Grado en Ingeniería de Tecnologías de Telecomunicaciones

Escuela Politécnica Superior de Linares

Universidad de Jaén

Escuela Politécnica Superior de Linares

DESARROLLO DE APP ESCAPEROOM MEDIANTE REALIDAD VIRTUAL. INTERACCIÓN MUSICAL

Alumno: Mohamed Yassine Abcha

Tutor: Prof. D. Raúl Mata Campos

Depto.: Departamento de Ingeniería de Telecomunicaciones

Universidad de Jaén

Escuela Politécnica Superior de Linares

Grado en Ingeniería de Tecnologías de Telecomunicación

Trabajo Fin de Grado

DESARROLLO DE APP ESCAPEROOM MEDIANTE REALIDAD VIRTUAL. INTERACCIÓN MUSICAL

Alumno: Mohamed Yassine Abcha Tutor: Prof. D. Raúl Mata Campos

Julio, 2023

MATA CAMPOS RAUL - 26480125H

Firmado digitalmente por MATA CAMPOS RAUL - 26480125H
Fecha: 2023.06.20 17:55:58 +02'00'

Agradecimientos

A mi familia, a mis padres y a mis hermanos, que siempre me han apoyado en mis estudios y en cada una de las decisiones que he tomado. Os estaré eternamente agradecido.

A mis amigos, por tantas risas y por mostrarme que era capaz de conseguir todo lo que me propusiera.

A Raúl Mata Campos, tutor de este proyecto, que desde el primer día me ha motivado para realizar este proyecto con total libertad y me ha prestado su ayuda siempre que la he necesitado.

A los compañeros que he conocido en este viaje y a los profesores que me han motivado a seguir aprendiendo cada día y a no poner límites a la creatividad.

RESUMEN:

El presente trabajo de fin de grado tiene como objetivo describir el proceso de desarrollo de un juego multijugador en realidad virtual no inmersiva, el cual consiste en un escape-room con incorporación de interacción musical. El propósito principal de este proyecto es diseñar una experiencia de escape-room que involucre a los usuarios en una aventura interactiva, utilizando procesamiento de audio. Para lograr esto, se llevará a cabo la creación de un entorno virtual detallado y realista, así como la implementación de una mecánica de juego desafiante y la integración de elementos musicales que permitan a los usuarios interactuar con la música y el entorno.

La herramienta utilizada para el desarrollo de la aplicación es Unity, un motor de juego ampliamente utilizado. Se implementarán diversas características con el fin de lograr una experiencia de usuario única y entretenida. La aplicación estará diseñada de manera que los usuarios puedan resolver rompecabezas y encontrar pistas para escapar de una habitación virtual.

Este trabajo de fin de grado proporcionará un detallado relato del proceso de diseño de la aplicación antes mencionada, abarcando desde la fase inicial de planificación hasta la etapa de implementación y pruebas del juego.

ABSTRACT:

The aim of this Bachelor thesis is to describe the development process of a multiplayer non immersive virtual reality game, which consists of an escape room with musical interaction. The main purpose of this project is to design an escape room experience that engages users in an interactive adventure, using sound processing. To achieve this, a detailed and realistic virtual environment will be created, along with the implementation of challenging gameplay mechanics and the integration of musical elements that allow users to interact with the music and environment.

Unity, a widely used game engine, is the tool employed for developing the application. Various features will be implemented to deliver a unique and entertaining user experience. The application will be designed in a way that allows users to solve puzzles and find clues to escape from a virtual room.

This thesis will provide a detailed account of the aforementioned application's design process, covering everything from the initial planning phase to the implementation and testing stage of the game.

TABLA DE CONTENIDOS:

I. MOTIVACIÓN 15

1. Necesidad Inicial 15
2. Interés Personal 19

II. INTODUCCIÓN 20

III. OBJETIVOS 22

IV. IDEAS CLAVE 23

V. MARCO TEÓRICO 25

1. Realidad Virtual vs Realidad Aumentada 25
2. Estado del arte de la Realidad Virtual 26
3. Estado del arte de la Escape Room 41
   - 3.1. Ideas Básicas 41
   - 3.2. Sus Orígenes 41
   - 3.3. Estado del arte 42
4. Escape Room con la Realidad Virtual 46
5. Teoría Musical 47
   - 5.1. Notas Musicales 48
   - 5.2. Teclado del Piano 49
   - 5.3. Protocolo MIDI 50
   - 5.4. Efectos de Sonido 55
     - 5.4.1. Eco 55
     - 5.4.2. Reverberación 56
6. Motores de Videojuegos 58
   - 6.1. Historia de los motores de videojuegos 58
   - 6.2. Motores de juegos actuales 59
   - 6.3. UNITY 60
     - 6.3.1. Historia de Unity 61
     - 6.3.2. Unity en el mercado 63
     - 6.3.3. Formación de Unity 64

6.3.4. Tienda de Unity Assets 64
6.3.5. Utilidades de Unity 65
6.3.6. Licencias de Unity 66
6.4.Netcodes – Unity Multijugador.....66
6.4.1. Netcode 67
6.4.2. Mejores netcodes en el mercado 68
6.4.3. Como elegir el netcode 68
6.5.Chuck 69
6.6.Chuck con Unity - Chunity 69
VI. DESARROLLO 70
1. Herramientas y programas usados para el desarrollo de la aplicación 70
1.1.Unity 70
1.2.Visual Studio 70
1.3.Chuck.....71
1.4.C# 71
1.5.Photon Pun2.....71
1.6.MATLAB 71
2. Arquitectura del sistema.....72
3. Procedimiento 73
3.1.Instalación del Software 74
3.2.Preparación de los sonidos para el proyecto.....76
3.3.Creación del proyecto en Unity y configuración inicial.....77
3.4.Creación de las escenas y estructura de la aplicación 80
3.5.Photon Networking.....89
VII. PRUEBAS 92
1. Prueba del creador del juego.....92
2. Prueba de otra gente.....93
2.1.Prueba de un amigo de la carrera.....93
2.2.Prueba de un amigo de Túnez 93
VIII. RESULTADOS Y CONCLUSIÓN.....94

1. Resultados 94
2. Conclusión 96
3. Ampliaciones y mejoras 97
IX. BIBLIOGRAFÍA 98
ANEXO A: Código de la aplicación 104
ANEXO B: Hoja de Evaluación del juego 152
ANEXO C: AUTOEVALUACIÓN 155
ANEXO D: EVALUACIÓN 1 158
ANEXO E: EVALUACIÓN 2 161

TABLA DE FIGURAS:

Figura 1: Definición Curva de Gartner....16
Figura 2: Curva de Gartner 2010....17
Figura 3: Curva de Gartner 2013....17
Figura 4: Curva de Gartner 2016....18
Figura 5: Aplicación RV con interacción musical....20
Figura 6: RV vs RA....25
Figura 7: Sensorama....26
Figura 8: La espada de Damocles....28
Figura 9: Maquina de juego de RV de Virtuality....29
Figura 10: Gafas Sega VR....30
Figura 11: Mando Virtual Boy....30
Figura 12: Gafas Virtual Boy....31
Figura 13: Google Cardboard....32
Figura 14: Samsung Gear VR....33
Figura 15: PlayStation VR....33
Figura 16: PlayStation VR2....34
Figura 17: Oculus Rift....35
Figura 18: Kit HTC Vive....35
Figura 19: Oculus Quest 2....36
Figura 20: Proyecto Estandarización API RV....37
Figura 21: Eras de los dispositivos de interacción para ambientes virtuales....37
Figura 22: Línea de tiempo de las tecnologías descritas....38
Figura 23: Guante háptico de Senso Devices....39
Figura 24: Traje háptico de Senso Devices....40
Figura 25: Virtuix Omni, plataforma de RV....40
Figura 26: Escape Room Joypolis....43
Figura 27: Escape Room Parapark....43
Figura 28: ciudades con más escape romos en el mundo....44

Figura 29: Línea temporal del escape room 45
Figura 30: Escapistas en “La quest virtual COSMOS” 47
Figura 31: Valor de cada nota.....48
Figura 32: Pentagrama con notas musicales.....49
Figura 33: Teclado de un piano 49
Figura 34: Notas de las teclas del piano 50
Figura 35: Cifrado Americano de las teclas del piano.....51
Figura 36: Ejemplo del uso de MIDI.....51
Figura 37: Representación MIDI.....53
Figura 38: Relación MIDI-Frecuencia de notas 53
Figura 39: Frecuencia de las diferentes notas musicales.....54
Figura 40: Código MIDI de las notas musicales 54
Figura 41: Mensaje MIDI.....55
Figura 42: comandos usados en mensajes MIDI.....55
Figura 43: Diagrama de bloques de un sistema con eco 56
Figura 44: Ondas de sonido alcanzando el receptor por diferentes trayectos 56
Figura 45: Diagrama de Bloques de un sistema con reverberación 58
Figura 46: www.unity.com.....61
Figura 47: Primeras versiones de Unity 62
Figura 48: Cuota de mercado mundial de motores de juego 63
Figura 49: Soportes Unity 64
Figura 50: Unity Asset.....65
Figura 51: Utilidades de Unity 65
Figura 52: Licencias de Unity 66
Figura 53: Netcodes.....67
Figura 54: Comparativa de los mejores netcodes.....68
Figura 55: Como elegir la mejor solución.....68
Figura 56: Chuck 69
Figura 57: Chunity.....70
Figura 58: Arquitectura de la aplicación 73

Figura 59: Descarga de Unity Hub 74
Figura 60: Descarga de Unity 75
Figura 61: Descarga de Chuck 75
Figura 62: MATLAB 76
Figura 63: Creación del proyecto en Unity 77
Figura 64: Importar Pun2 77
Figura 65: Acceder a la configuración del servidor de Photon 78
Figura 66: PhotonServerSettings en Unity 78
Figura 67: Aplicación Pun 79
Figura 68: XR Interaction Toolkit 80
Figura 69: Intro Scene 81
Figura 70: Jerarquía de la escena 81
Figura 71: EscapeScene 83
Figura 72: Cube SCENE SWITCH PIANO 84
Figura 73: Piano Scene 84
Figura 74: Guitar Scene 85
Figura 75: Drums Scene 86
Figura 76: Point Light Animation 86
Figura 77: Notas de la escena “EscapeScene” 87
Figura 78: Secret Scene 88
Figura 79: Mapa del juego 88
Figura 80: Logs de Conexión 89
Figura 81: Log de un jugador nuevo 89
Figura 82: Network Player 90
Figura 83: Network Player Components 90
Figura 84: Log PhotonView 91
Figura 85: proceso del juego 92

I. MOTIVACION:

1. Necesidad inicial:

En la era actual, disponemos de tecnologías avanzadas que nos ofrecen numerosas facilidades y comodidades para mantenernos conectados las veinticuatro horas del día. No obstante, esta situación conlleva diversas distracciones que frecuentemente nos impiden apreciar los pequeños detalles de la vida y disfrutar del presente. Nos encontramos inmersos en un mundo disperso, en constante cambio y con acceso inmediato a todo. Si bien podemos observar nuestro entorno y lo que se nos muestra a través de múltiples plataformas a nuestro alcance, la realidad es que no logramos estar verdaderamente presentes en ninguna de estas experiencias. La gran cantidad de información y su constante dinamismo suelen dificultar nuestra capacidad de concentración. Las distracciones nos rodean en exceso y nos impiden prestar una atención plena a lo que estamos experimentando.

Un claro ejemplo de esto es algo tan sencillo como leer un libro. En la actualidad, resulta prácticamente imposible disfrutar plenamente del acto de sumergirse en una historia. Nos cuesta no tener la televisión encendida de fondo, no estar pendientes del teléfono móvil y evitar caer en las distracciones de las redes sociales. Este ejemplo puede aplicarse a cualquier actividad cotidiana.

La demanda de llevar a cabo múltiples tareas simultáneamente, también conocida como multitarea, puede tener un impacto negativo en nosotros debido a que nuestro cerebro cuenta con capacidades limitadas para procesar una determinada cantidad de información en un periodo de tiempo dado. Otro aspecto importante a considerar es la atención. Cuando nos enfrentamos a dos fuentes de información complejas, la eficiencia de una disminuye como resultado de la otra. Esto se menciona en el artículo de F. Manes titulado "¿Cómo afectan las nuevas tecnologías a nuestro cerebro?" [1].

Gracias a los avances tecnológicos, muchas tareas han sido simplificadas, lo que significa que las nuevas generaciones no necesitan aprenderlas. Esta realidad ha sido objeto de elogios por parte de muchos, pero también ha generado críticas y preocupaciones, como la controversia en torno a la posible eliminación de la caligrafía en Finlandia [2]. Como

estudiantes, nos resulta cada vez más difícil concentrarnos para estudiar y nos enfrentamos al desafío de prestar atención durante una clase completa. Si este problema continúa evolucionando, podría resultar en que las nuevas generaciones no estén tan preparadas como deberían. Este fenómeno actual se conoce como Modernidad Líquida, que se refiere a una vida caracterizada por la falta de un rumbo determinado, ya que, al ser líquida, no mantiene una forma definida durante mucho tiempo. Esto conlleva a que nuestras vidas estén marcadas por la precariedad y la incertidumbre. Nuestra principal preocupación se centra en no quedarnos rezagados ante los rápidos cambios que ocurren en nuestro entorno y en evitar quedar obsoletos [3]. ¿Es posible encontrar una solución a este problema?

En los últimos años, hemos presenciado un resurgimiento de la realidad virtual (RV) como una tecnología emergente, acompañada por la realidad aumentada (RA) debido a las similitudes en sus fundamentos tecnológicos. Este resurgimiento ha sido respaldado tanto por el ciclo de sobre expectación de Gartner [4] como por las noticias sobre las inversiones millonarias realizadas por grandes empresas, lo que sugiere la anticipación de grandes innovaciones.

