UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Sistemas Informáticos y Computación

MindCubeXR: Una Aplicación de Realidad Extendida para la Rehabilitación Motora y Cognitiva de Personas con Daño Cerebral Adquirido

Trabajo Fin de Máster

Máster Universitario en Ingeniería y Tecnología de Sistemas Software

AUTOR/A: Marí Vilaplana, Ana

Tutor/a: Jaén Martínez, Francisco Javier

CURSO ACADÉMICO: 2023/2024

Agradecimientos

A lo largo del desarrollo de este Trabajo de Fin de Máster, he tenido la fortuna de contar con el apoyo y la colaboración de muchas personas, a quienes me gustaría expresar mi más sincero agradecimiento.

En primer lugar, quisiera expresar mi profunda gratitud a Francisco Javier Jaén Martínez por proporcionarme el material necesario para llevar a cabo este proyecto. Sus valiosas sugerencias y orientaciones han sido fundamentales para mejorar tanto la calidad como el enfoque de mi trabajo.

También quiero agradecer sinceramente a mis compañeros y amigos, cuyo apoyo y aliento fueron imprescindibles durante los momentos más difíciles. Su compañía y palabras de ánimo me brindaron la motivación constante para seguir adelante. En especial, me gustaría destacar a Dimitar Marinov Atanasov, quien estuvo a mi lado en los momentos más cruciales, brindándome fuerza y energía para dar lo mejor de mí. Su constante apoyo y sus palabras me recordaban cada día que el esfuerzo tendría su recompensa, por lo que le estoy profundamente agradecida.

Por último, deseo expresar mi más profundo agradecimiento a mi familia por su apoyo incondicional a lo largo de todo este proceso. Aunque no siempre comprendieran los detalles técnicos de mi proyecto, su constante respaldo, comprensión y confianza en mí fueron una fuente inagotable de inspiración y fortaleza, especialmente en los momentos más difíciles.

Resum

Este projecte s'enfoca en el desenvolupament d'una aplicació de tecnologia de realitat estesa (XR) per a abordar les necessitats de rehabilitació motora i estimulació cognitiva en individus afectats per uns certs tipus de dany cerebral adquirit. L'aplicació oferix una experiència interactiva i envolvent on els usuaris tenen la capacitat de construir puzles tridimensionals utilitzant cubs virtuals.

Esta activitat no sols promou la recuperació d'habilitats motores, sinó també la recuperació d'unes certes habilitats cognitives de reconeixement de formes i raonament especial que es veuen afectats en uns certs tipus de dany cerebral. Per a això s'empraran ferramentes avançades de desenvolupament d'aplicacions de realitat estesa. Este enfocament tecnològic permetrà una major personalització i adaptació a les necessitats individuals de cada usuari, la qual cosa maximitzarà l'efectivitat i la participació en el procés de rehabilitació i estimulació cognitiva.

La implementació es durà a terme en la plataforma Unity, i s'adaptarà per a funcionar sense problemes en dispositius Oculus Quest 2, brindant així una experiència immersiva i altament efectiva que té el potencial de transformar la forma en què s'aborden estos desafiaments de salut.

Paraules clau: Realitat Estesa (XR), rehabilitació motora, estimulació cognitiva, malalties cerebrals, Oculus Quest 2.

Resumen

Este proyecto se enfoca en el desarrollo de una aplicación de tecnología de realidad extendida (XR) para abordar las necesidades de rehabilitación motora y estimulación cognitiva en individuos afectados por ciertos tipos de daño cerebral adquirido. La aplicación ofrece una experiencia interactiva y envolvente donde los usuarios tienen la capacidad de construir puzzles tridimensionales utilizando cubos virtuales.

Esta actividad no solo promueve la recuperación de habilidades motoras, sino también la recuperación de ciertas habilidades cognitivas de reconocimiento de formas y razonamiento especial que se ven afectados en ciertos tipos de daño cerebral. Para ello se emplearán herramientas avanzadas de desarrollo de aplicaciones de realidad extendida. Este enfoque tecnológico permitirá una mayor personalización y adaptación a las necesidades individuales de cada usuario, lo que maximizará la efectividad y la participación en el proceso de rehabilitación y estimulación cognitiva.

La implementación se llevará a cabo en la plataforma Unity, y se adaptará para funcionar sin problemas en dispositivos Oculus Quest 2, brindando así una experiencia inmersiva y altamente efectiva que tiene el potencial de transformar la forma en que se abordan estos desafíos de salud.

Palabras clave: Realidad Extendida (XR), rehabilitación motora, estimulación cognitiva, enfermedades cerebrales, Oculus Quest 2.

Abstract

This project focuses on the development of an extended reality (XR) technology application to address the motor rehabilitation and cognitive stimulation needs of individuals affected by certain types of acquired brain injury. The application offers an interactive and immersive experience where users have the ability to build three-dimensional puzzles using virtual cubes.

This activity not only promotes the recovery of motor skills, but also the recovery of certain cognitive skills of shape recognition and special reasoning that are affected in certain types of brain damage. For this purpose, advanced tools for the development of extended reality applications will be used. This technological approach will allow greater customization and adaptation to the individual needs of each user, which will maximize the effectiveness and participation in the rehabilitation and cognitive stimulation process.

The implementation will run on the Unity platform, and will be adapted to run seamlessly on Oculus Quest 2 devices, providing an immersive and highly effective experience that has the potential to transform the way these health challenges are addressed.

Keywords: Extended Reality (XR), motor rehabilitation, cognitive simulation, brain disease, Oculus Quest 2.

Tabla de contenidos

Índice de figuras

Índice de códigos

CAPÍTULO 1
Introducción

El proyecto desarrollado aborda la rehabilitación de individuos con **daño cerebral adquirido (DCA)** mediante el uso de tecnologías de realidad extendida. Pero, ¿qué es el DCA? El daño cerebral adquirido es una lesión repentina en el cerebro ocasionada por eventos como traumatismos craneoencefálicos (TBI), interrupciones del flujo sanguíneo cerebral (ictus), tumores, infecciones anorexia o intervenciones quirúrgicas. Estas lesiones tienen un impacto significativo en la calidad de vida de los pacientes y sus familias debido a las secuelas que pueden abarcar desde anomalías en la comunicación y percepción hasta alteraciones físicas, cognitivas y emocionales.

En España, según la **encuesta de Discapacidad, Autonomía Personal y Situaciones de Dependencia (EDAD)** del INE, en 2018 se contabilizaron 420.064 personas afectadas por DCA, y cada año se registran más de 100.000 nuevos casos. Sin embargo, la **rehabilitación** tras el alta hospitalaria no es universal: un **12 %** de las personas con DCA no recibe rehabilitación, un **43 %** solo la recibe temporalmente, y solo un **45 %** sigue un tratamiento de rehabilitación de forma continuada (Figura 1.1). Esta situación subraya la necesidad de incrementar la oferta de plazas de rehabilitación y de incorporar nuevas tecnologías que faciliten la recuperación individualizada, especialmente en el área motora [1].

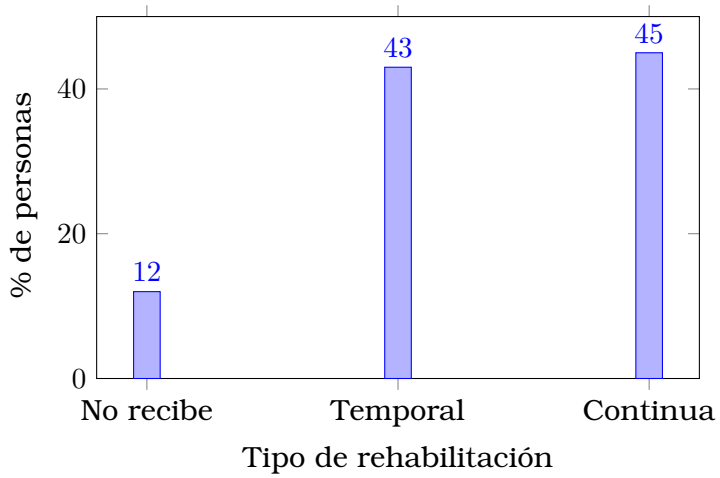

Figura 1.1: Porcentaje de personas con DCA en España según la rehabilitación recibida.

La aplicación de la realidad virtual (VR) en la neurorehabilitación ha mostrado resultados prometedores. La VR crea entornos tridimensionales inmersivos y altamente interactivos, lo que ofrece a los pacientes la oportunidad de realizar ejercicios de rehabilitación de manera más motivadora y adaptativa. Estudios recientes, como el de **Faria et al. (2023)**, han destacado la eficacia de la VR en la evaluación y

rehabilitación de personas con DCA, mejorando tanto las **funciones cognitivas** como **motoras** de los pacientes [2].

Investigaciones específicas, como la presentada por **Rábago y Wilken (2011)**, han demostrado mejoras significativas en la función motora utilizando programas de rehabilitación basados en VR. Este enfoque ha permitido integrar tareas de equilibrio y sensoriales en un entorno controlado, acelerando la recuperación de los pacientes [3]. Otro estudio de caso realizado por **Garber et al. (2014)** mostró que el tratamiento con VR puede ser un complemento efectivo a los programas de rehabilitación estándar, especialmente en pacientes con lesiones cerebrales traumáticas leves complicadas y recuperación retardada. La capacidad de adaptar rápidamente los escenarios terapéuticos a las necesidades individuales del paciente y proporcionar un entorno controlado y enfocado de rehabilitación fue crucial para los resultados positivos observados[4].

El uso continuado de **tecnologías XR** para la rehabilitación ha llevado a una mejora en la calidad de vida de los pacientes con DCA. La combinación de la **realidad virtual (VR)** y **realidad augmentada (AR)** permite abordar tanto los aspectos motores como cognitivos, ofreciendo un enfoque integral para la rehabilitación. Además, estas tecnologías incrementan la **motivación del paciente**, lo cual es esencial para el éxito del tratamiento.

1.1 Motivación

El principal motivo que me ha llevado a desarrollar este proyecto es mi profundo deseo de aplicar la tecnología para mejorar la calidad de vida de las personas afectadas por **daño cerebral adquirido (DCA)**. He podido observar de cerca como estas condiciones, como **ictus** y **traumatismos craneoencefálicos**, pueden tener un impacto devastador no solo en los propios pacientes que lo sufren, sino también en sus familiares y cuidadores. Las limitaciones en las **capacidades motoras y cognitivas** que resultan del DCA pueden hacer que la recuperación sea un proceso largo y arduo, con métodos tradicionales que a menudo resultan monótonos y desmotivadores.

Mi interés por la tecnología y su aplicación en la salud me ha llevado a explorar como las innovaciones en la **realidad extendida (XR)**, que incluyen **realidad virtual (VR)** y **aumentada (AR)**, pueden ofrecer soluciones más efectivas y atractivas para la rehabilitación. La capacidad de XR para crear entornos inmersivos y altamente interactivos puede transformar la forma en que los pacientes realizan sus ejercicios de rehabilitación, haciéndolos más motivadores y adaptados a sus necesidades individuales.

Más allá de los beneficios técnicos, mi motivación personal proviene de la convicción de que podemos utilizar la tecnología para hacer que el proceso de recuperación sea más humano y menos frustrante. Me gustaría que los pacientes con DCA sientan que sus terapias son más que una rutina repetitiva, que puedan ver y sentir su progreso de manera tangible y que encuentren en la tecnología una aliada en su camino hacia la recuperación.

Estoy segura de que al integrar XR en los programas de rehabilitación podemos no solo mejorar los resultados clínicos, sino también enriquecer la experiencia del paciente, haciendo que su camino hacia la recuperación sea más **interactivo, motivador y adaptado a sus necesidades específicas**.

1.2 Objetivos

El principal objetivo de este trabajo es desarrollar una **aplicación de realidad extendida (XR)** que facilite la rehabilitación motora y la estimulación cognitiva en pacientes con daño cerebral adquirido. La aplicación permitirá a los usuarios **interactuar con puzzles tridimensionales utilizando cubos virtuales**, promoviendo tanto la recuperación de habilidades motoras como cognitivas. Los objetivos se pueden dividir en principales y secundarios:

1.2.1. Objetivos principales

- **Desarrollar una aplicación XR en la plataforma Unity**:
  - Implementar una aplicación robusta y eficiente que funcione de manera óptima en dispositivos **Oculus Quest 2**.
  - Definir una arquitectura sólida que incluya tanto la interfaz de usuario (UI) como la lógica de negocio y los algoritmos de seguimiento de movimiento y interacción.
- **Facilitar la rehabilitación motora a través de actividades interactivas**:
  - Diseñar actividades que promuevan la coordinación y fuerza de los miembros superiores mediante la manipulación de cubos virtuales.
  - Asegurar que las actividades puedan ser personalizadas según las necesidades y capacidades individuales de cada paciente.
- **Estimular habilidades cognitivas específicas**:
  - Desarrollar puzzles tridimensionales que desafíen y mejoren el reconocimiento de formas y el razonamiento espacial.
  - Implementar niveles de dificultad ajustables para mantener el interés y la motivación del usuario.
- **Evaluar y monitorear el progreso del paciente**:
  - Incorporar funcionalidades que registren el desempeño del usuario y proporcionen retroalimentación en tiempo real.
  - Permitir que los terapeutas puedan acceder y analizar los datos para ajustar los programas de rehabilitación según sea necesario.

1.2.2. Objetivos secundarios

- **Promover la accesibilidad y usabilidad de la aplicación**:
  - Asegurar que la aplicación sea fácil de usar para personas con diferentes niveles de habilidad técnica y limitaciones físicas.
  - Adaptar la aplicación para ser multilingüe, comenzando con español e inglés.
- **Fomentar la colaboración interdisciplinaria**:
  - Colaborar con profesionales de la salud, incluyendo terapeutas físicos y ocupacionales, para asegurar que la aplicación cumpla con los estándares clínicos.

  - Facilitar la formación y el uso de la aplicación en entornos clínicos y domiciliarios.

- **Evaluar el impacto de la aplicación en la calidad de vida de los pacientes**:
  - Realizar estudios piloto para medir la efectividad de la aplicación en la mejora de las habilidades motoras y cognitivas.
  - Recoger y analizar el feedback de los usuarios y sus familias para mejorar continuamente la aplicación.

- **Contribuir en el campo de la neurorehabilitación con tecnología avanzada**:
  - Publicar los resultados y hallazgos del proyecto en conferencias y revistas académicas para compartir el conocimiento adquirido.
  - Inspirar a otras organizaciones y desarrolladores a adoptar tecnologías XR en sus propios programas de rehabilitación.

1.3 Impacto esperado

El impacto esperado de la aplicación **MindCubeXR** en la rehabilitación motora y cognitiva de pacientes con daño cerebral adquirido (DCA), es significativo tanto a nivel individual como en el ámbito de la neurorehabilitación en general. Se espera que este proyecto genere beneficios notables en las siguientes áreas:

- **Mejora en la recuperación de habilidades motoras y cognitivas**:
  - **Recuperación motora:** La utilización de actividades interactivas con cubos virtuales ayudará a mejorar la coordinación, fuerza y destreza de los miembros superiores de los pacientes. La interacción en un entorno controlado y motivador puede acelerar el proceso de recuperación comparado con métodos tradicionales.
  - **Estimulación cognitiva:** Los puzzles tridimensionales diseñados para desafiar el reconocimiento de formas y el razonamiento espacial contribuirán a la mejora de funciones cognitivas específicas, cruciales para la reintegración social y funcional de los pacientes.

- **Aumento de la motivación y adherencia al tratamiento**:

  La naturaleza inmersiva y lúdica de la realidad extendida (XR) tiene el potencial de aumentar la motivación de los pacientes para participar regularmente en las sesiones de rehabilitación. La retroalimentación en tiempo real y la personalización de las actividades aseguran que los pacientes se mantengan comprometidos con su proceso de recuperación.

- **Personalización y adaptabilidad del tratamiento**:

  MindCubeXR permitirá la personalización de los programas de rehabilitación para ajustarse a las necesidades específicas de cada paciente, considerando sus capacidades y limitaciones individuales. Esta flexibilidad es esencial para maximizar la efectividad de la rehabilitación.

- **Facilitación de la monitorización y evaluación del progreso**:

  La aplicación incorpora funcionalidades avanzadas de seguimiento del rendimiento, proporcionando datos detallados sobre el progreso del paciente. Esta

información es valiosa para los terapeutas, permitiéndoles ajustar los programas de rehabilitación en tiempo real y mejorar continuamente la calidad del tratamiento.

- **Reducción de costos y accesibilidad**:

  Al implementar una solución basada en dispositivos accesibles como Oculus Quest 2, MindCubeXR ofrece una alternativa costo-efectiva a las tecnologías de rehabilitación tradicionales. Además, su adaptabilidad para uso domiciliario se espera que facilite el acceso a la rehabilitación para pacientes que pueden tener dificultades para asistir a sesiones presenciales.

- **Contribución al campo de la neurorehabilitación**:

  Los datos y resultados generados por MindCubeXR contribuirán al conocimiento científico y clínico en el campo de la neurorehabilitación. La publicación de estos hallazgos en revistas académicas y conferencias fomentará la adopción de tecnologías XR en otros programas de rehabilitación y promoverá la innovación continua.

- **Impacto social y bienestar del paciente**:

  Al mejorar la efectividad de la rehabilitación motora y cognitiva, MindCubeXR tiene el potencial de mejorar significativamente la calidad de vida de los pacientes con DCA. La capacidad de recuperar habilidades perdidas y reintegrarse en su entorno social y laboral es un objetivo primordial del proyecto.

1.4 Estructura de la memoria

La estructura que se ha seguido en la memoria, se rige por una serie de capítulos, que se encargan de describir los puntos más principales del proyecto, cuyo contenido es el siguiente:

1. **Introducción:** Se presenta el tema principal del trabajo, el uso de tecnologías de realidad extendida (XR) en la rehabilitación de pacientes con daño cerebral adquirido. Se define la motivación, los objetivos del TFM, el impacto esperado de MindCubeXR en la rehabilitación motora y cognitiva y la estructura del trabajo.

2. **Fundamentos teóricos:** Se proporciona el marco teórico necesario para entender el contexto y la relevancia del proyecto. Incluye una revisión de las causas y efectos del daño cerebral adquirido, los principios de rehabilitación motora y cognitiva, y el uso de la XR en el ámbito de la salud.

3. **Estado del arte:** Se analiza las tecnologías y metodologías actuales en la rehabilitación con XR. Examina el contexto histórico, los avances recientes y las soluciones existentes, identificando sus fortalezas y debilidades. Presenta las soluciones actuales y la propuesta de MindCubeXR, destacando como se diferencia o mejora en comparación con el estado actual.

4. **Tecnologías Utilizadas:** Se detallan las herramientas y tecnologías empleadas en el proyecto, como Unity, NUnit, Unity Test Runner, Visual Studio 2022, Oculus Quest 2, entre otros. Describe el propósito de cada tecnología en el desarrollo de la aplicación y cómo contribuye a la implementación de MindCubeXR.

5. **Solución propuesta: MindCubeXR:** Se describe en detalle la solución desarrollada, MindCubeXR. Se explica la estructura del proyecto, la organización de directorios, la jerarquía de la escena principal y la arquitectura empleada. Además, se detalla la implementación de la lógica de la aplicación, incluyendo la creación y gestión de cubos, la interacción entre objetos y la programación de la lógica del juego.

6. **Pruebas y Validación:** Se presenta el proceso de pruebas de la aplicación. Incluyendo las pruebas unitarias, de integración y de usabilidad realizadas para garantizar la funcionalidad y la experiencia del usuario. Además, se detalla el diseño y la metodología de las pruebas, los criterios de evaluación y los resultados obtenidos.

7. **Conclusiones y Trabajos a Futuro:** Se realiza un resumen de los objetivos alcanzados, los resultados obtenidos y se sugieren futuras líneas de investigación y desarrollo.

8. **Bibliografía:** Se enumeran todas las fuentes bibliográficas utilizadas en el desarrollo del TFM.

9. **Anexos:** Se proporciona información adicional relevante para el proyecto, como los fragmentos de código desarrollados. Además, se especifican los diferentes Objetivos de Desarrollo Sostenible (ODS) relacionados estrechamente con el mismo.

CAPÍTULO 2
Fundamentos teóricos

2.1 Daño cerebral adquirido: causas, tipos y efectos

El daño cerebral adquirido (DCA) es una **lesión en el sistema nervioso** proveniente de un accidente posterior al nacimiento, una vez que el cerebro ha sido desarrollado. Concretamente, puede resultar de diversas causas, incluyendo eventos traumáticos como **traumatismos craneoencefálicos (TBI)** y **accidentes cerebrovasculares (ACVs)**, así como tumores, infecciones, anoxia y complicaciones de intervenciones quirúrgicas. Estos eventos pueden alterar significativamente la estructura y función del cerebro, afectando **la integridad física del tejido cerebral, la actividad metabólica y el funcionamiento de las células nerviosas** [5].

A continuación, se presenta un esquema (Figura 2.1) que ilustra los tipos de daño cerebral adquirido: [6]

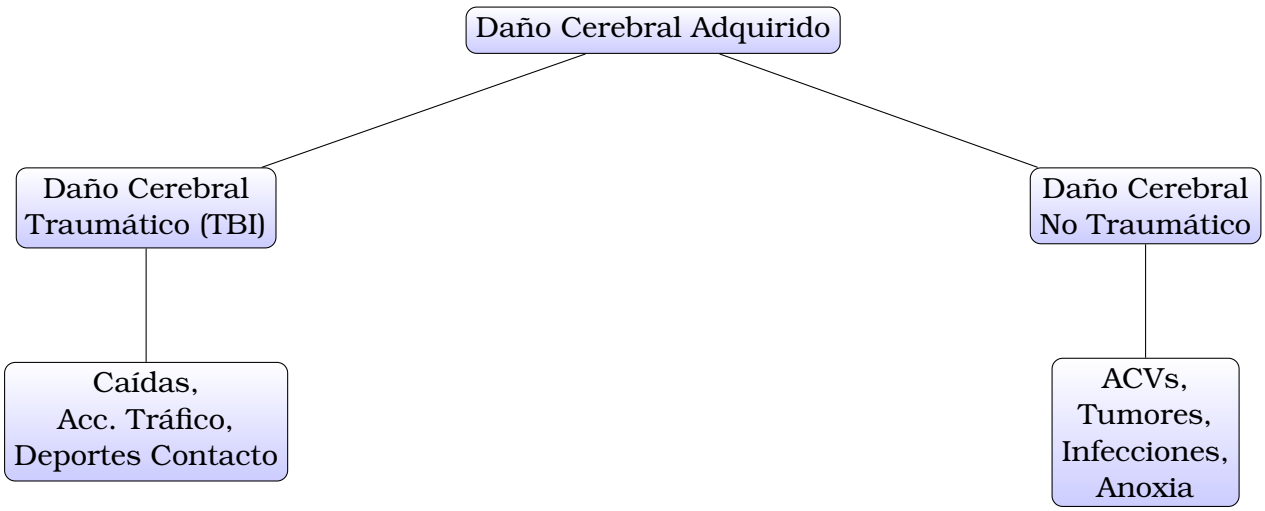

Figura 2.1: Esquema de los tipos de daño cerebral adquirido (DCA).

2.1.1. Tipos de DCA

El DCA se clasifica en dos categorías principales: traumático y no traumático.

Daño cerebral traumático (TBI)

El **daño cerebral traumático** incluye lesiones causadas por impactos externos, como caídas, accidentes de tráfico y deportes de contacto. Estos eventos pueden pro-

vocar fracturas de cráneo, hematomas subdurales, contusiones cerebrales y daños axonales difusos.

Daño cerebral no traumático

El **daño cerebral no traumático** abarca condiciones internas como derrames cerebrales, infecciones, tumores y anoxia.

- **Accidentes cerebrovasculares (ACVs)**: Interrupción del flujo sanguíneo al cerebro debido a un bloqueo o ruptura de los vasos sanguíneos.
- **Tumores cerebrales**: Crecimiento anormal de células en el cerebro que puede ser maligno o benigno.
- **Infecciones**: Como meningitis o encefalitis, que inflaman las membranas alrededor del cerebro o el propio tejido cerebral.
- **Anoxia**: Falta de oxígeno en el cerebro, normalmente causada por asfixia, ahogamiento o problemas cardíacos.

Los efectos del DCA pueden variar desde leves hasta severos, afectando las capacidades motoras, cognitivas, la comunicación y el comportamiento emocional de los pacientes. Estos efectos pueden ser temporales o permanentes, y su gravedad depende de la ubicación y extensión del daño.

2.2 Principios y enfoques de rehabilitación motora y cognitiva

La rehabilitación de pacientes con daño cerebral adquirido (DCA) se fundamenta en principios y enfoques específicos dirigidos a restaurar tanto las funciones motoras como cognitivas. Estos métodos son esenciales para ayudar a los pacientes a recuperar su independencia y mejorar su calidad de vida.

2.2.1. Rehabilitación Motora

La **rehabilitación motora** incluye una variedad de técnicas terapéuticas destinadas a mejorar la movilidad, la coordinación y la fuerza de los pacientes. La terapia física, los ejercicios de coordinación y las actividades para mejorar la movilidad son componentes clave de este enfoque. Estos métodos son cruciales para ayudar a los pacientes a recuperar la independencia en sus actividades diarias.

Los enfoques tradicionales, como la terapia física, los ejercicios de coordinación y el entrenamiento de movilidad, se centran en mejorar la fuerza muscular, la flexibilidad y la resistencia física. La evidencia sugiere que la práctica intensiva y repetitiva es vital para una rehabilitación motora efectiva, promoviendo así la neuroplasticidad y el aprendizaje motor [10].

2.2.2. Rehabilitación Cognitiva

La **rehabilitación cognitiva** se enfoca en mejorar funciones mentales como la memoria, la atención, el razonamiento y la resolución de problemas. Las técnicas utilizadas incluyen ejercicios de estimulación cognitiva, terapia ocupacional y estrategias conductuales diseñadas para desafiar y fortalecer las capacidades mentales afectadas.

El **objetivo principal** de la rehabilitación cognitiva es optimizar el comportamiento del paciente, entendiendo este como la expresión de la cognición. La intervención temprana y sostenida es crucial ya que puede prevenir complicaciones adicionales y promover una recuperación más rápida y completa. Además, un enfoque multidisciplinario que incluya fisioterapeutas, terapeutas ocupacionales, neuropsicólogos y otros profesionales de la salud es vital para abordar las diversas necesidades de los pacientes [11].

2.2.3. Enfoque Multidisciplinario

La rehabilitación efectiva a menudo requiere un enfoque multidisciplinario, involucrando a diversos profesionales de la salud como fisioterapeutas, terapeutas ocupacionales, neuropsicólogos y logopedas. Este esfuerzo colaborativo asegura que se aborden todos los aspectos de las necesidades del paciente, llevando a resultados más completos y efectivos en la rehabilitación [10].

2.3 Tecnología de realidad extendida (XR) y su aplicación en el ámbito de la salud

La tecnología de **realidad extendida** (XR), que incluye **la realidad virtual** (VR), la **realidad aumentada** (AR) y la **realidad mixta** (MR), está revolucionando el ámbito de la salud al ofrecer nuevas y efectivas herramientas para la rehabilitación y el tratamiento de diversas condiciones médicas. Estas tecnologías no solo mejoran la efectividad de las terapias tradicionales, sino que también abren nuevas posibilidades en el tratamiento y la educación médica.

2.3.1. Beneficios de la XR en la Rehabilitación

- **Entornos de Entrenamiento Seguros y Controlados:** La XR permite la creación de entornos simulados donde los pacientes pueden practicar habilidades motoras y cognitivas sin riesgos. Por ejemplo, la VR puede sumergir a los pacientes en entornos tridimensionales interactivos que replican situaciones de la vida real, mejorando la efectividad de la rehabilitación y aumentando la motivación y el compromiso de los pacientes.
- **Aumento de la Motivación y el Compromiso:** Las experiencias inmersivas de XR incrementan la adherencia al tratamiento, ya que los pacientes encuentran más atractiva y motivadora la interacción con entornos virtuales. Esto es particularmente útil en la rehabilitación de pacientes con daño cerebral adquirido (DCA), donde la motivación es clave para una recuperación efectiva [7].
- **Personalización y Flexibilidad:** La XR permite diseñar programas de rehabilitación personalizados, realizar un seguimiento detallado del progreso del paciente y ajustar los ejercicios en tiempo real para mejorar la calidad del tratamiento. Dispositivos como el Oculus Quest 2, combinados con plataformas de desarrollo como Unity, facilitan la creación de aplicaciones de XR personalizadas y accesibles, mejorando la experiencia terapéutica [8].

2.3.2. Aplicaciones Específicas de la XR

- **Realidad Virtual (VR) para la Rehabilitación:** La VR puede sumergir a los pacientes en entornos tridimensionales interactivos donde pueden practicar habilidades motoras y cognitivas en un entorno simulado que refleja situaciones

de la vida real. Un ejemplo es el uso de VR para **entrenar a pacientes en habilidades motoras** finas mediante la manipulación de objetos virtuales, lo que ayuda a mejorar la coordinación y la destreza.

- **Realidad Aumentada (AR) en la Cirugía y Diagnóstico:** La AR permite la superposición de información digital sobre el mundo real, lo que puede asistir en la realización de tareas médicas y diagnósticas. Por ejemplo, los cirujanos pueden usar AR para visualizar imágenes detalladas de anatomía sobre el paciente durante una operación, mejorando la precisión y reduciendo los riesgos quirúrgicos [8]. Un caso destacado es el uso de AR en la cirugía de fusión espinal, donde los cirujanos de la Universidad Johns Hopkins utilizaron cascos de AR para realizar la primera cirugía asistida por XR.
- **Realidad Mixta (MR) para la Educación Médica:** La MR combina VR y AR para crear entornos interactivos manipulables en tiempo real. Esto permite una interacción más natural y efectiva con los elementos virtuales, mejorando la experiencia de aprendizaje para los profesionales de la salud. La MR es especialmente útil en la formación de médicos y personal sanitario, permitiendo simulaciones de procedimientos médicos complejos en un entorno controlado.[9]
- **Evidencia y Eficacia de la XR en la Rehabilitación:** Estudios han demostrado que el uso de XR puede acelerar la recuperación de funciones motoras y cognitivas, proporcionando resultados positivos en comparación con los enfoques tradicionales. Por ejemplo, un estudio publicado en el ***Future Healthcare Journal*** destaca que las tecnologías XR pueden mejorar los resultados educativos y de rehabilitación al ofrecer experiencias de aprendizaje más inmersivas y personalizadas. Este estudio concluye que la implementación centrada en el aprendizaje y el uso de XR en contextos educativos y terapéuticos puede causar un cambio de paradigma, mejorando significativamente la calidad y efectividad del tratamiento [7].

2.3.3. Aplicaciones Prácticas y Futuras de la XR en la Salud

- **Telemedicina y Cuidado Remoto:** Las tecnologías XR están aumentando la accesibilidad y efectividad de la atención médica remota, como la telemedicina y la monitorización de pacientes. La VR, por ejemplo, crea un entorno virtual inmersivo para que pacientes y clínicos se conecten, casi como si estuvieran en la misma sala, mejorando la calidad de la atención remota y reduciendo barreras, especialmente para pacientes en comunidades remotas o desatendidas.
- **Gestión del Dolor y la Ansiedad:** La VR también se utiliza para gestionar el dolor y la ansiedad, proporcionando a los pacientes un escape terapéutico inmersivo que ayuda a distraerlos del dolor agudo o crónico. Un estudio de 2020 encontró que el **88 %** de los pacientes reportaron una reducción del dolor después de la terapia con VR, y más de 100 hospitales en los Estados Unidos utilizan VR como una herramienta efectiva para manejar el dolor y la ansiedad.
- **Tratamiento de Enfermedades Mentales:** Compañías como **XRHealth** ofrecen aplicaciones pre-cargadas para cascos de VR que incluyen sesiones de terapia guiada, juegos y sesiones privadas con un terapeuta. Estas herramientas pueden ser útiles para tratar diversas enfermedades mentales, como el **Trastorno Obsesivo Compulsivo (TOC)**, la depresión y el **trastorno de estrés postraumático (TEPT)**. La VR también puede ayudar a tratar fobias exponiendo a los pacientes a sus miedos en un entorno seguro e inmersivo [8].

CAPÍTULO 3
Estado del Arte

El desarrollo de tecnologías de realidad extendida (XR) ha revolucionado diversos campos, incluyendo la rehabilitación motora y cognitiva. Actualmente, existen múltiples aplicaciones que utilizan XR para apoyar la recuperación de pacientes con daño cerebral adquirido (DCA). Sin embargo, estas soluciones a menudo se centran en aspectos limitados de la rehabilitación, como ejercicios de movilidad básica o simples juegos cognitivos, sin aprovechar completamente el potencial de XR para ofrecer experiencias de rehabilitación más integrales y personalizadas.

3.1 Contexto histórico

La evolución de la tecnología de **Realidad Extendida (XR)**, que incluye la **Realidad Virtual (VR)**, la **Realidad Aumentada (AR)** y la **Realidad Mixta (MR)**, ha transformado significativamente la rehabilitación motora y cognitiva (Figura 3.1). Esta evolución ha sido impulsada por avances en hardware, software y la creciente accesibilidad de estas tecnologías.

