorLabel.text = $"Presentador: Jugador
{presentadorActual.PlayerId}";
        ShowToast("Presentador actualizado: " +
presentadorActual.PlayerId);
    }

    // ✅ Nuevo método para convertir content:// en path real usando el
plugin Java
    string ResolveContentUriToPath(string uri)
    {
#if UNITY_ANDROID && !UNITY_EDITOR
        if (string.IsNullOrEmpty(uri)) return uri;
        string normalizedUri = uri.Replace("/content:/", "content://");

        using (AndroidJavaClass unityPlayer = new
AndroidJavaClass("com.unity3d.player.UnityPlayer"))
        using (AndroidJavaObject activity =
unityPlayer.GetStatic<AndroidJavaObject>("currentActivity"))
        using (AndroidJavaClass pluginClass = new
AndroidJavaClass("com.example.contenturihelper.ContentUriHelper"))
        {
            string path =
pluginClass.CallStatic<string>("copyToTempFile", activity,
normalizedUri);
            return path ?? uri;
        }
#else
        return uri;
#endif
    }

    // --- Toast helpers ---
    public void ShowToast(string msg, float duration = 2f)
    {
        if (debugToastText == null) return;
        debugToastText.text = msg;
        debugToastText.gameObject.SetActive(true);
        if (toastCoroutine != null) StopCoroutine(toastCoroutine);
```

```
        toastCoroutine = StartCoroutine(HideToastAfter(duration));
    }

    private IEnumerator HideToastAfter(float duration)
    {
        yield return new WaitForSeconds(duration);
        if (debugToastText != null)
            debugToastText.gameObject.SetActive(false);
    }
}
```

11. Anexo B – Scripts de configuración y gestión de red

B.1 – NetworkManagerFusion.cs

Administra la conexión y sincronización de sesiones multijugador con Photon Fusion, incluyendo creación y unión a salas virtuales.

```
using Fusion;
using Fusion.Sockets;
using UnityEngine;
using UnityEngine.SceneManagement;
using System;
using System.Collections.Generic;

public class NetworkManagerFusion : MonoBehaviour,
INetworkRunnerCallbacks
{

    private Dictionary<PlayerRef, NetworkObject> playerAvatars = new();
    [SerializeField] private NetworkPrefabRef avatarPrefab;
    public static string currentRoomCode;
    public static NetworkManagerFusion Instance;

    [SerializeField] private NetworkRunner runnerPrefab;
    private NetworkRunner runnerInstance;

    private void Awake()
    {
        if (Instance == null) Instance = this;
        else
        {
            Destroy(gameObject);
            return;
```

```
        }

        DontDestroyOnLoad(gameObject);
    }

    public async void StartGameAsHost(string roomName)
    {
        runnerInstance = Instantiate(runnerPrefab);
        runnerInstance.name = "FusionRunner";
        runnerInstance.ProvideInput = true;
        runnerInstance.AddCallbacks(this);

        var sceneManager =
runnerInstance.GetComponent<NetworkSceneManagerDefault>() ??
                            runnerInstance.gameObject.AddComponent<Network
SceneManagerDefault>();

        // Obtén el SceneRef desde el índice de la escena que quieres
cargar
        SceneRef sceneRef = SceneRef.FromIndex(1); // Cambia "1" por el
índice real de tu escena de clase

        // Crea el NetworkSceneInfo correctamente
        NetworkSceneInfo sceneInfo = default;
        sceneInfo.AddSceneRef(sceneRef, LoadSceneMode.Single,
LocalPhysicsMode.None, true);

        // Llama a StartGame con los parámetros corregidos
        await runnerInstance.StartGame(new StartGameArgs
        {
            GameMode = GameMode.Host,
            SessionName = roomName,
            Scene = sceneInfo,
            SceneManager = sceneManager
        });

    }

    public async void StartGameAsClient(string roomName)
    {
        runnerInstance = Instantiate(runnerPrefab);
        runnerInstance.name = "FusionRunner";
        runnerInstance.ProvideInput = true;
        runnerInstance.AddCallbacks(this);

        var sceneManager =
runnerInstance.GetComponent<NetworkSceneManagerDefault>() ??
```

```
                        runnerInstance.gameObject.AddComponent<Network
SceneManagerDefault>();