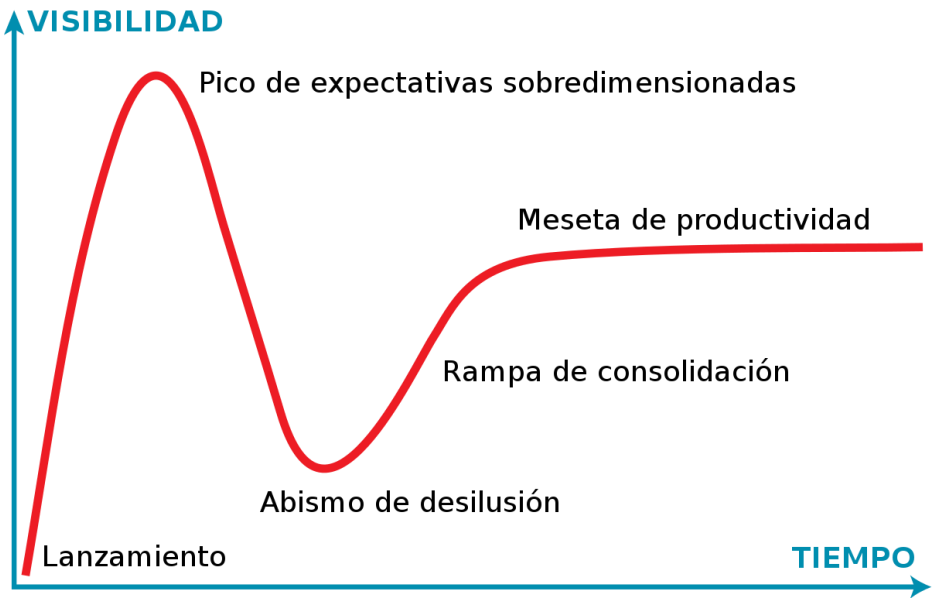


Figura 1: Definición Curva de Gartner

Fuente: www.wikipedia.com

Al analizar las figuras 2 y 3, se puede observar sorprendentemente que en el año 2010 la realidad virtual (RV) ni siquiera se consideraba como una tecnología emergente [5]. Sin embargo, para el año 2013, la RV ya figuraba en el ciclo, ubicada entre la etapa de "Abismo de desilusión" y la "Pendiente de iluminación", con una adopción de mercado inferior al 5% [6].

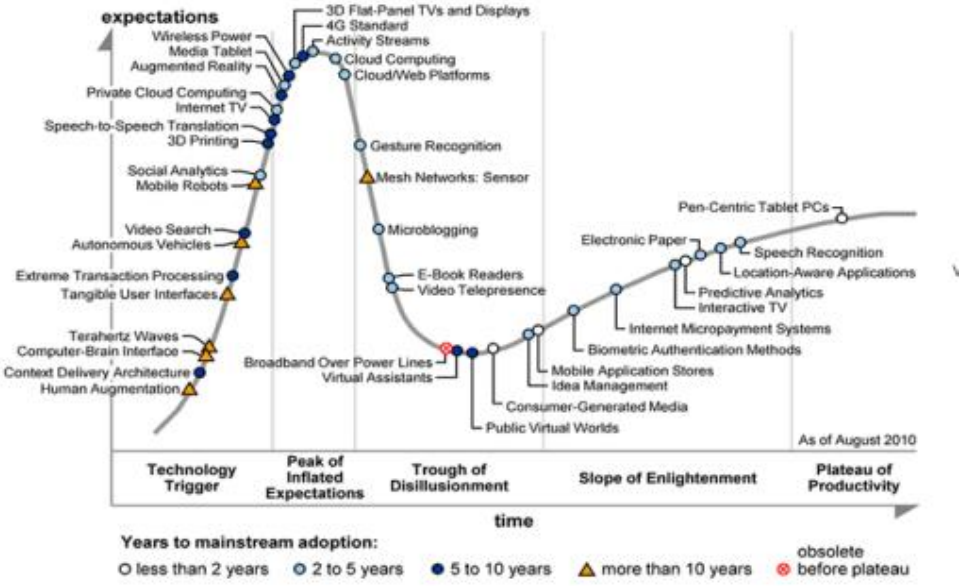


Figura 2: Curva de Gartner 2010

Fuente: www.virtualtravelog.net

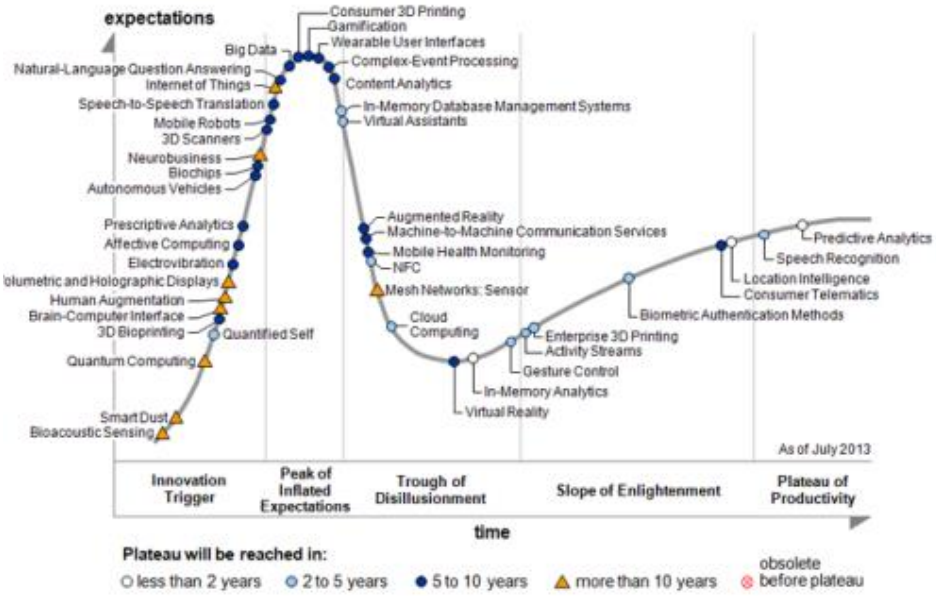


Figura 3: Curva de Gartner 2013

Fuente: www.virtualtravelog.net

En tan solo tres años, en julio de 2016, la realidad virtual (RV) se encuentra en plena fase de consolidación dentro del ciclo de Gartner, específicamente en la etapa de "La segunda generación de productos, algunos servicios". Este posicionamiento en el ciclo de Gartner demuestra claramente que la RV es una tecnología emergente y estamos presenciando actualmente diversas aplicaciones en campos como la salud, la construcción, el turismo, la educación, entre otros.

Considerando la evolución de la RV en el ciclo de Gartner, es evidente que estas dos tecnologías, la realidad virtual y la realidad aumentada, representan tanto el presente como el futuro. Ante esta situación, surge la pregunta: ¿hasta qué punto la RV y la RA pueden beneficiar a nuestra sociedad? ¿Es posible utilizar estas tecnologías para avanzar y resolver los problemas actuales, especialmente en el ámbito educativo?

Estas interrogantes plantean la posibilidad de aprovechar al máximo el potencial de la RV y la RA en el campo educativo, buscando así mejorar la forma en que se enseña y se aprende.

Figura 4: Curva de Gartner 2016

Fuente: www.gartner.com

2. Interés Personal:

Desde nuestra infancia en los años 2000, hemos sido testigos de la notable evolución tecnológica, especialmente en el ámbito de las consolas de videojuegos de renombradas compañías como Nintendo, Sega y Sony. Además, hemos presenciado el impresionante desarrollo de los dispositivos móviles, los cuales se están aprovechando de formas que en el pasado resultaban impensables. Durante este tiempo, hemos sido testigos del surgimiento de las tabletas, el crecimiento exponencial de Internet y la popularidad cada vez mayor de las televisiones inteligentes.

En la actualidad, nos encontramos en un momento en el que una nueva revolución tecnológica se está afianzando gradualmente en estos dispositivos. La realidad virtual y la realidad aumentada son tecnologías emergentes emocionantes en las que tenemos la oportunidad de participar e investigar su aplicabilidad en estos dispositivos. No solo estamos adquiriendo un conocimiento más amplio en este campo, sino que también estamos creando oportunidades profesionales para nosotros mismos.

II. INTRODUCCION:

Desde la década de los 90 hasta la actualidad, hemos presenciado un crecimiento exponencial en el desarrollo de dispositivos móviles y el acceso constante a Internet. Esto ha llevado a que nuestra sociedad esté profundamente inmersa en contenido multimedia, convirtiendo a los dispositivos móviles en elementos indispensables en nuestras vidas. Es evidente que cada vez se desarrollan más aplicaciones y estas se vuelven más complejas debido al aumento en la capacidad de procesamiento de los microprocesadores.

La aparición de Internet y, en particular, del contenido multimedia, ha generado un cambio significativo en la mentalidad de la sociedad en relación al uso de este tipo de contenido. La multimedia es una tecnología de comunicación digital que integra tanto el hardware como el software con el objetivo de humanizar las máquinas. Permite la integración de diversos medios a través de la computadora, fomentando la interacción con la misma. Este concepto está estrechamente relacionado con la realidad virtual.

Por otro lado, la realidad virtual (RV) es una tecnología que permite a los usuarios sumergirse en un entorno digital tridimensional y envolvente. La RV ha sido ampliamente aplicada en campos como la educación, el entretenimiento, la salud y la industria, demostrando ser una herramienta eficaz para mejorar la comprensión y la experiencia del usuario en estos ámbitos.

Figura 5: Aplicación RV con interacción musical

En la figura 5, se puede observar un ejemplo de una aplicación de realidad virtual con interacción musical. Para realizar dicha aplicación u otra similar, se necesitan varios elementos clave, entre los que se incluyen:

- Hardware: El hardware es uno de los componentes más importantes para la RV, ya que es el que permite la interacción del usuario con el entorno virtual. Los dispositivos más comunes son los cascos de RV, que pueden estar conectados a un ordenador o a un dispositivo móvil.
- Software: El software es el encargado de crear el entorno virtual, incluyendo los gráficos, la física y la interacción del usuario. Las herramientas de desarrollo de RV como Unity son ampliamente utilizadas para crear aplicaciones de RV.
- Contenido: El contenido es lo que hace que la aplicación de RV sea atractiva para los usuarios. Esto puede incluir entornos virtuales, modelos 3D, animaciones y sonidos.
- Interacción: La interacción es un elemento clave para la RV, ya que permite al usuario explorar y experimentar el entorno virtual de forma natural e intuitiva. Los controladores de RV y los sensores de movimiento son algunos de los elementos que se utilizan para lograr una interacción inmersiva.
- Ergonomía: La ergonomía es importante para garantizar una experiencia cómoda y segura para el usuario. Los cascos de RV deben ser ligeros, ajustables y permitir la ventilación adecuada. También se deben tener en cuenta factores como la fatiga visual y el mareo por movimiento.

En la actualidad existen numerosos programas de desarrollo de aplicaciones de realidad virtual. La gran ventaja de estos programas es que hay muchos en el mercado que son completamente gratuitos, lo cual se explicará con más detalle más adelante en la lista de Herramientas y Programas Usados.

III. OBJETIVOS:

El principal objetivo de este Trabajo Final de Grado es desarrollar una aplicación de realidad virtual (RV) que combine las estrategias de juego utilizadas en las Escape Room con la interacción musical. Para lograr esto, se han establecido los siguientes sub-objetivos:

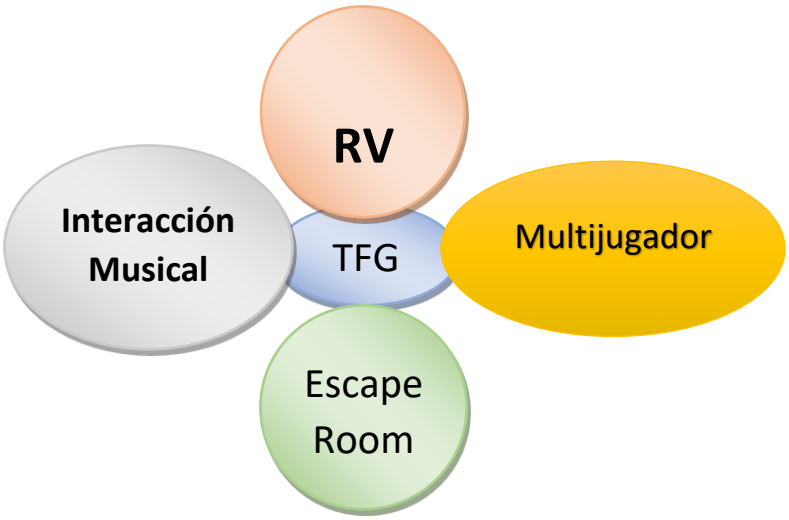
- Realizar una revisión bibliográfica y estudiar las técnicas de RV en general, así como las específicas para el desarrollo de aplicaciones de entretenimiento.
- Explorar los mecanismos que permiten incorporar la interacción musical en el juego.
- Investigar las aplicaciones existentes en el mercado que utilizan realidad aumentada (AR) en el contexto de las "escape room".
- Diseñar e implementar algoritmos, herramientas o librerías que formarán parte de la aplicación y cumplirán con los objetivos de análisis y procesamiento establecidos.
- Estudiar las herramientas de hardware y software necesarias para el desarrollo de la aplicación.
- Diseñar una actividad formativa basada en estas herramientas y su implementación.
- Realizar la implementación de la aplicación y verificar su correcto funcionamiento a través de pruebas y comprobaciones.
- Realizar una evaluación de la aplicación final mediante las experiencias de usuarios.

El cumplimiento de estos sub-objetivos permitirá alcanzar el objetivo principal

IV. Ideas Clave:

Este Trabajo Final de Grado se estructura en torno a tres elementos fundamentales que han sido claramente identificados y que, a su vez, constituyen los pilares del trabajo desarrollado.

El primer elemento es la tecnología de realidad virtual, que ha sido uno de los principales enfoques de investigación y desarrollo en este proyecto. El segundo elemento es la interacción musical, el cual ha sido abordado como un componente esencial en la creación de la aplicación. Por último, pero no menos importante, se encuentra el concepto de Escape Room.



A continuación, comentaremos brevemente estos tres pilares y el motivo por el cual es tan importante que estén juntos.

- **La tecnología de realidad virtual:** en respuesta a la evolución de los formatos multimedia, como imágenes, videos y animaciones interactivas, la realidad virtual representa un avance significativo tanto cualitativo, en términos de generar una sensación de presencia, como cuantitativo, al exigir estándares audiovisuales y ofrecer posibilidades de interacción. Estos avances anticipan un campo altamente prometedor para la innovación en la forma en que experimentamos la información y, por consiguiente, para el aprendizaje.

  Una cosa importante para mencionar, es que la realidad virtual utilizada en este TFG es la no inmersiva (RVNI). Consiste en una forma de experiencia virtual en

la que los usuarios interactúan con entornos y objetos virtuales utilizando interfaces y dispositivos de visualización menos inmersivos.

A diferencia de la realidad virtual inmersiva, que busca sumergir completamente al usuario en un entorno virtual tridimensional, la realidad virtual no inmersiva se basa en dispositivos más simples y accesibles, como pantallas de ordenador, tabletas o teléfonos móviles, para brindar una experiencia virtual.