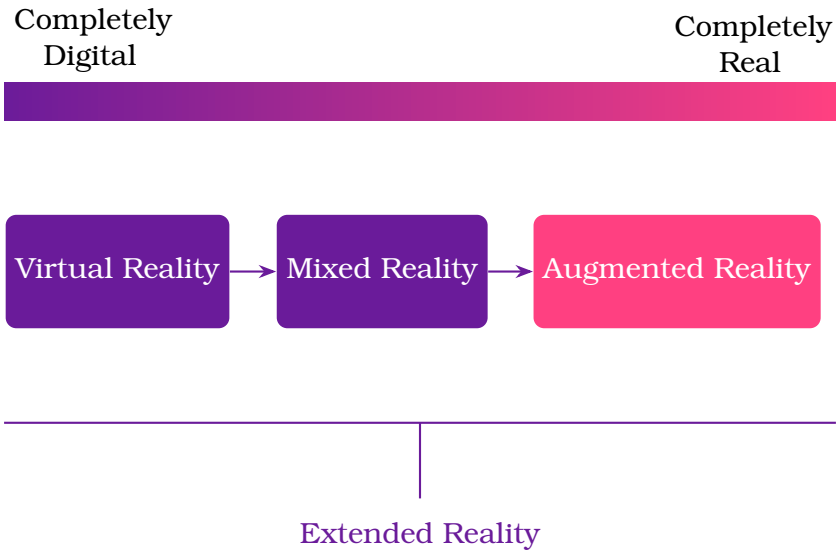

Figura 3.1: El espectro de la realidad virtual

3.1.1. Primeras Etapas y Desarrollo Inicial

La historia de la Realidad Virtual (VR) se remonta a más de un siglo atrás. Los primeros indicios de la exploración de entornos inmersivos pueden rastrearse hasta

1838 con la **invención del estereoscopio por Charles Wheatstone**. Este dispositivo óptico permitía recrear una ilusión tridimensional mediante el uso de espejos para reflejar imágenes diferentes en cada ojo, lo que marcó un avance significativo en el estudio de la percepción de profundidad binocular y la visión estereoscópica.

En 1939, el **View-Master** fue presentado como una evolución del estereoscopio, utilizando discos de cartón para mostrar imágenes tridimensionales. Este dispositivo se usó inicialmente para propósitos educativos y turísticos, pero más tarde se convirtió en un juguete popular, mostrando la capacidad de la VR para capturar la imaginación del público.

El **Sensorama**, desarrollado por **Morton Heilig en 1962**, fue un hito importante en la evolución de la VR. Este dispositivo integraba imágenes, sonido, olfato y movimiento para crear una experiencia inmersiva completa. Aunque nunca recibió el apoyo económico necesario para convertirse en un producto masivo, el Sensorama es considerado uno de los antecesores más importantes de los sistemas actuales de realidad virtual.

Otro avance significativo fue el **Aspen Movie Map**, desarrollado **en 1977 por el MIT**. Este programa de simulación permitía a los usuarios realizar paseos virtuales por Aspen, Colorado, anticipando tecnologías modernas como Google Street View. Este proyecto demostró el potencial de la VR para transportar a las personas a lugares remotos sin necesidad de moverse físicamente.

En 1985, NASA lanzó el proyecto **VIEW VR**, destinado a controlar robots en el espacio. Cada **headset**[1] costaba alrededor de $40.000 y contaba con una resolución monocromática, pantallas de baja frecuencia, reconocimiento por voz, controladores y guantes. Este proyecto, aunque limitado por las capacidades tecnológicas de la época, sentó las bases para futuros desarrollos en XR [12].

La década de 1990 vio la introducción de sistemas de VR más accesibles, como el **Virtuality y el Sega VR**. Estos dispositivos, aunque todavía costosos y con limitaciones tecnológicas, permitieron al público general tener una primera experiencia con entornos virtuales inmersivos. En 1991, Sega presentó su sistema VR, diseñado para potenciar los juegos arcade de su sistema de entretenimiento para el hogar **Mega Drive**. Aunque nunca se lanzó oficialmente al mercado, este sistema introdujo importantes avances en la tecnología de sensores y seguimiento de movimiento.

En 1995, **Nintendo lanzó el Virtual Boy**, el primer dispositivo de juegos en 3D portátil. Sin embargo, su falta de gráficos en color, soporte de software y comodidad llevaron a su fracaso comercial. A pesar de estos desafíos, estos primeros dispositivos marcaron el inicio de las innovaciones futuras en la tecnología XR [13].

3.1.2. Avances en el Siglo XXI

El **siglo XXI** trajo consigo **avances significativos** en la tecnología de XR, impulsados por mejoras en el procesamiento gráfico y la miniaturización de componentes. Empresas como **Oculus VR** (adquirida por Facebook) y **HTC** desarrollaron dispositivos más accesibles y avanzados, como el **Oculus Rift y el HTC Vive**, que ofrecían experiencias inmersivas de alta calidad a precios más asequibles.

Paralelamente, la **Realidad Aumentada (AR)** comenzó a ganar popularidad con aplicaciones móviles como **"Pokémon Go"**, que demostró el potencial de superponer

[1] Se asocia con dispositivos de realidad virtual (VR) o realidad aumentada (AR), pero en los años 80, estos dispositivos eran mucho más costosos y menos avanzados debido a las limitaciones tecnológicas de la época.

información digital en el mundo real a través de dispositivos portátiles. Este juego, lanzado en 2016, no solo captó la atención global, sino que también subrayó como la AR puede integrarse en la vida cotidiana para crear experiencias interactivas y atractivas.

La **Realidad Mixta (MR)**, aunque más reciente, ha **combinado elementos de VR y AR** para permitir interacciones más integradas y naturales entre el usuario y los entornos virtuales y físicos. La MR ha mostrado un gran potencial en aplicaciones industriales, médicas y de entretenimiento permitiendo a los usuarios manipular objetos virtuales dentro del mundo real con un alto grado de precisión [14, 15, 16, 17, 18].

Estos avances han abierto el camino para una nueva era de aplicaciones en **rehabilitación médica**, donde la VR y otras tecnologías XR se están utilizando para mejorar la recuperación cognitiva y motora en pacientes con diversas condiciones de salud. Por ejemplo, estudios recientes han demostrado la eficacia de la VR en la evaluación y entrenamiento de la memoria espacial en individuos con daño cerebral [17], así como en la rehabilitación cognitiva de pacientes que han sufrido un accidente cerebrovascular [18].

3.2 Crítica al Estado del Arte

En la actualidad, existen varias **aplicaciones de VR** que están transformando el campo de la rehabilitación, proporcionando a los pacientes experiencias terapéuticas más atractivas y efectivas. A continuación, se presentan y critican algunas de las aplicaciones más relevantes en este ámbito.

3.2.1. I Am Ready

`I Am Ready`[2] es una innovadora iniciativa de realidad virtual que forma parte de una colaboración entre la **Fundación Ricky Rubio y Nixi For Children**, enfocada en brindar apoyo a niños que enfrentan tratamientos oncológicos en diversos hospitales. Esta aplicación de realidad virtual no solo tiene como objetivo distraer, sino también preparar emocionalmente a los niños para enfrentar tratamientos médicos complejos, como la radioterapia y la protonterapia. Entre sus funcionalidades principales se pueden destacar las siguientes:

- **Simulación de Entornos Médicos:** La aplicación permite a los niños experimentar de manera inmersiva una visita a las salas de tratamiento antes de recibir la terapia real. Este enfoque les ayuda a familiarizarse con el entorno médico, reduciendo el miedo y la ansiedad que a menudo acompañan a estos procedimientos.
- **Reducción del Estrés:** Al permitir que los niños vivan una simulación del tratamiento, I Am Ready ayuda a que enfrenten los procedimientos sin la necesidad de sedación en muchos casos, lo que es beneficioso tanto para su bienestar emocional como para la eficacia del tratamiento.
- **Distribución Amplia y Colaborativa:** Hasta la fecha, se han distribuido más de 1.000 **Nixikits**[3] en más de 20 hospitales en España, Chile y Estados Unidos,

[2] https://therickyrubiofoundation.org/product/i-am-ready/

[3] **Nixikits:** Kits de realidad virtual diseñados para preparar emocionalmente a niños que enfrentan tratamientos médicos complejos, como la radioterapia, permitiéndoles experimentar de forma inmersiva los entornos médicos y reducir su ansiedad.

con contenido desarrollado en colaboración con médicos, psicólogos y técnicos de varias instituciones sanitarias de renombre.

I Am Ready ofrece una valiosa herramienta emocional y preparatoria para los niños que enfrentan tratamientos oncológicos, pero su alcance está limitado a contextos específicos de oncología pediátrica. La implementación en otros campos de la rehabilitación pediátrica podría explorar otros tipos de intervenciones, como la preparación para cirugías o tratamientos crónicos, ampliando así su aplicabilidad.

3.2.2. MindMaze

MindMaze[4] es una plataforma de realidad virtual que combina neurociencia, inteligencia artificial y realidad mixta para ayudar en la recuperación de pacientes con accidentes cerebrovasculares, lesiones cerebrales traumáticas u otras condiciones neurológicas. Entre sus aplicaciones más destacadas se encuentran:

- **MindPod Dolphin:** Esta aplicación utiliza un enfoque inmersivo para promover la recuperación de la habilidad motora y la función cognitiva en pacientes con accidentes cerebrovasculares. Se basa en un juego animado en un entorno oceánico donde los pacientes usan un chaleco antigravedad para entrenar el control motor fino de la extremidad superior.
- **MindMotion™ PRO:** Diseñado para tratar a pacientes con discapacidad grave tras un accidente cerebrovascular, este sistema permite a los pacientes controlar su brazo mediante estimulación eléctrica multicanal, generando movimientos significativos de las extremidades superiores.
- **TOAP Run:** Un videojuego serio desarrollado para pacientes con enfermedad de Parkinson que les ayuda a realizar movimientos más grandes y rápidos, reduciendo la incidencia de caídas y mejorando la capacidad motora.

MindMaze ofrece soluciones innovadoras y altamente interactivas, pero su implementación requiere infraestructura tecnológica avanzada y personal capacitado, lo que puede limitar su accesibilidad en entornos menos equipados. Además, el costo de estos sistemas puede ser excesivo para algunos centros de rehabilitación.

3.2.3. RehaCom

RehaCom[5] es un sistema de software para la rehabilitación cognitiva asistida por ordenador, desarrollado específicamente para ayudar en la recuperación de funciones cognitivas afectadas por daño cerebral, enfermedades neurodegenerativas u otras condiciones. Entre sus principales funcionalidades se encuentran:

- **Entrenamiento Autoadaptable:** Con 29 módulos de terapia disponibles en 27 idiomas, RehaCom adapta la dificultad de las tareas según el rendimiento del paciente, asegurando una progresión constante sin causar frustración.
- **Supervisión Remota:** Los terapeutas pueden prescribir programas personalizados que los pacientes realizan en casa, con la posibilidad de monitorear su progreso a distancia y ajustar la terapia en tiempo real.

[4] https://mindmaze.com/
[5] https://hasomed.de/es/productos/rehacom-terapia-cognitiva/

- **Evaluaciones Cognitivas:** RehaCom incluye módulos de screening que ayudan a detectar déficits cognitivos y a recomendar módulos de entrenamiento apropiados. Los resultados son presentados en gráficos y tablas que permiten un seguimiento detallado del progreso.

Aunque RehaCom es una herramienta poderosa y versátil, su complejidad y el costo de implementación pueden ser barreras significativas para su adopción en algunos entornos. La necesidad de una infraestructura tecnológica adecuada para la supervisión remota también puede limitar su uso.

3.2.4. Karuna Labs

`Karuna Labs`[6] se especializa en el **manejo del dolor crónico** a través de programas de realidad virtual que aprovechan la neuroplasticidad y la tecnología inmersiva. Los pacientes se sumergen en entornos virtuales diseñados para distraer al cerebro de la experiencia del dolor y promover la relajación.

Se encarga de ofrecer un enfoque innovador para el manejo del dolor, pero su enfoque en la neuroplasticidad y la distracción del dolor puede no ser aplicable a todos los tipos de rehabilitación, especialmente aquellos que requieren un enfoque más directo en la recuperación funcional.

3.3 Propuesta

En este proyecto se propone el desarrollo de una **aplicación de tecnología de realidad extendida (XR)** que aborde las necesidades de rehabilitación motora y estimulación cognitiva en individuos afectados por **daño cerebral adquirido (DCA)**. Esta aplicación ofrecerá una experiencia interactiva y envolvente en la que los usuarios podrán construir **puzzles tridimensionales utilizando cubos virtuales**. Esta actividad no solo promoverá la recuperación de habilidades motoras finas, sino que también fomentará el reconocimiento de formas y el razonamiento espacial, habilidades cognitivas frecuentemente afectadas por el DCA.

La aplicación XR tratará un tema emergente en el ámbito de la rehabilitación, donde se ha comprobado que no existe un enfoque similar al que se propone, especialmente en el uso combinado de realidad extendida para la rehabilitación tanto motora como cognitiva. Las investigaciones realizadas han demostrado que las actividades inmersivas pueden ser significativamente más efectivas que los métodos tradicionales, ofreciendo no solo una mejora en las capacidades físicas y cognitivas, sino también un aumento en la motivación y el compromiso del paciente.

Para diferenciarse de las aplicaciones existentes, en la propuesta se incluirán funcionalidades avanzadas como la personalización dinámica de la dificultad de los ejercicios según el progreso del usuario, la integración de feedback en tiempo real para informar sobre el desempeño y la adaptación específica a las necesidades individuales de cada paciente. Además, la aplicación se desarrollará en la **plataforma Unity** y estará optimizada para funcionar en dispositivos **Oculus Quest 2**, garantizando una experiencia inmersiva y accesible.

A pesar de que hay varias aplicaciones XR desarrolladas en el ámbito de la rehabilitación, muchas no logran satisfacer completamente las necesidades de los pacientes ni ofrecer una solución integrada que combine la rehabilitación motora y cognitiva de

[6] https://www.karunalabs.com/

manera efectiva. Esta insuficiencia ha llevado a que el impacto de estas tecnologías sea limitado y continúe siendo un área con un gran potencial sin explotar.

Por todas estas razones, se espera que la aplicación XR desarrollada logre cambiar esta situación y genere un interés significativo tanto en el ámbito clínico como en la sociedad en general. Además de destacar entre las demás aplicaciones existentes, creando un impacto positivo y significativo en la rehabilitación de pacientes con daño cerebral adquirido, mejorando su calidad de vida y promoviendo una recuperación más rápida y efectiva.

CAPÍTULO 4
Tecnologías utilizadas

En este capítulo se describen las **tecnologías empleadas** para llevar a cabo el desarrollo de la aplicación de realidad extendida (XR) para la rehabilitación motora y cognitiva en pacientes con daño cerebral adquirido (DCA). A continuación, se detallan las herramientas y tecnologías utilizadas en cada etapa del desarrollo del proyecto:

4.1 Unity

`Unity`[1] es un motor de desarrollo de videojuegos y aplicaciones interactivas en 3D y 2D, ampliamente utilizado en la industria para la creación de aplicaciones de realidad virtual (VR) y aumentada (AR). Desarrollado por Unity Technologies, se caracteriza por ser una de las herramientas más versátiles y potentes para la creación de contenido interactivo [20]. Algunas de las características por las que se compone son las siguientes:

- **Motor gráfico:** Soporta tanto gráficos tridimensionales como bidimensionales, ofreciendo herramientas especializadas para cada estilo, como la creación de terrenos en 3D y mapas de baldosas en 2D.
- **Física y colisiones:** Incluye un motor de física que permite simular interacciones físicas realistas dentro del entorno virtual.
- **Integración multiplataforma:** Permite exportar juegos y aplicaciones a una amplia variedad de plataformas, incluyendo Android, iOS, Windows, macOS, consolas de videojuegos y navegadores web.
- **Arquitectura fácil de entender:** Organiza los proyectos en escenas, que contienen los objetos del juego, permitiendo una estructura clara y una fácil manipulación de los elementos del proyecto.
- **API de scripting:** Proporciona una API robusta para acceder y manipular todas las características del motor, desde la posición y rotación de los objetos hasta la reproducción de audio y efectos visuales.
- **Gran almacén de activos:** Unity Asset Store proporciona una vasta colección de recursos, tanto gratuitos como de pago, que incluyen modelos 3D, texturas, sonidos y plantillas de código.
- **Capacidades de realidad vitual y aumentada:** Ofrece soporte nativo y extensible para dispositivos VR y AR, incluyendo la integración de tecnologías como ARCore, ARKit y XR Interaction Toolkit.

[1] https://unity.com/es

- **Opciones de canalización de renderizado:** Proporciona varias opciones para la renderización gráfica, incluyendo la capacidad de crear canalizaciones de renderizado personalizadas mediante la API Scriptable Render Pipeline.

Unity fue el núcleo del desarrollo de la aplicación XR, utilizado para crear los entornos virtuales, implementar las interacciones, y manejar la lógica del juego. Su capacidad para integrar múltiples tecnologías y SDKs facilitó el desarrollo de una experiencia inmersiva y altamente interactiva. Además, se selecciono para llevar a cabo el desarrollo la versión 2021.3.36f1 LTS por su estabilidad y soporte a largo plazo, lo que es crucial para el desarrollo de proyectos robustos y fiables.

4.2 NUnit

`NUnit`[2] es un framework de pruebas unitarias para .NET que permite a los desarrolladores escribir y ejecutar pruebas automatizadas para verificar que el código funcione correctamente. NUnit es ampliamente utilizado para garantizar la calidad del código mediante la ejecución de pruebas que verifican que los componentes individuales cumplan con las expectativas. Entre sus características principales se encuentran [21]:

- **Ejecución de pruebas automatizadas:** Permite ejecutar un conjunto de pruebas automáticamente para validar la funcionalidad de los componentes de software.
- **Asserts y atributos:** Proporciona una variedad de métodos de aserción y atributos para definir pruebas, establecer expectativas y validar resultados.
- **Compatibilidad con múltiples plataformas:** Funciona en varias plataformas .NET, incluyendo .NET Framework, .NET Core y Xamarin, facilitando la integración en diversos entornos de desarrollo.
- **Generación de informes de pruebas:** Ofrece capacidades para generar informes detallados sobre los resultados de las pruebas, facilitando la identificación y corrección de errores.

NUnit fue utilizado principalmente en el proyecto para realizar las pruebas unitarias y de integración, ha sido esencial para garantizar que el código de MindCubeXR sea robusto y confiable mediante la realización de pruebas exhaustivas y automatizadas que validan el correcto funcionamiento de cada componente del software.

4.3 Unity Test Runner

`Unity Test Runner`[3] es una herramienta integrada en el entorno de desarrollo Unity que permite realizar pruebas automatizadas dentro del mismo entorno. Esta herramienta es esencial para el desarrollo de aplicaciones interactivas y juegos, ya que facilita la ejecución de pruebas tanto unitarias como de integración. Sus características clave incluyen [22]:

- **Pruebas en el Editor:** Permite ejecutar pruebas directamente en el Editor de Unity, facilitando la validación de la lógica del juego y las interacciones en tiempo real.

[2] https://nunit.org/
[3] https://docs.unity3d.com/Manual/testing-editortestsrunner.html

- **Compatibilidad con NUnit:** Integra NUnit para realizar pruebas unitarias y de integración, utilizando el mismo framework para una consistencia en el proceso de pruebas.
- **Informes y resultados:** Ofrece informes detallados de los resultados de las pruebas, permitiendo a los desarrolladores identificar y solucionar problemas de manera eficiente.
- **Facilidad de integración:** Se integra de manera fluida en el flujo de trabajo de desarrollo, permitiendo ejecutar pruebas automáticamente durante el proceso de desarrollo y en diferentes fases del ciclo de vida del software.

Unity Test Runner se ha empleado para validar y asegurar que los componentes del juego, la interfaz de usuario y que las interacciones dentro del entorno de Unity funcionen correctamente, contribuyendo a una experiencia de usuario más estable y confiable, mediante las pruebas unitarias y de integración.

4.4 Visual Studio 2022

`Visual Studio 2022`[4] es un entorno de desarrollo integrado (IDE) desarrollado por Microsoft, utilizado para desarrollar aplicaciones en múltiples lenguajes de programación. Algunas de las características por las que destaca son las siguientes [23]:

- **Desarrollo multiplataforma:** Admite el desarrollo para Windows, Android, iOS, macOS, web y servicios en la nube desde un solo entorno.
- **Soporte para múltiples lenguajes:** Compatible con C#, C++, Python, JavaScript, entre otros, lo que lo hace versátil para diferentes necesidades de desarrollo.
- **Interfaz de usuario más moderna:** Presenta un diseño actualizado que mejora la experiencia visual y la organización de herramientas.
- **Herramientas de depuración avanzadas:** Proporciona herramientas robustas para la depuración y diagnóstico de aplicaciones, ayudando a identificar y corregir errores rápidamente.
- **Integración con Git:** Incluye integración nativa con sistemas de control de versiones como Git, facilitando la colaboración y gestión de código en equipos de desarrollo.
- **Extensibilidad:** Permite la instalación de extensiones para ampliar las funcionalidades del IDE, como herramientas de análisis de código, generadores de código, y más.
- **Soporte para inteligencia artificial y aprendizaje automático:** Incluye herramientas y extensiones para el desarrollo de aplicaciones basadas en IA y machine learning.

Visual Studio 2022 fue el IDE principal utilizado para el desarrollo del backend y la lógica de la aplicación, proporcionando un entorno sólido y eficiente para la codificación, depuración y pruebas del software desarrollado en C#.

[4] https://visualstudio.microsoft.com/es/

4.5 Oculus Quest 2

`Oculus Quest 2`[5] es un dispositivo de realidad virtual autónomo desarrollado por Meta (anteriormente Facebook). Puede operar de manera independiente o conectarse a un PC mediante Meta Quest Link. Se podrían destacar entre las características más principales las siguientes [24]:

- **Pantalla de alta resolución:** 1832 x 1920 píxeles por ojo, proporcionando una experiencia visual detallada.
- **Procesador Snapdragon XR2:** Equipado con un Snapdragon XR2, diseñado específicamente para aplicaciones de realidad virtual y aumentada, lo que permite un rendimiento fluido y una experiencia inmersiva sin necesidad de conexión a un ordenador.
- **Seguimiento de manos:** Capacidad para realizar un seguimiento preciso de las manos sin necesidad de controladores adicionales.
- **Conectividad Inalámbrica:** Soporta WiFi 6 y Bluetooth 5.1, lo que facilita la conectividad y la transmisión de datos rápida y eficiente para experiencias sin cables.
- **Almacenamiento interno:** Disponible en versiones de 64GB y 256GB para almacenar aplicaciones y contenido.
- **Sonido 3D:** Integra altavoces con sonido espacial 3D, que brindan una experiencia inmersiva al posicionar el sonido de manera precisa en el entorno virtual.
- **Tracking:** Utiliza cámaras internas para el seguimiento de la posición, eliminando la necesidad de sensores externos, lo que simplifica su uso y configuración.
- **Diseño y Comodidad:** A pesar de su peso de 503 gramos, las Oculus Quest 2 están diseñadas para ser relativamente cómodas, con una correa ajustable y una cubierta de gomaespuma suave en la zona interior. Incluyen un adaptador para gafas, aunque la luz puede filtrarse ligeramente si se utiliza este accesorio.
- **Compatibilidad con PC:** A través de la función Meta Quest Link, las Oculus Quest 2 pueden conectarse a un PC para ejecutar juegos y aplicaciones más exigentes gráficamente, aprovechando el hardware del ordenador.

Las Oculus Quest 2 fueron utilizadas como el dispositivo principal para ejecutar y probar la aplicación XR desarrollada. Su capacidad para operar de manera autónoma permitió realizar pruebas en un entorno realista y simular el uso por parte de los pacientes sin necesidad de estar conectadas a un PC. Además, su conectividad con un PC mediante Meta Quest Link fue crucial para el desarrollo y pruebas de aplicaciones más complejas, asegurando que el software fuera compatible y rindiera adecuadamente tanto en modo autónomo como conectado.

4.6 VRTK (Virtual Reality Toolkit)

VRTK[6] es un conjunto de herramientas para Unity que facilita la creación de interacciones en realidad virtual, proporcionando componentes reutilizables y configu-

[5] https://www.meta.com/es/quest/products/quest-2/

[6] https://assetstore.unity.com/packages/tools/integration/vrtk-virtual-reality-toolkit-vr-toolkit-64131

raciones predefinidas. Algunas de las características por las que destaca son las que se presentan a continuación [25]:

- **Componentes predefinidos:** Incluye scripts y configuraciones definidas, listas para usar, facilitando la implementación de interacciones como teletransportación y manipulación de objetos.
- **Extensibilidad:** Permite a los desarrolladores personalizar y extender las funcionalidades según las necesidades del proyecto.
- **Soporte multiplataforma:** Compatible con una variedad de dispositivos VR.

VRTK fue utilizado para acelerar el desarrollo de interacciones dentro de la aplicación XR, permitiendo implementar funcionalidades avanzadas sin necesidad de desarrollarlas desde cero.

4.7 Git

`Git`[7] es un sistema de control de versiones distribuido ampliamente utilizado en el desarrollo de software. Fue creado por Linus Torvalds en 2005 para facilitar el desarrollo del kernel de Linux, y desde entonces se ha convertido en una herramienta fundamental en la industria del software. Git permite a los desarrolladores trabajar de manera colaborativa, mantener un historial detallado de los cambios en el código y gestionar múltiples versiones de un proyecto de manera eficiente. A continuación, se detallan las principales características que hacen de Git una herramienta esencial en el desarrollo de software:

- **Open Source:** Es un software de código abierto, lo que ha contribuido a su adopción masiva y a la creación de una comunidad activa y su mejora constante, debido a que el código fuente está disponible para que cualquiera lo revise, modifique y distribuya bajo la licencia GPL (General Public License).
- **Escalabilidad:** Es altamente escalable capaz de manejar proyectos de cualquier tamaño y con cualquier número de colaboradores sin sacrificar el rendimiento. Esto lo convierte en una opción ideal tanto para pequeños proyectos personales como para grandes desarrollos corporativos.
- **Distribuido:** A diferencia de los sistemas de control de versiones centralizados, donde todos los cambios se almacenan en un único servidor, Git es distribuido. Esto significa que cada desarrollador tiene una copia completa del repositorio, incluyendo todo el historial del proyecto. Esta característica permite trabajar sin necesidad de estar conectado a un servidor central, mejorando la flexibilidad y la velocidad del desarrollo.
- **Seguridad:** Git asegura la integridad de los datos mediante el uso de SHA1 (Secure Hash Algorithm 1) para identificar y gestionar los objetos dentro del repositorio. Cada commit (confirmación de cambios) y archivo es verificado y rastreado mediante su checksum, lo que garantiza que los datos no puedan ser alterados sin que Git lo detecte.
- **Velocidad:** La mayoría de las operaciones en Git se realizan de manera local, lo que reduce drásticamente los tiempos de espera en comparación con otros sistemas de control de versiones.

[7] https://git-scm.com/

- **Soporte para desarrollo No Lineal:** Git facilita el desarrollo no lineal a través de su potente sistema de ramas y fusiones. Los desarrolladores pueden crear, modificar y fusionar ramas de manera eficiente, lo que les permite trabajar en diferentes características o correcciones de errores en paralelo sin interferir con la rama principal del proyecto.
- **Manejo de Ramas:** Git permite la creación de múltiples ramas dentro de un proyecto sin afectar a las demás. Las ramas permiten a los desarrolladores trabajar en diferentes características o versiones de un proyecto de manera aislada, y luego fusionar los cambios cuando estén listos.
- **Área de preparación (Staging Area):** El área de preparación de Git actúa como un espacio intermedio donde los cambios pueden ser revisados y organizados antes de ser confirmados en el repositorio.
- **Historia Limpia y Rebase:** Git permite mantener un historial de commits limpio y lineal mediante la técnica de rebase, que reorganiza los commits de una rama sobre otra, integrando los cambios de manera secuencial. Esto resulta en un historial de desarrollo más comprensible y fácil de seguir.
- **Integridad de Datos:** Git asegura la integridad de todos los datos versionados utilizando algoritmos criptográficos. Cada commit está vinculado a su predecesor, formando una cadena inmutable de historial que garantiza que ningún cambio en el código pueda pasar desapercibido.
- **Recuperación de Errores:** Git facilita la recuperación de errores mediante comandos que permiten deshacer cambios a diferentes niveles, desde la eliminación de cambios recientes en el área de preparación hasta la reescritura de commits anteriores.
- **Colaboración y Trabajo en Equipo:** Permite a los desarrolladores colaborar en proyectos compartidos, facilitando la integración de los cambios de múltiples contribuyentes. Herramientas como GitHub y GitLab, que se basan en Git, potencian estas capacidades al ofrecer funcionalidades adicionales para la gestión de proyectos y el seguimiento de problemas.

Git se utilizó como la principal herramienta de control de versiones en el desarrollo del proyecto. Su capacidad para manejar múltiples ramas de desarrollo permitió una organización eficiente del trabajo, permitiendo a los diferentes miembros del equipo colaborar en la implementación de nuevas características y correcciones de errores sin interferir con la versión estable del proyecto. Además, su robusta integración con Visual Studio y otros entornos de desarrollo facilitó la gestión del código y el control de versiones de manera integrada.

4.8 Meta Quest Link

`Meta Quest Link`[8] es una funcionalidad que permite conectar el Oculus Quest 2 a un PC, convirtiendo el visor en un dispositivo VR de alta gama que aprovecha el computo de procesamiento del PC. Algunas de las características por las que se compone son las que se pueden observar a continuación [26]:

[8] https://www.meta.com/es-es/help/quest/articles/headsets-and-accessories/oculus-link/set-up-link/

- **Conexión con PC:** Permite utilizar la capacidad de procesamiento de la GPU para ejecutar VR más exigentes.
- **Experiencia de alta calidad:** Mejora la calidad gráfica y el rendimiento en comparación con el uso autónomo del Quest.
- **Compatibilidad con múltiples plataformas:** Funciona con cualquier aplicación compatible con Oculus Rift.

Meta Quest Link se utilizó para realizar pruebas de la aplicación XR en un entorno más exigente, aprovechando el hardware del PC.

4.9 Meta XR SDK

`Meta XR SDK`[9] es un kit de desarrollo de software que proporciona herramientas y APIs para la creación de aplicaciones de realidad extendida (XR) específicamente diseñadas para dispositivos Meta, como Oculus Quest. Las características que se pueden resaltar son las que se describen a continuación [27]:

- **APIS para VR y AR:** Proporciona acceso a funcionalidades avanzadas de VR y AR, incluyendo seguimiento de manos y controladores.
- **Integración con Unity:** Facilita la integración de estas funcionalidades dentro de proyectos de Unity.
- **Soporte para interacciones avanzadas:** Incluye soporte para la manipulación de objetos en 3D y el reconocimiento de gestos.

Meta XR SDK fue utilizado para implementar interacciones avanzadas y optimizar la experiencia del usuario en la aplicación XR desarrollada.

4.10 Librerías y Plugins

Las **librerías** y **plugins** han sido fundamentales para mejorar la funcionalidad y la experiencia de usuario en la aplicación desarrollada. Las **librerías**, permiten a los desarrolladores realizar tareas específicas en sus proyectos sin tener que escribir código desde cero. Suelen incluir funciones, clases y métodos que abordan problemas comunes o implementan características estándar de manera eficiente. Mientras que los **plugins**, destacan por ser módulos o extensiones que se integran en una plataforma o entorno de desarrollo (como Unity, en este caso) para añadir nuevas funcionalidades o mejoras existentes. Los plugins permiten a los desarrolladores expandir las capacidades del software base sin modificar su código fuente.

4.10.1. Oculus Integration SDK

`Oculus Integration SDK`[10] es un conjunto de herramientas y recursos proporcionados por Oculus para desarrollar aplicaciones de realidad virtual específicamente optimizadas para los dispositivos Oculus, como las Oculus Quest y Rift. Este SDK facilita la integración de funciones avanzadas de VR en Unity, lo que permite a los desarrolladores aprovechar al máximo las capacidades de hardware de Oculus. Algunas de las características que proporciona son las siguientes [28]:

[9] https://assetstore.unity.com/packages/tools/integration/meta-xr-all-in-one-sdk-269657

[10] https://developer.oculus.com/documentation/unity/unity-import/

- **Soporte de Controladores:** Proporciona soporte completo para los controladores de Oculus, incluyendo mapeo de botones, seguimiento de movimiento y feedback háptico.

- **Seguimiento Posicional y de Manos:** Incluye APIs para implementar seguimiento posicional preciso, así como soporte para el seguimiento de manos, permitiendo interacciones naturales en el entorno VR.

- **Optimización de Desempeño:** Herramientas para optimizar el rendimiento de las aplicaciones en dispositivos Oculus, asegurando que se mantengan a una alta tasa de cuadros por segundo (FPS) para evitar mareos y mejorar la inmersión.

- **Integración con Unity:** Se integra fácilmente con Unity, permitiendo a los desarrolladores usar scripts y prefabs preconstruidos para configurar rápidamente escenas VR.

En el proyecto desrrollado, el Oculus Integration SDK se utilizó para implementar las funcionalidades VR principales, como el seguimiento de movimiento, la interacción con objetos virtuales mediante controladores y el ajuste del entorno VR para optimizar la experiencia del usuario en las Oculus Quest 2.

4.10.2. TextMeshPro

`TextMeshPro`[11] es un sistema avanzado de representación de texto que proporciona a los desarrolladores de Unity un control más detallado sobre el texto renderizado en sus aplicaciones. TextMeshPro ofrece una mejor calidad de texto y más flexibilidad que el sistema de texto predeterminado de Unity. Las características principales que lo definen son las siguientes:

- **Renderizado de Alta Calidad:** Ofrece texto nítido y claro con soporte para fuentes personalizadas y símbolos complejos.

- **Efectos de Texto Avanzados:** Permite aplicar sombras, bordes, gradientes y otros efectos visuales avanzados al texto.

- **Control Preciso del Formato:** Proporciona un control detallado sobre la alineación, el espaciado entre caracteres, palabras y líneas, así como sobre el interletraje.

- **Soporte Multilenguaje:** Compatible con una amplia gama de idiomas y sistemas de escritura, lo que es fundamental para aplicaciones que requieren localización.

TextMeshPro se utilizó en el proyecto para mejorar la visualización del texto en las interfaces de usuario (UI) dentro del entorno VR. Esto incluyó la creación de menús, paneles informativos y otros elementos de texto que requerían claridad y efectos visuales adicionales para mejorar la experiencia de usuario.