        // Obtén el SceneRef desde el índice de la escena que quieres
cargar
        SceneRef sceneRef = SceneRef.FromIndex(1); // Cambia "1" por el
índice real de tu escena de clase

        // Crea el NetworkSceneInfo correctamente
        NetworkSceneInfo sceneInfo = default;
        sceneInfo.AddSceneRef(sceneRef, LoadSceneMode.Single,
LocalPhysicsMode.None, true);

        // Llama a StartGame con los parámetros corregidos
        await runnerInstance.StartGame(new StartGameArgs
        {
            GameMode = GameMode.Client,
            SessionName = roomName,
            Scene = sceneInfo,
            SceneManager = sceneManager
        });


    }

    public void OnPlayerJoined(NetworkRunner runner, PlayerRef player)
    {
        Debug.Log($"Jugador conectado: {player}");

        if (!runner.IsServer) return;

        runner.Spawn(avatarPrefab, Vector3.zero, Quaternion.identity,
player, (runner, obj) =>
        {
            playerAvatars[player] = obj; // Guarda el avatar en el
diccionario

            var avatar = obj.GetComponent<NetworkAvatarSync>();


            //if (player == runner.LocalPlayer)

                avatar.rigLocalName = "Camera Rig";
                avatar.leftHandName = "L Hand";
                avatar.rightHandName = "R Hand";


            if (runner.IsServer)
```

```
            {
                runner.SetPlayerObject(player, obj);
            }
        });


    }

    public void OnPlayerLeft(NetworkRunner runner, PlayerRef player)
    {
        Debug.Log($"Jugador desconectado: {player}");

        if (!runner.IsServer) return;

        if (playerAvatars.TryGetValue(player, out var avatarObj))
        {
            runner.Despawn(avatarObj);
            playerAvatars.Remove(player);
        }
    }

    public void OnInput(NetworkRunner runner, NetworkInput input)
    {
        // Buscar el rig local y las manos en la escena
        GameObject rig = GameObject.Find("Camera Rig");
        Transform headReal = rig?.transform.Find("Camera Offset/Main
Camera") ?? Camera.main?.transform;
        Transform leftHandReal = GameObject.Find("L Hand")?.transform;
        Transform rightHandReal = GameObject.Find("R Hand")?.transform;

        // Si no se encuentran, no enviar input
        if (headReal == null || leftHandReal == null || rightHandReal ==
null)
            return;

        var avatarInput = new AvatarInputData
        {
            headPosition = headReal.position,
            headRotation = headReal.rotation,
            leftHandPosition = leftHandReal.position,
            leftHandRotation = leftHandReal.rotation,
            rightHandPosition = rightHandReal.position,
            rightHandRotation = rightHandReal.rotation
        };

        input.Set(avatarInput);
    }
```

```
    public void OnInputMissing(NetworkRunner runner, PlayerRef player,
NetworkInput input) { }
    public void OnShutdown(NetworkRunner runner, ShutdownReason
shutdownReason) { }
    public void OnConnectedToServer(NetworkRunner runner) { }
    public void OnDisconnectedFromServer(NetworkRunner runner) { }
    public void OnConnectRequest(NetworkRunner runner,
NetworkRunnerCallbackArgs.ConnectRequest request, byte[] token) { }
    public void OnConnectFailed(NetworkRunner runner, NetAddress
remoteAddress, NetConnectFailedReason reason) { }
    public void OnUserSimulationMessage(NetworkRunner runner,
SimulationMessagePtr message) { }
    public void OnSessionListUpdated(NetworkRunner runner,
System.Collections.Generic.List<SessionInfo> sessionList) { }
    public void OnCustomAuthenticationResponse(NetworkRunner runner,
System.Collections.Generic.Dictionary<string, object> data) { }
    public void OnHostMigration(NetworkRunner runner, HostMigrationToken
hostMigrationToken) { }
    public void OnObjectEnterAOI(NetworkRunner runner, NetworkObject obj,
PlayerRef player) { }
    public void OnObjectExitAOI(NetworkRunner runner, NetworkObject obj,
PlayerRef player) { }
    public void OnDisconnectedFromServer(NetworkRunner runner,
NetDisconnectReason reason) { }
    public void OnReliableDataReceived(NetworkRunner runner, PlayerRef
player, ReliableKey key, ArraySegment<byte> data) { }
    public void OnReliableDataProgress(NetworkRunner runner, PlayerRef
player, ReliableKey key, float progress) { }
    public void OnSceneLoadDone(NetworkRunner runner) { }
    public void OnSceneLoadStart(NetworkRunner runner) { }
}
```

B.2 – MenuManagerFusion.cs

Gestiona el flujo de menús para la conexión a sesiones y configuración previa antes de ingresar al aula virtual.