- **La interacción musical** en la realidad virtual se refiere a la capacidad de los usuarios de interactuar con la música en un entorno virtual. Esto puede incluir la capacidad de crear música en tiempo real, interactuar con elementos visuales en respuesta a la música, o incluso sentir la música a través de la retroalimentación háptica. La interacción musical en la RV ha demostrado ser una forma emocionante y creativa de explorar la música y la tecnología en conjunto. Desde la creación de nuevas formas de composición musical hasta la creación de entornos "reales" de conciertos virtuales, la interacción musical en la RV ha abierto nuevas posibilidades para la experiencia musical y el aprendizaje en un contexto digital.
- **La Escape Room:** la Escape Room proporciona un escenario propicio para la experimentación en el ámbito del entretenimiento. Este elemento diferencial se convierte en el núcleo central de nuestro trabajo, permitiéndonos explorar nuevas posibilidades y enfoques en esta área.
- **El Multijugador:** Esta parte no estaba prevista en la aplicación, pero resulta sumamente interesante agregarla. El multijugador, siendo nuestro cuarto y último componente, se ha convertido en una esencia fundamental de los juegos en la actualidad, debido a que posibilita que las personas jueguen en conjunto con sus amigos. Esta parte conlleva la mayor complejidad en términos de implementación, ya que implica la gestión de redes y usuarios.

V. Marco Teórico:

1. Realidad Virtual vs Realidad Aumentada:

En el presente capítulo se procederá a explicar la diferenciación entre la tecnología de realidad aumentada y realidad virtual, las cuales con frecuencia son confundidas debido a su estrecha relación.

La realidad virtual (RV) se encarga de construir un mundo ficticio en el cual nos sumergimos completamente, mientras que la realidad aumentada (RA) hace uso de nuestro entorno real como base y, a través de la cámara y la pantalla de un dispositivo, nos permite visualizar elementos que no existen en la realidad y, además, interactuar con ellos. Por su parte, la realidad mixta (RM) representa un híbrido entre la RV y la RA, generando espacios novedosos en los cuales se fusionan objetos y personas reales con elementos virtuales.

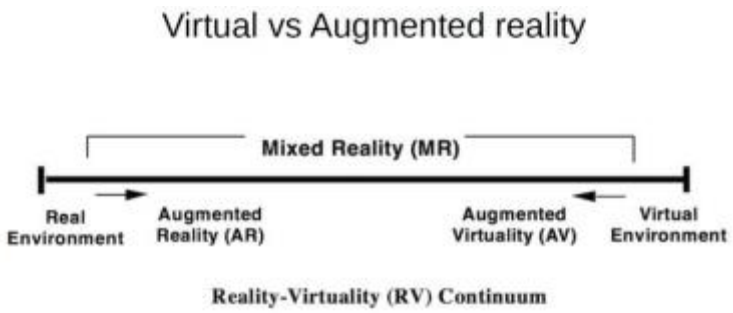


Figura 6: RV vs RA

Otra importante distinción entre estas tecnologías radica en los dispositivos necesarios para su uso. La realidad virtual y la realidad mixta requieren de dispositivos especiales proporcionados por el usuario, como gafas o un dispositivo específico. En cambio, para utilizar la realidad aumentada solo se necesita una aplicación instalada en un dispositivo móvil o tableta.

2. Estado del arte de la Realidad Virtual:

¿En qué fecha se originó la realidad virtual? La creencia común es que esta tecnología emergió en los últimos 15 o 20 años, pero esa afirmación es incorrecta. En realidad, el término "realidad virtual" se mencionó por primera vez en 1957 [8]. Por lo tanto, ¿por qué esta tecnología no tuvo éxito en ese momento? Uno de los principales motivos por los que la realidad virtual no prosperó en el pasado es la falta de capacidad computacional, incluyendo memoria, procesadores y tarjetas gráficas, que no eran tan potentes como las de hoy en día. Como resultado, los dispositivos de la época no podían transportarnos a una realidad paralela de manera convincente, y la involucración no era completa. A continuación, repasaremos los acontecimientos significativos de la realidad virtual, desde su concepción hasta nuestros días, con el fin de obtener una perspectiva general de su evolución.

En torno a los años 60, poco después de los primeros lanzamientos de televisión en color, se comenzaron a patentar los primeros dispositivos y cascos de realidad virtual, como el Sensorama en 1962 [9] y la "Telesphere Mask" en 1960 [10]. El principal problema era la falta de interactividad, debido a la imposibilidad en aquella época de generar gráficos 3D en tiempo real en una computadora.

En 1957, Morton Heilig, un cineasta, creó el Sensorama [11], una especie de gabinete teatral con estilo de sala de juegos públicos [12]. Este dispositivo estimulaba todos los sentidos, no solo la vista y el sonido, y contaba con altavoces estéreo, una pantalla estereoscópica en 3D, ventiladores, generadores de olores y una silla vibrante. El objetivo del Sensorama era sumergir completamente al usuario en la película. Además, Heilig produjo seis cortometrajes de dos minutos cada uno, que filmó, produjo y editó personalmente.

Morton Heilig presentó su siguiente invento, la máscara de Telesphere [14], que fue el primer dispositivo que permitía montar una pantalla en la cabeza. Aunque no era interactivo y no seguía los movimientos del usuario, este dispositivo permitía visualizar diapositivas en 3D con una amplia y estereoscópica visión y sonido estéreo.

Figura 7: Sensorama [13]

En 1961, dos ingenieros de la empresa Philco Corporation desarrollaron el primer precursor de los dispositivos con pantallas "montadas" en la cabeza [15], conocido hoy como HMD (Head-mounted Display) [16]: el Headsight [17]. Este dispositivo incorporaba una pantalla de video para cada ojo y un sistema de seguimiento de movimiento magnético, que se vinculaba a una cámara de circuito cerrado. Aunque el Headsight no se diseñó específicamente para aplicaciones de realidad virtual (ya que el término aún no existía), se desarrolló para permitir la visualización remota e inmersiva de situaciones peligrosas por parte de los militares. Los movimientos de la cabeza permitían que una cámara remota se moviera, permitiendo al usuario mirar naturalmente alrededor del ambiente. El Headsight fue el primer paso en la evolución de la pantalla en realidad virtual montada en la cabeza, aunque aún faltaba la integración de la computadora y la generación de imágenes.

En 1965, un artículo de Ivan Sutherland [18] presentó el concepto de "Ultimate Display" [19], el cual permitiría realizar simulaciones tan realistas que serían indistinguibles de la realidad. Este concepto incluía un mundo virtual visto a través de un dispositivo HMD, con sonido 3D y retroalimentación táctil para lograr una experiencia realista, así como ordenadores con hardware capaz de recrear escenas virtuales en tiempo real. También se

destacaba la importancia de la interacción realista de los usuarios con los objetos en el mundo virtual. Estos requisitos no difieren demasiado de los que presentan los dispositivos de realidad virtual actuales, y el artículo ha servido como un modelo básico para los conceptos de realidad virtual de la actualidad.

En la década de 1960, I. Sutherland creó el primer programa para diseñar mundos virtuales con imágenes en 3D, el cual incluía el primer sistema de realidad aumentada llamado The Ultimate Display [20]. Este sistema consistía en una pantalla montada sobre la cabeza que permitía a los usuarios ver un paisaje real con imágenes gráficas superpuestas, conectado a un ordenador que procesaba un ciclo de instrucción cada dos milisegundos y tenía una memoria de 4 KBytes. Esto significa que el ordenador de aquel entonces tenía una capacidad de 500 FLOPS (operaciones de punto flotante por segundo), mientras que hoy en día estamos en el rango de los GFLOPS (10^9 FLOPS), y en cuanto a la memoria, actualmente nos encontramos en los GBytes [21].

Además, en 1968, Ivan Sutherland marcó otro hito en la historia de la realidad virtual al crear, junto con su alumno Bob Sproull, el primer sistema de visualización de realidad virtual y aumentada (HMD). Aunque este sistema era primitivo tanto en términos de interfaz de usuario como de realismo, el HMD que debía llevar el usuario era tan pesado que debía suspenderse del techo. Los gráficos que componían el entorno virtual eran simples modelos de alambre. El dispositivo era impresionante y recibió el nombre de La espada de Damocles. [22]

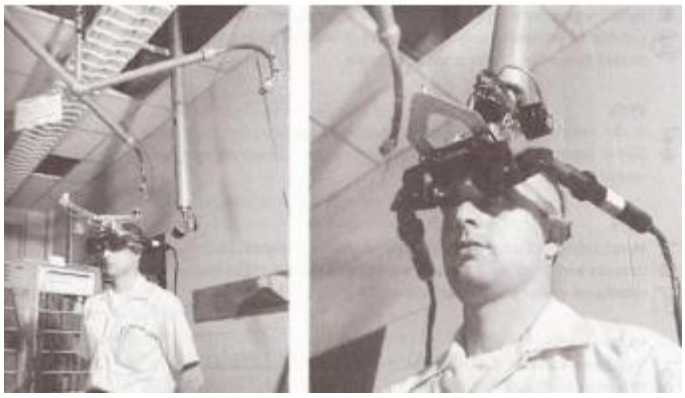


Figura 8: La espada de Damocles

Durante los años 70, se produjeron importantes avances algorítmicos en los gráficos de los ordenadores, lo que permitió que la siguiente generación de computadoras pudiera mostrar gráficos 3D interactivos más realistas. Sin embargo, hasta los años 80, estos gráficos no podían generarse en tiempo real debido a que los ordenadores no tenían un rendimiento suficiente para hacerlo.

A pesar de todos estos avances en la realidad virtual, sorprendentemente, aún no había un término para describir el campo. Fue en 1987 cuando Jaron Lanier [23], fundador del laboratorio de programación visual (VPL) [24], acuñó el término "realidad virtual". Finalmente, el área de investigación tenía un nombre. A través de su empresa de investigación en VPL, J. Lanier desarrolló una variedad de tecnologías de realidad virtual, como el Dataglove [25] (en colaboración con Tom Zimmerman) y la pantalla EyePhone [26] montada en la cabeza.

Figura 9: Maquina de juego de RV de Virtuality

En 1991, por primera vez, comenzaron a aparecer dispositivos de realidad virtual a los que el público podía acceder, aunque aún quedaba mucho camino por recorrer para llegar a lo que conocemos hoy en día como realidad virtual. El Grupo Virtuality [27] lanzó una serie de juegos y máquinas que incluían las primeras gafas de realidad virtual para jugar en máquinas de juego con imágenes 3D estereoscópicas inmersivas en tiempo real, con una latencia inferior a 50 ms. Algunas de estas unidades también se conectaron en red para ofrecer una experiencia de juego multijugador. [23]

En 1992, la Universidad de Illinois desarrolló el espacio CAVE (CAVE Automatic Virtual Environment) [28], una habitación en forma de cubo en la que se proyectan imágenes en las paredes, el suelo y el techo.

En 1993, Sega anunció las Sega VR [52] para la consola Sega Genesis. Las gafas de prototipo wrap-around incluían seguimiento de la cabeza, sonido estéreo y pantallas LCD en la visera. La idea principal de la empresa era lanzar el producto a un precio de alrededor de $200 en ese momento, o alrededor de $322 en 2015. Sin embargo, la empresa tuvo dificultades técnicas en el desarrollo que significaron que el dispositivo se quedaría en la fase de prototipo y nunca llegó al mercado, a pesar de haber desarrollado ya cuatro juegos para este producto. Este fue un gran fracaso para Sega.

Figura 10: Gafas Sega VR [29]

En 1995, Nintendo anunció la primera consola portátil que podía mostrar gráficos 3D realistas, la Nintendo Virtual Boy [30] (originalmente conocida como VR-32). A pesar de las expectativas de revolucionar el mercado de las consolas, el lanzamiento fue un fracaso comercial tanto en Japón como en América del Norte, incluso después de reducir los precios. Las razones detrás de esto fueron la falta de color en los gráficos, ya que los juegos solo estaban en rojo y negro, la falta de soporte de software y la dificultad de usar la consola en una posición cómoda.

Figura 11: Mando Virtual Boy

Figura 12: Gafas Virtual Boy[31]

Durante el período comprendido entre 1999 y 2004, apenas existía bibliografía disponible sobre el tema. Sin embargo, es conocido que durante los primeros años del siglo XXI se produjeron importantes avances en la tecnología de las computadoras, tales como procesadores más rápidos, mayores capacidades de memoria y mejores tarjetas gráficas, lo que supuso un gran impulso para la realidad virtual.

En julio de 2012, P. Luckey presentó Oculus Rift [32], un proyecto que se hizo conocido a través de Kickstarter [33]. Este dispositivo tenía como objetivo mejorar los HMD (dispositivos de visualización montados en la cabeza) para la realidad virtual que existían en ese momento. En 2014, Oculus fue adquirido por Facebook [34], quien ahora financia su desarrollo.

En marzo de 2014, Sony presentó Project Morpheus [35], un HMD para la realidad virtual que permite la visualización de un entorno virtual generado por la potencia de su consola de última generación, la PS4. Esta consola cuenta con un procesador de AMD de 8 núcleos, 8 GByte de memoria RAM compartida por la tarjeta gráfica AMD específica de la consola de Sony, y una pantalla AMOLED de alta resolución (1080p) de 5,7 pulgadas. Morpheus también incluye un sensor de seguimiento de posición de la cabeza, una característica común en estos dispositivos desde los primeros que aparecieron.

En mayo de 2015, se presentó FOVE [36] en la plataforma Kickstarter, un sistema de realidad virtual similar a Oculus Rift, pero con una característica diferenciadora: es posible utilizarlo como dispositivo de entrada a través de los ojos del usuario. Este dispositivo cuenta con un sensor que captura el movimiento ocular, lo que permite utilizarlo como sustituto del ratón.

Durante 2015, noticias como la adquisición de Oculus Rift por Facebook o la de Metaio por parte de Apple [37] dejaron claro el interés estratégico de la realidad virtual para las grandes empresas del sector tecnológico.

En 2014, Google lanzó la primera versión de sus gafas de realidad virtual, Cardboard [38], el dispositivo más económico y sencillo del mercado. Hay diferentes modelos [39] cuyos precios oscilan entre 7,99€ y 37,95€. Están fabricados en cartón, aunque algunos modelos ya están hechos de plástico. Según Wikipedia, "con apenas un cartón plegable recortado y dos lentes, es posible montar el teléfono inteligente y aprovechar las aplicaciones de Android VR" [40]. Este producto permite experimentar con la realidad virtual a un costo reducido, ya que se han desarrollado aplicaciones educativas. En 2015, se lanzó la segunda versión, que se ensambla en tres pasos y cuenta con un sujetador para la cabeza, pero sigue teniendo dos lentes de resina y está hecha de cartón.