[11] https://docs.unity3d.com/Packages/com.unity.textmeshpro@3.2/manual/index.html

4.10.3. Unity XR Interaction ToolKit

`Unity XR Interaction Toolkit`[12] es una herramienta de Unity que facilita la creación de experiencias de realidad virtual (VR) y realidad aumentada (AR) interactivas. Este toolkit proporciona componentes de interacción preconstruidos, como controladores de movimiento y sistemas de manipulación de objetos, que simplifican el desarrollo de experiencias inmersivas. Algunas de las características que han facilitado el desarrollo del proyecto son las siguientes:

- **Componentes de Interacción Predefinidos:** Incluye scripts y **prefabs**[13] para crear interacciones comunes en VR/AR, como agarre de objetos, teletransporte y selección de elementos.
- **Soporte Multidispositivo:** Es compatible con una amplia gama de dispositivos XR (eXtended Reality), incluyendo Oculus, HTC Vive y dispositivos AR móviles.
- **Sistema Modular:** Permite la personalización y extensión de las interacciones mediante un sistema modular, adaptándose a las necesidades específicas del proyecto.
- **Soporte para Interacciones Naturales:** Facilita la implementación de interacciones intuitivas como el agarre y manipulación de objetos usando las manos o controladores de movimiento.

El Unity XR Interaction Toolkit fue fundamental para implementar las interacciones de usuario dentro del entorno VR. Se utilizó para manejar la interacción de los usuarios junto a los objetos virtuales, los movimientos dentro del entorno y la realización de acciones específicas de manera intuitiva y natural.

[12] https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.0/manual/index.html

[13] Objeto predefinido que almacena un conjunto de componentes permitiendo reutilizarlo fácilmente en la escena sin necesidad de reconstruirlo desde cero.

CAPÍTULO 5
Interfaz Gráfica y Flujo de Interacción del Usuario

La interfaz gráfica y el flujo de interacción del usuario se caracterizan por ser componentes críticos en el diseño de MindCubeXR, ya que determinan la facilidad de uso y la eficiencia en la ejecución de las tareas por parte del usuario. Esta sección detalla cómo se ha enfocado la interfaz para proporcionar una experiencia intuitiva y cómo se gestionan las interacciones del usuario a lo largo de la aplicación.

5.1 Elementos de la Interfaz

La interfaz de usuario está diseñada para ser intuitiva y accesible, utilizando un espacio tridimensional en el que el usuario interactúa mediante controladores de realidad virtual. Los elementos principales de la interfaz incluyen una librería de puzzles, una pantalla de visualización del puzzle seleccionado y una mesa virtual en la que se generan los cubos y piezas magnéticas para resolver el puzzle.

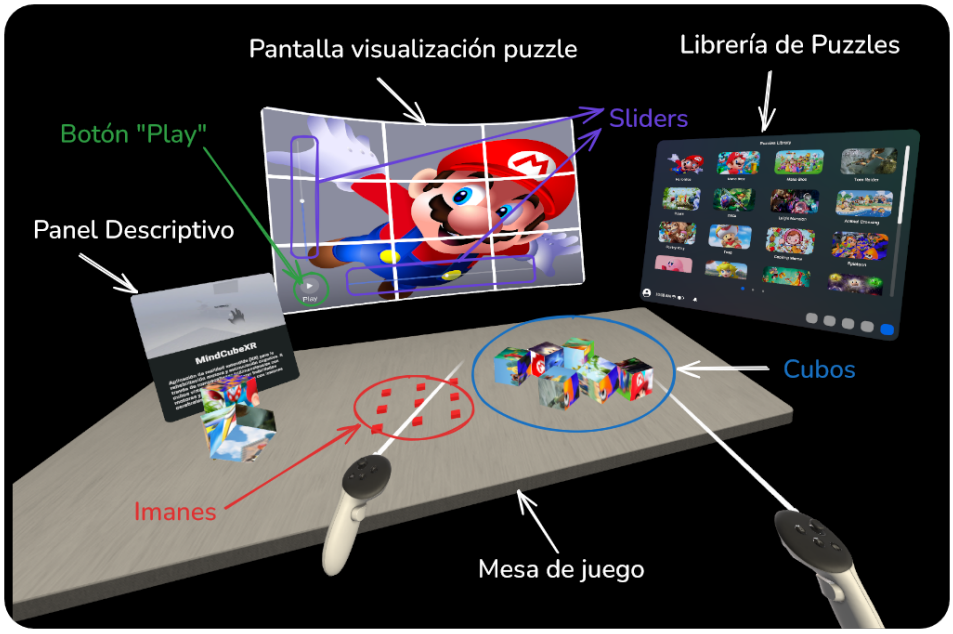

Figura 5.1: Escena MindCubeXR vista des de el simulador

Como se puede observar en la figura 5.1 así es como el usuario contempla des de el entorno de realidad virtual la escena, vista desde el simulador en este caso. Esta

imagen captura los principales elementos con los que interactúa el usuario y refleja la disposición de la interfaz en un entorno de realidad virtual.

5.1.1. Librería de Puzzles

Figura 5.2: Librería de Puzzles

La **librería de puzzles**, figura 5.2, está organizada en una cuadrícula de miniaturas, cada una representando un puzzle diferente. En la imagen se observan 16 miniaturas distribuidas en cuatro filas y cuatro columnas, cada una mostrando una imagen representativa del puzzle asociado. Los títulos de los puzzles están colocados debajo de cada miniatura, lo que facilita al usuario identificar rápidamente su opción preferida.

En este caso, cada miniatura corresponde a un título de un videojuego popular, lo que sugiere que los puzzles están tematizados entorno a personajes y escenarios reconocidos de videojuegos, como "Mario Bros", "Zelda", "Donkey Kong", y "Animal Crossing", entre otros. Esta organización permite que los usuarios encuentren fácilmente un puzzle que les interese.

En la parte inferior de la librería, se encuentra una barra de navegación que indica la página actual de la librería y permite al usuario desplazarse entre diferentes páginas de puzzles si existen más opciones disponibles, de manera horizontal y vertical.

El usuario interactúa con esta librería utilizando los controladores VR, desplazando su mano para resaltar y seleccionar un puzzle. Una vez seleccionado, **el puzzle elegido se carga en la pantalla central para su visualización** y configuración, lo que representa el siguiente paso en el flujo de interacción.

5.1.2. Pantalla de Visualización de Puzzle

Una vez que el usuario selecciona un puzzle desde la librería, éste se muestra en la **pantalla de visualización central**, que ocupa una posición destacada en el entorno virtual de MindCubeXR. Esta pantalla es crucial, ya que permite al usuario revisar y configurar el puzzle antes de comenzar a resolverlo.

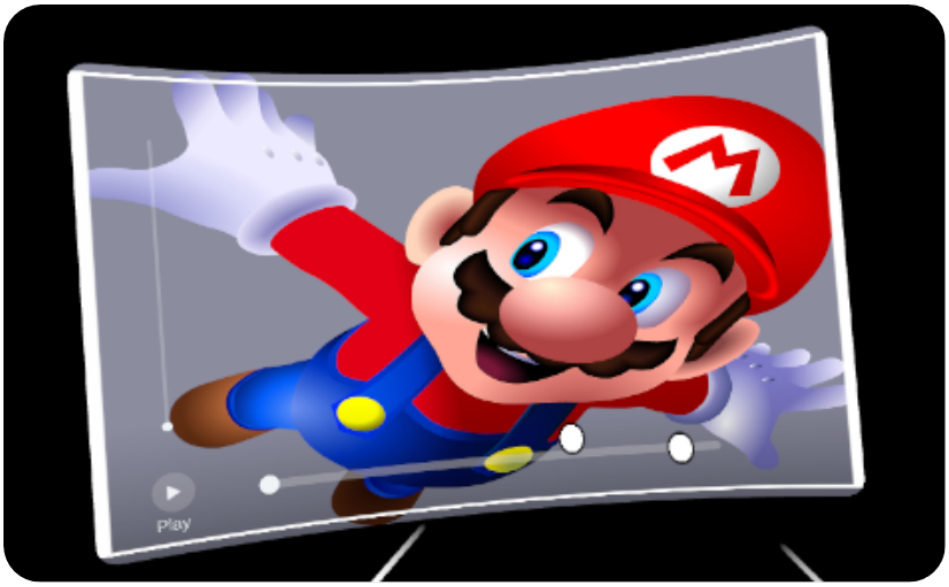

Figura 5.3: Pantalla de Visualización de Puzzle

La pantalla de visualización (Figura 5.3) **presenta la imagen del puzzle** seleccionado dividida en una cuadrícula que corresponde a la configuración inicial de filas y columnas. Esta cuadrícula es ajustable y permite al usuario personalizar la dificultad del puzzle.

- **Sliders de Configuración:** Debajo de la cuadrícula, se encuentran dos sliders que el usuario puede manipular con los controladores VR:
  - **Slider Vertical:** Ubicado a la izquierda de la pantalla, este slider permite al usuario seleccionar el número de filas en la cuadrícula del puzzle. Al mover el slider hacia arriba o hacia abajo, se aumenta o disminuye el número de filas, ajustando automáticamente como se dividirá la imagen del puzzle.
  - **Slider Horizontal:** Situado en la parte inferior de la pantalla, este slider permite ajustar el número de columnas en la cuadrícula. Al desplazar este slider hacia la derecha o izquierda, el usuario puede modificar la cantidad de columnas, lo que también cambia la cantidad de cubos que se generarán para completar el puzzle.
- **Botón "Play":** Una vez que el usuario ha configurado el número de filas y columnas deseadas, puede proceder a iniciar el puzzle presionando el botón "Play". Este botón está claramente visible en la pantalla y es accesible desde cualquier punto de la interfaz, asegurando que el usuario pueda comenzar a resolver el puzzle tan pronto como esté listo. Al pulsar "Play", los cubos correspondientes al número de filas y columnas seleccionadas se generan automáticamente sobre la mesa virtual, listos para ser ensamblados por el usuario.

La pantalla de visualización es, por lo tanto, un elemento interactivo que no solo muestra la imagen del puzzle, sino que también permite al usuario definir la complejidad del mismo antes de comenzar a resolverlo. Esto le proporciona al usuario un control directo sobre su experiencia, permitiendo que cada puzzle sea adaptado a su nivel de habilidad o preferencia.

5.1.3. Mesa Virtual

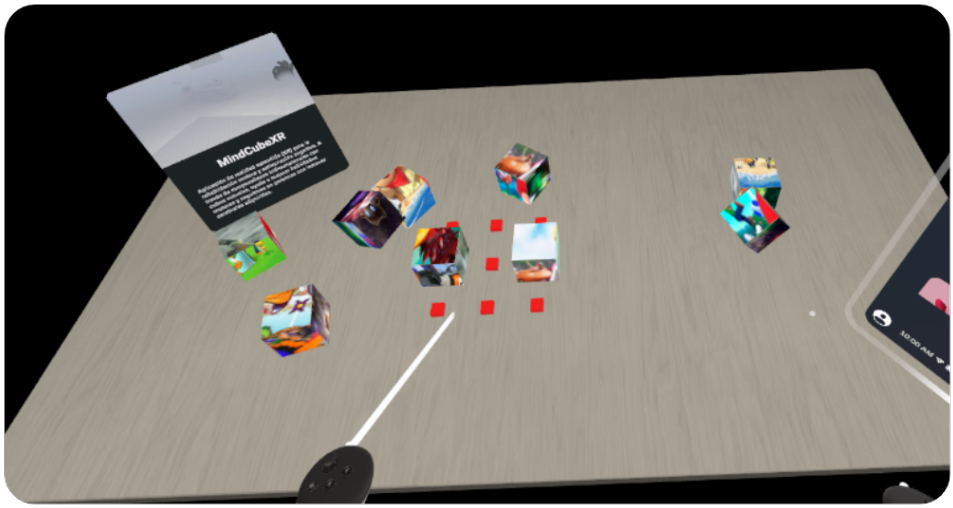

Figura 5.4: Mesa Virtual

La mesa virtual (Figura 5.4) es un componente clave en el entorno de MindCubeXR, donde el usuario interactúa directamente con las piezas del puzzle una vez configurado y seleccionado. Esta mesa se sitúa frente al usuario, en la parte inferior de la escena, proporcionando un espacio de trabajo virtual donde se materializan los elementos del puzzle.

- **Piezas del Puzzle:** Después de que el usuario haya configurado el número de filas y columnas mediante los "**Sliders**" en la pantalla de visualización y presionado el **botón** "**Play**", las piezas del puzzle se generan automáticamente sobre la mesa virtual. Estas piezas se representan como cubos tridimensionales, cada uno de los cuales corresponde a una sección de la imagen del puzzle seleccionada.

  - **Relación con Filas y Columnas:** El número de cubos generados en la mesa depende directamente de la configuración de filas y columnas que el usuario haya establecido. Por ejemplo, si el usuario selecciona una cuadrícula de 3x3, se generarán 9 cubos. Cada cubo lleva una parte de la imagen del puzzle y el objetivo del usuario es ensamblarlos correctamente para recrear la imagen original en 3D.

- **Imanes Virtuales:** Junto a los cubos, la mesa también presenta imanes virtuales (se encuentran representados como cuadrados más pequeños en color rojo) que el usuario puede usar para unir las piezas del puzzle. Estos imanes actúan como conectores entre los cubos, facilitando el ensamblaje correcto de las piezas. Los imanes aseguran que cuando el usuario acerca dos cubos que deberían estar juntos, estos se atraen y se conectan automáticamente, mejorando la fluidez y precisión en la interacción.

5.1.4. Panel Descriptivo de la Aplicación

Figura 5.5: Panel Descriptivo de la Aplicación

A la izquierda de la mesa virtual, se encuentra un **panel descriptivo** (Figura 5.5) de la aplicación MindCubeXR. Este panel proporciona al usuario información relevante sobre la aplicación y el puzzle seleccionado.

Este panel proporciona información general sobre la aplicación MindCubeXR, explicando sus características principales o brindando instrucciones sobre como interactuar con el entorno virtual. Esto es especialmente útil para usuarios nuevos o para aquellos que necesitan un recordatorio sobre las funcionalidades de la aplicación.

5.1.5. Paneles de Retroalimentación: Éxito y No éxito

Después de que el usuario haya ensamblado todas las piezas del puzzle en la mesa virtual, **la aplicación verifica automáticamente** si las piezas están colocadas correctamente según la configuración inicial de filas y columnas. Dependiendo del resultado, se muestra uno de los siguientes paneles de retroalimentación, diseñados para guiar al usuario en el siguiente paso:

- **Panel de Éxito ("Success"):**

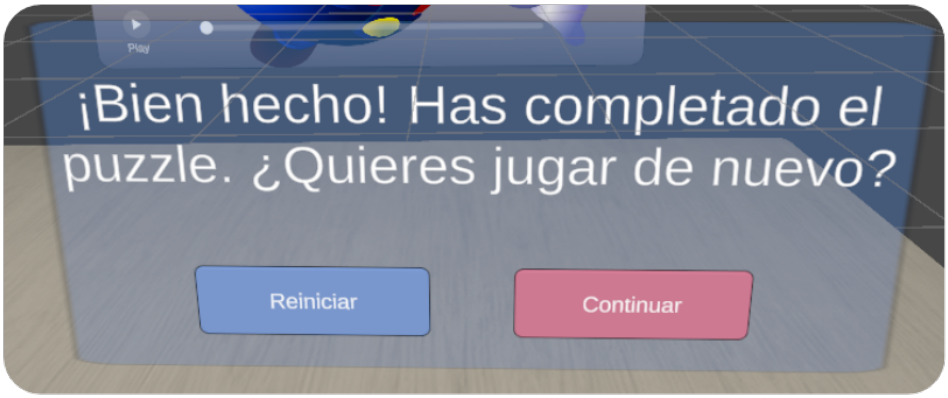

Figura 5.6: Panel Puzzle Completado

Si el puzzle ha sido completado correctamente (Figura 5.6), se despliega un panel de éxito en el centro del entorno virtual. Este panel es de color azul y contiene el siguiente mensaje: "¡Bien hecho! Has completado el puzzle. ¿Quieres jugar de nuevo?". El mensaje aparece en un formato claro y positivo, celebrando el logro del usuario.

- **Botones de Interacción:**
  - **Continuar:** Permite al usuario proceder con otro puzzle o seguir explorando la aplicación.
  - **Reiniciar:** Ofrece la opción de repetir el mismo puzzle desde el principio, permitiendo al usuario mejorar su tiempo o simplemente disfrutar de la experiencia nuevamente.

- **Panel de No éxito ("Warning"):**

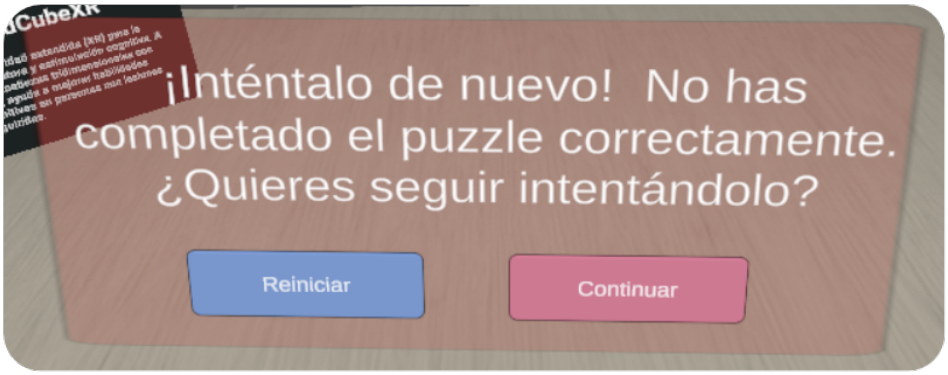

Figura 5.7: Panel Puzzle Erróneo

En caso de que el usuario no haya colocado correctamente las piezas, se muestra un panel de sin éxito de color rojo (Figura 5.7). Este panel presenta el mensaje: "¡Inténtalo de nuevo! No has completado el puzzle correctamente. ¿Quieres seguir intentándolo?". El tono del mensaje es alentador, incentivando al usuario a no rendirse y a intentar nuevamente hasta lograr el objetivo.

- **Botones de Interacción:**
  - **Continuar:** Permite al usuario continuar intentando resolver el puzzle desde el punto en que se encuentra, manteniendo las piezas en su posición actual.

- **Reiniciar:** Ofrece la opción de reiniciar el puzzle, devolviendo todas las piezas a su estado original para un nuevo intento.

Estos paneles emergentes están diseñados para proporcionar una retroalimentación inmediata y clara al usuario, ayudando a mantener la motivación y guiando sus siguientes pasos dentro de la aplicación. El uso de colores diferenciados (azul para éxito y rojo para sin éxito.) asegura que el usuario pueda interpretar rápidamente el resultado y tomar una decisión informada sobre como proceder.

5.2 Descripción del Flujo de Interacción del Usuario

El **flujo de interacción del usuario** dentro de la aplicación de realidad virtual (VR), tiene como objetivo guiar al usuario a través del proceso de selección, personalización y resolución de puzzles en un entorno inmersivo. A continuación, se detalla como se estructura dicha interacción, basándose en los elementos visuales y funcionales que componen la interfaz, así como las decisiones que el usuario puede tomar en cada etapa del proceso. Esta explicación permite entender como se logra una **experiencia intuitiva y eficiente**, proporcionando un equilibrio entre la personalización del reto (ajuste de dificultad) y la retroalimentación tras cada intento, ya sea exitoso o no.

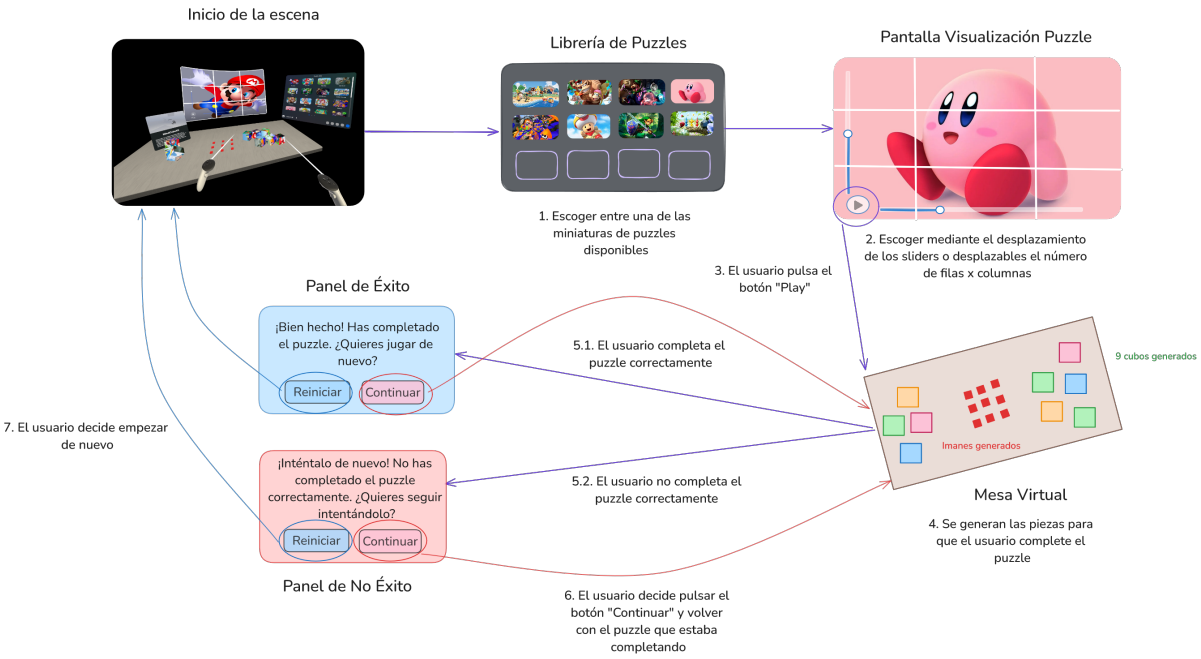

Figura 5.8: Diagrama de Flujo que muestra la interacción del usuario con la aplicación

En la interacción que se representa en la figura 5.8, el usuario comienza su experiencia en la aplicación de realidad virtual (VR) frente a una **mesa virtual**, una **pantalla central** donde se visualiza una imagen y diferentes **paneles** que actúa como el espacio inicial de la escena. Aquí, el usuario puede observar una serie de **miniaturas de puzzles** dispuestas en la "**Librería de Puzzles**", ubicada en la parte superior derecha de la interfaz. Estas miniaturas le permiten elegir entre las diversas opciones de puzzles disponibles para comenzar. El flujo de interacción comienza cuando el usuario selecciona uno de los puzzles, lo cual constituye el primer paso en el proceso, iniciando así la interacción con la interfaz.

Una vez seleccionado el puzzle, el usuario desvía su atención a la "**Pantalla de Visualización del Puzzle**", ubicada en la parte superior derecha de la imagen. En esta pantalla, tiene la posibilidad de ajustar el nivel de dificultad mediante el uso de **deslizadores o sliders**, que le permiten controlar la cantidad de filas y columnas que componen el puzzle. Este ajuste es un paso crucial, ya que permite personalizar la experiencia según las preferencias del usuario, aumentando o reduciendo el número de piezas. Este proceso es importante, ya que influye directamente en la complejidad del puzzle y, por lo tanto, en la duración e implicación del usuario.

Posteriormente, el usuario pulsa el **botón** "**Play**", que se encuentra en la misma Pantalla de Visualización, lo que da lugar a la **generación automática de piezas** del puzzle en la "**Mesa Virtual**" (situada en la parte central de la escena). En este punto, se crean **9 cubos** y una **9 imanes** que ayudarán al usuario a ensamblar correctamente las piezas del puzzle. La interacción con estas piezas se realiza mediante controles VR, mediante las manos, permitiendo al usuario manipularlas en tiempo real y empezar el proceso de resolución.

El flujo de la interacción culmina con dos posibles desenlaces. Si el usuario logra completar el puzzle correctamente, se le presenta un "**Panel de Éxito**", que lo felicita por haber resuelto el puzzle y le ofrece dos opciones: "**Reiniciar**" el puzzle para jugar de nuevo o "Continuar" con la sesión en la que se encuentra. Esta pantalla tiene como objetivo proporcionar una retroalimentación positiva y continuar la experiencia del usuario de manera fluida. Por otro lado, si el usuario no logra completar el puzzle de forma correcta, se muestra un "**Panel de No Éxito**", que le anima a intentarlo de nuevo o continuar con el que estaba realizando. Esto permite al usuario seguir participando en la actividad sin sensación de fracaso, ya que siempre tiene la opción de intentarlo nuevamente o avanzar a un nuevo reto.

Finalmente, dependiendo de la decisión del usuario, ya sea reiniciar o continuar, el flujo regresa a la **escena inicial** para permitirle seleccionar otro puzzle o modificar el anterior. De este modo, se cierra el ciclo de interacción y se asegura una **experiencia de usuario fluida y coherente**. Este flujo permite una interacción dinámica y progresiva, donde el usuario mantiene el control sobre sus decisiones y el ritmo de su experiencia en el entorno virtual.

CAPÍTULO 6
Solución propuesta: MindCubeXR

El **proceso de desarrollo** ha seguido una metodología estructurada que abarca múltiples fases de implementación, desde la conceptualización del diseño hasta la creación de un producto final interactivo y funcional. Este proceso no solo incluye la construcción de la lógica de juego, sino también la organización minuciosa de los recursos y la arquitectura empleada para garantizar la escalabilidad y el mantenimiento del proyecto.

En este capítulo, se describen en detalle las decisiones técnicas adoptadas durante la implementación del sistema, destacando los elementos clave que conforman la estructura del proyecto, la jerarquía de las escenas, y los patrones arquitectónicos empleados, como la **Programación Orientada a Componentes (COP)** y el **Modelo Vista Controlador (MVC)**, que permitieron estructurar de manera eficiente la interacción entre los diferentes componentes. Además, se detallará la implementación de la lógica del juego, mediante los diferentes **scripts** empleados para hacer funcionar correctamente la propia aplicación.

6.1 Estructura del Proyecto

El proyecto **MindCubeXR** se ha desarrollado empleando el motor de juego de Unity, lo que ha requerido una organización precisa tanto de los archivos como de los recursos utilizados. La estructura del proyecto que se ha seguido tiene el objetivo de facilitar el mantenimiento, la escalabilidad y la colaboración entre desarrolladores, siguiendo las mejores prácticas recomendadas para proyectos de realidad virtual (VR).

6.1.1. Organización de Directorios

La organización de los directorios dentro del proyecto sigue las mejores prácticas recomendadas para el desarrollo en Unity, especialmente para proyectos de realidad virtual (VR), donde la correcta gestión de recursos y scripts es esencial. El directorio principal "**Assets/**" (Figura 6.1) contiene todos los recursos empleados en el proyecto, incluyendo scripts, prefabs, texturas, materiales, tests, etc.

- **Oculus/**: Contiene los componentes relacionados con la integración de Oculus.
- **Plugins/**: Incluye los plugins necesarios para la extensión de funcionalidades.
- **Resources/**: Almacena recursos como imágenes, materiales y prefabs.
- **Samples/**: Contiene muestras y ejemplos que han servido de referencia y guía, para la implementación de diversas funcionalidades.

- **Scenes/**: Contiene las escenas principales del proyecto, es decir, los entornos donde se desarrollan las interacciones y eventos.
- **Scripts/**: En esta carpeta se almacenan todos los scripts escritos en C# que componen la lógica de juego, interacción y comportamiento del proyecto.
- **Tests/**: Este directorio almacena los scripts de pruebas escritos en C#, tanto de integración como de usabilidad. Estas pruebas permiten validar que las diferentes funcionalidades del proyecto se ejecuten correctamente y que la experiencia final cumpla con los requisitos establecidos.
- **TextMesh Pro/**: Contiene la integración con el sistema TextMesh Pro, una herramienta avanzada para la manipulación y presentación de texto dentro de Unity.
- **XR/**: Este directorio incluye componentes específicos para la realidad extendida (XR), englobando tanto realidad virtual como aumentada.
- **XRI/**: Contiene extensiones y utilidades especializadas para la interacción en XR.

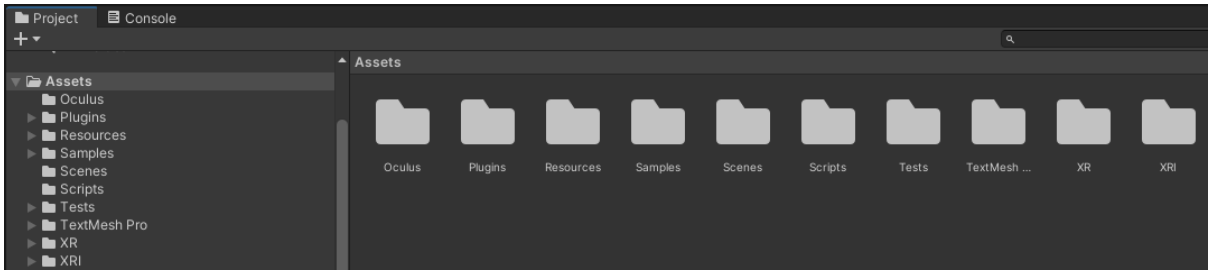

Figura 6.1: Estructura de directorios del proyecto MindCubeXR en Unity

6.1.2. Jerarquía de la Escena Principal

La jerarquía de la escena principal en Unity refleja la organización de los diferentes elementos y su interacción dentro del proyecto. Los objetos de juego, también conocidos como "**GameObjects**", como los cubos, los imanes, las zonas de interacción, etc, están organizados en una estructura que facilita su gestión y manipulación. (Figura 6.2).

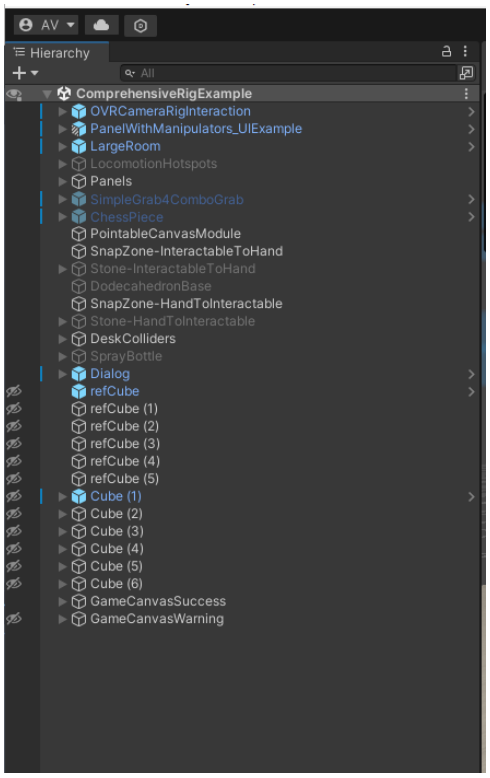

Figura 6.2: Jerarquía del proyecto MindCubeXR en Unity

A continuación, se describen algunos de los elementos clave que destacan dentro de esta estructura jerárquica:

- **OVRCameraRigInteraction:** Este GameObject contiene la configuración de la cámara y los controladores para la interacción en realidad virtual. Es esencial para gestionar la visión y el movimiento del usuario en el entorno de VR,
- **UI Cylinder:** UI Cylinder se encarga de agrupar todos los elementos de la interfaz de usuario que el jugador puede interactuar dentro del juego. Aquí se encuentran los paneles, botones y otros elementos que permiten al usuario controlar el juego.
- **refCube:** Estos objetos representan los imanes en el juego, que son puntos de acoplamiento para los cubos. Cada "**refCube**" es una instancia donde el cubo (descrito a continuación, denominado como "**Cube**") puede ser colocado por el usuario.
- **Cube:** Este es el cubo principal que el usuario debe manipular para completar los puzzles. Cada objeto "**Cube**" en la jerarquía corresponde a un cubo que el jugador puede mover y acoplar a los imanes ("**refCube**").

- **GameCanvasSuccess y GameCanvasWarning:** Estos GameObjects son utilizados para mostrar mensajes al usuario. "**GameCanvasSuccess**" se activa cuando el usuario completa correctamente un puzzle, mientras que "**GameCanvasWarning**" se muestra si el usuario comete un error.

- **Dialog:** Este GameObject contiene los elementos de diálogo y las interacciones que se presentan al usuario durante el juego. Incluye componentes como "Text" y "Image" que son utilizados para mostrar información al jugador.

- **Panels:** Dentro de este grupo se encuentran los paneles que contienen las diversas opciones y controles que el usuario puede manipular durante el juego, como el botón de reproducción/pausa ("**PlayPauseButton**") y el panel de control de la cuadrícula ("**GridControls**").

6.1.3. Arquitectura Empleada

Programación Orientada a Componentes (COP)

Como se ha mencionado anteriormente en el desarrollo, se ha empleado la **Programación Orientada a Componentes (COP)**, un enfoque ampliamente utilizado en el desarrollo de juegos de Unity. Este enfoque se basa en la organización de la funcionalidad del juego en componentes independientes y reutilizables que pueden ser asignados a diferentes objetos del juego, conocidos como **GameObjects**.

Cada uno de estos componentes, derivados de la clase base **MonoBehaviour**, maneja una parte específica de la funcionalidad del juego, como la física, el comportamiento o la interacción con el usuario. Esta modularidad permite mezclar y combinar componentes según sea necesario, creando un sistema flexible y escalable. En este enfoque, cada objeto en el juego (representado como un GameObject) puede estar compuesto de varios componentes (scripts) que definen su comportamiento. Por ejemplo, un cubo en MindCubeXR puede tener componentes que controlan su física, su interacción con el usuario y su apariencia visual.

```
1 public class GameGenerator : MonoBehaviour
2 {
3     public GameObject cubePrefab;
4     public GameObject magnetPrefab;
5    //Resto de Código
6 }
```

Listing 6.1: Inicialización clase GameGenerator.cs

El anterior fragmento de código 6.1 muestra como se definen los objetos en Unity usando componentes. Aquí, "**GameGenerator**" es un componente que se encarga de generar los cubos y los imanes necesarios para el puzzle. Estos componentes se asocian a un GameObject, que podría ser, por ejemplo, el objeto que representa la escena de juego.