```
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class MenuManagerFusion : MonoBehaviour
```

```
{
    [Header("Nombre de usuario")]
    public TMP_InputField nameInput;
    public Button acceptButton;
    public TextMeshProUGUI nameDisplay;
    public Button changeNameButton;

    [Header("Salas")]
    public TMP_InputField roomCodeInput;
    public Button createRoomButton;
    public Button joinRoomButton;

    private string playerName = "";

    void Start()
    {
        nameInput.gameObject.SetActive(true);
        acceptButton.gameObject.SetActive(true);
        nameDisplay.gameObject.SetActive(false);
        changeNameButton.gameObject.SetActive(false);

        acceptButton.onClick.AddListener(OnAcceptClicked);
        changeNameButton.onClick.AddListener(OnChangeClicked);

        createRoomButton.onClick.AddListener(OnCreateRoomClicked);
        joinRoomButton.onClick.AddListener(OnJoinRoomClicked);

        nameInput.text = "testUSRDEF";
    }

    void OnAcceptClicked()
    {
        playerName = nameInput.text.Trim();

        if (!string.IsNullOrEmpty(playerName))
        {
            PlayerPrefs.SetString("PlayerName", playerName);
            nameDisplay.text = playerName;

            nameInput.gameObject.SetActive(false);
            acceptButton.gameObject.SetActive(false);
            nameDisplay.gameObject.SetActive(true);
            changeNameButton.gameObject.SetActive(true);
        }
        else
        {
            Debug.LogWarning("El nombre no puede estar vac�o.");
        }
```

```
}

void OnChangeClicked()
{
    nameDisplay.gameObject.SetActive(false);
    changeNameButton.gameObject.SetActive(false);
    nameInput.gameObject.SetActive(true);
    acceptButton.gameObject.SetActive(true);
}

void OnCreateRoomClicked()
{
    if (ValidarNombre())
    {
        string roomName = "Sala_" + Random.Range(1000, 9999);
        NetworkManagerFusion.Instance.StartGameAsHost(roomName);
    }
}

void OnJoinRoomClicked()
{
    if (ValidarNombre())
    {
        string codigo = roomCodeInput.text.Trim();
        if (!string.IsNullOrEmpty(codigo))
        {
            NetworkManagerFusion.Instance.StartGameAsClient(codigo);
        }
        else
        {
            Debug.LogWarning("Ingresa un c◆digo de sala.");
        }
    }
}

bool ValidarNombre()
{
    if (string.IsNullOrEmpty(playerName))
    {
        Debug.LogWarning("Primero debes ingresar tu nombre.");
        return false;
    }
    return true;
}