Figura 13: Google Cardboard

En la misma época, surgió un dispositivo llamado The Leap de la empresa Leap Motion, que mostró el potencial del futuro de la realidad virtual (RV) y de la realidad aumentada (RA). Este dispositivo es un sensor de movimiento capaz de detectar los movimientos de las manos del usuario. El seguimiento manual se realiza en procesadores móviles con baja latencia y alta precisión, lo que permite una interacción sin precedentes entre el usuario y la computadora mediante gestos en lugar de los tradicionales teclado y ratón (LeapMotion, 2018).

Además, se han desarrollado andadores de RV que complementan los cascos de RV y acercan aún más a los usuarios a los entornos virtuales. Estos andadores son similares a las caminadoras de gimnasio y permiten a los usuarios desplazarse de forma segura y libre en cualquier dirección, lo que facilita movimientos de 360°. La primera opción fue el Virtuix Omni, que tiene una forma cóncava donde los pies de los usuarios se apoyan, pero sus movimientos pueden verse limitados por un arnés de seguridad que impide que se inclinen (VirtuixOmni, 2019). Una alternativa más ligera es el Cyberith Virtualizer, que

consiste en una alfombra deslizante que se mueve en respuesta a los pasos del usuario en cualquier dirección (Katvr, 2018).

A continuación, vamos a repasar brevemente los distintos dispositivos presentados y sus respectivas características.

Las gafas Samsung Gear VR [41], por un lado, destacan por su atractivo diseño, su excelente óptica, su campo de visión de 96º y la capacidad de seguimiento de nuestra cabeza, gracias a su variedad de sensores que incluyen acelerómetros, girosocopios y sensores de proximidad. Todo esto contribuye a crear una experiencia inmersiva más completa. No obstante, este dispositivo requiere un dispositivo móvil para funcionar y solo es compatible con modelos Samsung.

Figura 14: Samsung Gear VR [42]

En 2016, Sony dio a conocer las PlayStation VR, consideradas hoy en día como la gran innovación en la línea de productos de PlayStation (PlayStation, 2016). Este dispositivo está equipado con una pantalla OLED [43] que ofrece una velocidad de actualización de 120 Hz. Además, cuenta con un micrófono integrado, un campo de visión de 100º y sensores acelerómetros [44].

Figura 15: PlayStation VR[45]

En 2023, salió la PlayStation VR2 [46], el sucesor del PlayStation VR original y está diseñado para funcionar con la consola PlayStation 5. La PlayStation VR2 cuenta con una

pantalla de mayor resolución, un mejor seguimiento y un controlador rediseñado. También tiene retroalimentación háptica y detección de toque de los dedos para una experiencia más real y viva. El casco utiliza un solo cable para conectarse a la consola PlayStation 5 y admite tanto juegos de realidad virtual como juegos tradicionales. También incluye una cámara incorporada para seguir la posición del casco y los controladores.

Figura 16: PlayStation VR2

Dos empresas líderes en el mundo de la realidad virtual son Oculus y HTC, ambas revolucionarias en su campo. Las Oculus Rift salieron al mercado en marzo de 2017, y su principal competidora, las HTC Vive (lanzadas en 2016 por HTC), ofrecen una experiencia inmersiva similar. Para utilizar cualquiera de estos dispositivos, se necesita un ordenador potente para poder disfrutar de la experiencia de realidad virtual más realista posible.

Las Oculus tienen una pantalla OLED, una tasa de refresco de 90 Hz y un ángulo de visión de 110º. Además, cuentan con varios sensores para el seguimiento de la posición, incluyendo acelerómetro, giroscopio, magnetómetro y sistema posicional de 360º. [47]

Por su parte, HTC ofrece un kit que incluye las gafas de realidad virtual, una caja de conexiones para conectar las gafas al PC, un par de sensores de posición y unos mandos inalámbricos. Esto permite una inmersión muy realista. Las gafas de RV tienen una pantalla OLED, una tasa de refresco de 90 Hz, el audio está integrado y cuentan con sensores de acelerómetro, giroscopio, doble sistema de posición de láser (las gafas tienen 36 y los mandos 24 cada uno) y una cámara frontal.

En términos de especificaciones, las Oculus y las HTC son muy similares, aunque las HTC tienen una mayor compatibilidad con diferentes dispositivos gracias a sus mandos. Las Oculus sólo son compatibles con los Oculus Touch o el Xbox One, mientras que las HTC funcionan con los mandos HTC Vive, el mando Steam VR y cualquier Gamepad de PC. [48]

Figura 17: Oculus Rift [49]

Figura 18: Kit HTC Vive [50]

En 2019, Facebook transformó el panorama de la realidad virtual para siempre con el lanzamiento de Oculus Quest [51]. Oculus Quest fue el primer auricular independiente de realidad virtual, lo que significaba que se eliminaba toda la incomodidad de estar conectado a una PC o Mac. Aunque Quest no tenía el poder informático de una computadora de escritorio completa, era suficiente para proporcionar una experiencia de realidad virtual única. El éxito de Quest impulsó mejoras que llevaron al lanzamiento de Quest 2 [52] en 2020, con una mejor resolución de pantalla, frecuencia de actualización, almacenamiento y potencia de procesamiento.

Figura 19: Oculus Quest 2

En octubre de 2021, Facebook anunció una nueva dirección para la empresa. Las marcas como Facebook y Oculus se desvanecerán y, en su lugar, la empresa se centrará en el desarrollo del "metaverso". Pero, a principio de 2023, había rumores de salida próxima de la Oculus 3 [53].

Tal y como hemos venido informando, cada empresa está trabajando en su propio dispositivo y sistema, lo que hace que muchos productos difieran significativamente entre sí. Esto aporta variedad al mercado, pero a su vez supone un reto para los desarrolladores, ya que necesitan recursos considerables para poder trabajar en todas las plataformas existentes. Por este motivo, se está volviendo cada vez más importante estandarizar las API.

Para abordar esta cuestión, Khronos (khronos, 2016) y otros actores clave del sector, como Steam VR, Oculus, Gear VR, OSVR y Daydream, están colaborando en el desarrollo de unas API estándar que puedan utilizarse en cualquier dispositivo, independientemente del hardware que se utilice. Aunque no se puede garantizar una compatibilidad total, sí se podrán establecer unos requisitos mínimos que permitirán añadir valor a cada plataforma. [54]

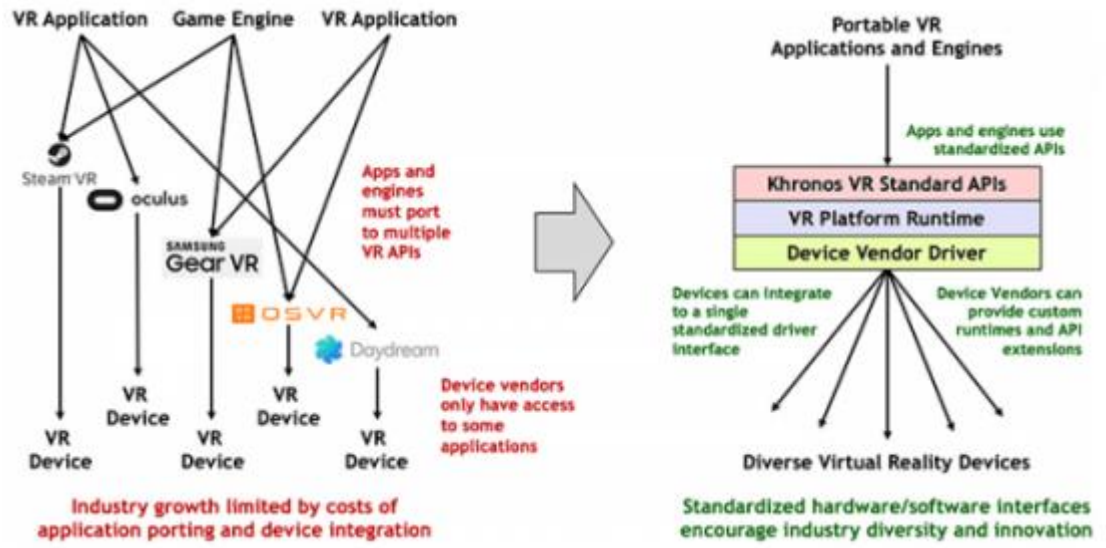


Figura 20: Proyecto Estandarización API RV

La Figura 21 muestra los predecesores de las tecnologías explicados hasta ahora, viajando a través de la historia para localizar como cada tecnología evoluciono desde su invención

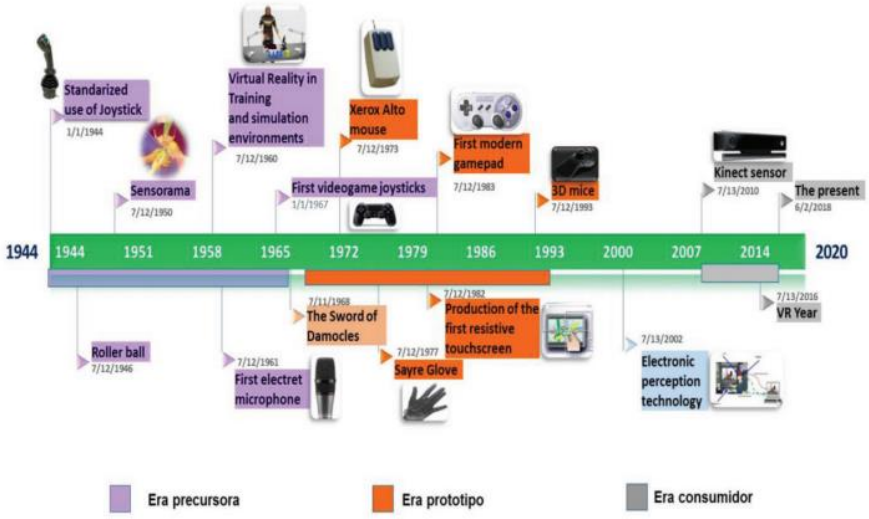


Figura 21: Eras de los dispositivos de interacción para ambientes virtuales

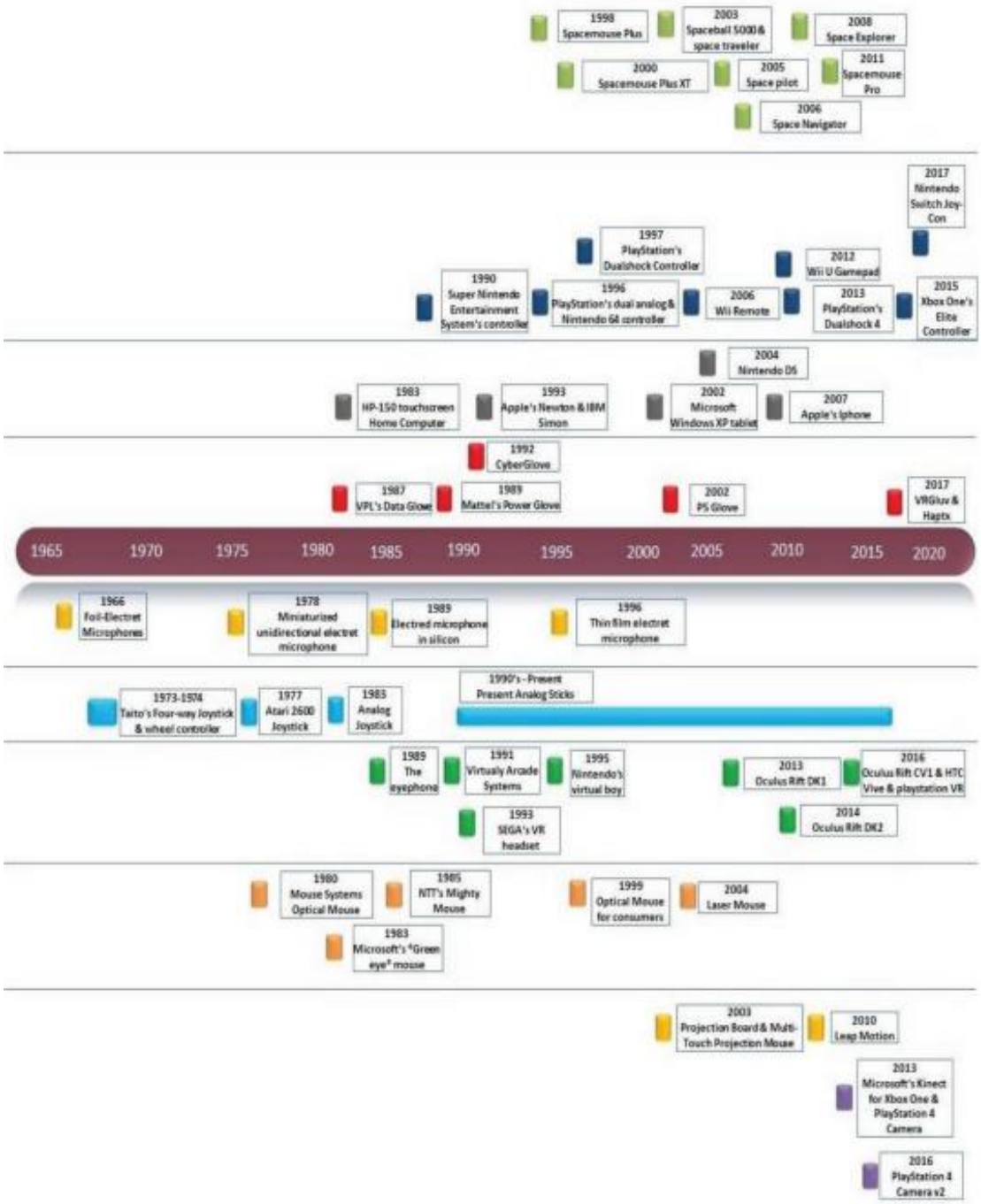


Figura 22: Línea de tiempo de las tecnologías descritas

Para concluir, es importante destacar que en estos últimos años no se han centrado únicamente en el diseño de gafas o mandos de realidad virtual. También se ha llevado a cabo investigación y desarrollo de diversos dispositivos hápticos, como guantes, trajes y plataformas, entre otros.