Además, la clase "**GameGenerator**" ofrece una amplia gama de métodos proporcionados por Unity para interactuar con el motor de juego y controlar el comportamiento de los objetos en la escena. Algunos de los métodos más relevantes son:

- "**Start()**": Este método 6.2 se ejecuta antes de la primera actualización del frame. Especialmente se ha empleado en el script de generacion de las piezas de puzzle para limpiar los cubos e imanes existentes, así como para inicializar los

paneles del juego a "false", evitando que aparezcan de nuevo en la escena al pulsar el botón "**Play**" con una nueva combinación de filas y columnas.

```
void Start()
{
    warningPanel.SetActive(false);
    successPanel.SetActive(false);

    restartButtonSuccess.onClick.AddListener(RestartGame);
    restartButtonWarning.onClick.AddListener(RestartGame);

    continueButtonSuccess.onClick.AddListener(CloseMessagePanel);
    continueButtonWarning.onClick.AddListener(CloseMessagePanel);

    ClearCurrentMagnets();
    ClearCurrentCubes();
}
```

Listing 6.2: Método Start() clase GameGenerator.cs

- "**Update()**": Este método 6.3 se llama una vez por frame y se utiliza para actualizar la lógica del juego de manera continua. En el caso de MindCubeXR, se emplea para gestionar la aparición de los diferentes mensajes en los que se indica al usuario si ha completado el puzzle correctamente o incorrectamente.

```
void Update()
{
    if (successPanel.activeSelf)
    {
        PositionPanel(successPanel);
    }

    if (warningPanel.activeSelf)
    {
        PositionPanel(warningPanel);
    }
}
```

Listing 6.3: Método Update() clase GameGenerator.cs

Modelo Vista Controlador (MVC)

Además del enfoque COP, MindCubeXR sigue el patrón arquitectónico **Modelo Vista Controlador (MVC)** para organizar la interacción entre los datos, la lógica del juego y la interfaz de usuario. Este enfoque permite mantener una clara separación de responsabilidades, facilitando el mantenimiento y la escalabilidad del código.

En primer lugar el **Modelo (Model)**, es la parte del sistema encargada de representar los datos y la lógica del juego. En este proyecto, este rol es desempeñado por los scripts como "GameGenerator.cs", "CubeInteraction.cs", "CollisionDetector.cs", etc. La responsabilidad de los scripts reside en manejar las reglas del juego, crear los elementos del puzzle y mantener el estado del juego. El script "**GameGenerator.cs**" se ocupa por ejemplo de generar los cubos y los imanes, que son los elementos esenciales del puzzle. El "**CubeInteraction.cs**", por otro lado, gestiona cómo interactúan los cubos generados con los imanes,determinando si están colocados correctamente. De esta forma, el "Modelo" asegura que toda la lógica del juego se mantenga coherente y actualizada, sin preocuparse por cómo se presentan estos datos al usuario, tal como se observa en la figura 6.3.

Por otra parte, la **Vista (View)** es la representación visual de los datos gestionados en la pantalla por el Modelo. En MindCubeXR, la Vista incluye todos los elementos visuales del puzzle y la interfaz de usuario con los que interactúa el usuario final, como los cubos del puzzle y los paneles de control. Componentes como "**ImagePanelController**" son fundamentales en esta capa, ya que determinan como se muestran las imágenes y otros elementos visuales, asegurando que el usuario tenga una experiencia visual clara y atractiva. La figura 6.3 ilustra como la Vista se encuentra directamente conectada al Controlador, recibiendo las actualizaciones necesarias para reflejar el estado actual del juego.

Por otro lado, el **Controlador (Controller)** actúa como intermediario entre el Modelo y la Vista. Por ejemplo, scripts como "**CubeInteraction**" funcionan como Controladores, procesando las entradas del usuario, como la colocación de un cubo sobre un imán, y actualizando tanto el Modelo como la Vista en consecuencia. Esto significa que el Controlador interpreta las acciones del usuario y las traduce en cambios dentro del juego, asegurando que la Vista se actualice para reflejar estos cambios. En el esquema 6.3, se puede observar como el Controlador está en el centro de la interacción, conectando de manera efectiva las otras dos capas.

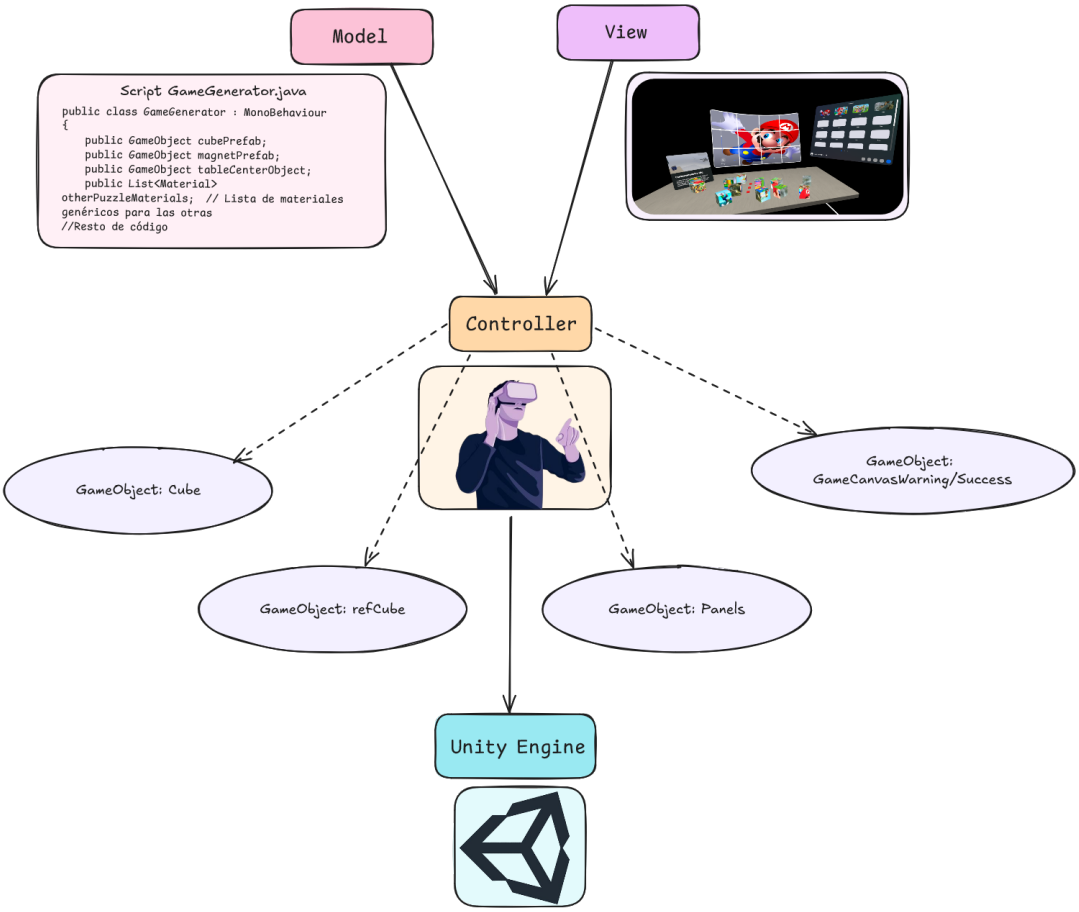

Figura 6.3: Esquema de la Arquitectura MVC empleada en MindCubeXR

El esquema anterior (Figura 6.3), representa la interacción y conexión entre las tres capas del patrón MVC en MindCubeXR. El **Controlador** recibe las entradas del usuario, actualiza el **Modelo** para reflejar los cambios en los datos y, finalmente, actualiza la **Vista** para que el usuario pueda ver estos cambios en tiempo real. Este flujo continuo asegura que cada componente del sistema tenga una función específica, lo que no solo mejora la eficiencia del desarrollo, sino que también permite una mayor flexibilidad y modularidad en el código.

6.2 Implementación de la Lógica

La implementación de la lógica de la aplicación se centra en garantizar que las mecánicas del puzzle funcionen de manera precisa y coherente dentro del entorno de realidad virtual. Para lograrlo, se han empleado componentes clave que se encuentran incluidos en la propia estructura del cubo virtual y el sistema de imanes.

6.2.1. Modelo de cubos virtuales

El modelo de cubos virtuales en MindCubeXR es uno de los elementos centrales que define la experiencia del usuario, ya que no solo representa las piezas que componen el puzzle, sino que también integra una serie de comportamientos y características físicas esenciales para la interacción en un entorno de realidad virtual.

Cada cubo en el juego es una entidad compleja que combina múltiples componentes, cada uno con una función específica. Uno de los primeros componentes clave es

el **Collider**, que establece la forma física del cubo. Este componente es fundamental porque define como el cubo interactúa con otros objetos dentro del juego, permitiendo detectar colisiones y asegurando que los cubos se comporten de manera coherente cuando son movidos o manipulados por el usuario.

En conjunto con el Collider, el **Rigidbody** desempeña un papel crucial al aplicar las leyes de la física al cubo. Este componente controla aspectos como la masa, la gravedad y la inercia, lo que garantiza que los cubos respondan de manera realista a las fuerzas aplicadas en el entorno virtual. Por ejemplo, cuando un usuario coge o suelta un cubo, el Rigidbody se encarga de que el cubo reaccione correctamente, cayendo al suelo o manteniéndose suspendido según corresponda.

La representación visual de los cubos es gestionada por el **Mesh Renderer**, que aplica las texturas y materiales a la geometría del cubo. En MindCubeXR, esto es especialmente importante ya que cada cubo muestra una parte de la imagen que forma el puzzle completo. Así, la apariencia de los cubos es fundamental para guiar al usuario en la tarea de ensamblar la imagen correctamente.

Para manejar la interactividad del cubo, se incluyen componentes como el **Snap Interactor** y el **Grabbable**. El Snap Interactor permite que el cubo se acople automáticamente a puntos específicos en el tablero de juego, como los imanes, cuando se coloca en la posición correcta. Esto no solo facilita la jugabilidad, sino que también proporciona una respuesta inmediata al usuario, confirmando que el cubo ha sido colocado correctamente. Por su parte, el Grabbable habilita la capacidad de agarrar y manipular los cubos dentro del entorno de realidad virtual, permitiendo al usuario interactuar de manera intuitiva y directa con los elementos del juego.

Finalmente, el `Material`[1] aplicado a cada cubo define las texturas y colores de sus caras, lo que es vital para la jugabilidad de MindCubeXR. Cada material corresponde a una sección específica de la imagen del puzzle, proporcionando las pistas visuales necesarias para que el usuario pueda completar el desafío.

En conjunto, estos componentes crean una experiencia de juego inmersiva y dinámica, donde cada cubo no solo es una pieza del puzzle, sino una entidad interactiva completa, con propiedades físicas, visuales y de interacción cuidadosamente diseñadas. La figura 6.4 ilustra como estos componentes se integran para formar el modelo de cubos virtuales en MindCubeXR, destacando su papel fundamental en la lógica del juego y la interacción del usuario.

[1] **Material:** recurso que define cómo se visualiza un objeto, y en este caso, cada material corresponde a una parte específica de la imagen del puzzle.

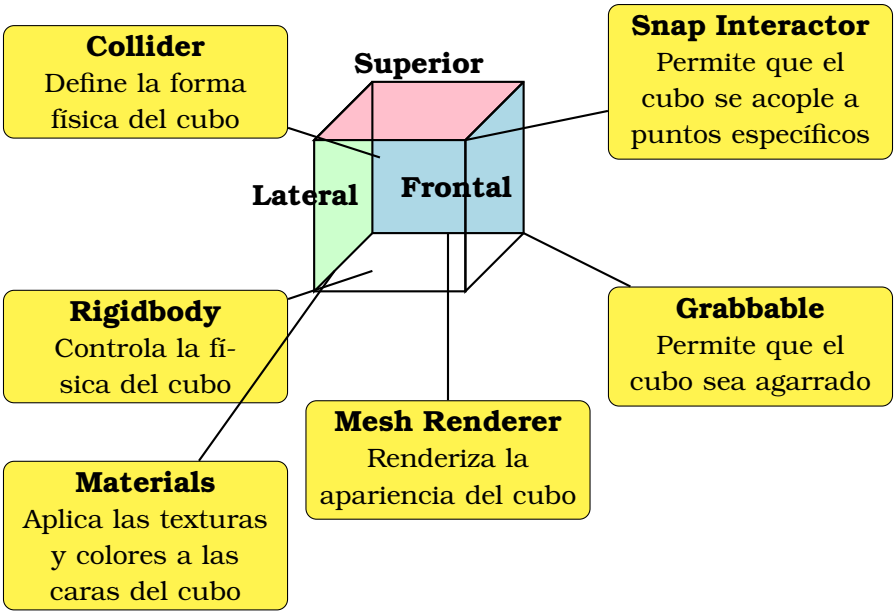

Figura 6.4: Esquema del Cubo con Componentes en MindCubeXR

6.2.2. Estructura General de los Scripts

MindCubeXR se basa en una arquitectura modular, donde cada funcionalidad específica está encapsulada en un script independiente, lo que facilita la mantenibilidad y escalabilidad del código. Estos scripts se adhieren al enfoque de Programación Orientada a Componentes (COP) descrito anteriormente, lo que permite una reutilización eficiente y la fácil integración de nuevas características.

Los scripts principales que se emplean para controlar la lógica del juego son:

1. **GameGenerator:** Encargado de la generación de los cubos y los imanes, así como de la inicialización del juego.
2. **CubeInteraction:** Gestiona la interacción entre los cubos y los imanes, incluyendo la detección de colisiones y la validación de la posición y orientación correctas.
3. **CollisionDetector:** Supervisa las colisiones entre los cubos y los imanes, activando la lógica necesaria cuando se produce una interacción.
4. **ImagePanelController:** Gestiona la selección de imágenes para los puzzles, permitiendo al usuario personalizar la apariencia del puzzle.

Como se ha podido ver cada uno de los scripts tiene una funcionalidad y propósito específico, pero para entender mejor como se ha llevado a cabo la lógica de cada uno es necesario dar una explicación en más profundidad de cada uno de ellos poniendo especial atención en los fragmentos de código más relevantes:

Generación y Gestión de Cubos: "GameGenerator.cs"

El primero de ellos, el script "**GameGenerator**" (anexo A.1 sección A.1.1), desempeña un papel fundamental en el desarrollo de MindCubeXR, actuando como el núcleo encargado de la creación, configuración y gestión de los cubos y los imanes que constituyen el puzzle en el entorno de realidad virtual. Este script se caracteriza

por su alta configurabilidad, lo que permite adaptar dinámicamente el juego a diferentes escenarios y configuraciones. A continuación, se detallan sus componentes más relevantes:

```
public GameObject cubePrefab;
public GameObject magnetPrefab;
public GameObject tableCenterObject;
public List<Material> otherPuzzleMaterials;
public Image selectedImage;
public Transform imagesPanel;

//Resto de Código

 public GameObject warningPanel;
 public GameObject successPanel;
 public TextMeshProUGUI successMessageText; // Para el panel de éxito
 public TextMeshProUGUI warningMessageText; // Para el panel de advertencia
 public Button playButton; // Botón de Play
 public Button continueButtonSuccess;
```

Listing 6.4: Definición de Variables y Componentes Clave del script GameGenerator.cs

Como se puede observar en el fragmento de código 6.4 las variables incluyen referencias a "**`Prefabs`**"[2] de cubos e imanes ("**cubePrefab**", "**magnetPrefab**"), una lista de materiales que se aplican a las caras de los cubos para representar las distintas partes del puzzle ("**otherPuzzleMaterials**"), así como también, elementos de la propia interfaz de usuario como paneles de advertencia y éxito ("**warningPanel**", "**successPanel**"), y botones de control que facilitan la interacción del usuario ("**playButton**", "**continueButtonSuccess**").

Estas variables permiten al script ser altamente configurable, facilitando la adaptación del juego a diferentes necesidades, lo que es crucial para un entorno de desarrollo de realidad virtual, donde la flexibilidad y la adaptabilidad son fundamentales.

Método Start()

Continuando con el contenido del script, el primero de los métodos que se puede destacar es el método "**Start()**". Este método 6.5 se ejecuta al iniciar la escena y establece el estado inicial de los componentes del juego. Además, se asegura que los paneles de advertencia y éxito estén desactivados al comienzo del juego ("**warningPanel.SetActive(false)**", "**successPanel.SetActive(false)**"), garantizando una interfaz limpia y enfocada en la experiencia del usuario.

```
warningPanel.SetActive(false);
successPanel.SetActive(false);

restartButtonSuccess.onClick.AddListener(RestartGame);
restartButtonWarning.onClick.AddListener(RestartGame);

continueButtonSuccess.onClick.AddListener(CloseMessagePanel;
continueButtonWarning.onClick.AddListener(CloseMessagePanel;

ClearCurrentMagnets();
ClearCurrentCubes();
}
```

[2] **Prefab:** recurso que permite crear múltiples instancias de un objeto en la escena sin tener que recrear el objeto cada vez, en este caso los cubos y los imanes serían prefabs

Listing 6.5: Método Start del script GameGenerator.cs

Método GameGenerator()

El método "**GameGenerator()**" es la función más importante dentro del script, ya que es el responsable de la creación dinámica de los cubos y los imanes que constituyen el puzzle. Este método es esencial para el funcionamiento del juego, ya que configura la disposición y características de los elementos del puzzle cada vez que se reinicia el juego.

El proceso comienza con la obtención de los componentes clave de la escena, como la cuadrícula, que se corresponde con la división de filas y columnas que el usuario realiza mediante el desplazamiento de los sliders en la pantalla principal ("**gridCreator**") y el fondo curvado de la pantalla principal ("**curvedBackground**"). Con el objetivo de obtener los diferentes materiales que se asociaran a posteriori con cada cubo que compone el puzzle principal.(Línea de código 6.6).

```
Material[] materials = DivideImageIntoMaterials(curvedBackgroundTexture2D,
    gridCreator.rows, gridCreator.columns);
```

Listing 6.6: Línea de inicialización del array de materiales del script GameGenerator.cs

Método DivideImageIntoMaterials()

Posteriormente, el método divide la imagen de fondo en múltiples materiales mediante el uso del método "**DivideImageIntoMaterials()**", cuya implementación se puede observar a continuación:

```
private Material[] DivideImageIntoMaterials(Texture2D image, int rows, int columns)
{
    Material[] materials = new Material[rows * columns];
    int partWidth = image.width / columns;
    int partHeight = image.height / rows;

    for (int r = 0; r < rows; r++)
    {
        for (int c = 0; c < columns; c++)
        {
            Texture2D partTexture = new Texture2D(partWidth, partHeight);
            partTexture.SetPixels(image.GetPixels(c * partWidth, (rows - 1 - r) *
    partHeight, partWidth, partHeight));
            partTexture.Apply();

            Material material = new Material(Shader.Find("Unlit/Texture"));
            material.mainTexture = partTexture;
            materials[r * columns + c] = material;
        }
    }

    return materials;
}
```

Listing 6.7: Método DivideMaterials() del script GameGenerator.cs

El anterior método 6.7 se encarga de dividir una imagen grande (que representa el puzzle completo) en partes más pequeñas, cada una de las cuales se convierte en

una textura aplicada a una cara de los cubos. El uso de "**SetPixels()**" y "**Apply()**" en "**Texture2D**" es clave, ya que permite crear sub-imágenes que corresponden a las distintas piezas del puzzle. Esto hace posible que los cubos individuales, cuando se ensamblen correctamente, formen la imagen completa, proporcionando un desafío visual para el usuario.

Esto asegura que cada cubo del puzzle tenga una parte única de la imagen original, lo que es crucial para la lógica del juego, ya que los usuarios deben reensamblar la imagen correctamente.

Una vez que se generan los materiales, el script procede a instanciar los cubos y los imanes en la escena. Los cubos son posicionados de manera aleatoria con rotaciones también aleatorias para aumentar la dificultad del puzzle. A cada cubo se le asigna un material específico y se le añaden componentes de interacción para que puedan ser manipulados en el entorno de realidad virtual.

```
for (int r = 0; r < gridCreator.rows; r++)
{
    for (int c = 0; c < gridCreator.columns; c++)
    {
        Vector3 cubePosition = new Vector3(Random.Range(-0.5f, 0.5f), 1.2f, Random.
    Range(0.2f, 0.6f));
        Quaternion cubeRotation = Quaternion.Euler(Random.Range(0f, 90f), Random.
    Range(0f, 90f), Random.Range(0f, 90f));
        GameObject cube = Instantiate(cubePrefab, cubePosition, cubeRotation);
        cube.name = $"Cube_{r}_{c}";
        cube.tag = "cube";
        cube.AddComponent<CubeInteraction>();
    }
}
```

Listing 6.8: Creación cubos del puzzle del script GameGenerator.cs

El fragmento de código anterior 6.8 muestra como el juego crea dinámicamente el puzzle, asegurando que cada pieza esté colocada de manera única y ofreciendo un reto al usuario para resolverlo.

Método IsPuzzleComplete()

Además, en el propio script se dispone de un método que permite la comprobación de la completitud del puzzle, que recibe el nombre de "**IsPuzzleComplete()**". Este método es invocado por un método anterior ("**CheckPuzzleCompletion()**"). La función principal del mismo es realizar una comprobación detallada de la alineación de cada cubo con su correspondiente posición en la cuadrícula de imanes. Como se observa a continuación:

```
private bool IsPuzzleComplete()
{
    GameObject[] cubes = GameObject.FindGameObjectsWithTag("cube";

    foreach (GameObject cube in cubes)
    {
        string[] splitName = cube.name.Split('_');
        int row, col;
        if (!int.TryParse(splitName[1], out row) || !int.TryParse(splitName[2], out
     col))
        {
            return false;
        }
```

```
13
14          Vector3 targetPosition = GetMagnetPosition(row, col);
15          if (Vector3.Distance(cube.transform.position, targetPosition) > 0.05f ||
16              Quaternion.Angle(cube.transform.rotation, Quaternion.identity) > 5.0f)
17          {
18              return false;
19          }
20      }
21
22      return true;
23  }
```

Listing 6.9: Método IsPuzzleComplete() del script GameGenerator.cs

Como se puede observar en el código 6.9 se verifica la posición y la orientación de cada cubo en relación con su posición objetivo, que es determinada por "**GetMagnetPosition()**". El uso de umbrales (por ejemplo, "**0.05f**" para la distancia y "**5.0f**" para el ángulo) permite cierta flexibilidad en la colocación, lo que es esencial para una buena experiencia de usuario en realidad virtual, donde la precisión absoluta puede ser difícil de lograr. Este método asegura que solo se considere completo el puzzle cuando todos los cubos estén correctamente colocados.

Gestión de la Interacción entre Cubos e Imanes: "CollisionDetector.cs"

La clase "**CollisionDetector**" (anexo A.1 sección A.1.2) es otro de los componentes cruciales dentro de la lógica de MindCubeXR. Su función principal es gestionar las interacciones entre los cubos del puzzle y los imanes en el entorno de realidad virtual, asegurando que los cubos se posicionen correctamente en la cuadrícula establecida por los imanes. A través de sus métodos clave, controla el anclaje y liberación de los cubos, así como su correcta alineación, garantizando una experiencia de usuario coherente y realista. La correcta implementación de este script es esencial para el funcionamiento suave y preciso del puzzle en el entorno de realidad virtual y su lógica asegura que los cubos se comporten de manera predecible y conforme a las expectativas del jugador.

El script comienza declarando las variables claves que se van a necesitar para el posterior control y manejo de los cubos durante la interacción:

```
1  bool released = true;
2  int iter = 0;
3  int iter_released = -int.MaxValue;
4  GameObject anchoredObject = null;
5  float x_anchored, y_anchored, z_anchored = 0;
6  Rigidbody rigidbody;
```

Listing 6.10: Declaración de variables clave del script CollisionDetector.cs

El fragmento de código 6.10 anterior ilustra las diferentes variables que se han inicializado, las más destacadas:

- "**bool released**": Esta variable controla si un cubo está liberado o anclado a un imán. Su estado determina si el cubo es libre de moverse o si está fijado en su lugar.
- "**GameObject anchoredObject**": Mantiene una referencia al cubo que está actualmente anclado a un imán. Esto es esencial para realizar los ajustes necesarios en su posición y rotación.

- "**Rigidbody rigidbody**": Representa el componente de física del cubo, permitiendo manipular su comportamiento físico, como el movimiento y la colisión.

Las variables detalladas anteriormente, como se ha dicho son esenciales para el seguimiento del estado del cubo y para asegurar que se gestionen adecuadamente las interacciones físicas entre el cubo y los imanes.

Siguiendo con el script, se pueden obsevar una serie de métodos pero se va a centrar la atención de la explicación en los más destacados. Empezando por el método "**OnTriggerEnter(Collider other)**".

Método OnTriggerEnter(Collider other)

Este método se ejecuta cuando un objeto entra en el área de colisión del **CollisionDetector**[3]. Su objetivo es identificar si el objeto es un cubo y, si lo es, anclarlo al imán correspondiente.

```
1  void OnTriggerEnter(Collider other)
2  {
3      var objectcolliding = findParentWithTag("cube", other.gameObject);
4      if (objectcolliding == null) return;
5
6      foreach (var refCube in GameObject.FindGameObjectsWithTag("refCube"))
7          if ((refCube != this.gameObject) && (refCube.GetComponent<CollisionDetector
       >().IsNotReleased(objectcolliding.name))) return;
8
9      anchoredObject = objectcolliding;
10     rigidbody = anchoredObject.GetComponent<Rigidbody>();
11
12     anchoredObject.transform.position = new Vector3(this.gameObject.transform.
       position.x,anchoredObject.transform.position.y,this.gameObject.transform.
       position.z);
13
14     released = false;
15 }
```

Listing 6.11: Método OnTriggerEnter(Collider other) del script CollisionDetector.cs

Como se puede obsevar en el método del código anterior 6.11, se determina si un cubo debe ser anclado a un imán cuando entra en su área de colisión. Para poder determinarlo se realiza una verificación ("**if((refCube != this.gameObject) && (refCube.GetComponent<CollisionDetector>().IsNotReleased(objectcolliding.name)))**") para asegurar que el cubo no esté ya anclado y, si no lo está, se ajusta su posición y se ancla físicamente a la posición del imán ("**anchoredObject.transform.position = new Vector3(this.gameObject.transform.position.x,this.gameObject.transform.position.z);** "). De esta forma se asegura que los cubos se alineen correctamente con la cuadrícula del puzzle.

Método OnTriggerExit(Collider other)

A continuación, otro de los métodos que se podrían destacar sería el método "**OnTriggerExit(Collider other)**". Este método se invoca cuando un cubo sale del área de colisión del propio imán. Su función principal es liberar el cubo anclado, permitiendo que vuelva a moverse libremente en el espacio de juego.

[3]Se trata de un script asociado a los imanes, que permite detectar cuando el usuario acerca o aleja el cubo de su área de colisión, lo que provoca que el imán atraiga o desprenda el cubo de la posición.

```
private void OnTriggerExit(Collider other)
{
    if (anchoredObject == null || anchoredObject != findParentWithTag("cube", other
    .gameObject)) return;

    if (!released)
    {
        if (iter < 15) return;

        anchoredObject = null;
    }
}
```

Listing 6.12: Método OnTriggerExit(Collider other) del script CollisionDetector.cs

Como se puede observar en el fragmento anterior, lo que se dispone a hacer este método es una comprobación permitiendo saber si el cubo no esta anclado (”**if (anchoredObject == null || anchoredObject != findParentWithTag(”cube”, other.gameObject))**”). En caso de que no lo esté no hace nada, pero si lo está entonces procede a ejecutar las siguientes líneas de código para desvincularlo del imán (”**anchoredObject = null**”). Esto permite que los cubos puedan ser reubicados o manipulados nuevamente por el usuario, manteniendo la flexibilidad necesaria para una interacción fluida en la propia escena.

Método Update()

Finalmente, otro de los métodos importantes a destacar en el script anterior sería el método ”Update()”. El método ”**Update()**” se ejecuta en cada frame y se encarga de gestionar el estado cinemático del cubo, así como de ajustar su rotación para asegurar que esté correctamente alineado en la cuadrícula del puzzle.

```
if (rigidbody != null && anchoredObject != null) {
        if ((!released) && (iter == 10)) {
            rigidbody.isKinematic = true;
        }

        var eulerAngles = anchoredObject.transform.rotation.eulerAngles;

        var dif0 = Mathf.Abs(0 - eulerAngles.y);
        var minY = Mathf.Min(dif0, ...);  // Comparación de ángulos

        if (minY == dif0) eulerAngles.y = 0;
        // Otros ajustes de rotación

        anchoredObject.transform.rotation = Quaternion.Euler(eulerAngles);
}
```

Listing 6.13: Método Update() del script CollisionDetector.cs

En este código 6.13, se asegura que el cubo, una vez anclado, permanezca correctamente alineado en la cuadrícula del puzzle. El ajuste de rotación automático (”**anchoredObject.transform.rotation = Quaternion.Euler(eulerAngles)**”) es crucial, ya que ayuda a mantener los cubos bien orientados, lo que es especialmente importante en un entorno de realidad virtual donde la precisión visual es fundamental.

Creación y Manipulación Dinámica de la Cuadrícula:: "GridCreator.cs"

La clase "**GridCreator**" (anexo A.1 sección A.1.3) se encarga de la creación y manipulación dinámica de la cuadrícula que sirve como base para la disposición de los cubos en el entorno de realidad virtual. Esta cuadrícula determina la organización espacial del puzzle y su estructura es clave para la jugabilidad, y la interacción del usuario.

Método Update()

El script comienza con la declaración de varias variables que definen la configuración y el comportamiento de la cuadrícula. Estas incluyen el número de filas y columnas ("**rows**", "**columns**"), así como un prefab que representa cada celda de la cuadrícula ("**gridCellPrefab**"). Además, se definen constantes que establecen las dimensiones del panel y las imágenes utilizadas para construir la cuadrícula.

```
public int rows;
public int columns;
public GameObject gridCellPrefab;
private GameObject[,] gameGrid;
private const int PANEL_WIDTH = 800;
private const int PANEL_HEIGHT = 500;
private const int IMAGE_WIDTH = 725;
private const int IMAGE_HEIGHT = 725;
public UnityEngine.UI.Slider columnsSlider, rowsSlider;
```

Listing 6.14: Declaración de Variables y Componentes Clave del script CollisionDetector.cs

Las variables "**rows**" y "**columns**" son fundamentales para determinar el tamaño de la cuadrícula, mientras que "**gridCellPrefab**" permite instanciar visualmente cada celda en la escena. La matriz "**gameGrid**" almacena referencias a todas las celdas generadas, lo que facilita su manipulación posterior. Al inicio, el usuario debe seleccionar el número de filas y columnas que desea para el puzzle mediante dos sliders, uno horizontal y otro vertical. Dependiendo de la posición de los sliders, se dibujan líneas sobre la imagen en la pantalla, facilitando la selección de la complejidad del puzzle.

Método createGrid()

El método "**createGrid()**" es el encargado de generar la cuadrícula en la escena. Este método calcula las dimensiones y la posición de cada celda en función de las dimensiones del panel y de la imagen de fondo. Luego, instancia las celdas en sus posiciones correspondientes y ajusta sus escalas para asegurar que se adapten correctamente al espacio disponible.

```
private void createGrid()
{
    var cell_x_size = PANEL_WIDTH / columns;
    var cell_y_size = PANEL_HEIGHT / rows;
    gameGrid = new GameObject[columns, rows];

    for (int y = 0; y < rows; y++)
        for (int x = 0; x < columns; x++)
        {
            gameGrid[x, y] = Instantiate(gridCellPrefab, ...);
            gameGrid[x, y].transform.localPosition = new Vector3(...);
        }
```

```
13 }
```

Listing 6.15: Método createGrid() del script CollisionDetector.cs

Este código de este método 6.15 asegura que cada celda se posicione correctamente dentro del panel, lo que es crucial para mantener la coherencia visual y funcional del puzzle. La utilización de "**Instantiate**" para crear cada celda permite que la cuadrícula se genere dinámicamente, lo que es vital para la adaptabilidad del juego.

Además, se tienen otros métodos 6.16 ("**OnColumnsSliderValueChange()**" y "**OnRowsSliderValueChange()**") que permiten al usuario ajustar el tamaño de la cuadrícula mediante sliders de la interfaz. Estos métodos actualizan la cuadrícula en tiempo real, destruyendo la cuadrícula existente y creando una nueva con el tamaño ajustado. Son fundamentales para la adaptabilidad del juego, permitiendo que la cuadrícula se ajuste a diferentes configuraciones de puzzle, lo que enriquece la experiencia de usuario al ofrecer un entorno flexible y dinámico.

```
1 private void createGrid()
2 public void OnColumnsSliderValueChange() {
3     changeSize(rows, (int)columnsSlider.value);
4 }
```

Listing 6.16: Método OnColumnsSliderValueChange() del script CollisionDetector.cs

Control de la Selección de Imágenes para el Puzzle: "ImagePanelController.cs"

El script "**ImagePanelController.cs**" (anexo A.1 sección A.1.5) se caracteriza por permitir a los usuarios la selección y actualización dinámica de la imagen de fondo del puzzle, ofreciendo una experiencia personalizada dentro del entorno de realidad virtual.

Este script utiliza una variable pública llamada "**gameGeneraror**", que es esencial para coordinar la selección de imágenes con la lógica de generación del puzzle. Además, esta variable actúa como un puente con el componente "**GameGenerator**", permitiendo que el "**ImagePanelController**" interactúe con él para actualizar la imagen seleccionada que se aplicará en el puzzle.