public string ObtenerNombreJugador()
{
    return playerName;
```

```
    }
}
```

B.3 – NetworkAvatarSync.cs

Sincroniza en red la posición y rotación de la cabeza y manos de los avatares, vinculando el modelo 3D con el rig físico del usuario.

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

public struct AvatarInputData : INetworkInput
{
    public Vector3 headPosition;
    public Quaternion headRotation;
    public Vector3 leftHandPosition;
    public Quaternion leftHandRotation;
    public Vector3 rightHandPosition;
    public Quaternion rightHandRotation;
}

public class NetworkAvatarSync : NetworkBehaviour
{
    [Header("Cabeza y manos del rig avatar (asignar en prefab)")]
    public Transform headAvatar;
    public Transform leftHandAvatar;
    public Transform rightHandAvatar;

    [System.Serializable]
    public class FingerAssignment
    {
        [Tooltip("Nombre para referencia en editor (ej: Thumb,
Index...)")]
        public string fingerName;

        [Tooltip("Joints del avatar: solo asigna el KNUCKLE (padre)")]
        public Transform avatarKnuckleJoint;

        // Offset de corrección para cada joint
```

```
        public Vector3 knuckleOffsetEuler;
        public Vector3 middleOffsetEuler;
        public Vector3 distalOffsetEuler;

        // Interno: los joints detectados desde el prefab (Middle y
Distal)
        [HideInInspector] public Transform avatarMiddleJoint;
        [HideInInspector] public Transform avatarDistalJoint;

        [HideInInspector] public Transform realKnuckle;
        [HideInInspector] public Transform realMiddle;
        [HideInInspector] public Transform realDistal;
    }

    [Header("Dedos mano izquierda")]
    public FingerAssignment[] leftFingers;

    [Header("Dedos mano derecha")]
    public FingerAssignment[] rightFingers;

    [Header("Nombre de objetos reales en escena (AutoHand)")]
    public string rigLocalName = "Camera Rig";
    public string leftHandName = "L Hand";
    public string rightHandName = "R Hand";

    private Transform headReal;
    private Transform leftHandReal;
    private Transform rightHandReal;

    [Header("Ajuste de rotación para correctionOffset (editable en
runtime)")]
    public float correctionOffsetX = 0f;
    public float correctionOffsetY = 0f;
    public float correctionOffsetZ = 0f;

    private Quaternion correctionOffset; // Ajuste según orientación de
tu modelo

    [Networked] private Vector3 HeadPosition { get; set; }
    [Networked] private Quaternion HeadRotation { get; set; }
    [Networked] private Vector3 LeftHandPosition { get; set; }
    [Networked] private Quaternion LeftHandRotation { get; set; }
    [Networked] private Vector3 RightHandPosition { get; set; }
    [Networked] private Quaternion RightHandRotation { get; set; }

    public override void Spawned()
    {
        // Buscar rig local solo si tienes autoridad de entrada
```

```
        if (Object.HasInputAuthority)
        {
            GameObject rig = GameObject.Find(rigLocalName);
            headReal = rig?.transform.Find("Camera Offset/Main Camera")
?? Camera.main?.transform;
            leftHandReal = GameObject.Find(leftHandName)?.transform;
            rightHandReal = GameObject.Find(rightHandName)?.transform;

            if (!leftHandReal || !rightHandReal || !headReal)
            {
                Debug.LogError(" No se encontraron referencias del rig
real.");
                return;
            }
        }

        if (!headAvatar || !leftHandAvatar || !rightHandAvatar)
        {
            Debug.LogError(" No se asignaron referencias de avatar en el
prefab.");
            return;
        }

        // Mapear joints del avatar
        SetupAvatarFingerJoints(leftFingers);