- **Guantes hápticos:**

Estos guantes permiten la posición y el movimiento de nuestras manos y dedos, y proporcionan feedback háptico en cada dedo. En el caso específico de los guantes de Senso Devices, integran 7 sensores inerciales, 5 motores de vibración y sensores para el seguimiento de SteamVR [55], lo que les permite ofrecer una latencia inferior a 10ms y una autonomía de 10 horas.

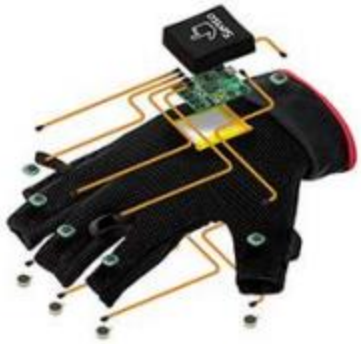


Figura 23: Guante háptico de Senso Devices

- **Traje háptico:**

Por otra parte, el traje nos brinda seguimiento completo de todo el cuerpo, gracias a sus 15 módulos que funcionan mediante USB o WiFi y que integran un sensor inercial, un motor de vibración y sensores de SteamVR. Estos módulos se distribuyen por los brazos, piernas y torso. La idea de crear el traje surgió a partir de la demanda de los usuarios que deseaban aplicar la misma tecnología al resto del cuerpo. Esta solución permite el seguimiento de cuerpo completo en juegos o su uso para la captura de movimiento.

Tanto los guantes como el traje incluyen sensores de SteamVR [55], pero su uso es opcional. Es decir, si se desea obtener un posicionamiento más preciso que el proporcionado por los sensores inerciales.

Figura 24: Traje háptico de Senso Devices

- **Plataformas:**

En este caso, nos referimos a una plataforma de la empresa norteamericana Virtuix, creadora del sistema de realidad virtual Virtuix Omni. Recientemente se anunció la participación de un startup español del fondo Startcaps Ventures.

La Virtuix Omni es una plataforma omnidireccional y periférica para videojuegos de realidad virtual. Para simular el efecto de caminar, la plataforma utiliza una base o piso resbaladizo, para lo cual se requieren zapatos especiales que reduzcan la fricción al caminar. El jugador se encuentra completamente dentro de un "anillo" que, además de absorber el peso del jugador, cuenta con un arnés que se ajusta a la cintura y se apoya en una base [56].

Figura 25: Virtuix Omni, plataforma de RV

3. Estado del arte de la Escape Room:

3.1. Ideas Básicas:

Cuando nos encontramos con la palabra "Escape", solemos asociarla con el término "Escapismo". Pero ¿qué significa exactamente el término escapismo? El escapismo se refiere a la práctica de escapar de situaciones de encierro físico o de otras trampas. Los escapistas, también conocidos como artistas del escape, se liberan de esposas, camisas de fuerza, jaulas, cofres, cajas de acero, barriles, bolas, edificios en llamas, tanques de agua y otros peligros, a menudo combinados. [57]

Aunque la definición anterior es útil, no debemos confundir el concepto de escapismo con el de "escape room". Un escape room es un juego de aventuras en el que los jugadores se encuentran bloqueados en una habitación y deben usar los elementos que se encuentran en ella para resolver una serie de pruebas y escapar antes de que se acabe el tiempo. Este juego se desarrolla en un local con una o varias salas, llenas de enigmas y objetos extraños. Las salas de escape room son controladas por el encargado del juego, quien explica el misterio que se debe resolver y proporciona pistas en caso de ser necesario.

Las salas de escape room son una actividad de ocio cada vez más popular, en la que se puede pasar un buen rato y trabajar en equipo para conseguir escapar de una habitación completamente cerrada mientras se resuelven misterios y enigmas.

3.2. Sus orígenes:

En España, algunas de las salas de escape más famosas, como Cubick [58], Lever Escape Room [59] o Mad Escape Room [60], se originaron en California, Estados Unidos, en 2006. En concreto, en Silicon Valley se creó Origin, el primer juego en vivo en espacio cerrado que intentaba dar vida a una novela de Agatha Christie. Este escape room era muy diferente a las que conocemos hoy en día, pero aun así tuvo éxito y se expandió por Asia.

En Japón, comenzaron a hacer réplicas de Origin y de otras salas novedosas. Por lo tanto, en 2008, ya había varias salas de escape disponibles. Finalmente, en 2011 nació Parapark [61]. El formato que conocemos hoy en Europa se inició en Budapest, Hungría, cuando Attila Gyurkovics creó un juego en el que un grupo de personas tenía que buscar la forma de salir de una habitación en un tiempo limitado. Este juego se basó en la Teoría del Flow, desarrollada por el psicólogo húngaro Mihaly Csíkszentmihályi, como se cita en La línea del tiempo, iPlay. [62]

La Teoría del Flow es uno de los fundamentos y orígenes del escape room tal y como las conocemos hoy en día. Fue Cskszentmihalyi [63] quien investigó y desarrolló esta teoría, que no solo es la base del escape room, sino también de muchas otras situaciones.

3.3. Estado del arte:

Según Cubick Room Escape [64], las Escape Room tiene como origen ideológico

en:

- Los libros enigma de los años 70. En estos libros se incluían papeles de diferente tipo y tarjetas con el fin de encontrar mensajes secretos.
- Lo ordenadores. “Los primeros y más básicos juegos de rol, llamados aventura conversacional, donde únicamente tenías una pantalla con texto que te describía una situación y donde tú introducías, también mediante texto, la acción precisa que querías que realizase tu personaje. Estas acciones, te hacían interactuar con el espacio que el juego te ofrecía para avanzar”.
- La literatura y aventura: “Lo que realmente destaca es su aspecto literario, la capacidad de utilizar la lectura de una manera diferente. Tras ello, la computación, los juegos y la inclusión del vídeo nos hicieron acercarnos más a la imagen que tenemos a día de hoy de las salas de escapismo.”

Como se mencionó en la sección 3.2, las salas de escape room tuvieron su origen en el año 2006 en Silicon Valley con la creación de Origin. Posteriormente, este fenómeno se expandió por Asia y en 2011 Attila Gyurkovics creó Parapark [65], la primera escape room como la conocemos hoy en día.

Es importante destacar que, aunque la primera escape room nació en Hungría con Parapark [66], el concepto de escape room ya existía en otras propuestas de ocio que se ofrecían en Japón.

En el año 2008 surgieron los "juegos de escape" y sus salas tanto en Tokio como en Osaka, aunque su concepto era ligeramente diferente al de hoy en día. En aquel entonces, se enfatizaba más en la experiencia y la aventura que en la resolución de enigmas o la toma de decisiones, a diferencia de los juegos de escape actuales. Además, en esas salas los jugadores eran acompañados por un miembro del equipo que, a modo de actor, lideraba el grupo en su recorrido por las distintas salas.

Figura 26: Escape Room Joypolis [67]

El tipo de escape room que conocemos hoy en día surgió con Parapark en el año 2011, creado por Attila Gyurkovics en Budapest, Hungría. Hasta ese momento, solo se habían experimentado juegos de enigmas en papel o en los primeros videojuegos modernos. La franquicia internacional fue un éxito abrumador y se extendió rápidamente por Europa, llegando a España a través de Parapark Barcelona en el año 2012.

Figura 27: Escape Room Parapark

De acuerdo con MAD Escape Room [68], el fenómeno de las salas de escape ha experimentado un notable aumento en los últimos años. En el año 2022, se contabilizaban más de 3.400 salas de escape en todo el mundo, y en ciudades como Nueva York o Chicago se pueden encontrar decenas de ellas [69]. En la imagen adjunta se pueden observar las ciudades con mayor número de salas de escape en el mundo.

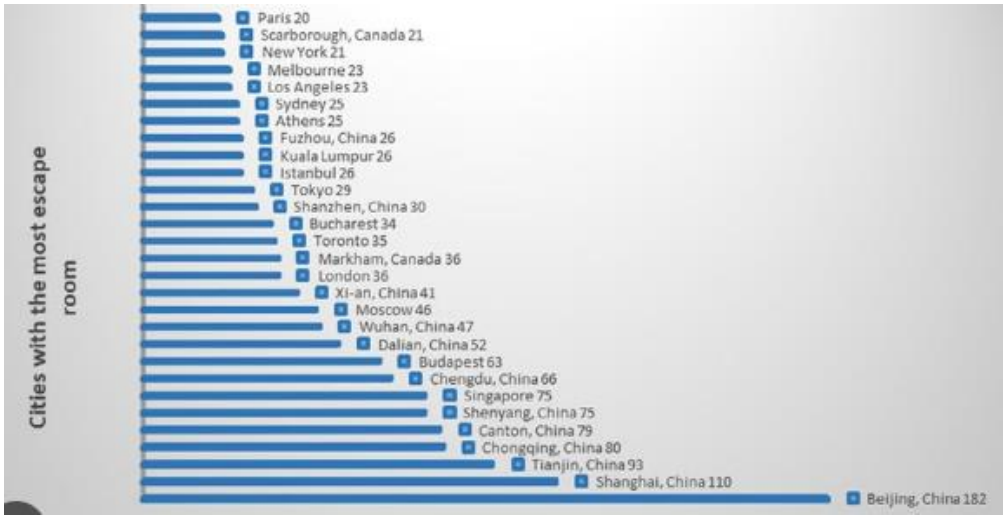


Figura 28: ciudades con más escape rooms en el mundo

Cuando el fenómeno de las salas de escape apareció en 2012, se consideraba principalmente como una actividad de ocio que permitía a los participantes evadirse durante una hora, fomentar el trabajo en equipo y disfrutar de un buen momento. Sin embargo, ha resultado ser mucho más poderoso de lo que se pensaba, como demuestra su impresionante crecimiento actual.

No obstante, con el transcurso del tiempo se ha observado que esta actividad ha alcanzado un nivel de éxito superior al esperado. ¿A qué se debe tal fenómeno?

- En primer lugar, se debe destacar el papel fundamental del trabajo en equipo en esta actividad. A diferencia de otras formas de entretenimiento, no se basa en la competencia entre los jugadores, sino en la cooperación del grupo. Se enseña la importancia de sumar esfuerzos para resolver el misterio y alcanzar el objetivo propuesto.
- Otro aspecto que contribuye a su triunfo radica en la naturaleza única de cada escape. Antes de ingresar a la sala, se genera un suspenso que despierta el interés. Además, cada grupo de participantes logra crear su propia experiencia,

descubriendo pistas y resolviendo enigmas en distintos momentos del juego. Aunque al final todos los grupos logren (casi siempre) escapar, la vivencia en sí misma resulta ser única. Desde los momentos previos, con la intriga sobre lo que se encontrará, hasta los posteriores, recordando anécdotas y momentos que han dejado una huella en todo el grupo.

- Además, esta actividad logra que los participantes se sumerjan en un mundo aparte, desconectándose de sus vidas y concentrándose plenamente en la resolución de los desafíos. Esto genera una sensación de protagonismo en una historia en la que solo existen ellos, y en la cual el éxito depende exclusivamente de su desempeño.

Los escape-rooms también nos brindan la oportunidad de trabajar en equipo, aceptar todas las opiniones como válidas, practicar la escucha activa y llegar a un acuerdo. Al descubrir el potencial del escape room para brindar la "esencia de la felicidad", se ha comenzado a explorar cómo esta actividad puede beneficiar a otros campos, como la educación y el turismo.

Surge así una nueva forma de turismo, conocida como turismo de escape, en la que se incluye una partida de escape room en nuestra ruta de viaje o se elige un destino en función de la oferta de salas de escape disponibles. Moscú es un buen lugar para comenzar a explorar esta forma innovadora de conocer nuevos lugares, ya que cuenta con más de 50 salas, mientras que en Budapest hay edificios enteros llenos de salas de escape, cada piso con nuevos desafíos para enfrentar.

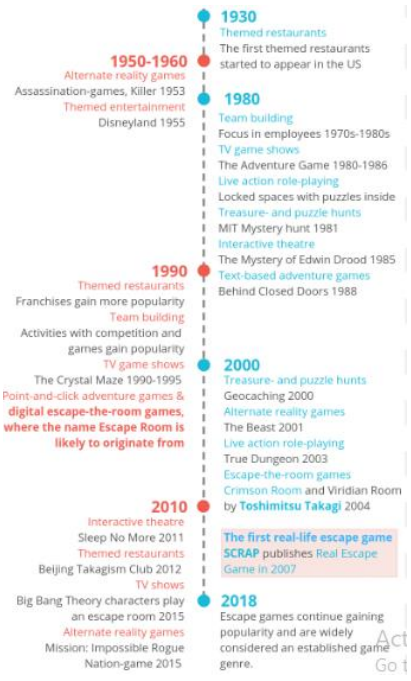


Figura 29: Línea temporal del escape room

4. Escape Room con la Realidad Virtual:

Como hemos comentado anteriormente, las empresas líderes en tecnología, tales como HTC, Google, Facebook (Meta) y Sony, entre otras, están invirtiendo fuertemente en realidad virtual, especialmente en el ámbito de los videojuegos, según señalan Forbes [70] y The Motley Fool [71] en sus respectivos artículos. Este hecho es relevante porque las escape room se originaron como juegos, pero han demostrado tener enormes posibilidades en el campo de la educación, como se ha discutido en este trabajo. Si se adapta a la realidad virtual, esta actividad tiene un enorme potencial, ya que permitiría a los usuarios vivir experiencias más inmersivas y jugar desde la comodidad de sus hogares.

Actualmente, ya existen dispositivos que permiten jugar a escape rooms en realidad virtual, y la sensación de inmersión es cada vez más realista. Solo es necesario decidir si se prefiere realizar la actividad en casa, como por ejemplo con Escape Room VR Three Stories [72], que solo está disponible para los usuarios de Samsung Gear VR que posean un Samsung Note 5 o un S6, o en un espacio habilitado, como las salas de escape tradicionales.

En Barcelona, ya existe la primera escape room en realidad virtual [73], llamada "La quest virtual COSMOS" [74], que se encuentra en un espacio habilitado para tal fin. Según se menciona en un artículo de El Periódico [74], los usuarios pueden mover objetos, interactuar con amigos en persona, disparar rayos láser, probar habilidades de telequinesis y hasta volar. El creador de la experiencia, Yuri Popov, afirma que "Lo más increíble es que nuestro cerebro no diferencia lo que es realidad y lo que es realidad virtual. Por eso es tan increíble. Tú sabes que es falso, pero tu cerebro cree que es real".

Además, Yuri Popov añade, “La realidad virtual es el futuro de los juegos. En dos, tres años, mucha gente tendrá gafas de realidad virtual en casa”. Por lo que no se confirma, pero todo apunta a que la realidad virtual por fin ha llegado para quedarse.