El método más significativo en esta clase es "**SelectImage(bool active)**", que se encarga de actualizar la imagen de fondo del puzzle según la selección del usuario en el panel de imágenes de la interfaz. Cuando el usuario selecciona una imagen (mediante la interacción con un botón en la interfaz), este método se invoca automáticamente, actualizando de inmediato la imagen en la pantalla curva principal de la escena.

```
1  public void SelectImage(bool active) {
2      if (active) {
3          Image toggleButtonImage = this.gameObject.GetComponentInChildren<Image>();
4          GameObject curvedBackground = GameObject.FindGameObjectWithTag("
       curvedBackground");
5          curvedBackground.GetComponent<Image>().sprite = toggleButtonImage.sprite;
6
7          // Actualiza la imagen seleccionada en el GameGenerator
8          gameGenerator.OnImageSelected(toggleButtonImage);
9      }
10 }
```

Listing 6.17: Método SelectImage(bool active) del script CollisionDetector.cs

Como se muestra el fragmento de código 6.17, el método "**SelectImage()**" comienza obteniendo la imagen seleccionada por el usuario en la interfaz de usuario, mediante la instrucción "**Image toggleButtonImage = this.gameObject.GetComponentInChildren<Image>()**". A continuación, se cambia el "**Sprite**"[4] del objeto de fondo ("**curvedBackground**") para reflejar la nueva imagen seleccionada, lo que permite una visualización inmediata de la elección del usuario. Finalmente, el método informa a "**GameGenerator**" sobre la nueva imagen seleccionada, asegurando que se utilice correctamente en la generación del puzzle.

Gestión de la Física y Selección de Objetos: "PhysiscsGrabbable.cs"

El script "**PhysicsGrabbable.cs**" es una pieza central para la interacción física. Proveniente del SDK de Oculus, este script se encarga de gestionar la manipulación de objetos mediante eventos de selección, deselección y la aplicación de fuerzas físicas para simular el comportamiento realista de los objetos en el espacio virtual. A continuación, se explican las funcionalidades más destacadas que se han podido observar en el script.

La primera de ellas es el manejo de componentes 6.18, el script utiliza "**Rigidbody**"[5] para manejar la física del objeto, incluyendo su masa, movimiento y la aplicación de fuerzas y toque cuando es manipulado. Además, emplea la interfaz "**IPointable**" para definir los puntos de interacción, permitiendo que los usuarios puedan seleccionar o interactuar con los objetos mediante dispositivos de entrada como controladores de realidad virtual.

```
1 [SerializeField, Interface(typeof(IPointable))]
2 private UnityEngine.Object _pointable;
3
4 [SerializeField]
5 private Rigidbody _rigidbody;
```

Listing 6.18: Manejo de Componentes del script PhysiscsGrabbable.cs

A continuación, la línea de código 6.19 se dispone de una opción para escalar la masa del objeto de acuerdo con su tamaño, lo que es fundamental cuando el objeto cambia de tamaño en la aplicación, manteniendo una física realista.

```
1 private bool _scaleMassWithSize = true;
```

Listing 6.19: Escalado de Masa con el Tamaño del script PhysiscsGrabbable.cs

En el propio script también se tienen métodos ("**DisablePhysics()**" y "**ReenablePhysics()**") que permiten desactivar y reactivar la física para evitar comportamientos no deseados, como la caída del objeto mientras se manipula.

```
1 private void DisablePhysics()
2 {
3     CachePhysicsState();
```

[4] **Sprite:** Se trata de imágenes 2D planas (normalmente en formatos como PNG o JPEG) que pueden ser animadas, escaladas o rotadas en el entorno del juego. En este caso se corresponden con las imágenes de los puzzles.

[5] **Rigidbody:** Componente utilizado para añadir física realista a los objetos de un juego o aplicación. Un GameObject con un Rigidbody puede moverse, rotar y ser afectado por fuerzas físicas como la gravedad, la fricción, colisiones o fuerzas externas.

```
4       _rigidbody.LockKinematic();
5   }
6
7   private void ReenablePhysics()
8   {
9       if (_scaleMassWithSize)
10      {
11          float initialScaledVolume = _initialScale.x * _initialScale.y *
        _initialScale.z;
12          Vector3 currentScale = _rigidbody.transform.localScale;
13          float currentScaledVolume = currentScale.x * currentScale.y * currentScale.
        z;
14          float changeInMassFactor = currentScaledVolume / initialScaledVolume;
15          _rigidbody.mass *= changeInMassFactor;
16      }
17      _rigidbody.UnlockKinematic();
18      _rigidbody.isKinematic = false;
19  }
```

Listing 6.20: Ajuste de las Propiedades Físicas del Objeto del script PhysiscsGrabbable.cs

En el código anterior 6.20, el método "**DisablePhysics**", se guarda el estado físico actual del objeto ("**CachePhysicsState()**") y se bloquea su estado cinemático ("**_rigidbody.LockKinematic()**"), lo que permite que el objeto sea manipulado sin que la física estándar de Unity lo afecte hasta que sea liberado. Mientras, que el método "**ReenablePhysics()**" se encarga de ajustar la masa del objeto basándose en la diferencia de volumen antes y después del cambio de escala ("**_scaleMassWithSize**"). Esto es esencial para mantener una experiencia de interacción física coherente, donde objetos de diferentes tamaños se comporten de manera realista.

```
1   public void ApplyVelocities(Vector3 linearVelocity, Vector3 angularVelocity)
2   {
3       _hasPendingForce = true;
4       _linearVelocity = linearVelocity;
5       _angularVelocity = angularVelocity;
6   }
7
8   private void FixedUpdate()
9   {
10      if (_hasPendingForce)
11      {
12          _hasPendingForce = false;
13          _rigidbody.AddForce(_linearVelocity, ForceMode.VelocityChange);
14          _rigidbody.AddTorque(_angularVelocity, ForceMode.VelocityChange);
15      }
16  }
```

Listing 6.21: Aplicación de Velocidades del script PhysiscsGrabbable.cs

Por último, en el código anterior 6.21, una vez que el objeto ha sido soltado, el script puede aplicar las velocidades lineales y angulares ("**ApplyVelocities**") que tenía en el momento de la liberación, para simular un lanzamiento o movimiento continuo.

Gestión de la Interacción del Cubo con el Puzzle: "CubeInteraction.cs"

El script "**CubeInteraction.cs**" (anexo A sección A.1.4) se encarga de manejar la lógica detrás de la interacción de los cubos con los imanes en el entorno del puzzle. Este componente es el que se encarga de determinar si un cubo ha sido correctamente

colocado en la cuadrícula y de gestionar el progreso del usuario en la resolución del puzzle. A continuación, se destacan los elementos clave de este script:

Empezando por las variables 6.22, las más destacadas en este caso son en primera instancia "**gameGenerator**" que permite almacenar una referencia al componente "**GameGenerator**", que es el responsable de la creación y configuración general del puzzle. Gracias a esta referencia, "**CubeInteraction**" puede acceder a la configuración del puzzle, como el número de filas y columnas, y validar si un cubo ha sido colocado correctamente.

Por otro lado, se tiene la variable "**cubesPlacedCorrectly**" que lleva la cuenta del número total de cubos que han sido colocados correctamente en sus respectivas posiciones. Es una variable compartida por todos los cubos, lo que permite hacer un seguimiento global del progreso del puzzle.

Por último, la siguiente variable más destacada sería la variable "**isPlaced**" que es de tipo "**boolean**" y se asegura que cada cubo solo se cuente una vez cuando se coloca en la posición correcta. Esto es fundamental para evitar que el contador "**cubesPlacedCorrectly**" incremente más de una vez por un mismo cubo.

```
1  private GameGenerator gameGenerator;
2  public static int cubesPlacedCorrectly = 0; // Contador de cubos colocados
      correctamente
3  private static int totalCubes; // Número total de cubos a colocar
4  private bool isPlaced = false; // Variable para asegurarse de que solo se cuente
      una vez
```

Listing 6.22: Variables Fundamentales del script CubeInteraction.cs

Continuando con el resto del código, el método principal que destaca en este script es "**OnTriggerEnter(Collider other)**", que se invoca cuando un cubo entra en contacto con un imán en la cuadrícula. Este método es vital para el juego, ya que determina si un cubo ha sido colocado correctamente y, si es así, actualiza el estado del puzzle.

```
1  void OnTriggerEnter(Collider other)
2  {
3      if (!isPlaced && other.CompareTag("refCube"))
4      {
5          Debug.Log($"Cubo {gameObject.name} colocado correctamente en {other.
       gameObject.name}");
6          cubesPlacedCorrectly++;
7          isPlaced = true;
8
9          if (cubesPlacedCorrectly == totalCubes && gameGenerator != null)
10         {
11             Debug.Log("Puzzle completado. Verificando...");
12             gameGenerator.CheckPuzzleCompletion();
13         }
14     }
15 }
```

Listing 6.23: Método OnTriggerEnter(Collider other) del script CubeInteraction.cs

En este fragmento de código 6.23, se puede observar que cuando un cubo entra en contacto con un imán, el script verifica si el cubo ha sido colocado correctamente. Si el cubo no ha sido previamente marcado como colocado ("**!isPlaced**") y el objeto con el que colisiona es un imán ("**other.CompareTag(refCube")**"), se incrementa el contador "**cubesPlacedCorrectly**" y el cubo se marca como colocado ("**isPlaced =**

true"). Posteriormente, si todos los cubos han sido colocados, se llama al método "**CheckPuzzleCompletion()**" para verificar si el puzzle ha sido resuelto correctamente. Este proceso asegura que el puzzle solo se considere completo cuando todos los cubos estén en su lugar correcto.

Por último, se tiene el método 6.24 que se encarga de la gestión de restar del contador "**cubesPlacedCorrectly**" cuando un cubo previamente colocado es removido de su posición correcta, permitiendo que los jugadores puedan corregir sus movimientos sin que el juego registre erróneamente la colocación de los cubos. De esta manera, el sistema asegura que solo se consideren los cubos que estén correctamente posicionados al finalizar el puzzle.

```
1 void OnTriggerExit(Collider other)
2 {
3     if (isPlaced && other.CompareTag("refCube"))
4     {
5         cubesPlacedCorrectly--;
6         isPlaced = false;
7     }
8 }
```

Listing 6.24: Método OnTriggerExit(Collider other) del script CubeInteraction.cs

CAPÍTULO 7
Pruebas y Validación

En cualquier proyecto de desarrollo de software, la **fase de pruebas** es crucial para garantizar la calidad, fiabilidad y robustez del producto final. Las pruebas permiten identificar y corregir errores, verificar que el sistema cumple con los requisitos especificados, y asegurar que todas las funcionalidades operan de manera coherente bajo diferentes escenarios. En el contexto del presente proyecto, que involucra el desarrollo de una aplicación de realidad extendida (XR) para la rehabilitación motora y cognitiva, la realización de pruebas adquiere una importancia particular debido a la interacción directa del usuario con un entorno virtual complejo.

Para asegurar que la aplicación desarrollada cumple con los estándares de calidad esperados, **se implementaron diferentes pruebas** (pruebas unitarias, de integración y de usabilidad) para garantizar que la aplicación fuera robusta, intuitiva y eficiente en un entorno real de uso. A continuación, se describen los pasos seguidos para la implementación de las pruebas unitarias, se presenta un ejemplo detallado de una prueba crítica realizada para el script "GameGenerator", y finalmente, se discuten los resultados generales obtenidos.

7.1 Pruebas unitarias

Las **pruebas unitarias** permiten validar el correcto funcionamiento de componentes individuales de manera aislada. Estas pruebas tienen como objetivo cubrir todas las posibles situaciones, garantizando así el correcto funcionamiento del código. En este proyecto, las pruebas unitarias fueron una parte integral del proceso de desarrollo, siguiendo la metodología ágil, que enfatiza la importancia de la calidad del código en cada iteración.

En el proyecto, se implementaron pruebas unitarias (anexo A, sección A.2.) para asegurar que los elementos clave de la aplicación, especialmente aquellos relacionados con la generación y gestión del puzzle tridimensional, operaran de acuerdo con las expectativas. Estas pruebas se realizaron utilizando el framework "**NUnit**" y el sistema de pruebas integrado de Unity, conocido como "**Unity Test Runner**", lo que permitió ejecutar pruebas tanto en modo de edición como en modo de juego.

7.1.1. Prueba Unitaria del GameGenerator.cs

Para poder mostrar el proceso que se ha seguido para realizar la correcta implementación de las mismas se procederá a explicar el ejemplo que se había mencionado anteriormente de la prueba realizada sobre el script "GameGenerator", que se encarga de generar los cubos e imanes que conforman el puzzle, así como de validar si el usuario ha completado correctamente el puzzle correctamente o no.

```
[UnityTest]
public IEnumerator GameGenerator_CreatesAndValidatesPuzzleCorrectly()
{
    // Configurar el GameObject y añadir el script GameGenerator
    var gameGenerator = new GameObject().AddComponent<GameGenerator>();
    gameGenerator.rows = 3;
    gameGenerator.columns = 3;

    // Configurar otros componentes necesarios
    gameGenerator.successPanel = new GameObject();
    gameGenerator.successMessageText = gameGenerator.successPanel
                                                    .AddComponent<TextMeshProUGUI
    >();
    gameGenerator.successPanel.SetActive(false);  // Desactivar al inicio

    // Ejecutar la generación del puzzle
    gameGenerator.GenerateGame();

    // Esperar un frame para asegurar que todo se ha creado
    yield return null;

    // Verificar que se han creado el número correcto de cubos e imanes
    var cubes = GameObject.FindGameObjectsWithTag("cube");
    Assert.AreEqual(gameGenerator.rows * gameGenerator.columns, cubes.Length
                                                          , "El número de cubos
    generados no es el correcto.");

    // Simular la colocación correcta de los cubos y verificar la completitud del
    puzzle
    gameGenerator.CheckPuzzleCompletion();
    Assert.IsTrue(gameGenerator.successPanel.activeSelf, "El mensaje de éxito no se
     mostró al completar el puzzle.");
}
```

Listing 7.1: Test del script GameGenerator.cs

El test que ilustra el código anterior 7.1 comienza creando un nuevo "**GameObject**" al cual se le agrega el componente "**GameGenerator**". Este script es el encargado de manejar la lógica de generación del puzzle, y se le configuran las dimensiones del puzzle mediante las propiedades "**rows**" y "**columns**", asignándole valores de "**3x3**", lo que implica un total de "**9 cubos e imanes**" que deben ser generados.

Además, se configuran otros elementos necesarios para la prueba, como el "**successPanel**" y el "**successMessageText**", que son usados para mostrar un mensaje de éxito cuando el puzzle se completa correctamente. El "**successPanel**" se desactiva inicialmente para evitar resultados falsos positivos.

Una vez configurado el entorno, se invoca el método "**GenerateGame()**" del "**GameGenerator**", el cual es responsable de crear todos los elementos del puzzle en la escena. Después de llamar a este método, se incluye un "**yield return null**" para esperar un frame, asegurando que todos los elementos se hayan generado antes de proceder con las validaciones.

A continuación, en el test se verifica verifica que el número de cubos generados coincide con el número de celdas definidas por las propiedades "**rows**" y "**columns**". Esto se realiza utilizando "**GameObject.FindGameObjectsWithTag("cube")**", que devuelve todos los objetos con la etiqueta "**cube**" en la escena, y se compara su longitud con el número esperado. Mediante esta validación se asegura asegurar que la lógica de generación del puzzle está funcionando correctamente y que se están creando todos los elementos necesarios.

Finalmente, el test simula la colocación correcta de todos los cubos y llama al método "**CheckPuzzleCompletion()**" para validar si el puzzle se considera completado. Se verifica que, al completar el puzzle, el "**successPanel**" se activa, mostrando el mensaje de éxito al usuario.

7.1.2. Resultados de las Pruebas unitarias

Después de implementar todas las pruebas unitarias, se procedió a ejecutar el conjunto completo de pruebas utilizando el Test Runner integrado en Unity. Como se muestra en la figura 7.1, todas las pruebas se ejecutaron correctamente.

En total, se implementaron **25 pruebas unitarias**, cubriendo las funcionalidades críticas de los scripts más relevantes del proyecto, como "**GameGenerator**", "**CollisionDetector**", "**CubeInteraction**", "**GridCreator**", e "**ImagePanelController**". El hecho de que todas las pruebas hayan pasado exitosamente demuestra que las funciones clave del sistema están operando según lo esperado.

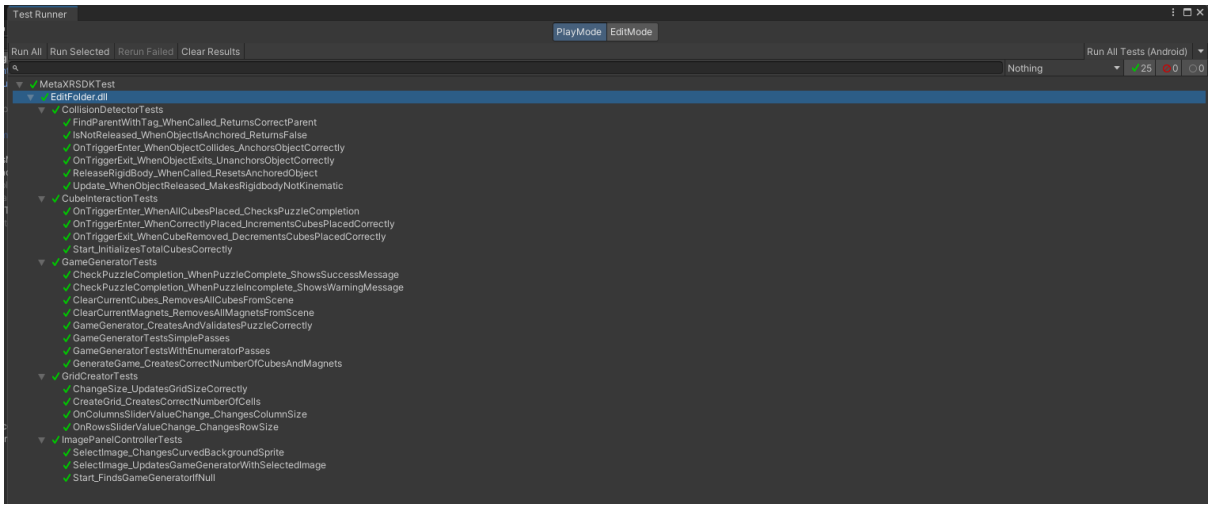

Figura 7.1: Resultados de las Pruebas Unitarias

7.2 Pruebas de integración

Las **pruebas de integración** permiten asegurar que los distintos módulos del proyecto funcionan correctamente cuando interactúan entre sí. A diferencia de las pruebas unitarias, que se centran en componentes individuales de manera aislada, las pruebas de integración verifican que los sistemas colaboran correctamente para lograr la funcionalidad deseada.

En el desarrollo de este proyecto, las pruebas de integración se han diseñado para validar la correcta interacción entre los componentes clave del sistema, asegurando que las funcionalidades críticas se ejecuten correctamente cuando los módulos están integrados. Estas pruebas abarcan desde la generación de la cuadrícula y la colocación de los cubos hasta la gestión de la interfaz de usuario y la respuesta a las acciones del usuario. Las pruebas se realizaron de la misma manera que las unitarias, mediante el framework "**NUnit**" y "**Unit Test**".

Los casos en los que se ha centrado la atención incluyen el comprobar que la cuadrícula del juego se genera correctamente y los cubos se posicionan adecuadamente dentro de la misma (Integración entre "**GameGenerator**" y "**GridCreator**"), validar que los cubos generados por el "GameGenerator" interactúan correctamente con el sistema de detección de colisiones (Integración entre "**GameGenerator**" y

"**CollisionDetector**"), verificar que la selección de imágenes en la interfaz de usuario se refleje correctamente en el juego, actualizando la imagen mostrada en el fondo (Integración entre "**ImagePanelController**" y "**GameGenerator**") y por último, se evaluó como los controles de la interfaz de usuario, como deslizadores o botones, afectan la configuración y generación del juego (Integración entre la **interfaz de usuario (UI) y el juego**). (Anexo A sección A.3).

A continuación, se procederá a dar un ejemplo de prueba de integración realizada que se relaciona con uno de los aspectos críticos del sistema, más concretamente, con la interacción entre el "**GameGenerator**", encargado de crear la cuadrícula y los cubos del juego, y el "**CollisionDetector**", que gestiona las colisiones entre los cubos durante el juego. Es esencial que, una vez que los cubos se generan y se colocan en la cuadrícula, el sistema de colisiones funcione de manera precisa para que el juego se comporte como se espera.

7.2.1. Prueba Integración entre GameGenerator y CollisionDetector

La prueba de integración denominada "**GameGenerator_IntegratesWithCollisionDetector_AnchorsCubesCorrectly**" se ha diseñado con el objetivo de garantizar que los cubos generados por el GameGenerator se anclan correctamente a los imanes mediante el "**CollisionDetector**". Esta funcionalidad es esencial para asegurar que la interacción física dentro del juego, en la que los cubos se colocan sobre los imanes, se produce de manera precisa y coherente.

```
1  using System.Collections;
2  using NUnit.Framework;
3  using UnityEngine;
4  using UnityEngine.TestTools;
5
6  public class GameGeneratorCollisionDetectorIntegrationTests
7  {
8      [UnityTest]
9      public IEnumerator
       GameGenerator_IntegratesWithCollisionDetector_AnchorsCubesCorrectly()
10     {
11         // Configurar el GameObject y añadir los componentes necesarios
12         var gameGenerator = new GameObject().AddComponent<GameGenerator>();
13         gameGenerator.rows = 2;
14         gameGenerator.columns = 2;
15         gameGenerator.GenerateGame();
16
17         yield return null; // Esperar un frame para que los cubos sean generados
18
19         // Simular la colocación de cubos en la posición de los imanes
20         foreach (var cube in GameObject.FindGameObjectsWithTag("cube"))
21         {
22             var collisionDetector = cube.AddComponent<CollisionDetector>();
23             cube.transform.position = gameGenerator.GetMagnetPosition(0, 0);
24             yield return null; // Esperar un frame para que las interacciones sean
       procesadas
25             Assert.IsNotNull(collisionDetector); // Verificar que el componente se
       añadió correctamente
26         }
27     }
28 }
```

Listing 7.2: Ejemplo de Prueba de Integración: Interacción entre GameGenerator y CollisionDetector

El test que se ilustra anteriormente 7.2, sigue una serie de pasos clave. En primer lugar, se configura un nuevo "**GameObject**" al que se le añade el componente "**GameGenerator**". Este componente se encarga de crear una cuadrícula de juego con las dimensiones especificadas, en este caso, una cuadrícula de 2x2.

Después de que los cubos se generan, se espera un frame ("**yield return null;**") para asegurar que todos los elementos se han creado correctamente en la escena. A continuación, se simula una interacción general del juego en la que los cubos son arrastrados hacia las posiciones de los imanes, lo cual es fundamental para la mecánica del juego. Para cada cubo, se añade un componente "**CollisionDetector**", que es el encargado de manejar las colisiones.

Finalmente, se verifica que cada cubo efectivamente tenga el componente "**CollisionDetector**" mediante la instrucción "**Assert.IsNotNull(collisionDetector)**". Esta verificación es crucial, ya que confirma que los cubos han sido correctamente preparados para interactuar físicamente con los imanes en la escena, asegurando así que la integración entre el "**GameGenerator**" y el "**CollisionDetector**" es exitosa.

7.2.2. Resultados de las Pruebas de integración

Las **pruebas de integración realizadas** confirmaron que los componentes del sistema, cuando se integran, **interactúan de manera correcta y consistente**. En particular, el caso de prueba presentado verificó que la generación de la cuadrícula y los cubos del juego, junto con la detección de colisiones, funcionan correctamente cuando se combinan. El "**GameGenerator**" genera los cubos según lo esperado, y el "**CollisionDetector**" gestiona las colisiones de forma precisa, lo cual es fundamental para la jugabilidad del sistema.

La ejecución de las pruebas de integración **mostró resultados positivos**, ya que todas las pruebas en la categoría "**IntegrationTests.dll**" fueron superadas exitosamente, como se observa en la figura 7.2. Los **tests** pasaron **sin errores**, lo que indica que los diferentes componentes del sistema están bien integrados y se comportan como se espera cuando trabajan en conjunto.

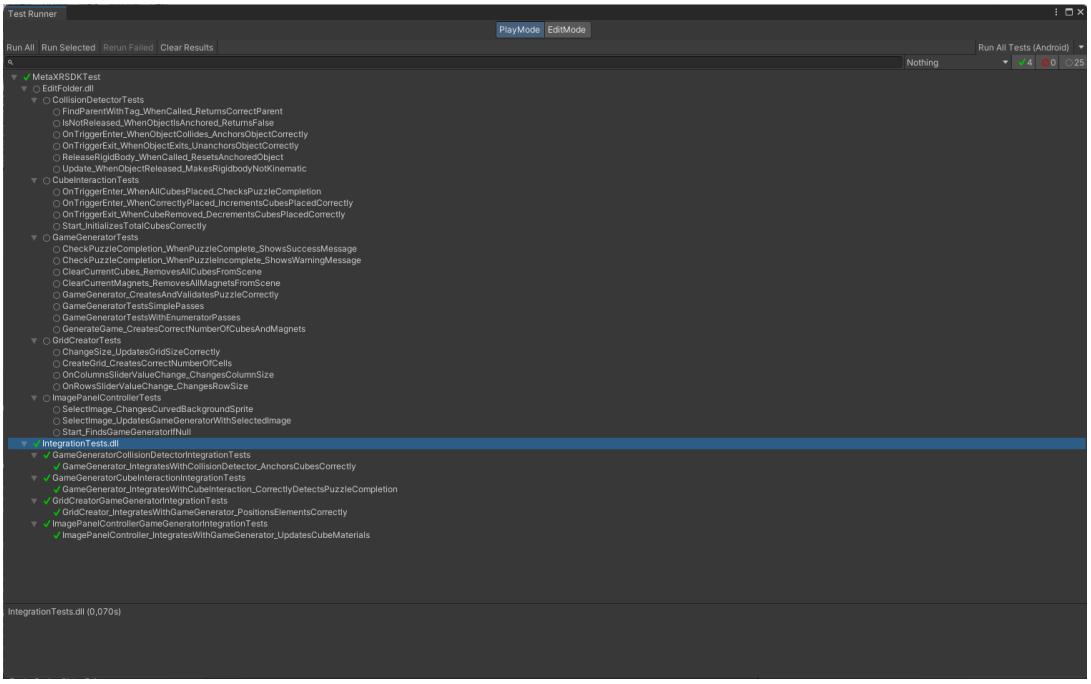

Figura 7.2: Resultados de las Pruebas de Integración

7.3 Pruebas de usabilidad

Una vez finalizado el desarrollo del proyecto, se consideró fundamental realizar una **evaluación de su usabilidad** para comprender mejor como los usuarios interactúan con el sistema y obtener información valiosa sobre su experiencia general. La evaluación se llevó a cabo a través de una **encuesta diseñada específicamente para este propósito**, con el objetivo de identificar posibles áreas de mejora y validar que los objetivos de diseño y funcionalidad fueron alcanzados. Mediante, la encuesta que se creó se recopilaron datos cuantitativos y cualitativos sobre diversos aspectos de la interacción con la aplicación, tales como la facilidad de uso, la claridad de la interfaz, la eficiencia en la interacción y la satisfacción general del usuario.

7.3.1. Diseño y Metodología

Como se ha indicado anteriormente, para evaluar la usabilidad de la aplicación de realidad virtual desarrollada, se diseñó una **`encuesta en Google Forms`**[1]. El propósito de esta encuesta fue recopilar opiniones de los usuarios sobre diversos aspectos de la interfaz, los controles y la eficiencia general del sistema. La encuesta se organizó en seis secciones principales: información demográfica, facilidad de uso, claridad de la interfaz, eficiencia en la interacción, satisfacción general del usuario y comentarios abiertos. La mayoría de las preguntas fueron diseñadas con una **escala de Likert de cinco puntos**[2], que iba desde "**Totalmente en desacuerdo**" hasta "**Totalmente de acuerdo**". Esta escala permitió captar la intensidad de las percepciones de los usuarios sobre diferentes aspectos de la aplicación.

Dado que el acceso a un amplio grupo de usuarios de diferentes rangos de edad no fue posible, la encuesta se distribuyó entre amigos, familiares y una persona que ha sufrido un ictus, quienes proporcionaron retroalimentación basada en sus experiencias individuales con MindCubeXR. En total, se obtuvieron 30 respuestas, lo que permitió realizar un análisis cualitativo y cuantitativo de la experiencia de usuario.

En cuanto a la **información demográfica** (Figuras 7.3 y 7.4), la mayoría de los participantes tenían entre 18 y 24 años (30 %), seguidos por grupos de menos de 18 años (20 %) y aquellos de 25 a 34 años (16, %). Grupos de mayor edad también estuvieron representados, con un 20 % de personas de entre 45 y 54 años y un 13,3 % de más de 55 años.

En términos de **género**, el 53,3 % de los encuestados se identificaron como femeninos, el 46,7 % como masculinos y ninguno prefirió no decirlo.

[1] https://forms.gle/PHJ3jLNYhCSGwYyg6

[2] Se caracteriza por ser un tipo de escala de medición que se utiliza comúnmente en encuestas y estudios para evaluar actitudes, opiniones o percepciones de las personas.

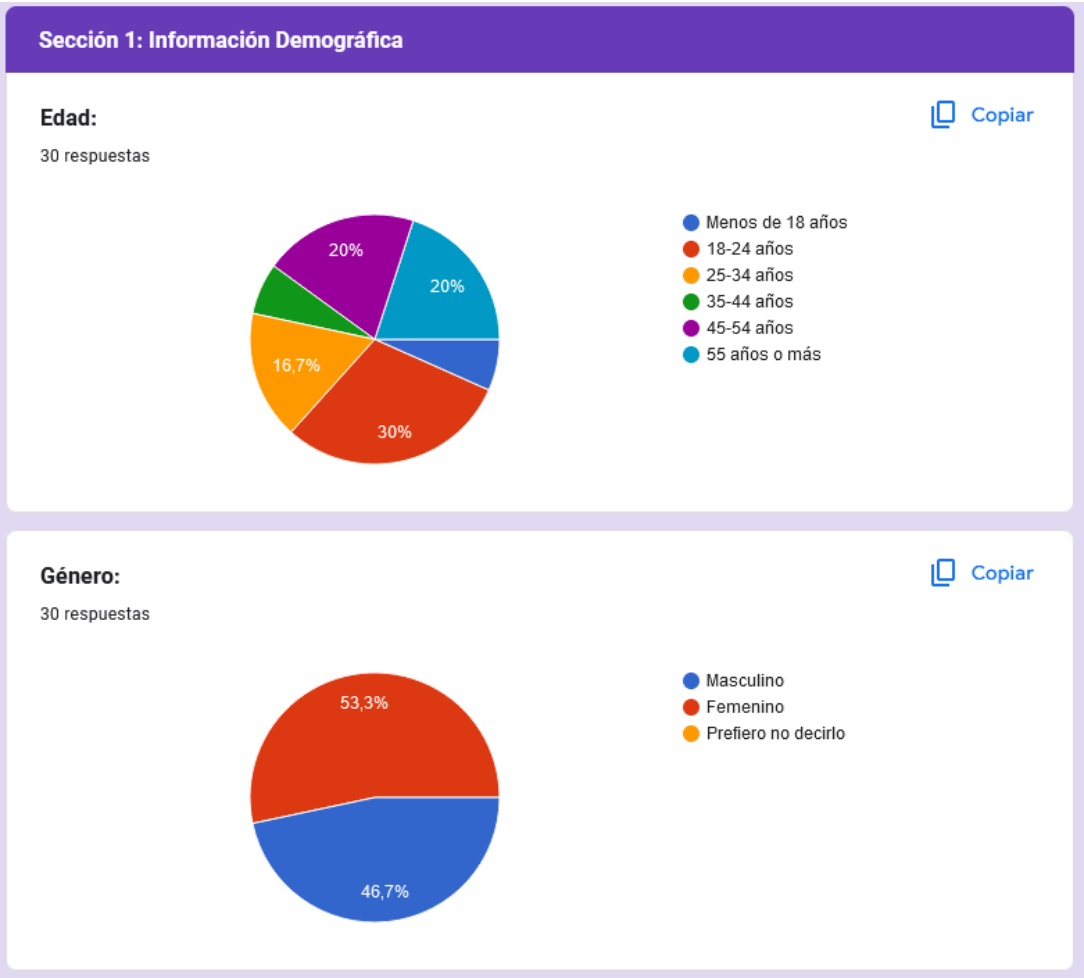

Figura 7.3: Gráfico que ilustra las respuestas de los usuarios sobre su edad y género

Respecto al **nivel educativo**, el 46,7 % contaba con estudios universitarios, mientras que un 33,3 % tenía un título técnico superior, y el 20 % había completado la secundaria.

En cuanto a la **experiencia previa con aplicaciones de realidad virtual (VR)**, el 60 % de los participantes no tenía experiencia alguna, un 20 % tenía una experiencia básica, un 16,7 % una experiencia intermedia y solo un 3,3 % reportó experiencia avanzada.