        SetupAvatarFingerJoints(rightFingers);

        // Mapear joints reales desde AutoHand solo si tienes autoridad
        if (Object.HasInputAuthority)
        {
            SetupRealFingers(leftFingers, leftHandReal);
            SetupRealFingers(rightFingers, rightHandReal);
        }

        // Oculta render del propio avatar (descomenta si lo necesitas)
        //foreach (var renderer in GetComponentsInChildren<Renderer>())
        //    renderer.enabled = false;

        Debug.Log(" Avatar sincronizado: " + Object.InputAuthority);
    }

    public override void FixedUpdateNetwork()
    {
        // Si recibimos input, actualizamos las variables de red
        if (GetInput(out AvatarInputData inputData))
        {
            HeadPosition = inputData.headPosition;
            HeadRotation = inputData.headRotation;
```

```
            LeftHandPosition = inputData.leftHandPosition;
            LeftHandRotation = inputData.leftHandRotation;
            RightHandPosition = inputData.rightHandPosition;
            RightHandRotation = inputData.rightHandRotation;
        }

        // Aplica la posición/rotación a los avatares en todos los
clientes
        headAvatar.SetPositionAndRotation(HeadPosition, HeadRotation);
        leftHandAvatar.SetPositionAndRotation(LeftHandPosition,
LeftHandRotation);
        rightHandAvatar.SetPositionAndRotation(RightHandPosition,
RightHandRotation);

        // Sincroniza los dedos solo para el propio jugador
        if (Object.HasInputAuthority)
        {
            SyncFingers(leftFingers);
            SyncFingers(rightFingers);
        }

        correctionOffset = Quaternion.Euler(correctionOffsetX,
correctionOffsetY, correctionOffsetZ);

    }

    void SyncFingers(FingerAssignment[] fingers)
    {
        foreach (var finger in fingers)
        {
            if (finger.avatarKnuckleJoint && finger.realKnuckle)
            {
                var offset = Quaternion.Euler(finger.knuckleOffsetEuler);
                finger.avatarKnuckleJoint.localRotation = offset *
finger.realKnuckle.localRotation;
            }

            if (finger.avatarMiddleJoint && finger.realMiddle)
            {
                var offset = Quaternion.Euler(finger.middleOffsetEuler);
                finger.avatarMiddleJoint.localRotation = offset *
finger.realMiddle.localRotation;
            }

            if (finger.avatarDistalJoint && finger.realDistal)
            {
                var offset = Quaternion.Euler(finger.distalOffsetEuler);
```

```
                finger.avatarDistalJoint.localRotation = offset *
finger.realDistal.localRotation;
            }
        }
    }

    void SetupAvatarFingerJoints(FingerAssignment[] fingers)
    {
        foreach (var f in fingers)
        {
            if (!f.avatarKnuckleJoint) continue;

            if (f.avatarKnuckleJoint.childCount > 0)
            {
                f.avatarMiddleJoint = f.avatarKnuckleJoint.GetChild(0);
                if (f.avatarMiddleJoint.childCount > 0)
                    f.avatarDistalJoint =
f.avatarMiddleJoint.GetChild(0);
            }
        }
    }

    void SetupRealFingers(FingerAssignment[] fingers, Transform handRoot)
    {
        var fingerComponents =
handRoot.GetComponentsInChildren<Autohand.Finger>();
        foreach (var f in fingers)
        {
            foreach (var real in fingerComponents)
            {
                if (real.name.ToLower().Contains(f.fingerName.ToLower()))
                {
                    f.realKnuckle = real.knuckleJoint;
                    f.realMiddle = real.middleJoint;
                    f.realDistal = real.distalJoint;
                    break;
                }
            }
        }
    }
}
```