Figura 30: Escapistas en “La quest virtual COSMOS”

5. Teoría Musical:

Para llevar a cabo este Trabajo de Fin de Grado, es necesario poseer ciertos conocimientos mínimos de música, ya que el juego involucra instrumentos musicales. La música se define como la combinación de sonidos en el tiempo y se asume que dicha combinación es correcta, ya que existen combinaciones que pueden resultar desagradables al oído.

La representación gráfica de los sonidos se lleva a cabo en un pentagrama, que consta de 5 líneas horizontales equidistantes donde se representan las notas y las demás notaciones musicales.

El arte de la música se rige por los principios fundamentales de la melodía, la armonía y el ritmo. Como se ha mencionado anteriormente, la música es una manifestación artística que tiene como objetivo expresar sentimientos, emociones, pensamientos, ideas o circunstancias. Es importante destacar que, en la actualidad, la música también se utiliza con fines terapéuticos.

La melodía se define como una combinación de sonidos que busca lograr una sonoridad agradable al oído y que tenga sentido musical. La armonía, por su parte, consiste en la combinación de diferentes sonidos que se tocan al mismo tiempo para acompañar la melodía y lograr una sonoridad agradable.

A continuación, se explicarán las nociones básicas de música que ayudan al jugador del juego explicado en este Trabajo de Fin de Grado.

5.1. Notas Musicales:

En el pentagrama, cada sonido se representa mediante una nota. Hay siete notas diferentes que se colocan una tras otra en el siguiente orden: Do, Re, Mi, Fa, Sol, La, Si. Para la aplicación que se utilizará en este trabajo, se empleará el sistema latino de notación. Sin embargo, existen otros sistemas de notación musical, como el sistema inglés (también conocido como denominación literal), cuya representación es: C, D, E, F, G, A, B, y el sistema alemán de notación musical, que es muy similar al sistema inglés, pero con una letra diferente para la nota Si, que se representa con la letra H.

Dentro de las notas musicales, es posible clasificarlas según su duración. En la siguiente figura se pueden ver las diferentes notas junto con su duración correspondiente. Cada nota tiene su equivalente en silencio, y en la misma figura también se pueden apreciar los distintos silencios clasificados según su duración.

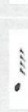
denominación	figura	Duración 4/4	Figuras que caben para completar el compás de 4/4
REDONDA		4/4	1
BLANCA		2/4	2
NEGRA		1/4	4
CORCHEA		1/8	8
SEMICORCHEA		1/16	16
FUSA		1/32	32
SEMIFUSA		1/64	64

Figura 31: Valor de cada nota

El nombre de las notas musicales cambia en función de su ubicación en el pentagrama. Para identificar correctamente una nota, es necesario fijarse en el inicio del pentagrama, donde se encuentra una figura llamada clave. La clave sirve como punto de referencia para relacionar la ubicación de las notas con los sonidos correspondientes.

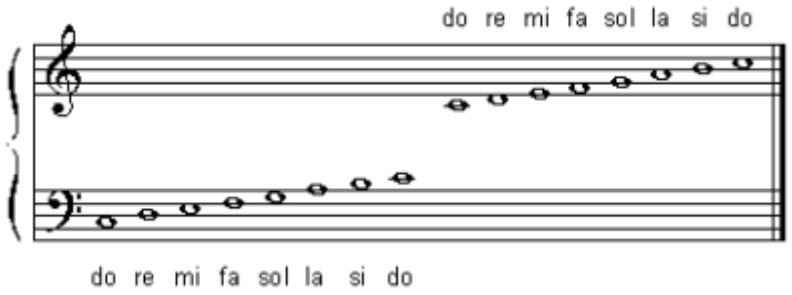


Figura 32: Pentagrama con notas musicales

5.2. Teclado del Piano:

El teclado del piano está formado por teclas negras y blancas. Las teclas negras están en grupos de dos y tres teclas. Los sonidos del piano son más agudos hacia la derecha del teclado (hacia arriba), y más graves hacia la izquierda (hacia abajo).

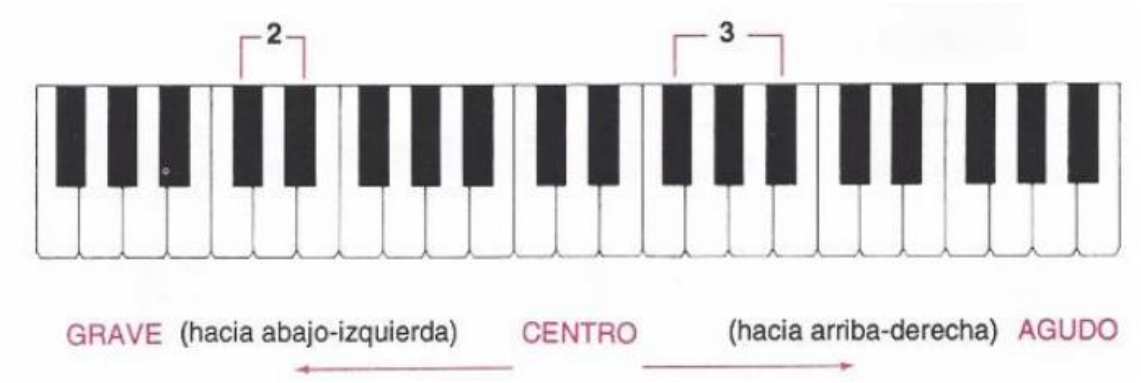


Figura 33: Teclado de un piano

En este TFG, solo nos interesan las teclas blancas del piano. Como hemos comentado antes, nuestro sistema musical tiene siete notas. El orden de las notas es DO, RE, MI, FA, SOL, LA Y SI. Estas notas corresponden a las teclas blancas del piano:

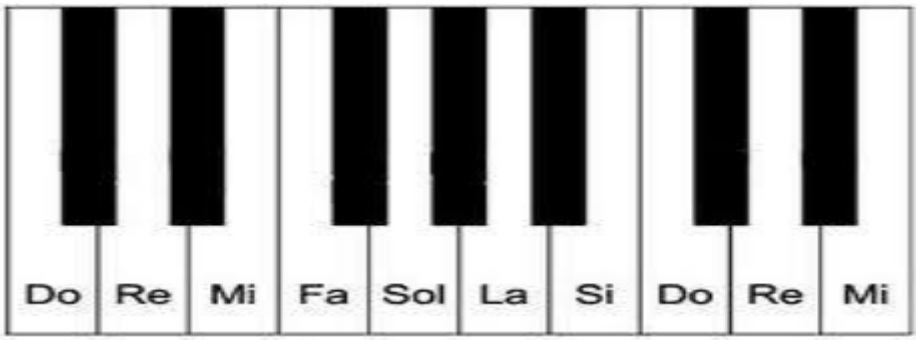


Figura 34: Notas de las teclas del piano

Las notas musicales pueden ser nombradas también por algunas letras del abecedario en lugar de los nombres clásicos que ya conocemos: do – re – mi – fa – sol – la – si. Estas letras son utilizadas en algunas partituras. Sobre todo, de música popular. Estas mismas notas pueden ser identificadas con las siguientes letras:

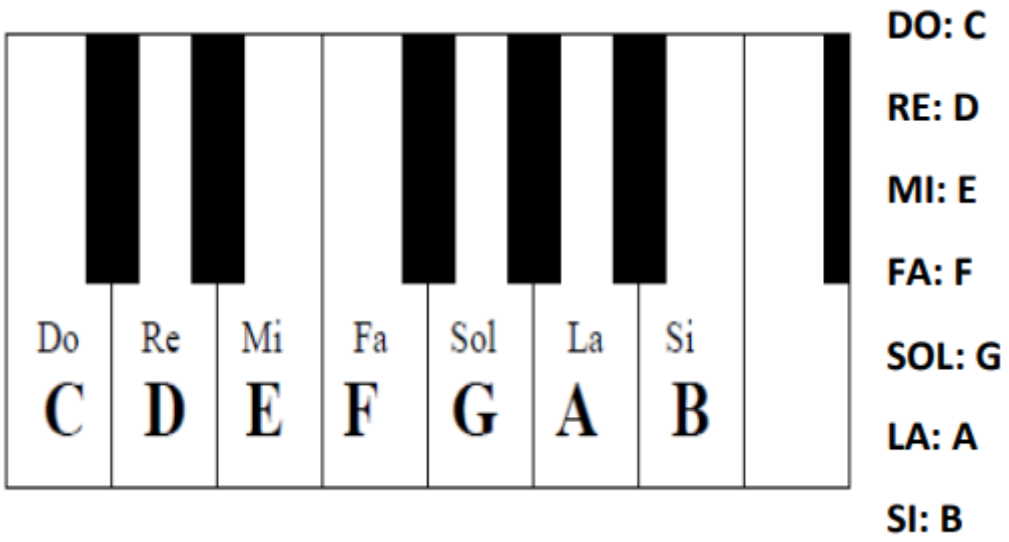


Figura 35: Cifrado Americano de las teclas del piano

5.3. Protocolo MIDI:

La evolución de la música ha sido influenciada por los avances tecnológicos a lo largo de los años. En 1983 se estandarizó el protocolo MIDI, coincidiendo con la popularidad de los dispositivos electrónicos musicales. El MIDI, o Interfaz Digital de Instrumentos Musicales, es un protocolo de comunicación digital que permite la interacción entre instrumentos y ordenadores. Sin embargo, la representación musical en MIDI es diferente a la de los seres humanos, ya que los datos deben ser comprensibles por las máquinas.

El protocolo MIDI utiliza mensajes digitales que contienen información sobre los parámetros musicales, como la nota a tocar, su duración, intensidad y notación musical. Estos mensajes le indican al receptor cuándo y cómo debe reproducir la nota.

El canal MIDI es un aspecto importante del protocolo, ya que permite que un instrumento maestro transmita información a un instrumento esclavo. Es similar a los canales de televisión, donde el instrumento maestro decide qué canal transmitir y el instrumento esclavo selecciona el canal que desea recibir. Hasta 16 canales pueden ser transmitidos de manera independiente, aunque el esclavo solo recibirá el canal seleccionado, al igual que un televisor que transmite todos los canales, pero solo muestra uno.

En la Figura siguiente se muestra un ejemplo del funcionamiento de los canales MIDI.

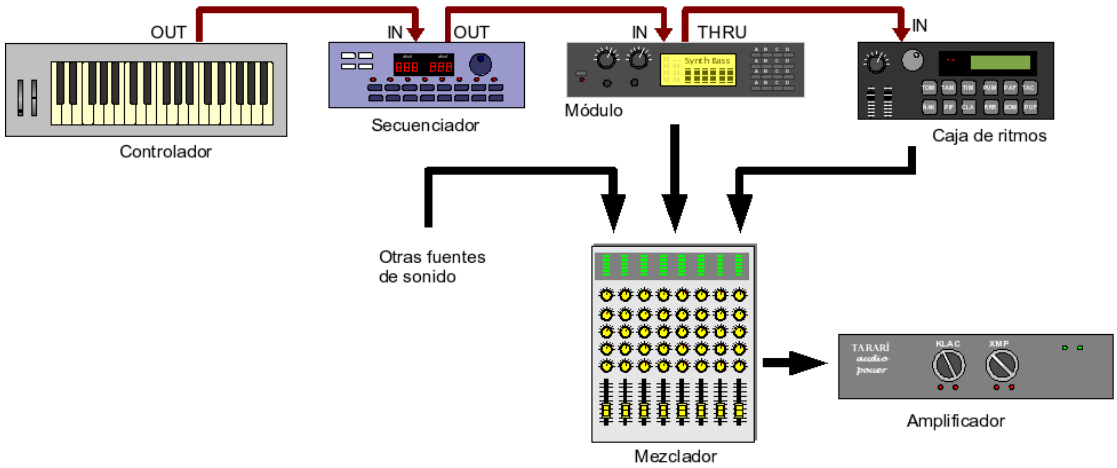


Figura 36: Ejemplo del uso de MIDI

Los principales dispositivos son los siguientes:

- Controladores. Son las fuentes primarias de mensajes Midi y por eso llevan una entrada MIDI-OUT. Los más frecuentes son teclados, pero existen símiles electromecánicos de diversos tipos de instrumentos: flautas, saxofones, guitarras y otros. Los controladores no producen ningún sonido.
- Módulos de sonido de diversos tipos. Son los receptores de mensajes y por eso tienen una entrada MIDI-IN. Son los consumidores primarios de mensajes Midi, y los traducen en señal de audio. La conversión depende del diseño del módulo y de los parámetros que se le hayan fijado. Estos son: en general, cada módulo tendrá sus sonidos, que se podrán modificar en mayor o menor medida mediante algunos parámetros. Los módulos suelen llevar una salida MIDI-THRU que reproduce los mensajes midi recibidos por la entrada para poder conectar otros módulos en cascada.
- Sintetizadores completos, que combinaban un controlador de tipo teclado con un módulo de sonido. Pueden funcionar de forma autónoma conectando el controlador con el módulo interno; pero también admiten conexiones de entrada y de salida con otros dispositivos MIDI. Como son productores y receptores de mensajes, sus conexiones son tres: MIDI-IN, MIDI-OUT y MIDI-THRU.
- Cajas de ritmos, que combinan un módulo de sonido (especializado en instrumentos de percusión) con una fuente de mensajes programable. Sus usuarios lo usan para programar esquemas rítmicos propios de una batería. Suelen disponer de las tres conexiones Midi.
- Secuenciadores: dispositivos que almacenan la información Midi de manera que pueden repetirla después de manera controlada. Un secuenciador permite el tratamiento de la información Midi almacenada. También suelen disponer de las tres conexiones Midi.
- Dispositivos de encaminamiento de mensajes: concentradores, bifurcadores y otros, que permiten combinaciones flexibles de diversos componentes.

Con este concepto, es posible tocar 16 partes distintas en 16 instrumentos mediante un solo cable MIDI. Hay cuatro modos disponibles:

- Modo 1: OMNI ON, POLY. En este modo, se recibe información de todos los canales de forma polifónica.
- Modo 2: OMNI ON, MONO. Aquí, se recibe información de todos los canales, pero solo suena una nota a la vez.
- Modo 3: OMNI OFF, POLY. Se recibe información del canal MIDI seleccionado de forma polifónica, lo que resulta muy útil con secuenciadores.
- Modo 4: OMNI OFF, MONO. En este modo, se recibe información de los canales MIDI seleccionados, pero solo suena una nota a la vez. Este modo resulta muy útil para controladores de guitarra.