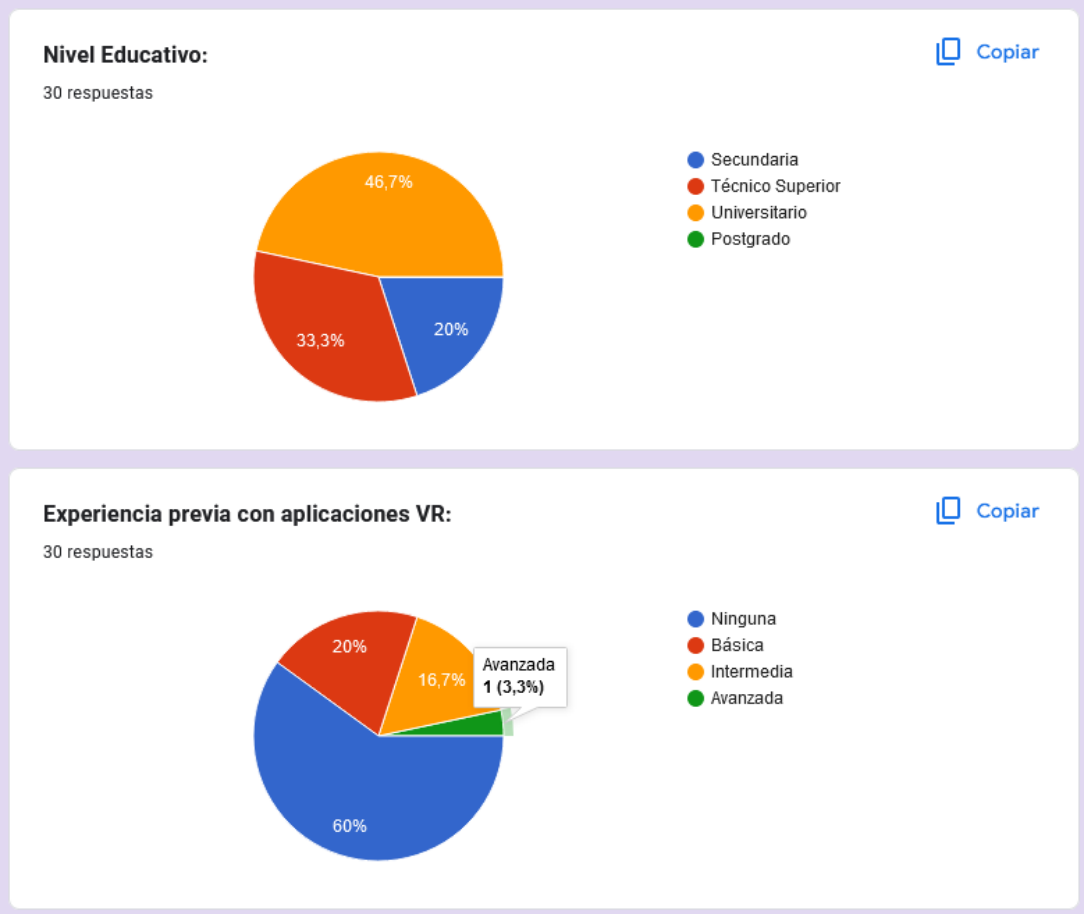

Figura 7.4: Gráfico que ilustra las respuestas de los usuarios sobre su nivel educativo y experiencia previa con aplicaciones VR

7.3.2. Criterios de Evaluación

Los criterios de evaluación se centraron en cuatro áreas clave:

1. **Facilidad de Uso:** Se evaluó lo fácil que fue para los usuarios navegar y utilizar la aplicación.

2. **Claridad de la Interfaz:** Se midió la claridad y comprensibilidad de los elementos visuales y las instrucciones dentro de la aplicación.

3. **Eficiencia en la Interacción:** Se analizó la rapidez y fluidez con la que los usuarios podían generar y resolver los puzzles, así como la respuesta de la aplicación a sus interacciones.

4. **Satisfacción General:** Se evaluó el grado de satisfacción del usuario con la experiencia completa ofrecida por la aplicación.

7.3.3. Resultados y Análisis

Los resultados de la encuesta reflejan un alto grado de satisfacción en general, con algunos puntos específicos que merecen atención para futuras mejoras:

- **Facilidad de Uso:**

- **Interfaz de la Aplicación en VR:** Como se puede observar en la figura 7.5, 63,3 % de los participantes estuvo "**De acuerdo**" en que la interfaz es fácil de usar, mientras que un 20 % estuvo "**Totalmente de acuerdo**". Este es un resultado positivo que sugiere que la mayoría de los usuarios encontró la aplicación accesible y sencilla. Sin embargo, el 16,7 % que se mantuvo neutral indica que hay espacio para mejorar la experiencia de usuario, posiblemente simplificando aún más la interfaz o proporcionando más guías y tutoriales.

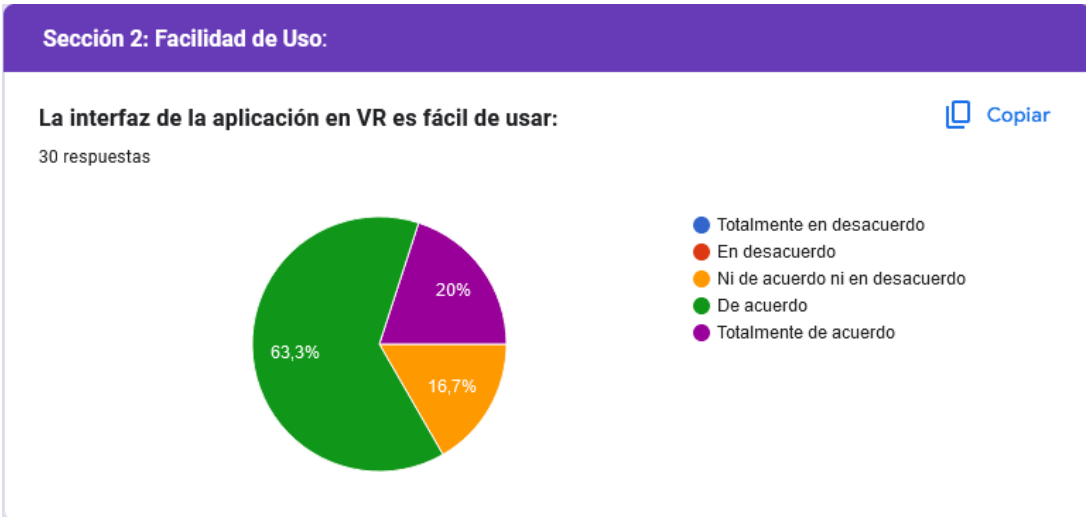

Figura 7.5: Gráfico que ilustra las respuestas de los usuarios sobre la facilidad de uso de la interfaz en VR

- **Controles VR:** En cuanto a los controles VR el 66,7 % estuvo de acuerdo que los controles, es decir, las manos para desenvolverte en el entorno de realidad inmersiva, son intuitivas para interactuar con los cubos y el entorno de la escena, y un 20 % estuvo totalmente de acuerdo, lo que indica también una buena aceptación por parte de los usuarios. En cambio un 13,3 % no estuvo ni de acuerdo ni en desacuerdo principalmente debido a que no se desenvolvieron en un inicio con tanta soltura en el entorno de VR. (Figura 7.6)

Figura 7.6: Gráfico que ilustra las respuestas de los usuarios sobre la facilidad de uso de los controles en VR

- **Selección y Colocación de Cubos:** Como ilustra la figura 7.7, el 53,3 % estuvo de acuerdo en que tanto el proceso de escoger las filas y columnas del puzzle, así como también el hecho de colocar los cubos en las propias casillas es un proceso sencillo y directo, y un 40 % estuvo totalmente de acuerdo. Esto refuerza la percepción de facilidad en la interacción con el entorno. En cambio, sólo un 6,7 % no estuvieron ni de acuerdo ni en desacuerdo porque probablemente experimentaron algún tipo de dificultad a la hora de colocar los cubos o desplazar con los sliders horizontal y vertical con el objetivo de escoger por cuantas filas y columnas querían que estuviera formado el puzzle.

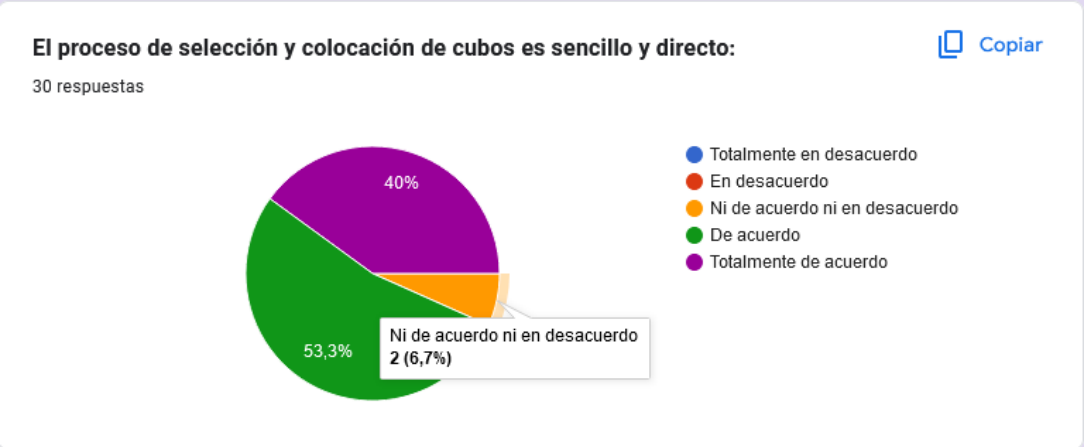

Figura 7.7: Gráfico que ilustra las respuestas de los usuarios sobre la facilidad de uso de la selección y colocación de los cubos

- **Claridad de la Interfaz:**

  - **Elementos Claros y Fáciles de Entender:** En la figura 7.8 se puede observar que el 53,3 % de los usuarios estuvo de acuerdo en que los elementos son claros, tanto los diferentes paneles como la pantalla principal donde se selecciona el puzzle y las diferentes alertas en las que se indica si el usuario ha completado o no el propio puzzle son claros, de la misma forma un 40 % estuvo totalmente de acuerdo, lo que refleja una buena claridad en el diseño de la interfaz.

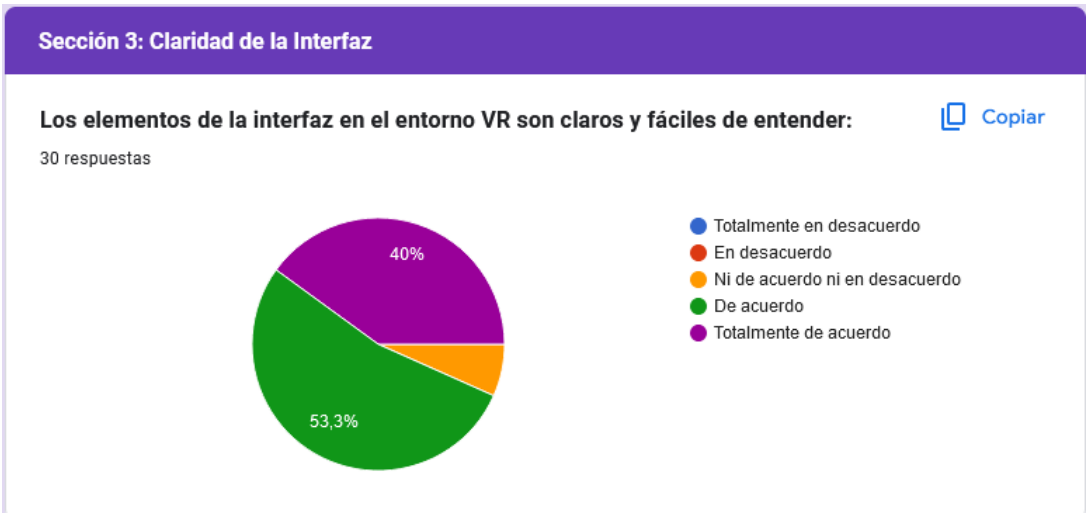

Figura 7.8: Gráfico que ilustra las respuestas de los usuarios sobre la claridad de los elementos de la interfaz

- **Instrucciones y Guías:** En este caso en la figura 7.9 se muestra que el 48,7 % estuvo de acuerdo de que con poco que sepan como moverse por la aplicación de VR pueden desenvolverse perfectamente en el entorno de VR, con un 16,7 % si opinaron que las instrucciones podrían ser más claras y que podría haber muchas más, de ahí al ni de acuerdo ni desacuerdo.

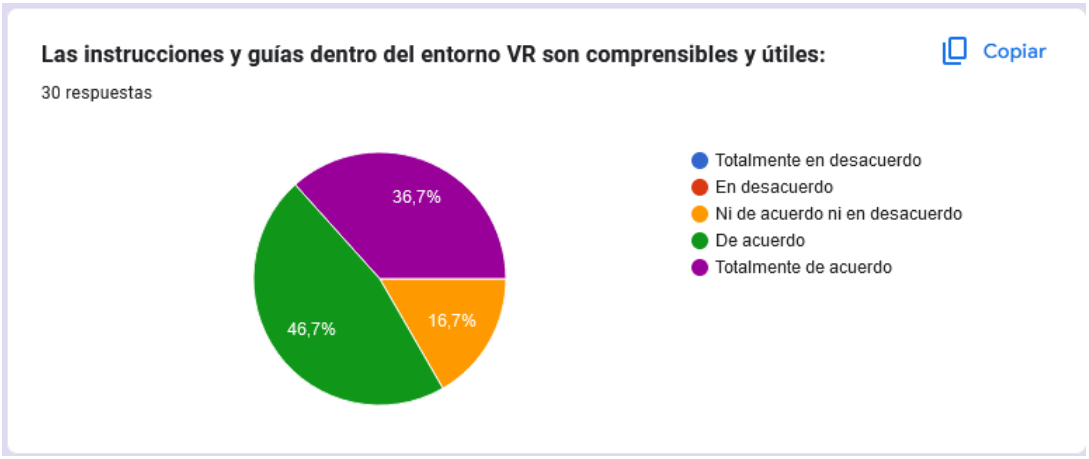

Figura 7.9: Gráfico que ilustra las respuestas de los usuarios sobre la claridad de las instrucciones y guías facilitadas

- **Eficiencia de la Interacción:**

  - **Resolución de Puzzles:** En cuanto a la figura 7.10, se puede ver según los resultados obtenidos que el 50 % estuvo de acuerdo en que pueden resolver puzzles de manera eficiente usando la aplicación, y de igual manera un 46,7 % coincidieron en los mismo. Por el contrario, el resto no estuvieron ni de acuerdo ni en desacuerdo, seguramente porque les costo un poco más ver entre todos los cubos que se generaban donde posicionar cada uno de ellos.

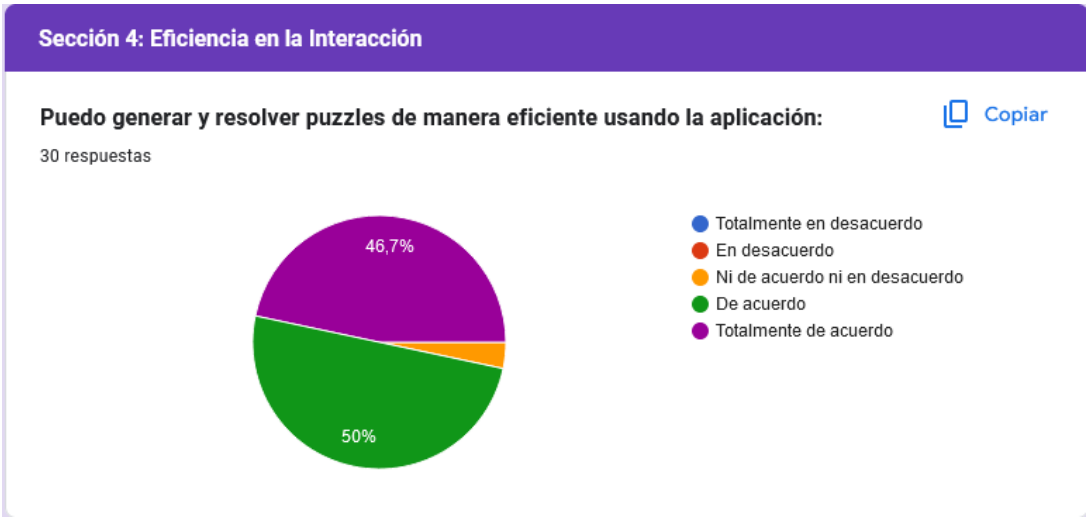

Figura 7.10: Gráfico que ilustra las respuestas de los usuarios sobre la resolución de puzzles en VR

- **Respuesta de la Aplicación:** El 56,7 % estuvieron de acuerdo en que la aplicación responde rápidamente a las interacciones, y un 33,3 %estuvo también de acuerdo con ello lo que indica que el rendimiento de la aplicación es adecuado y solo parece que el 10 % noto que no era tan rápido como se esperaba. (Figura 7.11)

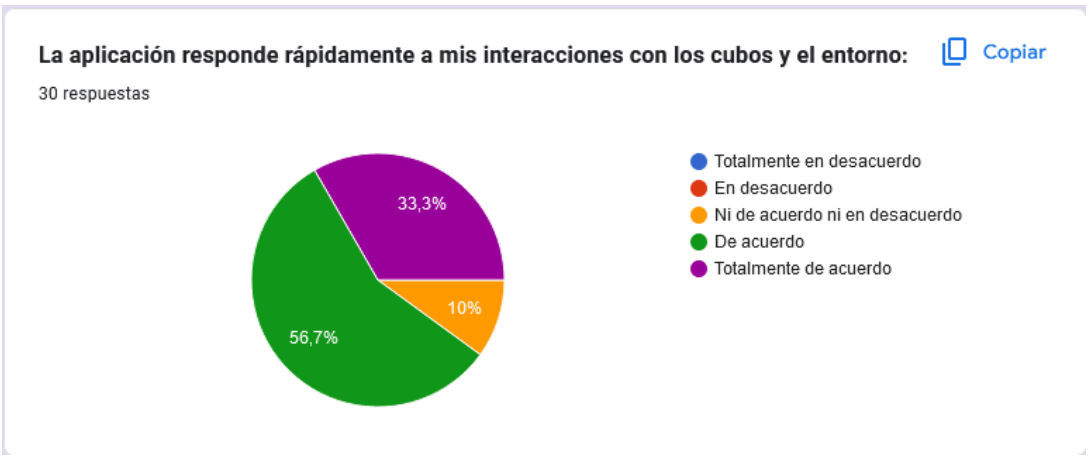

Figura 7.11: Gráfico que ilustra las respuestas de los usuarios sobre la respuesta de la aplicación

- **Problemas Técnicos:** En cuanto a los problemas técnicos (Figura 7.12), el 50 % de los usuarios indicaron no haber experimentado problemas técnicos significativos, mientras que un 30 % estuvo totalmente de acuerdo con esta afirmación, lo que sugiere una buena estabilidad en la aplicación. Solo el 20 % notó algún retraso, lo que debería investigarse y abordarse en futuras iteraciones del desarrollo.

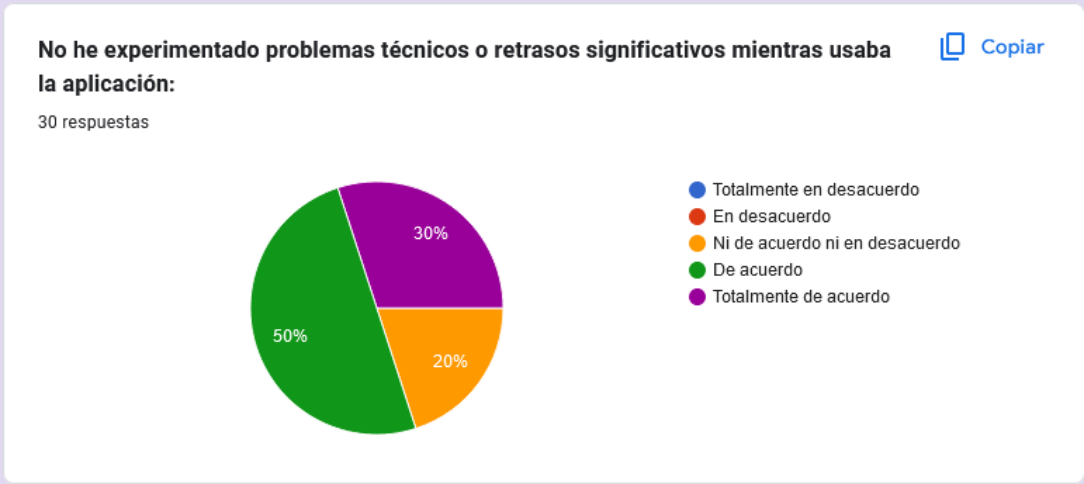

Figura 7.12: Gráfico que ilustra las respuestas de los usuarios sobre los problemas técnicos

- **Satisfacción General:**

  Como se observa en la figura 7.13, la satisfacción en general con la aplicación fue alta, más concretamente de entre los participantes un 66,7 % la recomendarían sin lugar a duda y lo mismo con el 33,3 %, estarían de acuerdo en recomendarla a otros. Este es un buen indicador del éxito del proyecto, aunque los comentarios abiertos proporcionaron sugerencias valiosas para seguir mejorando la aplicación.

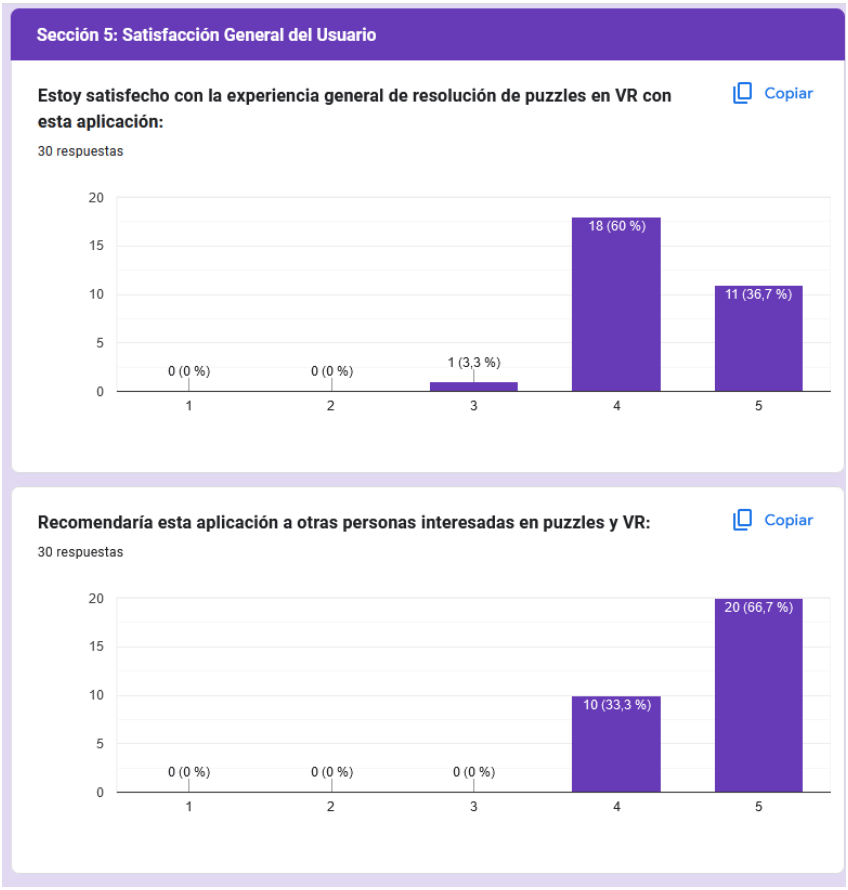

Figura 7.13: Gráfico que ilustra las respuestas de los usuarios sobre la satisfacción con MindCubeXR

- **Comentarios Abiertos:**

  Los comentarios proporcionados por los usuarios han sido cruciales para identificar áreas de mejora y refinar la experiencia de la aplicación. Las sugerencias sobre la inclusión de música, la mejora del rendimiento y la adición de más guías y tutoriales destacan como áreas prioritarias para futuras versiones. Asimismo, la alta valoración de la interactividad y la sensación de realismo en la manipulación de los cubos refuerza la efectividad del diseño actual, indicando que la aplicación logra su objetivo principal de proporcionar una experiencia inmersiva y personalizable en la resolución de puzzles en VR.

  - **Aspectos más Atractivos:** Los usuarios destacaron varios aspectos positivos de la aplicación que contribuyeron a una experiencia de juego inmersiva y satisfactoria:
    - **Interactividad y personalización:** Varios encuestados apreciaron la posibilidad de personalizar la dificultad de los puzzles mediante los sliders para elegir el número de filas y columnas. Comentaron que esta característica les permitió ajustar la experiencia a sus preferencias personales.
    - **Inmersión y realismo:** Muchos usuarios valoraron positivamente como podían interactuar con los cubos del puzzle, mencionando que parecía que estaban manipulando cubos reales. La sensación de inmersión se reforzaba por la respuesta física y visual de los cubos dentro del entorno VR.
    - **Claridad de la interfaz:** Algunos participantes destacaron la simplicidad y claridad de la interfaz, lo que facilitó su uso, especialmente para aquellos menos experimentados con la tecnología VR.
    - **Innovación en la experiencia de puzzle:** La generación dinámica de cubos y la capacidad de interactuar con ellos utilizando imanes fue vista como una característica innovadora y atractiva. Además, la retroalimentación inmediata al completar correctamente o incorrectamente un puzzle fue muy valorada.

    Algunos comentarios:
    - "La parte que más me ha gustado es que tú decides con los sliders el número de filas y columnas que deseas."
    - "Me ha gustado mucho la forma en la que se generan las piezas del puzzle y como el usuario puede cogerlas con la mano."
    - "Parecía que los cubos fueran reales mientras completaba el puzzle. Me ha gustado la experiencia."
  - **Sugerencias de Mejora** A pesar de la satisfacción general, los usuarios también propusieron algunas mejoras para optimizar la experiencia de juego:
    - **Incorporación de música y sonidos:** Varios participantes sugirieron añadir música de fondo o efectos sonoros para hacer la experiencia más envolvente y completa.
    - **Optimización del rendimiento:** Algunos usuarios notaron retrasos ocasionales y sugirieron mejoras en la velocidad de respuesta y rendimiento general de la aplicación.
    - **Más tutoriales y guías:** Para facilitar la adaptación, especialmente para usuarios novatos, se recomendó incluir más tutoriales interactivos y guías dentro del entorno VR.

- **Niveles de dificultad y temporizador:** Se sugirió que la aplicación ofreciera más opciones para seleccionar niveles de dificultad y la inclusión de un temporizador para añadir un reto adicional a la experiencia de juego.

Algunos comentarios:

- "Estaría bien que el usuario seleccionara el nivel de dificultad de otra manera, además de añadir música como complemento."
- "Experimenté algunos retrasos ocasionales, lo que afectó la fluidez de la experiencia."
- "Me gustaría que hubiera alguna especie de guía al principio para coger de manera más rápida las funcionalidades."

7.3.4. Limitaciones del Estudio

Es importante destacar que el tamaño de la muestra fue limitado y no incluyó un rango amplio de usuarios con diferentes niveles de experiencia en VR. Además, el uso de un participante con antecedentes médicos específicos (ictus) proporcionó una perspectiva útil, pero no puede considerarse representativa de toda la población con condiciones similares. Por lo tanto, se recomienda realizar pruebas adicionales con una muestra más grande y diversa para obtener resultados más generalizables.

CAPÍTULO 8
Conclusiones y Trabajos a Futuro

El desarrollo de **MindCubeXR** ha logrado cumplir con los objetivos fundamentales establecidos al inicio del proyecto, **proporcionando una plataforma inmersiva e interactiva** que combina realidad extendida (XR), incluyendo realidad virtual, aumentada y mixta. La aplicación ha demostrado ser eficaz en la creación de entornos virtuales personalizados y accesibles, facilitando tanto el desarrollo como la interacción en estos espacios digitales.

Durante el proyecto, las tecnologías avanzadas utilizadas han sido cruciales para lograr un entorno visualmente atractivo y técnicamente robusto. Un logro destacado ha sido la capacidad de ofrecer una interacción fluida y natural en un entorno virtual, lo que ha facilitado que los usuarios se involucren activamente en la resolución de puzzles. La satisfacción de los usuarios ha sido notable, ya que han valorado positivamente la facilidad de uso y la claridad de la interfaz. El entorno de realidad virtual ha proporcionado una experiencia novedosa que ha contribuido significativamente a la percepción positiva de la aplicación.

El aprendizaje obtenido durante este proyecto ha sido significativo tanto a nivel técnico como personal. La experiencia ha permitido consolidar conocimientos en el desarrollo de entornos XR y fortalecer habilidades en la resolución de problemas, como la verificación de la correcta posición de los cubos en el puzzle mediante la orientación y posición de imanes. MindCubeXR ha establecido una base sólida que no solo responde a las necesidades actuales del mercado, sino que también abre puertas a futuras innovaciones en el campo de la realidad extendida, especialmente en el ámbito de la salud.

En conclusión, el proyecto ha dado lugar a una aplicación sólida y prometedora, con un alto potencial para seguir evolucionando y adaptándose a las necesidades del mercado. La satisfacción obtenida hasta ahora respalda la viabilidad del producto y su capacidad para seguir creciendo y mejorando en futuras versiones.

8.1 Trabajos a Futuro

Se espera que la aplicación no se quede en un Trabajo de Final de Máster y siga evolucionando hacia nuevas versiones con funcionalidades mucho más refinadas. Con el propósito de que sea útil para la sociedad en un futuro, especialmente que sea útil para aquellas personas que requieren de rehabilitación por su diagnóstico médico. Algunas de las mejoras que se han visualizado de cara a futuro son las siguientes:

1. **Personalización de Tutoriales y Guías:** Incorporar la opción de personalizar tutoriales y guías que ayuden a los usuarios a desenvolverse dentro del entorno

de realidad virtual. Estos tutoriales podrían adaptarse al nivel de experiencia del usuario, facilitando una curva de aprendizaje más suave y eficiente.

2. **Integración de Música como Complemento:** Tras analizar los resultados de las encuestas, se podría añadir música de fondo para mejorar la inmersión y la experiencia general del usuario mientras interactúa con los puzzles. La música podría ser seleccionada o personalizada según las preferencias del usuario.

3. **Mejoras en la Interfaz de Usuario:** Continuar optimizando la interfaz, no solo en términos de usabilidad y estética, sino también introduciendo nuevas funcionalidades. Por ejemplo, un sistema de recompensas que motive a los usuarios a seguir avanzando o un contador que registre el tiempo empleado en resolver cada puzzle, permitiendo así que los usuarios puedan autoevaluar su rendimiento y progresión.

4. **Nuevas Funcionalidades y Retos:** Implementar nuevas funcionalidades que mantengan a los usuarios comprometidos, como la creación de desafíos temporales o puzzles exclusivos que puedan desbloquearse con puntos o logros dentro de la aplicación.

5. **Ampliación de la Interactividad:** Integrar más mecanismos de interacción avanzada, como el reconocimiento de voz o la integración con dispositivos adicionales de realidad aumentada, para enriquecer aún más la experiencia del usuario.

6. **Mejoras en la Usabilidad:** Optimizar la interfaz de usuario para hacerla aún más intuitiva, con un enfoque en la accesibilidad y la adaptación a diferentes perfiles de usuario.

7. **Escalabilidad y Rendimiento:** A medida que la aplicación crezca, será necesario optimizar su rendimiento y escalabilidad, asegurando que pueda manejar un mayor número de usuarios y escenarios más complejos sin comprometer la experiencia de usuario.

8. **Internacionalización y Personalización:** Implementar opciones de personalización más avanzadas y soporte para múltiples idiomas, permitiendo una mayor adopción global de la aplicación.

9. **Investigación y Desarrollo:** Explorar nuevas tecnologías emergentes en el campo de la realidad virtual y aumentada, con el objetivo de mantener la aplicación en la vanguardia tecnológica.