B.4 – AvatarUIController.cs

Controla las interfaces gráficas (canvases) asociadas a cada avatar, permitiendo que el host mutee a otros jugadores o asigne permisos de presentación, y que cada usuario gestione su propio micrófono.

```
using UnityEngine;
using UnityEngine.UI;
using Fusion;
using Photon.Voice.Unity;
using TMPro;

public class AvatarUIController : NetworkBehaviour
{
    [Header("UI")]
    public Toggle presenterToggle;    // Toggle con icono invertido
    public Toggle muteToggle;         // Toggle para mute/unmute por host

    [Header("UI Extra")]
    public Toggle selfMuteToggle;         // Toggle para que el usuario
se mutee a sí mismo
    public Toggle reclaimPresenterToggle; // Toggle para que el host se
reasigne como presentador
    public Canvas avatarUICanvas;         // Canvas principal del UI del
avatar


    [Header("Voice")]
    public Speaker voiceSpeaker;      // Speaker en la cabeza del avatar
(audio 3D)

    [Networked]
    public NetworkBool IsMutedByHost { get; set; }

    [Networked]
    public NetworkBool IsSelfMuted { get; set; }

    [Networked]
    public NetworkBool NoEsPresentador { get; set; } = true; // true = no
presenta (toggle activado con X)

    private PresentacionHandler presentacionHandler;
    private bool lastMuteState;
    private bool lastPresenterState;

    [Header("Name Tag")]
```

```
    public TextMeshProUGUI playerNameTag; // Asigna en el inspector

    [Networked]
    public NetworkString<_32> PlayerName { get; set; } // Tamaño
ajustable

    public override void Spawned()
    {
        presentacionHandler = FindObjectOfType<PresentacionHandler>();

        bool soyHost = Runner.IsServer;

        if (presenterToggle != null)
        {
            presenterToggle.interactable = soyHost;
            presenterToggle.onValueChanged.AddListener((state) =>
            {
                if (!Runner.IsServer) return;
                NoEsPresentador = state;
                SyncPresenterToggles(NoEsPresentador);

                if (!state)
                {
                    presentacionHandler.AsignarPresentador(Object.InputAu
thority);
                }
            });
            presenterToggle.SetIsOnWithoutNotify(NoEsPresentador);
        }

        if (muteToggle != null)
        {
            muteToggle.interactable = soyHost;
            muteToggle.onValueChanged.AddListener((state) =>
            {
                if (!Runner.IsServer) return;
                IsMutedByHost = state;
            });

            muteToggle.SetIsOnWithoutNotify(IsMutedByHost);
        }

        // Aplicar estado inicial
        ApplyMute(IsMutedByHost);
        SyncPresenterToggles(NoEsPresentador);

        if (Object.HasInputAuthority)
        {
```

```
            // Recupera el nombre guardado por MenuManagerFusion
            string nombre = PlayerPrefs.GetString("PlayerName",
"Jugador");
            PlayerName = nombre;
        }

        UpdateNameTag(PlayerName.ToString());

        // Toggle de auto-mute (siempre interactuable)
        if (selfMuteToggle != null)
        {
            selfMuteToggle.interactable = true;
            selfMuteToggle.onValueChanged.AddListener((state) =>
            {
                if (Object.HasInputAuthority)
                {
                    IsSelfMuted = state;
                    ApplySelfMute(state);
                }
            });
            selfMuteToggle.SetIsOnWithoutNotify(IsSelfMuted);
        }

        // Toggle para que el host se reasigne como presentador
        if (reclaimPresenterToggle != null)
        {
            reclaimPresenterToggle.interactable = soyHost;
            reclaimPresenterToggle.onValueChanged.AddListener((state) =>
            {
                if (!Runner.IsServer) return;
                NoEsPresentador = state;
                SyncPresenterToggles(NoEsPresentador);

                if (!state)
                {
                    presentacionHandler.AsignarPresentador(Object.InputAu
thority);
                }
            });
            reclaimPresenterToggle.SetIsOnWithoutNotify(NoEsPresentador);
        }
        // Canvas solo visible para el dueño
        if (avatarUICanvas != null)
            avatarUICanvas.enabled = Object.HasInputAuthority;
    }

    public override void Render()
    {
```

```
        // 🔄 Detección de cambios manual, ya que Changed<> no existe en
Fusion 2.x

        if (avatarUICanvas != null)
            avatarUICanvas.enabled = Object.HasInputAuthority;

        if (lastMuteState != IsMutedByHost)
        {
            lastMuteState = IsMutedByHost;
            ApplyMute(IsMutedByHost);
        }

        // Detecta si el presentador actual ha cambiado
        if (presentacionHandler != null)
        {
            var actual = presentacionHandler.GetPresentadorActual();
            bool soyPresentador = actual == Object.InputAuthority;

            // Si el estado de presentador ha cambiado, actualiza el
toggle y la propiedad
            if (NoEsPresentador == soyPresentador)
            {
                NoEsPresentador = !soyPresentador;
                SyncPresenterToggles(NoEsPresentador);
            }
        }
        else
        {
            // Fallback: actualiza solo si cambia el valor local
            if (lastPresenterState != NoEsPresentador)
            {
                lastPresenterState = NoEsPresentador;
                SyncPresenterToggles(NoEsPresentador);
            }
        }

        UpdateNameTag(PlayerName.ToString());
    }

    private void ApplyMute(bool muted)
    {
        //  Apaga/enciende Speaker para que nadie lo escuche
        if (voiceSpeaker != null)
            voiceSpeaker.enabled = !muted;

        //  Si soy dueño local, apago mi transmisión global
        if (Object.HasInputAuthority)
        {
```

```
            var runnerVoice = FindObjectOfType<MyRunnerVoice>();
            if (runnerVoice != null)
                runnerVoice.SetMute(muted);
        }

        //  Actualizar toggle visual si existe
        if (muteToggle != null)
            muteToggle.SetIsOnWithoutNotify(muted);
    }
    private void SyncPresenterToggles(bool noEsPresentador)
    {
        if (presenterToggle != null)
            presenterToggle.SetIsOnWithoutNotify(noEsPresentador);

        if (reclaimPresenterToggle != null)
            reclaimPresenterToggle.SetIsOnWithoutNotify(noEsPresentador);
    }


    private void UpdateNameTag(string nombre)
    {
        if (playerNameTag != null)
            playerNameTag.text = nombre;
    }

    private void ApplySelfMute(bool muted)
    {
        if (Object.HasInputAuthority)
        {
            var runnerVoice = FindObjectOfType<MyRunnerVoice>();
            if (runnerVoice != null)
                runnerVoice.SetMute(muted);
        }
        if (selfMuteToggle != null)
            selfMuteToggle.SetIsOnWithoutNotify(muted);
    }
}
```