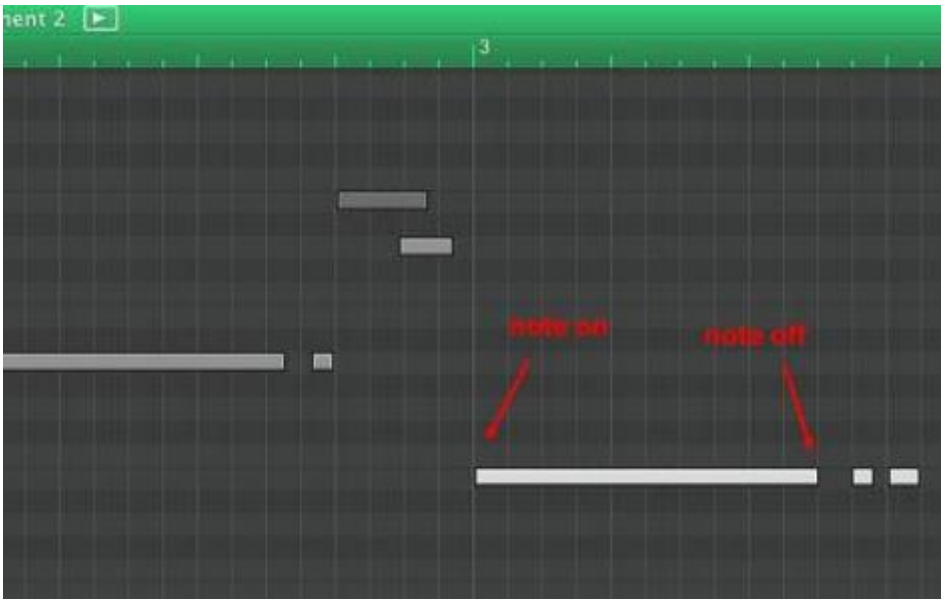


Figura 37: Representación MIDI

Cada nota musical está asociada con una frecuencia y, cada frecuencia, está asociada a un valor que corresponde la nota MIDI. La relación las podemos sacar de cualquiera de estas dos ecuaciones:

$$N_{MIDI} = \left(69 + 12 \cdot log_2 \frac{f(Hz)}{440}\right)$$

$$f = 440 \cdot 2^{\frac{N_{MIDI}-69}{12}}$$

Figura 38: Relación MIDI-Frecuencia de notas

En la siguiente figura, se pueden ver las notas musicales con notación americana, su correspondiente octava, junto con la frecuencia y luego con el código del protocolo MIDI de cada nota.

	C	C#	D	Eb	E	F	F#	G	G#	A	Bb	B
0	16.35	17.32	18.35	19.45	20.60	21.83	23.12	24.50	25.96	27.50	29.14	30.87
1	32.70	34.65	36.71	38.89	41.20	43.65	46.25	49.00	51.91	55.00	58.27	61.74
2	65.41	69.30	73.42	77.78	82.41	87.31	92.50	98.00	103.8	110.0	116.5	123.5
3	130.8	138.6	146.8	155.6	164.8	174.6	185.0	196.0	207.7	220.0	233.1	246.9
4	261.6	277.2	293.7	311.1	329.6	349.2	370.0	392.0	415.3	440.0	466.2	493.9
5	523.3	554.4	587.3	622.3	659.3	698.5	740.0	784.0	830.6	880.0	932.3	987.8
6	1047	1109	1175	1245	1319	1397	1480	1568	1661	1760	1865	1976
7	2093	2217	2349	2489	2637	2794	2960	3136	3322	3520	3729	3951
8	4186	4435	4699	4978	5274	5588	5920	6272	6645	7040	7459	7902

Figura 39: Frecuencia de las diferentes notas musicales

	Note Numbers											
Octava	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
-1	0	1	2	3	4	5	6	7	8	9	10	11
0	12	13	14	15	16	17	18	19	20	21	22	23
1	24	25	26	27	28	29	30	31	32	33	34	35
2	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59
4	60	61	62	63	64	65	66	67	68	69	70	71
5	72	73	74	75	76	77	78	79	80	81	82	83
6	84	85	86	87	88	89	90	91	92	93	94	95
7	96	97	98	99	100	101	102	103	104	105	106	107
8	108	109	110	111	112	113	114	115	116	117	118	119
9	120	121	122	123	124	125	126	127				

Figura 40: Código MIDI de las notas musicales

Los mensajes MIDI están estructurados en octetos. Cuando se transmiten, a cada octeto se le añade un bit de paridad para detección de errores.

El primer octeto de un mensaje contiene un comando codificado en cuatro bits de la forma1MMM. (en hexadecimal, 0x8 a 0xF) y un identificador de canal de forma CCCC (por eso hay 16 canales, que generalmente se numeran del 1 al 16).

Figura 41: Mensaje MIDI

Cada comando va acompañado de sus parámetros característicos. Los parámetros se alojan en octetos cuyo bit más significativo es siempre 0. Los siete bits restantes codifican un valor comprendido entre 0 y 127. Los comandos admitidos en esta codificación son:

Mensaje	Cód.	Parámetro 1	Parámetro 2
Note Off	0x8	número de nota	velocidad
Note On	0x9	número de nota	velocidad
Note Aftertouch	0xA	número de nota	presión
Controller	0xB	núm. de controlador	controller value
Program Change	0xC	núm. de programa	—
Channel Pressure	0xD	presión	—
Pitch Bend	0xE	*pitch value (LSB)*	*pitch value (MSB)*

Figura 42: comandos usados en mensajes MIDI

Los comandos Note On y Note Off, por ejemplo, codifican la tecla en un rango suficiente para representar las 88 notas del piano y otras que no existen en los instrumentos analógicos tradicionales.

5.4. Efectos de Sonido:

5.4.1. Eco:

El retraso es el efecto de sonido más básico. El retraso de una señal x[n] considerando muestras discretas puede definirse como:

$$y[n] = x[n - D]$$

El retraso es un efecto simple a partir del cual pueden entenderse otros efectos de sonido más complejos. De esta forma, un eco puede expresarse como:

$$y[n] = x[n] + ax[n - TD]$$

En la expresión TD, da cuenta de la muestra a partir de la cual se empieza a percibir la señal de eco. En concreto, si se denota por fs a la frecuencia de mestreo, y D representa el tiempo de retardo en segundos, la señal eco se percivirá (en notación de Matlab)

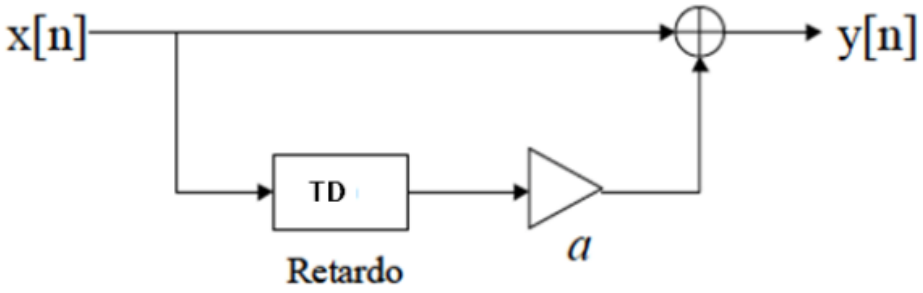


Figura 43: Diagrama de bloques de un sistema con eco

después de:

$$TD = \text{round}\,(fs \times D).$$

A la señal x[n] original se le añade una versión retrasada y atenuada de si misma con un valor a (a<1) que representa las pérdidas por reflexiones de la señal durante la transmisión.

5.4.2. Reverberación:

La reverberación se define como la suma de múltiples reflexiones del sonido en las diversas superficies de un recinto a la señal de sonido original. Desde cualquier fuente de sonido hay un camino directo hacia cualquier receptor, pero el sonido puede también llegar al receptor a través reflexiones en las paredes o techo.

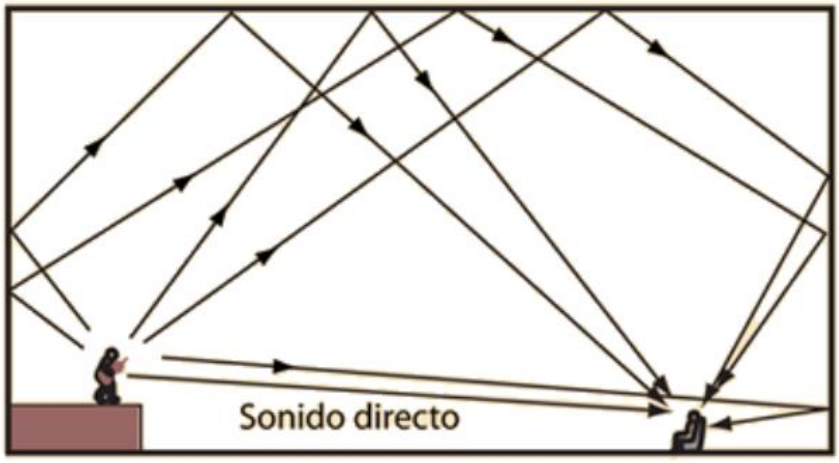


Figura 44: Ondas de sonido alcanzando el receptor por diferentes trayectos

Las ondas de sonido reflejadas que alcanzan al receptor llegan con un cierto retardo respecto de la onda de sonido que alcanza al receptor de forma directa. Además, las superficies del reciento absorben energía de las ondas que sufren reflexiones, de forma que estas ondas llegan también atenuadas. La secuencia de atenuaciones y retardos sobre una señal acústica se conoce como reverberación, siendo un efecto que provoca la sensación de espacio de un recinto. Es importante resaltar que la reverberación no es una serie de ecos como pudiera derivarse. La diferencia es que el eco implica un retraso de la señal original con retardo que se suma a la señal original con retardos mucho mayores y se percibe como señal duplicada).

Sin embargo, en reverberación, las ondas acústicas retardadas llegan en tiempos tan cortos que no se perciben copias del sonido original sino un efecto de entorno o sala. Este tipo de onda que llega típicamente con retardos de tiempo de 10 a 30 ms.

Es decir, si estoy en una sala, y me llegan dos frentes de ondas sonoras, uno directo y otro reflejado en las paredes del recinto, si el desfase es mayor de 1/10 de segundos percibiré dos sonidos distintos y por tanto eco. En cambio, si el desfase es menor percibiré un mismo

sonido prolongado, sentiré la reverberación.

Una señal con reverberación puede calcularse sucesivamente como:

$$y[n] = x[n] + ay[n - TD]$$

En donde el valor de TD viene dado por TD round fs D = × ( ) , siendo D el valor del retardo en segundos y fs la frecuencia de muestreo. Observar, que hasta la muestra TD+1 la expresión que da cuenta de la reverberación no se puede aplicar pues se accedería a una muestra con índice “n” cero o negativo, cosa no permitida en MATLAB. Para remediar este caso, cuando se implemente la reverberación y desde la muestra n=1 hasta la muestra n TD = el valor de la señal reverberada será simplemente:

$$y[n] = x[n]$$

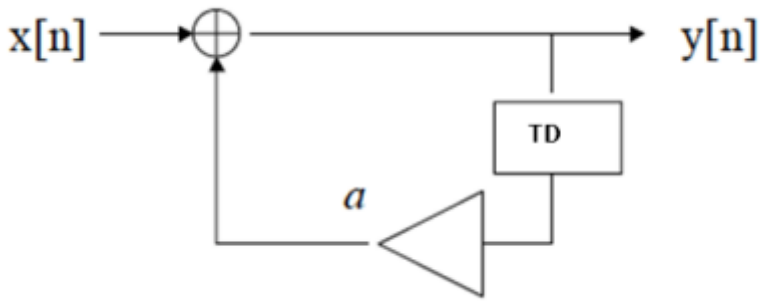


Figura 45: Diagrama de Bloques de un sistema con reverberación

6. Motores de videojuegos:

6.1. Historia de los motores de videojuegos:

Antes de los motores de juego, cada juego se escribía de forma independiente, por lo que había que reescribir todas las características básicas cada vez. Los famosos motores de juego, como DOOM y Quake de id software, Unreal de Epic Game y Source de Valve, han cambiado el desarrollo de juegos. Los motores de juego se han convertido en kits de desarrollo de software reutilizables con todas las funciones, que pueden adquirirse bajo licencia y utilizarse para crear casi cualquier juego imaginable [75].

Las empresas de juegos desarrollan sus propios motores de juego; utilizan sus propios motores de juego para desarrollar sus propios juegos. Más tarde, como algunos de estos motores de juego tienen tanto éxito, otras empresas de juegos también quieren utilizarlos, lo que permite a las empresas de juegos propietarias ganar dinero de otras empresas de juegos. De esta manera, algunas empresas de juegos renuncian a sus motores de juegos internos y pasan a utilizar esos motores de juegos de éxito.

Con el paso del tiempo, algunos motores de juegos muy famosos y de gran éxito evolucionan varias generaciones. Algunos ejemplos de estos motores de juegos son Unreal Engine [76], Unity [77], CryEngine [78], etc.

Todos estos motores de juego pueden ser reutilizados por muchos juegos. En la actualidad, cada vez más empresas de videojuegos tienden a utilizar estos motores de juegos de éxito en lugar de desarrollar sus propios motores de juegos desde cero.

6.2. Motores de juego actuales:

Hoy en día, las principales funcionalidades típicas que ofrecen los motores de juego son un motor de renderizado, un motor físico, un sistema de sonido, un sistema de animación, conexión en red, etc. Además, una de las razones por las que los motores de juego se han hecho más populares entre los desarrolladores de juegos es su compatibilidad con lenguajes de programación de alto nivel como C#, Java, Python, etc.

Los motores de juego no sólo sirven para desarrollar juegos, sino que también se han aplicado a otros campos como la visualización, la formación y la simulación militar.

Entre los cientos de motores de juego que existen, los más populares son Unreal Engine, CryEngine, Source Engine y Unity. Sus plataformas de destino son Windows y videoconsolas como Xbox, PlayStation y Nintendo. Pero ahora, tras el auge de los dispositivos móviles, Unity se ha convertido en la plataforma de desarrollo de juegos dominante a nivel mundial. Unity es fácil de usar y cubre casi todas las plataformas: Móvil, Escritorio, AR, VR, Consola, Web, Smart TVs.