Bibliografía

[1] *(Neuro) Rehabilitación: Informe de daño cerebral.* Grupo 5 CIAN. (2022). [en linea]. Disponible en: https://www.grupo5.net/neuro-rehabilitacion-informe-dca-cerebral/ [Consultado el 5 de junio de 2024]

[2] *Faria, A. L., Latorre, J., Silva Cameirão, M., Bermúdez i Badia, S., & Llorens, R. (2023). Ecologically valid virtual reality-based technologies for assessment and rehabilitation of acquired brain injury: a systematic review.*Frontiers in Psychology. [en linea]. Disponible en: https://riunet.upv.es/handle/10251/204346 [Consultado el 5 de julio de 2024]

[3] *Rábago, C. A., & Wilken, J. M. (2011). Application of a mild traumatic brain injury rehabilitation program in a virtual reality environment: A case study.* Journal of Neurologic Physical Therapy, 35(4), 185-193. [en linea]. Disponible en: https://pubmed.ncbi.nlm.nih.gov/220274 [Consultado el 5 de julio de 2024]

[4] *Garber, B. G., Rusu, C., & Zamorski, M. A. (2014). Deployment-related mild traumatic brain injury, mental health problems, and post-concussive symptoms in Canadian armed forces personnel.* BMC Psychiatry, 14, 235. [en linea]. Disponible en: https://pubmed.ncbi.nlm.nih.gov/25410348/ [Consultado el 7 de julio de 2024]

[5] *Dra. Mar Grau Escriva (2023). Daño cerebral adquirido.* TOPDOCTORS. [en linea]. Disponible en: https://www.topdoctors.es/diccionario-medico/dano-cerebral-adquirido [Consultado el 7 de julio de 2023]

[6] *Goldman, L., Siddiqui, E. M., Khan, A., Jahan, S., Rehman, M. U., Mehan, S., Sharma, R., Budkin, S., Kumar, S. N., Sahu, A., Kumar, M., & Vaibhav, K. (2022). Understanding Acquired Brain Injury: A Review.* Biomedicines, 10(9), 2167. doi: 10.3390/biomedicines10092167. [en linea]. Disponible en: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9496189/ [Consultado el 7 de julio de 2024]

[7] *Logeswaran, A., Munsch, C., Chong, Y. J., Ralph, N., & McCrossnan, J. (2021). The role of extended reality technology in healthcare education: Towards a learner-centred approach.* Future Healthc J, 8(1), e79–e84. doi: 10.7861/fhj.2020-0112. [en linea]. Disponible en: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8004346/ [Consultado el 7 de julio de 2024]

[8] *Circuit Stream. (2023). Extended Reality (XR) in Healthcare: Top Use Cases for 2023 & Career Opportunities.* [en linea]. Disponible en: https://circuitstream.com [Consultado el 10 de julio de 2024]

[9] *Shepherd, M., Adapa, K., & van der Kruk, S. (2022). Patient education using extended reality technologies: A review.* Journal of Medical Education, 8(1), 101-117. doi: 10.1007/s10270-022-00604-8. [en linea]. Disponible en: https:

//journals.sagepub.com/doi/full/10.1177/00178969231198955 [Consultado el 10 de julio de 2024]

[10] *Mozo, J. F. (2024). Fundamentos de la rehabilitación cognitiva.* Blog de ISEP. Publicado el 11 de junio de 2024. [en linea]. Disponible en: https://www.isep.es/actualidad/fundamentos-de-la-rehabilitacion-cognitiva/ [Consultado el 11 de julio de 2024]

[11] *Kitago, T., & Krakauer, J. W. (2013). Motor learning principles for neurorehabilitation.* Handbook of Clinical Neurology, 110, 93-103. doi: 10.1016/B978-0-444-52901-5.00008-3. [en linea]. Disponible en: https://www.sciencedirect.com/ [Consultado el 11 de julio de 2024]

[12] *Virtual ON (2023). Historia de la Realidad Virtual.* Virtual ON. [en linea]. Disponible en: https://virtualon.es/historia-de-la-realidad-virtual/ [Consultado el 12 de julio de 2024]

[13] *Hoffman, H. G., et al. (2000). Use of virtual reality for adjunctive treatment of adolescent burn pain during wound care: A case report.* Pain. [en linea]. Disponible en: https://www.sciencedirect.com/science/article/pii/S0304395999002754 [Consultado el 12 de julio de 2024]

[14] *Onirix (2023). Realidad Extendida: Explorando el Futuro de las Tecnologías Inmersivas.* Onirix. [en linea]. Disponible en: https://www.onirix.com/es/realidad-extendida/ [Consultado el 13 de julio de 2024]

[15] *Mertz, L. (2019). Virtual Reality Is Taking the Hurt Out of Pain. IEEE Pulse*, May/-June 2019, pp. 3-8. [en linea]. Disponible en: https://ieeexplore.ieee.org/document/8720288 [Consultado el 13 de julio de 2024]

[16] *Borodkin, L. (2022). Evolution of Virtual and Augmented Reality Technologies in Historical and Archaeological Research.* IEEE, pp. 1-10. [en linea]. Disponible en: https://ieeexplore.ieee.org/document/10038138 [Consultado el 16 de julio de 2024]

[17] *Szabó, P., et al. (2023). Virtual Reality-based Application for Rehabilitation: Corsi-Test.* IEEE International Conference on Cognitive Aspects of Virtual Reality. [en linea]. Disponible en: https://ieeexplore.ieee.org/document/10395839 [Consultado el 16 de julio de 2024]

[18] *John, N. W., et al. (2019). Virtual Reality Environment for the Cognitive Rehabilitation of Stroke Patients.* IEEE. [en linea]. Disponible en: https://ieeexplore.ieee.org/document/8864513 [Consultado el 20 de julio de 2024]

[19] *Realidad Virtual y Salud: Apps móviles que están cambiando la rehabilitación.* Doonamis (2023). [en linea]. Disponible en: https://www.doonamis.com/realidad-virtual-salud-apps-moviles-rehabilitacion/ [Consultado el 20 de julio de 2024]

[20] *What is Unity? – A Top Game Engine for Video Games.* Zenva (2023). [en linea]. Disponible en: https://gamedevacademy.org/what-is-unity/ [Consultado el 22 de julio de 2024]

[21] *NUnit Tutorial: A Complete Guide With Examples and Best Practices.* Learning Hub. [en linea]. Disponible en: https://www.lambdatest.com/learning-hub/nunit-tutorial [Consultado el 22 de julio de 2024]

[22] *Getting Started with the Unity Test Runner*. Medium (2020). [en linea]. Disponible en: https://medium.com/xrpractices/getting-started-with-the-unity-test-runner-246ec95238bc [Consultado el 22 de julio de 2024]

[23] *Visual Studio 2022 es un entorno de desarrollo integrado (IDE) altamente utilizado para crear aplicaciones de software en múltiples plataformas, entre sus novedades y características destacan*. Danysoft (2022). [en linea]. Disponible en: https://www.danysoft.com/novedades-y-caracteristicas-destacadas-en-visual-studio-2022/ [Consultado el 22 de julio de 2024]

[24] *Oculus Quest 2, análisis: una de las mejores (y asequibles) opciones para iniciarse en la realidad virtual*. Xataka (2020). [en linea]. Disponible en: https://www.xataka.com/analisis/oculus-quest-2-analisis-caracteristicas-precio-especificaciones [Consultado el 22 de julio de 2024]

[25] *Virtual Reality Toolkit an Open source Framework | Introduction | Setting up environment*. Medium (2022). [en línea]. Disponible en: https://sumitkrsharma-ai.medium.com/virtual-reality-toolkit-an-open-source-framework-introduction-setting-up-environment-b11817cb8927 [Consultado el 22 de julio de 2024]

[26] *Meta Quest Link ahora con funciones, usabilidad y características de rendimiento mejoradas*. realovirtual (2024). [en línea]. Disponible en: https://www.realovirtual.com/noticias/13848/meta-quest-link-ahora-funciones-usabilidad-caracteristicas-rendimiento-mejoradas [Consultado el 23 de julio de 2024]

[27] *How To Create Spatial Interactions with Meta XR Interaction SDK*. MixedRealityNow (2024). [en línea]. Disponible en: https://mixedrealitynow.com/getting-started-with-meta-xr-interaction-sdk-quest-3-how-to-crucial-interactions [Consultado el 23 de julio de 2024]

[28] *Interaction SDK Overview*. Meta Quest (2024). [en línea]. Disponible en: https://developer.oculus.com/documentation/unity/unity-isdk-interaction-sdk-overview/ [Consultado el 23 de julio de 2024]

APÉNDICE A
Códigos utilizados

A.1 Scripts MindCubeXR

A.1.1. GameGenerator.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.XR.Interaction.Toolkit;
using TMPro;

public class GameGenerator : MonoBehaviour
{
    public GameObject cubePrefab;
    public GameObject magnetPrefab;
    public GameObject tableCenterObject;
    public List<Material> otherPuzzleMaterials;  // Lista de materiales genéricos para las otras caras
    public Image selectedImage; // Esta será la imagen seleccionada cuando el usuario haga clic
    public Transform imagesPanel;

    private List<Vector3> magnetPositions = new List<Vector3>(); // Lista de posiciones de los imanes
    private GridCreator gridCreator;

    public GameObject warningPanel;
    public GameObject successPanel;
    public TextMeshProUGUI successMessageText; // Para el panel de éxito
    public TextMeshProUGUI warningMessageText; // Para el panel de advertencia
    public Button playButton; // Botón de Play
    public Button continueButtonSuccess;
    public Button restartButtonSuccess;
    public Button continueButtonWarning;
    public Button restartButtonWarning;


    private bool puzzleCompleted = false; // Variable para controlar si el puzzle está completado

    public float cubeSize = 0.1f; // Tamaño del cubo, ajustar según sea necesario
    public float magnetHeightOffset = 0.005f; // Altura adicional para que solo la parte roja del imán sea visible

    public int rows;
```

```
37      public int columns;
38
39      void Start()
40      {
41          // Asegúrate de que el panel esté desactivado al inicio
42          warningPanel.SetActive(false);
43          successPanel.SetActive(false);
44
45          restartButtonSuccess.onClick.AddListener(RestartGame);
46          restartButtonWarning.onClick.AddListener(RestartGame);
47
48          continueButtonSuccess.onClick.AddListener(CloseMessagePanel);
49          continueButtonWarning.onClick.AddListener(CloseMessagePanel);
50
51          // Desactivar inicialmente los imanes y cubos
52          ClearCurrentMagnets();
53          ClearCurrentCubes();
54      }
55
56      void Update()
57      {
58          if (successPanel.activeSelf)
59          {
60              PositionPanel(successPanel);
61          }
62
63          if (warningPanel.activeSelf)
64          {
65              PositionPanel(warningPanel);
66          }
67      }
68
69      void RestartGame()
70      {
71          // Lógica para reiniciar el juego
72          CubeInteraction.cubesPlacedCorrectly = 0;
73          warningPanel.SetActive(false);
74          successPanel.SetActive(false);
75          puzzleCompleted = false;
76          ClearCurrentMagnets();
77          ClearCurrentCubes();
78          GenerateGame();
79      }
80
81      void CloseMessagePanel()
82      {
83          // Ocultar el panel de mensajes sin reiniciar el juego
84          if (warningPanel.activeSelf)  // Verifica si el WarningPanel está activo
85          {
86              warningPanel.SetActive(false);
87          }
88          else if (successPanel.activeSelf)  // Verifica si el SuccessPanel está
    activo
89          {
90              successPanel.SetActive(false);
91          }
92
93      }
94
95      public void CheckPuzzleCompletion()
96      {
97          if (IsPuzzleComplete())
98          {
```

```
99              puzzleCompleted = true;
100             ShowMessageSuccess("¿Bien hecho! Has completado el puzzle. ¿Quieres
    jugar de nuevo?", Color.white, true);
101         }
102         else
103         {
104             ShowMessageWarning("¿Inténtalo de nuevo!  No has completado el puzzle
    correctamente. ¿Quieres seguir intentándolo?\"", Color.red, false);
105         }
106     }
107
108     private bool IsPuzzleComplete()
109     {
110         GameObject[] cubes = GameObject.FindGameObjectsWithTag("cube");
111
112         foreach (GameObject cube in cubes)
113         {
114             string[] splitName = cube.name.Split('_');
115             if (splitName.Length < 3)
116             {
117                 Debug.LogError($"El nombre del cubo no tiene el formato esperado: {
    cube.name}");
118                 return false;
119             }
120
121             int row, col;
122             if (!int.TryParse(splitName[1], out row) || !int.TryParse(splitName[2],
     out col))
123             {
124                 Debug.LogError($"No se pudieron parsear los índices de la cuadrí
    cula desde el nombre del cubo: {cube.name}");
125                 return false;
126             }
127
128             Vector3 targetPosition = GetMagnetPosition(row, col);
129
130             // Verificación de alineación y orientación
131             if (Vector3.Distance(cube.transform.position, targetPosition) <= 0.05f
    &&
132                 Quaternion.Angle(cube.transform.rotation, Quaternion.identity) <=
    5.0f)
133             {
134                 cube.transform.position = targetPosition;
135                 cube.transform.rotation = Quaternion.identity;
136                 Debug.Log($"Cubo {cube.name} alineado correctamente.");
137             }
138             else
139             {
140                 Debug.Log($"Cubo {cube.name} no está alineado correctamente.");
141                 return false;
142             }
143         }
144
145         return true;
146     }
147
148
149     private void ShowMessageSuccess(string message, Color color, bool
    showRestartButtonSuccess)
150     {
151         successMessageText.text = message;
152         successMessageText.color = color;
153         successPanel.SetActive(true);
```

```
154         restartButtonSuccess.gameObject.SetActive(showRestartButtonSuccess);
155         continueButtonSuccess.gameObject.SetActive(!showRestartButtonSuccess);
156         PositionPanel(successPanel);
157     }
158 
159     private void ShowMessageWarning(string message, Color color, bool
    showRestartButtonWarning)
160     {
161         warningMessageText.text = message;
162         warningMessageText.color = color;
163         warningPanel.SetActive(true);
164         restartButtonWarning.gameObject.SetActive(showRestartButtonWarning);
165         continueButtonWarning.gameObject.SetActive(!showRestartButtonWarning);
166         PositionPanel(warningPanel);
167     }
168 
169 
170     void PositionPanel(GameObject panel)
171     {
172         Transform userTransform = Camera.main.transform;
173 
174         // Coloca el panel enfrente de la cámara, ligeramente hacia arriba
175         Vector3 positionOffset = userTransform.forward * 0.5f + userTransform.up *
    0.1f;  // 0.5f es la distancia frontal, 0.1f es la altura
176         panel.transform.position = userTransform.position + positionOffset;
177 
178         // Asegura que el panel esté mirando directamente hacia la cámara
179         panel.transform.rotation = Quaternion.LookRotation(userTransform.forward);
180 
181         // Ajustar el tamaño del canvas
182         panel.transform.localScale = new Vector3(0.01f, 0.01f, 0.01f);
183 
184     }
185 
186     void GenerateGame()
187     {
188         var grid = GameObject.FindGameObjectWithTag("Grid");
189         var curvedBackground = GameObject.FindGameObjectWithTag("curvedBackground")
    ;
190 
191         if (grid != null && curvedBackground != null)
192         {
193             gridCreator = grid.GetComponent<GridCreator>();
194             rows = gridCreator.rows;
195             columns = gridCreator.columns;
196             var curvedBackgroundImage = curvedBackground.GetComponent<Image>();
197             Sprite curvedBackgroundSprite = curvedBackgroundImage.sprite;
198             Texture2D curvedBackgroundTexture2D = SpriteToTexture2D(
    curvedBackgroundSprite);
199 
200             Material[] materials = DivideImageIntoMaterials(
    curvedBackgroundTexture2D, gridCreator.rows, gridCreator.columns);
201 
202             if (materials != null && materials.Length > 0)
203             {
204                 ClearCurrentMagnets(); // Limpiar los imanes actuales
205                 ClearCurrentCubes(); // Limpiar las piezas del puzzle actuales
206                 GenerateMagnetPositions(gridCreator.rows, gridCreator.columns); //
    Generar posiciones de los imanes
207 
208                 int magnetIndex = 0; // Inicializar índice de imanes
209 
210                 for (int r = 0; r < gridCreator.rows; r++)
```

```
211                 {
212                     for (int c = 0; c < gridCreator.columns; c++)
213                     {
214                         // Generar cubo en posición aleatoria
215                         Vector3 cubePosition = new Vector3(Random.Range(-0.5f, 0.5f
    ), 1.2f, Random.Range(0.2f, 0.6f));
216                         Quaternion cubeRotation = Quaternion.Euler(Random.Range(0f,
     90f), Random.Range(0f, 90f), Random.Range(0f, 90f));
217                         GameObject cube = Instantiate(cubePrefab, cubePosition,
    cubeRotation);
218 
219                         // Asignar el tag "cube" al cubo
220                         cube.tag = "cube";
221 
222                         // Obtener los renderers de las caras del cubo
223                         Renderer[] cubeRenderers = cube.GetComponentsInChildren<
    Renderer>();
224 
225                         if (cubeRenderers != null && cubeRenderers.Length > 0)
226                         {
227                             foreach (Renderer renderer in cubeRenderers)
228                             {
229                                 // Asignar el material de la imagen dividida a una
    cara específica del cubo
230                                 if (renderer.name == "Face1")  // Asumimos que la
    cara con nombre "Face1" es donde queremos la pieza del puzzle
231                                 {
232                                     renderer.material = materials[r * gridCreator.
    columns + c];
233                                 }
234                                 else
235                                 {
236                                     if (otherPuzzleMaterials != null &&
    otherPuzzleMaterials.Count > 0)
237                                     {
238                                         int randomIndex = Random.Range(0,
    otherPuzzleMaterials.Count);
239                                         renderer.material = otherPuzzleMaterials[
    randomIndex];
240                                     }
241                                     else
242                                     {
243                                         GenerateMaterialsFromPanelImages();
244                                         int randomIndex = Random.Range(0,
    otherPuzzleMaterials.Count);
245                                         renderer.material = otherPuzzleMaterials[
    randomIndex];
246                                     }
247                                 }
248                             }
249 
250                             // Configurar el nombre del objeto
251                             cube.name = $"Cube_{r}_{c}";
252 
253                             // Añadir el componente XRGrabInteractable y el script
    para detección de eventos de soltar cubo
254                             var interactable = cube.AddComponent<XRGrabInteractable
    >();
255                             cube.AddComponent<CubeInteraction>();
256                         }
257                         else
258                         {
```

```
259                             Debug.LogWarning("El objeto clonado no tiene
    componentes Renderer en las caras del cubo.");
260                             Destroy(cube);
261                         }
262
263                         // Instanciar imán en la posición predefinida
264                         var magnet = Instantiate(magnetPrefab, magnetPositions[
    magnetIndex], Quaternion.identity);
265                         magnet.tag = "refCube"; // Asegúrate de que el imán tenga
    el tag correcto
266
267                         magnetIndex++; // Mover al siguiente imán
268                     }
269                 }
270             }
271             else
272             {
273                 Debug.LogError("No se pudieron generar los materiales");
274             }
275         }
276         else
277         {
278             Debug.LogError("No se encontraron los objetos con los tags 'Grid' o '
    curvedBackground'.");
279         }
280     }
281
282     private Material[] DivideImageIntoMaterials(Texture2D image, int rows, int
    columns)
283     {
284         Material[] materials = new Material[rows * columns];
285         int partWidth = image.width / columns;
286         int partHeight = image.height / rows;
287
288         for (int r = 0; r < rows; r++)
289         {
290             for (int c = 0; c < columns; c++)
291             {
292                 Texture2D partTexture = new Texture2D(partWidth, partHeight);
293                 partTexture.SetPixels(image.GetPixels(c * partWidth, (rows - 1 - r)
     * partHeight, partWidth, partHeight));
294                 partTexture.Apply();
295
296                 Material material = new Material(Shader.Find("Unlit/Texture"));
297                 material.mainTexture = partTexture;
298                 materials[r * columns + c] = material;
299             }
300         }
301
302         return materials;
303     }
304
305     private Texture2D SpriteToTexture2D(Sprite sprite)
306     {
307         Texture2D texture = new Texture2D((int)sprite.rect.width, (int)sprite.rect.
    height);
308         texture.SetPixels(sprite.texture.GetPixels(
309             (int)sprite.rect.x,
310             (int)sprite.rect.y,
311             (int)sprite.rect.width,
312             (int)sprite.rect.height
313         ));
314         texture.Apply();
```

```
315
316         return texture;
317     }
318
319     void GenerateMagnetPositions(int gridRows, int gridColumns)
320     {
321         // Limpiar los imanes actuales antes de generar nuevas posiciones
322         ClearCurrentMagnets();
323
324         if (tableCenterObject == null)
325         {
326             Debug.LogError("No se ha asignado un objeto de referencia para el
    centro de la mesa.");
327             return;
328         }
329
330         Vector3 tableCenter = tableCenterObject.transform.position; // Obtener la
    posición del centro de la mesa
331
332         // Calcular el tamaño del puzzle en relación a las filas y columnas
333         float puzzleWidth = gridColumns * cubeSize;
334         float puzzleHeight = gridRows * cubeSize;
335
336         magnetPositions.Clear();
337
338         // Generar posiciones de imanes
339         for (int r = 0; r < gridRows; r++)
340         {
341             for (int c = 0; c < gridColumns; c++)
342             {
343                 // Calcular posición en relación con el centro de la mesa
344                 Vector3 magnetPosition = new Vector3(
345                     tableCenter.x - puzzleWidth / 2 + c * cubeSize * 0.8f,
346                     tableCenter.y + magnetHeightOffset, // Ajustar altura para que
    solo la parte roja del imán sea visible
347                     tableCenter.z - puzzleHeight / 2 + r * cubeSize * 0.8f
348                 );
349
350                 magnetPositions.Add(magnetPosition);
351
352                 // Crear el nuevo imán en la posición calculada
353                 if (magnetPrefab != null)
354                 {
355                     GameObject newMagnet = Instantiate(magnetPrefab, magnetPosition
    , Quaternion.identity);
356                     newMagnet.tag = "refCube"; // Asegúrate de que el nuevo imán
    tenga el tag correcto
357
358                     // Ajustar la escala del imán para que solo se vea la parte
    roja
359                     newMagnet.transform.localScale = new Vector3(newMagnet.
    transform.localScale.x, 0.01f, newMagnet.transform.localScale.z);
360                 }
361                 else
362                 {
363                     Debug.LogError("No se pudo cargar el prefab de imán.");
364                 }
365             }
366         }
367
368         Debug.Log($"Se generaron {magnetPositions.Count} posiciones de imanes.");
369     }
370
```

```
371     void ClearCurrentMagnets()
372     {
373         GameObject[] magnets = GameObject.FindGameObjectsWithTag("refCube");
374         foreach (GameObject magnet in magnets)
375         {
376             Destroy(magnet);
377         }
378     }
379 
380     void ClearCurrentCubes()
381     {
382         GameObject[] cubes = GameObject.FindGameObjectsWithTag("cube");
383         foreach (GameObject cube in cubes)
384         {
385             Destroy(cube);
386         }
387     }
388 
389     List<Vector3> GetCurrentMagnetPositions()
390     {
391         List<Vector3> positions = new List<Vector3>();
392         GameObject[] magnets = GameObject.FindGameObjectsWithTag("refCube");
393 
394         foreach (GameObject magnet in magnets)
395         {
396             positions.Add(magnet.transform.position);
397         }
398 
399         return positions;
400     }
401 
402     Vector3 GetMagnetPosition(int row, int col)
403     {
404         if (row * gridCreator.columns + col < magnetPositions.Count)
405         {
406             return magnetPositions[row * gridCreator.columns + col];
407         }
408         else
409         {
410             Debug.LogError("La posición del imán está fuera de los límites.");
411             return Vector3.zero;
412         }
413     }
414 
415     public void OnImageSelected(Image image)
416     {
417         selectedImage = image; // Establece la imagen seleccionada
418         otherPuzzleMaterials.Clear(); // Limpia la lista para regenerar los
    materiales
419         GenerateMaterialsFromPanelImages(); // Regenera la lista de materiales
    excluyendo la seleccionada
420     }
421 
422     void GenerateMaterialsFromPanelImages()
423     {
424         foreach (Transform child in imagesPanel)
425         {
426             // Busca el transform "Content" dentro de cada Toggle
427             Transform contentTransform = child.Find("Content");
428 
429             if (contentTransform != null)
430             {
431                 // Busca el transform "Background" dentro de "Content"
```

```
                Transform backgroundTransform = contentTransform.Find("Background")
    ;

                if (backgroundTransform != null)
                {
                    // Busca el componente Image dentro de "Background"
                    var img = backgroundTransform.GetComponent<Image>();

                    if (img != null && img.sprite != null && img.sprite !=
    selectedImage.sprite)
                    {
                        // Convertir el Sprite a Texture2D
                        Texture2D texture = SpriteToTexture2D(img.sprite);

                        // Dividir la imagen en partes según las filas y columnas
                        Material[] materials = DivideImageIntoMaterials(texture,
    rows, columns);

                        // Añadir todos los materiales generados a la lista
    otherPuzzleMaterials
                        otherPuzzleMaterials.AddRange(materials);
                        Debug.Log("Imagen añadida a otherPuzzleMaterials: " + img.
    sprite.name);
                    }
                    else
                    {
                        Debug.LogWarning("El Image en " + backgroundTransform.name
    + " no tiene un sprite asignado o es el sprite seleccionado.");
                    }
                }
                else
                {
                    Debug.LogWarning("No se encontró 'Background' en " + child.name
    );
                }
            }
            else
            {
                Debug.LogWarning("No se encontró 'Content' en " + child.name);
            }
        }

        Debug.Log("Total de materiales generados: " + otherPuzzleMaterials.Count);
    }

}
```

Listing A.1: GameGenerator.cs

Autores: *Ana Marí Vilaplana y Francisco Javier Jaén Martínez.*

A.1.2. CollisionDetector.cs

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

public class CollisionDetector : MonoBehaviour
{
```

```
7       bool released = true;
8       int iter = 0;
9       int iter_released = -int.MaxValue;
10      GameObject anchoredObject = null;
11      float x_anchored,y_anchored,z_anchored = 0;
12      Rigidbody rigidbody;
13
14      void OnTriggerEnter(Collider other)
15      {
16          //if (anchoredObject != null) return;
17          var objectcolliding = findParentWithTag("cube", other.gameObject);
18          print("other gameobject is: " + other.gameObject);
19          print("object colliding is: " + objectcolliding);
20          if (objectcolliding == null) return;
21          //if (released)
22          //{
23              foreach (var refCube in GameObject.FindGameObjectsWithTag("refCube"))
24                  if ((refCube != this.gameObject) && (refCube.GetComponent<
    CollisionDetector>().IsNotReleased(objectcolliding.name))) return;
25
26              iter = 0;
27              print("Collider game object is " + other.gameObject.name);
28
29              anchoredObject = objectcolliding;
30              if (anchoredObject == null) return;
31              rigidbody = anchoredObject.GetComponent<Rigidbody>();
32              //if (rigidbody.velocity==Vector3.zero)
33                  //rigidbody.isKinematic = true;
34              x_anchored = this.gameObject.transform.position.x;
35              y_anchored = anchoredObject.transform.position.y;
36              z_anchored = this.gameObject.transform.position.z;
37              anchoredObject.transform.position = new Vector3(x_anchored,y_anchored,
    z_anchored);
38
39              print("isKinematic changed to true");
40
41              released = false;
42              print("Another object has ENTERED the trigger and y= " + anchoredObject
    .transform.position.y);
43
44          //}
45
46
47      }
48      private void OnTriggerExit(Collider other)
49      {
50          if (anchoredObject == null) return;
51          if (anchoredObject != findParentWithTag("cube", other.gameObject)) return;
52          if (!released)
53          {
54              // this is to avoid the ill effects of OntriggerExit/enter called
    multiple times when isKinematic is false
55              if (iter<15) return;
56              //var rigidbody = anchoredObject.GetComponent<Rigidbody>();
57              //rigidbody.isKinematic = false;
58              //rigidbody.WakeUp();
59              print("Another object has EXITED the trigger, other object is:" + other
    .gameObject.name);
60              iter_released = iter;
61              print("isKinematic is = " + rigidbody.isKinematic+" .. "+rigidbody.
    IsSleeping());
62              print("anchored object is: " + anchoredObject.name);
63              anchoredObject = null;
```

```
64              //released = true;
65
66
67          }
68
69      }
70      // Start is called before the first frame update
71      void Start()
72      {
73
74      }
75
76      GameObject findParentWithTag(string tagToFind)
77      {
78          return findParentWithTag(tagToFind, this.gameObject);
79      }
80      GameObject findParentWithTag(string tagToFind, GameObject startingObject)
81      {
82
83          if (startingObject.tag.Equals(tagToFind))
84              return startingObject;
85          var parent = startingObject.transform.parent;
86          while (parent != null)
87          {
88              if (parent.tag == tagToFind)
89              {
90                  return parent.gameObject as GameObject;
91              }
92              parent = parent.transform.parent;
93          }
94          return null;
95      }
96      // Update is called once per frame
97      void Update()
98      {
99          iter++;
100
101         if (rigidbody != null) {
102
103             // needed because when releasing object by opening hand it is set by
    oculus SDK to kinematic
104             // and is not affected by Physics
105             if ((!released) && (rigidbody.isKinematic)&&(iter==iter_released+10)) {
106                 //rigidbody.isKinematic = false;
107                 iter_released = -int.MaxValue;
108                 released = true;
109                 print("Released!!!!!!!!!!!!!!!!!!!!!!!!!!!");
110             }
111             if ((released) && (rigidbody.isKinematic)) {
112                 rigidbody.isKinematic = false;
113                 anchoredObject = null;
114                 rigidbody = null;
115
116
117                 //rigidbody.velocity = Vector3.zero;
118             }
119
120         }
121         if (anchoredObject != null)
122         {
123
124             if ((!released) && (iter == 10)) {
125
```

```
126                 rigidbody.isKinematic = true;
127                 foreach (var gameobject in GameObject.FindGameObjectsWithTag("
    refCube"))
128                     gameobject.GetComponent<CollisionDetector>().releaseRigidBody(
    this.gameObject.name, rigidbody.gameObject.name);
129 
130             }
131             //print("anchored object is " + anchoredObject.ToString());
132             var eulerAngles = anchoredObject.transform.rotation.eulerAngles;
133             // This code is to automatically adjust to 0 the rotation of the cube
    around the axis that is up
134             // helps the user to have the cubes always well aligned on the table
135             //anchoredObject.transform.position = new Vector3(x_anchored,
    y_anchored, z_anchored);
136 
137             if (rigidbody.isKinematic) {
138 
139                 /*if (Mathf.Abs(Vector3.Dot(Vector3.up, anchoredObject.transform.up
    )) >0.9f)
140                     eulerAngles.y = 0;
141                 else if (Mathf.Abs(Vector3.Dot(Vector3.up, anchoredObject.transform
    .right)) >0.9f)
142                     eulerAngles.x = 0;
143                 else
144                     eulerAngles.z = 0;
145                 */
146                 print(eulerAngles.x + " ** " + eulerAngles.y + " ** " + eulerAngles
    .z);
147 
148                 // Calculates the minimum rotation for the cube to be properly
    alligned
149                 var dif0 = Mathf.Abs(0 - eulerAngles.y);
150                 var dif90= Mathf.Abs(90 - eulerAngles.y);
151                 var dif180 = Mathf.Abs(180 - eulerAngles.y);
152                 var dif270 = Mathf.Abs(270 - eulerAngles.y);
153                 var dif360= Mathf.Abs(360 - eulerAngles.y);
154                 var minY=Mathf.Min(dif0,dif90,dif180,dif270,dif360);
155 
156                 if (minY==dif0)
157                     eulerAngles.y = 0;
158                 else if (minY == dif90)
159                     eulerAngles.y = 90;
160                 else if (minY == dif180)
161                     eulerAngles.y = 180;
162                 else if (minY == dif270)
163                     eulerAngles.y = 270;
164                 else eulerAngles.y = 360;
165                 anchoredObject.transform.rotation = Quaternion.Euler(eulerAngles);
166 
167             }
168         }
169     }
170     public void releaseRigidBody(string commanderName,string gameObjectName)
171     {
172         if (this.gameObject.name.Equals(commanderName)) return;
173         if ((rigidbody!=null) && (rigidbody.gameObject.name.Equals(gameObjectName))
    ) {
174             rigidbody = null;
175             anchoredObject = null;
176             released = true;
177             print("Releasing. Commander= " + commanderName + ", gameObjectName= " +
     gameObjectName);
178         }
```

```
179     }
180     public bool IsNotReleased(string objectName)
181     {
182         if (rigidbody != null)
183             return ((!released) && (rigidbody.gameObject.name.Equals(objectName)));
184         return false;
185     }
186 }
```

Listing A.2: CollisionDetector.cs

Autor: *Francisco Javier Jaén Martínez.*

A.1.3. GridCreator.cs

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class GridCreator : MonoBehaviour
6  {
7      public int rows;
8      public int columns;
9      public GameObject gridCellPrefab;
10     private GameObject[,] gameGrid;
11     private const int PANEL_WIDTH = 800;
12     private const int PANEL_HEIGHT = 500;
13     private const int IMAGE_WIDTH = 725;
14     private const int IMAGE_HEIGHT = 725;
15     public UnityEngine.UI.Slider columnsSlider,rowsSlider;
16     private void createGrid()
17     {
18         var cell_x_size = PANEL_WIDTH / columns;
19         var cell_y_size = PANEL_HEIGHT / rows;
20         var x_scale = (float)cell_x_size / IMAGE_WIDTH;
21         var y_scale = (float)cell_y_size / IMAGE_HEIGHT;
22         var x_offset = (PANEL_WIDTH - cell_x_size) / 2;
23         var y_offset = (PANEL_HEIGHT - cell_y_size) / 2;
24
25         gameGrid = new GameObject[columns, rows];
26         for (int y = 0; y < rows; y++)
27             for (int x = 0; x < columns; x++)
28             {
29                 gameGrid[x, y] = Instantiate(gridCellPrefab, new Vector3(-x_offset
    + x * cell_x_size, -y_offset + y * cell_y_size), Quaternion.identity);
30                 gameGrid[x, y].transform.parent = transform;
31                 gameGrid[x, y].transform.localPosition = new Vector3(-x_offset + x
    * cell_x_size, -y_offset + y * cell_y_size);
32                 gameGrid[x, y].transform.localScale = new Vector3(x_scale, y_scale,
     1f);
33             }
34
35     }
36
37     public void OnColumnsSliderValueChange()
38     {
39         changeSize(rows, (int)columnsSlider.value);
40     }
41     public void OnRowsSliderValueChange()
```

```
42      {
43          changeSize((int)rowsSlider.value, columns);
44      }
45
46      // Start is called before the first frame update
47      void Start()
48      {
49          columnsSlider.onValueChanged.AddListener(delegate {
        OnColumnsSliderValueChange(); });
50          rowsSlider.onValueChanged.AddListener(delegate { OnRowsSliderValueChange();
         });
51          createGrid();
52      }
53
54      // Update is called once per frame
55      void Update()
56      {
57
58      }
59      public void changeSize(int newRows,int newColumns)
60      {
61          for (int y = 0; y < rows; y++)
62              for (int x = 0; x < columns; x++)
63                  Destroy(gameGrid[x, y]);
64          rows = newRows;
65          columns = newColumns;
66          createGrid();
67      }
68  }
```

Listing A.3: GridCreator.cs

Autor: *Francisco Javier Jaén Martínez.*

A.1.4. CubeInteraction.cs

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class CubeInteraction : MonoBehaviour
6   {
7       private GameGenerator gameGenerator;
8       public static int cubesPlacedCorrectly = 0; // Contador de cubos colocados
        correctamente
9       private static int totalCubes; // Número total de cubos a colocar
10
11      private bool isPlaced = false; // Variable para asegurarse de que solo se
        cuente una vez
12
13      void Start()
14      {
15          // Buscar y asignar el componente GameGenerator
16          gameGenerator = FindObjectOfType<GameGenerator>();
17
18          if (gameGenerator != null)
19          {
20              totalCubes = gameGenerator.rows * gameGenerator.columns;
21              cubesPlacedCorrectly = 0;
```

```
22          }
23          else
24          {
25              Debug.LogError("No se encontró el componente GameGenerator.");
26          }
27      }
28
29      void OnTriggerEnter(Collider other)
30      {
31          if (!isPlaced && other.CompareTag("refCube"))
32          {
33              Debug.Log($"Cubo {gameObject.name} colocado correctamente en {other.
    gameObject.name}");
34              // Incrementar el contador de cubos colocados correctamente
35              cubesPlacedCorrectly++;
36              isPlaced = true; // Marcar este cubo como colocado
37
38              // Solo verificar la completitud del puzzle si todos los cubos han sido
     colocados
39              if (cubesPlacedCorrectly == totalCubes && gameGenerator != null)
40              {
41                  Debug.Log("Puzzle completado. Verificando...");
42                  gameGenerator.CheckPuzzleCompletion();
43              }
44          }
45      }
46
47      void OnTriggerExit(Collider other)
48      {
49          if (isPlaced && other.CompareTag("refCube"))
50          {
51              // Decrementar el contador si el cubo es removido
52              cubesPlacedCorrectly--;
53              isPlaced = false;
54          }
55      }
56  }
```

Listing A.4: CubeInteraction.cs

Autora: *Ana Marí Vilaplana.*

A.1.5. ImagePanelController.cs

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.UI;
5
6   public class ImagePanelController : MonoBehaviour
7   {
8       public GameGenerator gameGenerator; // Referencia al script GameGenerator
9       // Start is called before the first frame update
10      void Start()
11      {
12          // Encuentra automáticamente el GameGenerator si no está asignado
13          if (gameGenerator == null)
14          {
15              gameGenerator = FindObjectOfType<GameGenerator>();
```

```
16          }
17      }
18
19      // Update is called once per frame
20      void Update()
21      {
22
23      }
24      public void SelectImage(bool active)
25      {
26          if (active) {
27              Image toggleButtonImage = this.gameObject.GetComponentInChildren<Image
        >();
28              GameObject curvedBackground = GameObject.FindGameObjectWithTag("
        curvedBackground");
29              curvedBackground.GetComponent<Image>().sprite = toggleButtonImage.
        sprite;
30
31              // Actualiza la imagen seleccionada en el GameGenerator
32              gameGenerator.OnImageSelected(toggleButtonImage);
33          }
34      }
35  }
```

Listing A.5: ImagePanelController.cs

Autora: *Ana Marí Vilaplana.*

A.2 Pruebas Unitarias

A.2.1. GameGeneratorTests.cs

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using NUnit.Framework;
4   using UnityEngine;
5   using UnityEngine.TestTools;
6   using TMPro;
7   using UnityEngine.UI;
8
9   public class GameGeneratorTests
10  {
11      private GameObject gameObject;
12      private GameGenerator gameGenerator;
13
14      [SetUp]
15      public void SetUp()
16      {
17          // Configurar el entorno de pruebas creando un nuevo GameObject
18          gameObject = new GameObject();
19          gameGenerator = gameObject.AddComponent<GameGenerator>();
20
21          // Inicializar los componentes necesarios
22          gameGenerator.warningPanel = new GameObject().AddComponent<GameObject>();
23          gameGenerator.successPanel = new GameObject().AddComponent<GameObject>();
24          gameGenerator.successMessageText = new GameObject().AddComponent<
        TextMeshProUGUI>();
25          gameGenerator.warningMessageText = new GameObject().AddComponent<
        TextMeshProUGUI>();
```

```
26          gameGenerator.playButton = new GameObject().AddComponent<Button>();
27          gameGenerator.continueButtonSuccess = new GameObject().AddComponent<Button
        >();
28          gameGenerator.restartButtonSuccess = new GameObject().AddComponent<Button
        >();
29          gameGenerator.continueButtonWarning = new GameObject().AddComponent<Button
        >();
30          gameGenerator.restartButtonWarning = new GameObject().AddComponent<Button
        >();
31
32          // Configuración básica
33          gameGenerator.rows = 2;
34          gameGenerator.columns = 2;
35      }
36
37      [Test]
38      public void CheckPuzzleCompletion_WhenPuzzleComplete_ShowsSuccessMessage()
39      {
40          // Simular que el puzzle está completo
41          gameGenerator.CheckPuzzleCompletion();
42
43          // Verificar que se muestra el mensaje de éxito
44          Assert.IsTrue(gameGenerator.successPanel.activeSelf);
45          Assert.AreEqual("¿Bien hecho! Has completado el puzzle. ¿Quieres jugar de
        nuevo?", gameGenerator.successMessageText.text);
46      }
47
48      [Test]
49      public void CheckPuzzleCompletion_WhenPuzzleIncomplete_ShowsWarningMessage()
50      {
51          // Simular que el puzzle no está completo
52          gameGenerator.CheckPuzzleCompletion();
53
54          // Verificar que se muestra el mensaje de advertencia
55          Assert.IsTrue(gameGenerator.warningPanel.activeSelf);
56          Assert.AreEqual("¿Inténtalo de nuevo!  No has completado el puzzle
        correctamente. ¿Quieres seguir intentándolo?\"", gameGenerator.
        warningMessageText.text);
57      }
58
59      [UnityTest]
60      public IEnumerator GenerateGame_CreatesCorrectNumberOfCubesAndMagnets()
61      {
62          // Ejecutar el método para generar el juego
63          gameGenerator.GenerateGame();
64
65          // Esperar un frame para asegurarse de que todo se ha creado
66          yield return null;
67
68          // Verificar que se han creado el número correcto de cubos y imanes
69          var cubes = GameObject.FindGameObjectsWithTag("cube");
70          var magnets = GameObject.FindGameObjectsWithTag("refCube");
71
72          Assert.AreEqual(gameGenerator.rows * gameGenerator.columns, cubes.Length);
73          Assert.AreEqual(gameGenerator.rows * gameGenerator.columns, magnets.Length)
        ;
74      }
75
76      [UnityTest]
77      public IEnumerator ClearCurrentCubes_RemovesAllCubesFromScene()
78      {
79          // Crear cubos para simular una escena con objetos
80          gameGenerator.GenerateGame();
```

```
81
82          // Esperar un frame
83          yield return null;
84
85          // Limpiar los cubos
86          gameGenerator.ClearCurrentCubes();
87
88          // Esperar un frame
89          yield return null;
90
91          // Verificar que no quedan cubos en la escena
92          var cubes = GameObject.FindGameObjectsWithTag("cube");
93          Assert.AreEqual(0, cubes.Length);
94      }
95
96      [UnityTest]
97      public IEnumerator ClearCurrentMagnets_RemovesAllMagnetsFromScene()
98      {
99          // Crear imanes para simular una escena con objetos
100         gameGenerator.GenerateGame();
101
102         // Esperar un frame
103         yield return null;
104
105         // Limpiar los imanes
106         gameGenerator.ClearCurrentMagnets();
107
108         // Esperar un frame
109         yield return null;
110
111         // Verificar que no quedan imanes en la escena
112         var magnets = GameObject.FindGameObjectsWithTag("refCube");
113         Assert.AreEqual(0, magnets.Length);
114     }
115
116     [UnityTest]
117     public IEnumerator GameGenerator_CreatesAndValidatesPuzzleCorrectly()
118     {
119         // Configurar el GameObject y añadir el script GameGenerator
120         var gameGenerator = new GameObject().AddComponent<GameGenerator>();
121         gameGenerator.rows = 3;
122         gameGenerator.columns = 3;
123
124         // Configurar otros componentes necesarios
125         gameGenerator.successPanel = new GameObject();
126         gameGenerator.successMessageText = gameGenerator.successPanel.AddComponent<
    TextMeshProUGUI>();
127         gameGenerator.successPanel.SetActive(false); // Desactivar al inicio
128
129         // Ejecutar la generación del puzzle
130         gameGenerator.GenerateGame();
131
132         // Esperar un frame para asegurar que todo se ha creado
133         yield return null;
134
135         // Verificar que se han creado el número correcto de cubos e imanes
136         var cubes = GameObject.FindGameObjectsWithTag("cube");
137         Assert.AreEqual(gameGenerator.rows * gameGenerator.columns, cubes.Length, "
    El número de cubos generados no es el correcto.");
138
139         // Simular la colocación correcta de los cubos y verificar la completitud
    del puzzle
140         gameGenerator.CheckPuzzleCompletion();
```

```
141         Assert.IsTrue(gameGenerator.successPanel.activeSelf, "El mensaje de éxito
    no se mostró al completar el puzzle.");
142     }
143
144     [TearDown]
145     public void TearDown()
146     {
147         // Limpiar después de cada prueba
148         Object.DestroyImmediate(gameObject);
149     }
150 }
```

Listing A.6: GameGeneratorTests.cs

Autora: *Ana Marí Vilaplana.*

A.2.2. CollisionDetectorTests.cs

```
1  using System.Collections;
2  using NUnit.Framework;
3  using UnityEngine;
4  using UnityEngine.TestTools;
5
6  public class CollisionDetectorTests
7  {
8      private GameObject testObject;
9      private CollisionDetector collisionDetector;
10     private GameObject cubeObject;
11     private Rigidbody cubeRigidbody;
12
13     [SetUp]
14     public void SetUp()
15     {
16         // Configurar el entorno de pruebas creando un nuevo GameObject
17         testObject = new GameObject("TestObject");
18         collisionDetector = testObject.AddComponent<CollisionDetector>();
19
20         // Crear un objeto cubo para las pruebas
21         cubeObject = new GameObject("Cube");
22         cubeObject.tag = "cube";
23         cubeRigidbody = cubeObject.AddComponent<Rigidbody>();
24     }
25
26     [TearDown]
27     public void TearDown()
28     {
29         // Limpiar después de cada prueba
30         Object.DestroyImmediate(testObject);
31         Object.DestroyImmediate(cubeObject);
32     }
33
34     [Test]
35     public void OnTriggerEnter_WhenObjectCollides_AnchorsObjectCorrectly()
36     {
37         // Configurar el objeto de prueba como si estuviera en una colisión
38         var otherObject = new GameObject("OtherObject");
39         var collider = otherObject.AddComponent<BoxCollider>();
40
41         // Simular la colisión
```

```
42          collisionDetector.OnTriggerEnter(collider);
43
44          // Validar que el objeto se ancla correctamente
45          Assert.IsNotNull(collisionDetector, "El objeto no se ancló correctamente.")
    ;
46      }
47
48      [Test]
49      public void OnTriggerExit_WhenObjectExits_UnanchorsObjectCorrectly()
50      {
51          // Configurar el objeto de prueba como si hubiera salido de una colisión
52          var otherObject = new GameObject("OtherObject");
53          var collider = otherObject.AddComponent<BoxCollider>();
54
55          // Simular la colisión
56          collisionDetector.OnTriggerEnter(collider);
57          collisionDetector.OnTriggerExit(collider);
58
59          // Validar que el objeto se desancla correctamente
60          Assert.IsNull(collisionDetector, "El objeto no se desancló correctamente.")
    ;
61      }
62
63      [Test]
64      public void FindParentWithTag_WhenCalled_ReturnsCorrectParent()
65      {
66          // Configurar la jerarquía de objetos
67          var parentObject = new GameObject("ParentObject");
68          parentObject.tag = "cube";
69          var childObject = new GameObject("ChildObject");
70          childObject.transform.parent = parentObject.transform;
71
72          // Ejecutar el método
73          var result = collisionDetector.findParentWithTag("cube", childObject);
74
75          // Validar que se encontró el padre correcto
76          Assert.AreEqual(parentObject, result, "No se encontró el padre correcto con
     el tag especificado.");
77      }
78
79      [UnityTest]
80      public IEnumerator Update_WhenObjectReleased_MakesRigidbodyNotKinematic()
81      {
82          // Configurar la simulación como si el objeto estuviera anclado
83          collisionDetector.Update();
84
85          // Simular el comportamiento
86          yield return null;
87
88          // Validar que el Rigidbody no es cinemático después de la liberación
89          Assert.IsFalse(cubeRigidbody.isKinematic, "El Rigidbody sigue siendo cinemá
    tico después de la liberación.");
90      }
91
92      [Test]
93      public void ReleaseRigidBody_WhenCalled_ResetsAnchoredObject()
94      {
95          // Configurar el objeto de prueba para que esté anclado
96          collisionDetector.releaseRigidBody("Commander", "Cube");
97
98          // Validar que el objeto se reseteó correctamente
99          Assert.IsNull(collisionDetector, "El objeto anclado no se reseteó
    correctamente.");
```

```
100     }
101
102     [Test]
103     public void IsNotReleased_WhenObjectIsAnchored_ReturnsFalse()
104     {
105         // Simular que el objeto está anclado
106         collisionDetector.releaseRigidBody("Commander", "Cube");
107
108         // Validar que IsNotReleased retorna false
109         Assert.IsFalse(collisionDetector.IsNotReleased("Cube"), "El método
    IsNotReleased no funcionó correctamente.");
110     }
111 }
```

Listing A.7: CollisionDetectorTests.cs

Autora: *Ana Marí Vilaplana.*

A.2.3. CubeInteractionTests.cs

```
1  using System.Collections;
2  using NUnit.Framework;
3  using UnityEngine;
4  using UnityEngine.TestTools;
5
6  public class CubeInteractionTests
7  {
8      private GameObject cubeObject;
9      private CubeInteraction cubeInteraction;
10     private GameObject gameGeneratorObject;
11     private GameGenerator gameGenerator;
12     private GameObject refCubeObject;
13
14     [SetUp]
15     public void SetUp()
16     {
17         // Configurar el entorno de pruebas creando un nuevo GameObject para el
    cubo y el GameGenerator
18         cubeObject = new GameObject("CubeObject");
19         cubeInteraction = cubeObject.AddComponent<CubeInteraction>();
20
21         gameGeneratorObject = new GameObject("GameGeneratorObject");
22         gameGenerator = gameGeneratorObject.AddComponent<GameGenerator>();
23
24         refCubeObject = new GameObject("RefCubeObject");
25         refCubeObject.tag = "refCube";
26
27         // Asignar filas y columnas
28         gameGenerator.rows = 3;
29         gameGenerator.columns = 3;
30
31         // Inicializar el GameGenerator en el CubeInteraction
32         cubeInteraction.gameGenerator = gameGenerator;
33
34         // Inicializar los valores estáticos
35         CubeInteraction.cubesPlacedCorrectly = 0;
36     }
37
38     [TearDown]
```

```
39      public void TearDown()
40      {
41          // Limpiar después de cada prueba
42          Object.DestroyImmediate(cubeObject);
43          Object.DestroyImmediate(gameGeneratorObject);
44          Object.DestroyImmediate(refCubeObject);
45      }
46
47      [Test]
48      public void Start_InitializesTotalCubesCorrectly()
49      {
50          // Verificar que totalCubes se inicializa correctamente
51          Assert.AreEqual(9, CubeInteraction.cubesPlacedCorrectly);
52      }
53
54      [Test]
55      public void OnTriggerEnter_WhenCorrectlyPlaced_IncrementsCubesPlacedCorrectly()
56      {
57          // Simular la activación del trigger
58          var collider = refCubeObject.AddComponent<BoxCollider>();
59          cubeInteraction.OnTriggerEnter(collider);
60
61          // Verificar que el contador de cubos colocados correctamente se incrementa
62          Assert.AreEqual(1, CubeInteraction.cubesPlacedCorrectly);
63          Assert.IsTrue(cubeInteraction.isPlaced);  // Verificar que el cubo se ha
    marcado como colocado
64      }
65
66      [Test]
67      public void OnTriggerEnter_WhenAllCubesPlaced_ChecksPuzzleCompletion()
68      {
69          // Simular que todos los cubos están colocados
70          CubeInteraction.cubesPlacedCorrectly = gameGenerator.rows * gameGenerator.
    columns - 1;
71
72          // Simular la activación del trigger para el último cubo
73          var collider = refCubeObject.AddComponent<BoxCollider>();
74          cubeInteraction.OnTriggerEnter(collider);
75
76          // Verificar que se ha llamado a la función CheckPuzzleCompletion del
    GameGenerator
77          Assert.AreEqual(9, CubeInteraction.cubesPlacedCorrectly);  // Ahora todos
    los cubos están colocados
78      }
79
80      [Test]
81      public void OnTriggerExit_WhenCubeRemoved_DecrementsCubesPlacedCorrectly()
82      {
83          // Simular la activación y luego la salida del trigger
84          var collider = refCubeObject.AddComponent<BoxCollider>();
85          cubeInteraction.OnTriggerEnter(collider);
86          cubeInteraction.OnTriggerExit(collider);
87
88          // Verificar que el contador de cubos colocados correctamente se decrementa
89          Assert.AreEqual(0, CubeInteraction.cubesPlacedCorrectly);
90          Assert.IsFalse(cubeInteraction.isPlaced);  // Verificar que el cubo se ha
    marcado como no colocado
91      }
92  }
```

Listing A.8: CubeInteractionTests.cs

Autora: *Ana Marí Vilaplana.*

A.2.4. GridCreatorTests.cs

```
using System.Collections;
using NUnit.Framework;
using UnityEngine;
using UnityEngine.TestTools;
using UnityEngine.UI;

public class GridCreatorTests
{
    private GameObject gridCreatorObject;
    private GridCreator gridCreator;
    private GameObject cellPrefab;

    [SetUp]
    public void SetUp()
    {
        // Crear un objeto de prueba para GridCreator
        gridCreatorObject = new GameObject("GridCreatorObject");
        gridCreator = gridCreatorObject.AddComponent<GridCreator>();

        // Crear un prefab para las celdas de la cuadrícula
        cellPrefab = new GameObject("GridCellPrefab");
        gridCreator.gridCellPrefab = cellPrefab;

        // Configurar sliders para las filas y columnas
        var sliderObject = new GameObject("Slider");
        gridCreator.columnsSlider = sliderObject.AddComponent<Slider>();
        gridCreator.rowsSlider = sliderObject.AddComponent<Slider>();

        // Inicializar filas y columnas
        gridCreator.rows = 3;
        gridCreator.columns = 3;
    }

    [TearDown]
    public void TearDown()
    {
        // Limpiar después de cada prueba
        Object.DestroyImmediate(gridCreatorObject);
        Object.DestroyImmediate(cellPrefab);
    }

    [Test]
    public void CreateGrid_CreatesCorrectNumberOfCells()
    {
        // Ejecutar la creación de la cuadrícula
        gridCreator.SendMessage("createGrid");

        // Verificar que se crearon el número correcto de celdas
        var cells = GameObject.FindGameObjectsWithTag("Untagged");
        Assert.AreEqual(gridCreator.rows * gridCreator.columns, cells.Length);
    }

    [Test]
    public void ChangeSize_UpdatesGridSizeCorrectly()
    {
        // Ejecutar la creación inicial de la cuadrícula
        gridCreator.SendMessage("createGrid");

        // Cambiar el tamaño de la cuadrícula
```

```
60          gridCreator.changeSize(2, 2);
61
62          // Verificar que se ha cambiado el tamaño correctamente
63          var cells = GameObject.FindGameObjectsWithTag("Untagged");
64          Assert.AreEqual(4, cells.Length);  // 2x2 = 4 celdas
65      }
66
67      [Test]
68      public IEnumerator OnColumnsSliderValueChange_ChangesColumnSize()
69      {
70          // Configurar el slider para cambiar el tamaño de las columnas
71          gridCreator.columnsSlider.value = 4;
72
73          // Ejecutar el método OnColumnsSliderValueChange
74          gridCreator.OnColumnsSliderValueChange();
75
76          // Esperar un frame para que la cuadrícula se actualice
77          yield return null;
78
79          // Verificar que el tamaño de la cuadrícula ha cambiado
80          var cells = GameObject.FindGameObjectsWithTag("Untagged");
81          Assert.AreEqual(4 * gridCreator.rows, cells.Length);
82      }
83
84      [Test]
85      public IEnumerator OnRowsSliderValueChange_ChangesRowSize()
86      {
87          // Configurar el slider para cambiar el tamaño de las filas
88          gridCreator.rowsSlider.value = 4;
89
90          // Ejecutar el método OnRowsSliderValueChange
91          gridCreator.OnRowsSliderValueChange();
92
93          // Esperar un frame para que la cuadrícula se actualice
94          yield return null;
95
96          // Verificar que el tamaño de la cuadrícula ha cambiado
97          var cells = GameObject.FindGameObjectsWithTag("Untagged");
98          Assert.AreEqual(4 * gridCreator.columns, cells.Length);
99      }
100 }
```

Listing A.9: GridCreatorTests.cs

Autora: *Ana Marí Vilaplana.*

A.2.5. ImagePanelController.cs

```
1  using System.Collections;
2  using NUnit.Framework;
3  using UnityEngine;
4  using UnityEngine.TestTools;
5  using UnityEngine.UI;
6
7  public class ImagePanelControllerTests
8  {
9      private GameObject imagePanelControllerObject;
10     private ImagePanelController imagePanelController;
11     private GameObject gameGeneratorObject;
```

```
12      private GameGenerator gameGenerator;
13
14      [SetUp]
15      public void SetUp()
16      {
17          // Crear un objeto de prueba para ImagePanelController
18          imagePanelControllerObject = new GameObject("ImagePanelControllerObject");
19          imagePanelController = imagePanelControllerObject.AddComponent<
    ImagePanelController>();
20
21          // Crear un objeto de prueba para GameGenerator
22          gameGeneratorObject = new GameObject("GameGeneratorObject");
23          gameGenerator = gameGeneratorObject.AddComponent<GameGenerator>();
24
25          // Asignar el GameGenerator al ImagePanelController
26          imagePanelController.gameGenerator = gameGenerator;
27      }
28
29      [TearDown]
30      public void TearDown()
31      {
32          // Limpiar después de cada prueba
33          Object.DestroyImmediate(imagePanelControllerObject);
34          Object.DestroyImmediate(gameGeneratorObject);
35      }
36
37      [Test]
38      public void Start_FindsGameGeneratorIfNull()
39      {
40          // Desasignar el GameGenerator para probar la asignación automática
41          imagePanelController.gameGenerator = null;
42
43          // Ejecutar Start
44          imagePanelController.Start();
45
46          // Verificar que el GameGenerator se ha encontrado y asignado automá
    ticamente
47          Assert.IsNotNull(imagePanelController.gameGenerator);
48      }
49
50      [Test]
51      public void SelectImage_UpdatesGameGeneratorWithSelectedImage()
52      {
53          // Crear un objeto de imagen de prueba
54          var imageObject = new GameObject("ImageObject");
55          var imageComponent = imageObject.AddComponent<Image>();
56          imageComponent.sprite = Sprite.Create(Texture2D.blackTexture, new Rect(0,
    0, 100, 100), Vector2.zero);
57
58          // Asignar la imagen al ImagePanelController
59          imagePanelControllerObject.AddComponent<Image>().sprite = imageComponent.
    sprite;
60
61          // Ejecutar SelectImage
62          imagePanelController.SelectImage(true);
63
64          // Verificar que la imagen seleccionada ha sido actualizada en el
    GameGenerator
65          Assert.AreEqual(imageComponent.sprite, imagePanelController.gameGenerator.
    selectedImage.sprite);
66
67          // Limpiar
68          Object.DestroyImmediate(imageObject);
```

```
69      }
70
71      [Test]
72      public IEnumerator SelectImage_ChangesCurvedBackgroundSprite()
73      {
74          // Crear un objeto de imagen de prueba
75          var imageObject = new GameObject("ImageObject");
76          var imageComponent = imageObject.AddComponent<Image>();
77          imageComponent.sprite = Sprite.Create(Texture2D.blackTexture, new Rect(0,
        0, 100, 100), Vector2.zero);
78
79          // Asignar la imagen al ImagePanelController
80          imagePanelControllerObject.AddComponent<Image>().sprite = imageComponent.
        sprite;
81
82          // Crear el fondo curvado y asignarle un sprite
83          var curvedBackgroundObject = new GameObject("CurvedBackground");
84          var curvedBackgroundImage = curvedBackgroundObject.AddComponent<Image>();
85          curvedBackgroundObject.tag = "curvedBackground";
86
87          // Ejecutar SelectImage
88          imagePanelController.SelectImage(true);
89
90          // Esperar un frame para que se actualice el sprite
91          yield return null;
92
93          // Verificar que el sprite del fondo curvado se ha actualizado
94          Assert.AreEqual(imageComponent.sprite, curvedBackgroundImage.sprite);
95
96          // Limpiar
97          Object.DestroyImmediate(imageObject);
98          Object.DestroyImmediate(curvedBackgroundObject);
99      }
100 }
```

Listing A.10: ImagePanelController.cs

Autora: *Ana Marí Vilaplana.*

A.3 Pruebas de Integración

A.3.1. GameGeneratorCollisionDetectorIntegrationTests.cs

```
1   using System.Collections;
2   using NUnit.Framework;
3   using UnityEngine;
4   using UnityEngine.TestTools;
5
6   public class GameGeneratorCollisionDetectorIntegrationTests
7   {
8       [UnityTest]
9       public IEnumerator
        GameGenerator_IntegratesWithCollisionDetector_AnchorsCubesCorrectly()
10      {
11          // Configurar el GameObject y añadir los componentes necesarios
12          var gameGenerator = new GameObject().AddComponent<GameGenerator>();
13          gameGenerator.rows = 2;
14          gameGenerator.columns = 2;
```

```
15          gameGenerator.GenerateGame();
16
17          yield return null; // Esperar un frame para que los cubos sean generados
18
19          // Simular la colocación de cubos en la posición de los imanes
20          foreach (var cube in GameObject.FindGameObjectsWithTag("cube"))
21          {
22              var collisionDetector = cube.AddComponent<CollisionDetector>();
23              cube.transform.position = gameGenerator.GetMagnetPosition(0, 0);
24              yield return null; // Esperar un frame para que las interacciones sean
    procesadas
25              Assert.IsNotNull(collisionDetector); // Verificar que el componente se
    añadió correctamente
26          }
27      }
28  }
```

Listing A.11: GameGeneratorCollisionDetectorIntegrationTests.cs

Autora: *Ana Marí Vilaplana.*

A.3.2. GameGeneratorCubeInteractionIntegrationTests.cs

```
1   using System.Collections;
2   using NUnit.Framework;
3   using UnityEngine;
4   using UnityEngine.TestTools;
5
6   public class GameGeneratorCubeInteractionIntegrationTests
7   {
8       [UnityTest]
9       public IEnumerator
        GameGenerator_IntegratesWithCubeInteraction_CorrectlyDetectsPuzzleCompletion()
10      {
11          var gameGenerator = new GameObject().AddComponent<GameGenerator>();
12          gameGenerator.rows = 2;
13          gameGenerator.columns = 2;
14          gameGenerator.successPanel = new GameObject();
15          gameGenerator.successMessageText = gameGenerator.successPanel.AddComponent<
    TMPro.TextMeshProUGUI>();
16          gameGenerator.successPanel.SetActive(false);
17
18          gameGenerator.GenerateGame();
19          yield return null; // Esperar un frame para que los cubos sean generados
20
21          foreach (var cube in GameObject.FindGameObjectsWithTag("cube"))
22          {
23              cube.transform.position = gameGenerator.GetMagnetPosition(0, 0); //
    Simular que el cubo se ha colocado correctamente
24              yield return null; // Esperar un frame para que las interacciones sean
    procesadas
25          }
26
27          gameGenerator.CheckPuzzleCompletion();
28          Assert.IsTrue(gameGenerator.successPanel.activeSelf, "El mensaje de éxito
    no se mostró al completar el puzzle.");
29      }
30  }
```

Listing A.12: GameGeneratorCubeInteractionIntegrationTests.cs

Autora: *Ana Marí Vilaplana.*

A.3.3. GridCreatorGameGeneratorIntegrationTests.cs

```
using System.Collections;
using NUnit.Framework;
using UnityEngine;
using UnityEngine.TestTools;

public class GridCreatorGameGeneratorIntegrationTests
{
    [UnityTest]
    public IEnumerator
    GridCreator_IntegratesWithGameGenerator_PositionsElementsCorrectly()
    {
        var gridCreator = new GameObject().AddComponent<GridCreator>();
        gridCreator.rows = 3;
        gridCreator.columns = 3;
        gridCreator.gridCellPrefab = new GameObject();
        gridCreator.createGrid();

        var gameGenerator = new GameObject().AddComponent<GameGenerator>();
        gameGenerator.rows = gridCreator.rows;
        gameGenerator.columns = gridCreator.columns;
        gameGenerator.GenerateGame();

        yield return null; // Esperar un frame para que los cubos e imanes sean
    generados

        var cubes = GameObject.FindGameObjectsWithTag("cube");
        Assert.AreEqual(gridCreator.rows * gridCreator.columns, cubes.Length, "El n
    úmero de cubos generados no es el correcto.");
        // Aquí se podría agregar la verificación de la posición de los cubos
    generados
    }
}
```

Listing A.13: GridCreatorGameGeneratorIntegrationTests.cs

Autora: *Ana Marí Vilaplana.*

A.3.4. ImagePanelControllerGameGeneratorIntegrationTests.cs

```
using System.Collections;
using NUnit.Framework;
using UnityEngine;
using UnityEngine.TestTools;

public class ImagePanelControllerGameGeneratorIntegrationTests
{
    [UnityTest]
```

```
9       public IEnumerator
        ImagePanelController_IntegratesWithGameGenerator_UpdatesCubeMaterials()
10      {
11          var gameGenerator = new GameObject().AddComponent<GameGenerator>();
12          gameGenerator.rows = 2;
13          gameGenerator.columns = 2;
14          gameGenerator.imagesPanel = new GameObject().transform;
15
16          var imagePanelController = new GameObject().AddComponent<
        ImagePanelController>();
17          imagePanelController.gameGenerator = gameGenerator;
18
19          // Simular la selección de una imagen
20          imagePanelController.SelectImage(true);
21          gameGenerator.GenerateGame();
22
23          yield return null; // Esperar un frame para que los cubos sean generados
24
25          // Verificar que los materiales de los cubos se actualizan correctamente
26          foreach (var cube in GameObject.FindGameObjectsWithTag("cube"))
27          {
28              // Aquí se podría agregar la verificación de que los materiales del
        cubo coincidan con la imagen seleccionada
29              Assert.NotNull(cube.GetComponent<Renderer>().material);
30          }
31      }
32  }
```

Listing A.14: ImagePanelControllerGameGeneratorIntegrationTests.cs

Autora: *Ana Marí Vilaplana.*

APÉNDICE B
Objetivos de Desarrollo Sostenible

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenible	**Alto**	**Medio**	**Bajo**	**N/A**
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**	X			
ODS 4. **Educación de calidad.**	X			
ODS 5. **Igualdad de género.**			X	
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**		X		
ODS 9. **Industria, innovación e infraestructuras.**	X			
ODS 10. **Reducción de las desigualdades.**		X		
ODS 11. **Ciudades y comunidades sostenibles.**			X	
ODS 12. **Producción y consumo responsables.**				X
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**			X	

Reflexión sobre la relación del TFM con los ODS y con el/los ODS más relacionados.

El proyecto MindCubeXR, diseñado para la rehabilitación cognitiva de personas con daño cerebral adquirido, tiene una relación variada con los **Objetivos de Desarrollo Sostenible (ODS)**, lo que refleja su impacto en diversas áreas sociales, económicas y tecnológicas. Tras un análisis exhaustivo, se han identificado una fuerte relación con los **ODS 3, 4, 8, 9 y 10**, cada uno de ellos con razones específicas que justifican su alta relevancia.

- **ODS 3: Salud y bienestar**. La relación con este objetivo es **alta**, ya que la aplicación se centra en mejorar la salud y el bienestar de los usuarios a través de una herramienta innovadora de rehabilitación cognitiva. Este enfoque contribuye directamente a la mejora de la calidad de vida y a la recuperación funcional de las personas con daño cerebral adquirido, cumpliendo así uno de los propósitos clave de este ODS.
- **ODS 4: Educación de calidad**. MindCubeXR también tiene una relación **alta** con este objetivo, debido a que ofrece un entorno de aprendizaje avanzado y personalizado que puede ser utilizado por terapeutas y pacientes. La aplicación facilita la adquisición de habilidades cognitivas a través de ejercicios interactivos, contribuyendo al acceso a una educación de calidad, especialmente para aquellos en procesos de rehabilitación.
- **ODS 8: Trabajo decente y crecimiento económico**. Este proyecto tiene una relación **media** con el ODS 8, ya que la rehabilitación efectiva de los pacientes podría facilitar su reintegración al mercado laboral, contribuyendo así al crecimiento económico. Además, la tecnología desarrollada puede impulsar la creación de empleo en los sectores tecnológico y sanitario, promoviendo un crecimiento económico sostenible.
- **ODS 9: Industria, innovación e infraestructuras**. MindCubeXR está fuertemente alineado con el ODS 9, con una relación **alta**, dado que promueve la innovación tecnológica en el campo de la salud mediante el uso de la realidad virtual. Este enfoque representa un avance en las infraestructuras de salud y demuestra como la tecnología emergente puede mejorar la atención y rehabilitación de los pacientes.
- **ODS 10: Reducción de las desigualdades**. Aunque la relación con el ODS 10 es de nivel **medio**, MindCubeXR puede contribuir a reducir las desigualdades al hacer que herramientas avanzadas de rehabilitación sean accesibles para una mayor parte de la población, independientemente de su condición socioeconómica. Esto podría ayudar a mitigar las disparidades en el acceso a servicios de salud de calidad.

Además de estos objetivos principales, MindCubeXR también muestra relaciones más tenues o indirectas con otros ODS. **ODS 5: Igualdad de género** tiene una relación **baja**, ya que, aunque la aplicación es accesible para ambos géneros, no aborda específicamente la igualdad de género. De manera similar, **ODS 11: Ciudades y comunidades sostenibles** tiene una relación **baja**, ya que el proyecto contribuye indirectamente a la creación de comunidades más inclusivas y accesibles, aunque no es su enfoque principal. Asimismo, **ODS 17: Alianzas para lograr objetivos** podría requerir colaboraciones entre diferentes instituciones, como universidades, centros

de investigación, hospitales y empresas tecnológicas, aunque esto no es el núcleo del proyecto.

Por último, otros ODS como **ODS 1 (Fin de la pobreza)**, **ODS 2 (Hambre cero)**, **ODS 6 (Agua limpia y saneamiento)**, **ODS 7 (Energía asequible y no contaminante)**, **ODS 12 (Producción y consumo responsables)**, **ODS 13 (Acción por el clima)**, **ODS 14 (Vida submarina)**, **ODS 15 (Vida de ecosistemas terrestres)** y **ODS 16 (Paz, justicia e instituciones sólidas)** se consideran no aplicables **(N/A)** debido a que el proyecto no tiene una relación directa con estos objetivos.