Las tendencias de los motores de juego son la compatibilidad con distintas plataformas, sobre todo para mejorar el rendimiento de las plataformas móviles, y la compatibilidad con los últimos y numerosos dispositivos de realidad virtual y el continuo aprovechamiento del hardware más reciente para mejorar el rendimiento gráfico. En este sentido, cada año en la Game Developers Conference (GDC) se presentan nuevas tecnologías y juegos. Así, en las últimas GDC, la realidad virtual y la realidad aumentada se han convertido en tendencia [79]. Dispositivos de RV como Oculus Rift, Samsung Gear VR, HTC Vive, PlayStation VR11, se han introducido en el mercado de los juegos.

Además, todos los motores de juego más populares son compatibles con estos dispositivos de RV.

Todos los motores de juego intentan mejorar el rendimiento y aprovechar al máximo el nuevo hardware para competir entre sí. Y debido al gran éxito de Unity, los motores de juegos están abriendo su política de licencias para beneficiar más a sus usuarios. Unreal Engine se hizo de código abierto para sus usuarios, de modo que cualquier usuario puede acceder a su código fuente en Github y utilizarlo gratuitamente si sus ingresos son inferiores a 3.000 dólares estadounidenses.

6.3. UNITY:

Para la realización de este TFG se va a utilizar Unity. Unity es un motor de videojuegos multiplataforma. Fue creado por Unity Technologies. Está disponible como plataforma de desarrollo para Windows, MacOS y Linux. Está programado en C, C++ y C#.

Unity Technologies ha desarrollado un motor de videojuegos multiplataforma llamado Unity. Esta herramienta de desarrollo está disponible para Microsoft Windows, OS X y Linux, y cuenta con soporte de compilación para diversas plataformas. Actualmente, numerosas empresas utilizan este software y demandan profesionales capacitados para trabajar en este entorno.

Unity ofrece una optimización de tiempo y características multiplataforma, así como una Store de assets donde se pueden encontrar diversos elementos para crear una aplicación o un videojuego, como modelos, animaciones, sonidos, objetos 3D y más. Además, Unity permite a los usuarios crear y personalizar sus propios elementos.

Este motor de desarrollo es compatible con ambientes 2D y 3D, proporcionando una interfaz sencilla y fácil de manejar. También ofrece una función de prueba que permite a los desarrolladores visualizar y modificar el resultado final de sus creaciones directamente en el motor.

Figura 46: www.unity.com

6.3.1. Historia de Unity:

La historia de Unity Technologies comenzó en 2004, cuando David Helgason, Nicholas Francis y Joachim Ante decidieron cambiar el rumbo de su compañía de desarrollo de videojuegos tras el fracaso de su juego "GooBall". A pesar de que el juego no tuvo el éxito esperado, durante su desarrollo se crearon herramientas muy potentes que sembraron la idea de democratizar el desarrollo de videojuegos.

La idea era crear un motor de videojuegos que pudiera ser utilizado por pequeñas y grandes empresas por igual, con un entorno amigable para programadores, artistas y diseñadores, que pudiera llegar a diferentes plataformas sin necesidad de programar el juego específicamente para cada una de ellas. Esta idea parecía utópica hace diez años, pero hoy en día se ha convertido en una realidad.

Inicialmente, el motor solo estaba disponible para Mac y se presentó en dos versiones: Indie y Profesional. La versión Indie era una opción económica para los pequeños estudios que estaban comenzando y no podían permitirse un gasto considerable. Tenía muchas funciones y su precio empezaba en 300 dólares. Por otro lado, la versión Pro, con un costo de 1.500 dólares, ofrecía todas las funciones del motor. Es importante señalar que incluso en la versión Indie del motor se permitía vender el producto resultante. A pesar de que el motor funcionaba bien, estaba lejos de ser una alternativa al mismo nivel de los grandes motores de la época.

En el año 2008, Unity dio un gran salto al aprovechar el lanzamiento del iPhone y su popularidad, junto con su compatibilidad con la plataforma. Poco después, se lanzó la versión para Android. El éxito de Unity se volvió imparable. En 2009, la compañía dio el paso definitivo en su estrategia al hacer que la versión Indie del motor fuera gratuita para cualquiera que quisiera iniciarse en la plataforma. Incluso con esta versión gratuita, se permitía la distribución del juego.

Figura 47: Primeras versiones de Unity

La idea de Unity fue rápidamente adoptada por los desarrolladores independientes, convirtiéndolo en uno de los motores gráficos más utilizados. Sin embargo, aún necesitaba dar un salto de calidad para conquistar a los estudios de desarrollo que exigían una mayor calidad gráfica. Este salto se logró con la versión 4.0, que incluyó acuerdos con Sony, Microsoft y Nintendo para que el motor fuera compatible con sus sistemas.

La versión actual es compatible con una amplia variedad de plataformas, incluyendo PC, Mac, Linux, iOS, Android, BlackBerry, PlayStation, Xbox, Wii, Wii U y Web, acercando cada vez más el sueño de desarrollar una sola vez y publicar en todas partes. Aunque al final, nunca es tan fácil como parece y se necesitan varios días de ajustes para cada versión.

6.3.2. Unity en el mercado:

El motor de juegos Unity es, con diferencia, la plataforma de desarrollo de juegos dominante en todo el mundo. Desarrollar un videojuego multiplataforma puede ser una tarea abrumadora y los críticos tienen razón al afirmar que es una fuente constante de fallos y errores. Recuerdo la época de auge de Nokia, cuando los valientes estudios de desarrollo que se aventuraban a crear juegos móviles tenían que hacer casi una versión diferente para cada plataforma.

O hace unos años, cuando los "ports" de Xbox 360 a PS3 resultaban en una versión defectuosa en la consola de Sony. El concepto de multiplataforma siempre ha sido complicado, pero con Unity ya no es tan intimidante. El motor ha sido diseñado desde el principio para brindar soporte a todas las plataformas posibles y actualmente presenta un catálogo de opciones envidiable.

La principal ventaja de Unity frente a otros programas de desarrollo de videojuegos radica en sus características. Estas características permiten un aprendizaje más rápido de lo normal, ahorro de tiempo al desarrollar un solo juego para más de 25 plataformas en lugar de crear versiones diferentes para cada dispositivo y la posibilidad de llegar a una audiencia más amplia y universal.

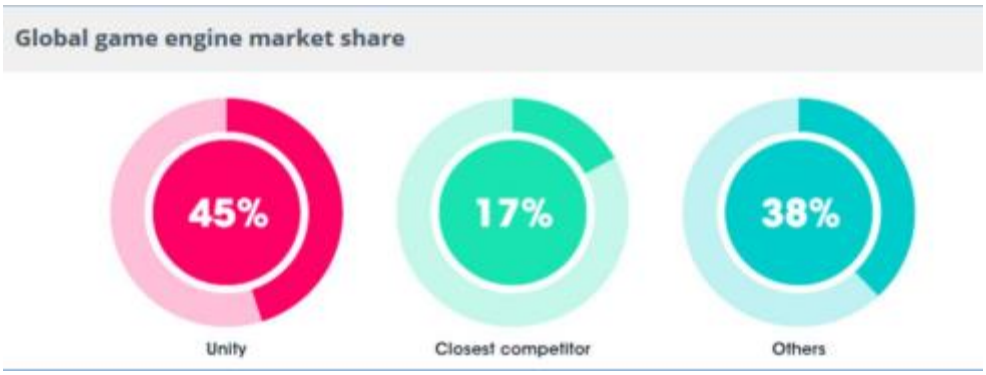


Figura 48: Cuota de mercado mundial de motores de juego

6.3.3. Formación de Unity:

Los programadores de Unity pueden utilizar C#, UnityScript o Boo como lenguajes de programación. En el sitio web oficial hay muchos recursos de formación gratuitos. Para los principiantes, los tutoriales son el mejor punto de partida. Estos tutoriales están categorizados por temas; es fácil encontrar el material que el desarrollador necesita en cada momento. La documentación de Unity es el mejor lugar para buscar la descripción de las APIs de Unity.

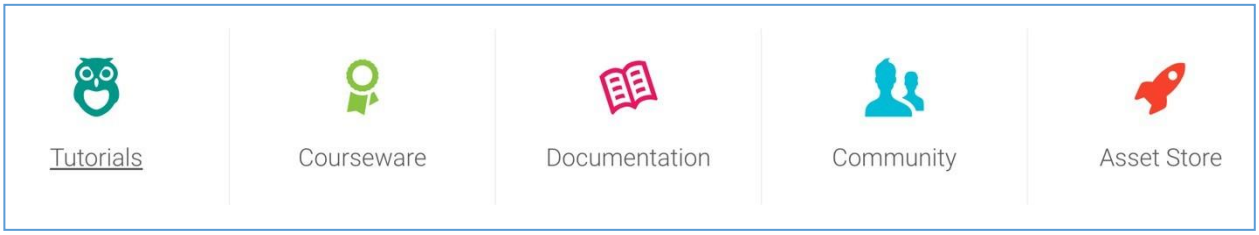


Figura 49: Soportes Unity

6.3.4. Tienda de Unity Assets:

La tienda de activos de Unity proporciona más de 15000 modelos 3D gratuitos y de pago, extensiones del editor, scripts, shaders, materiales, archivos de audio, animaciones y mucho más para ayudar a los desarrolladores a potenciar sus proyectos de Unity. Es el mejor lugar para encontrar algunos activos básicos para iniciar un prototipo o demo.

El editor de Unity tiene algunos activos estándar incluidos, que los desarrolladores pueden utilizar para crear prototipos o demos, e incluso un proyecto completo. Gracias a los activos estándar de Unity y a la gran cantidad de activos gratuitos y de pago de la tienda de activos de Unity, un pequeño equipo formado únicamente por programadores con un presupuesto limitado puede iniciar y finalizar un proyecto sin necesidad de diseñadores 3D.

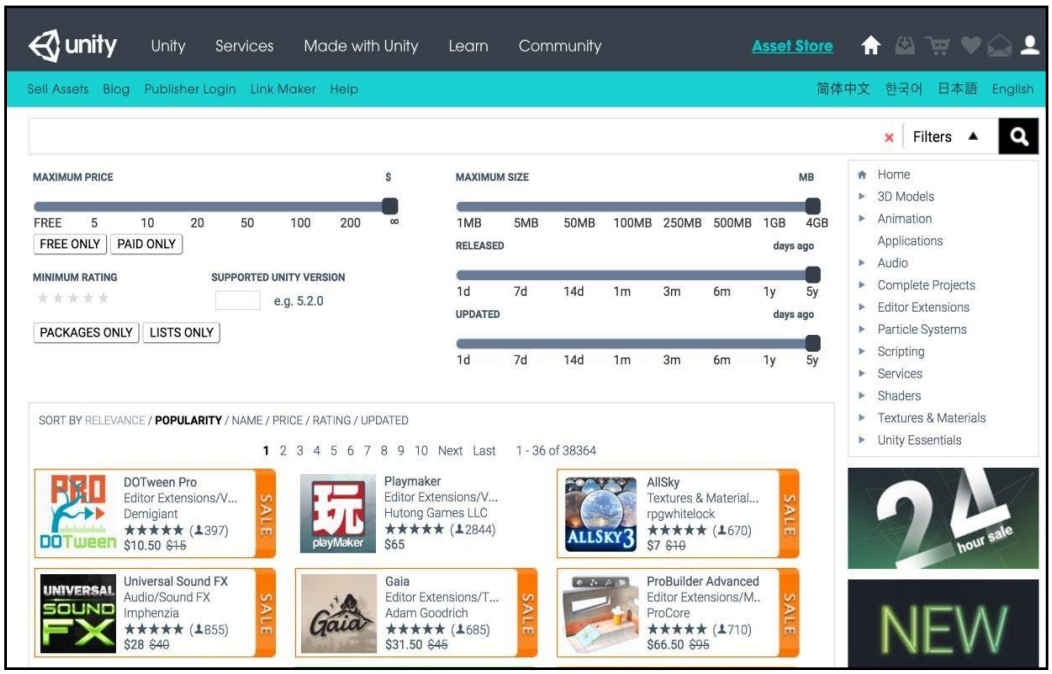


Figura 50: Unity Asset [80]

6.3.5. Utilidades de Unity:

Unity no es sólo un motor de juegos para desarrollar juegos, también integra muchas unidades y servicios útiles para ayudar a los desarrolladores de juegos a analizar sus juegos, mejorar su proceso de desarrollo, proporcionar servicios de infraestructura fundamentales para crear fácilmente juegos multijugador que les permitan monetizar sus juegos.

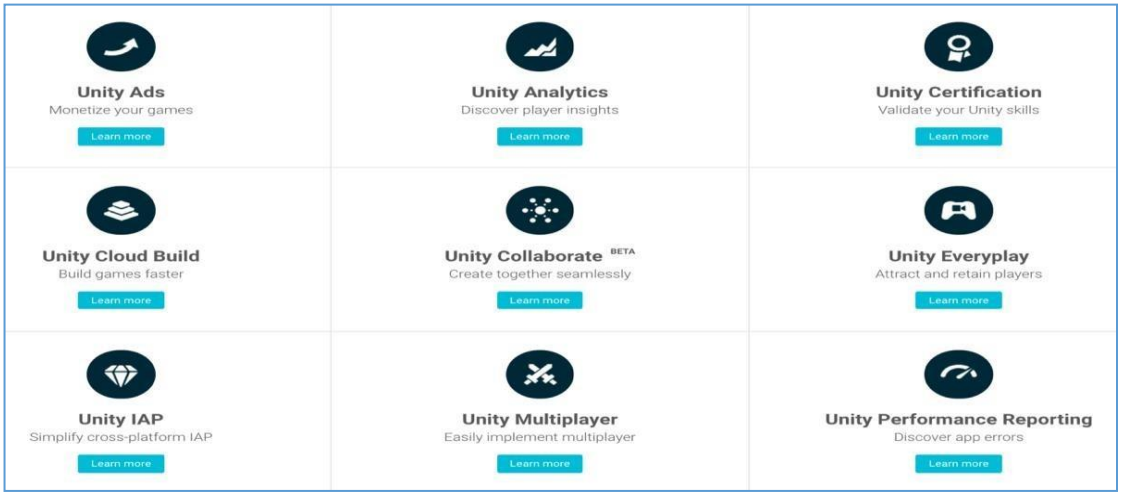


Figura 51: Utilidades de Unity [81]

6.3.6. Licencias de Unity:

Unity ofrece 4 licencias para diferentes tipos de desarrolladores. Para equipos pequeños o desarrolladores individuales, la licencia personal es gratuita. Para cualquiera que comience a usar Unity, la versión personal es la mejor elección. La política de licencias de Unity es muy acertada. Esa es una de las razones por las que está dominando el mercado de los motores de juegos. Por ello, otros motores de juegos como Unreal tuvieron que cambiar su antigua política de licencias.

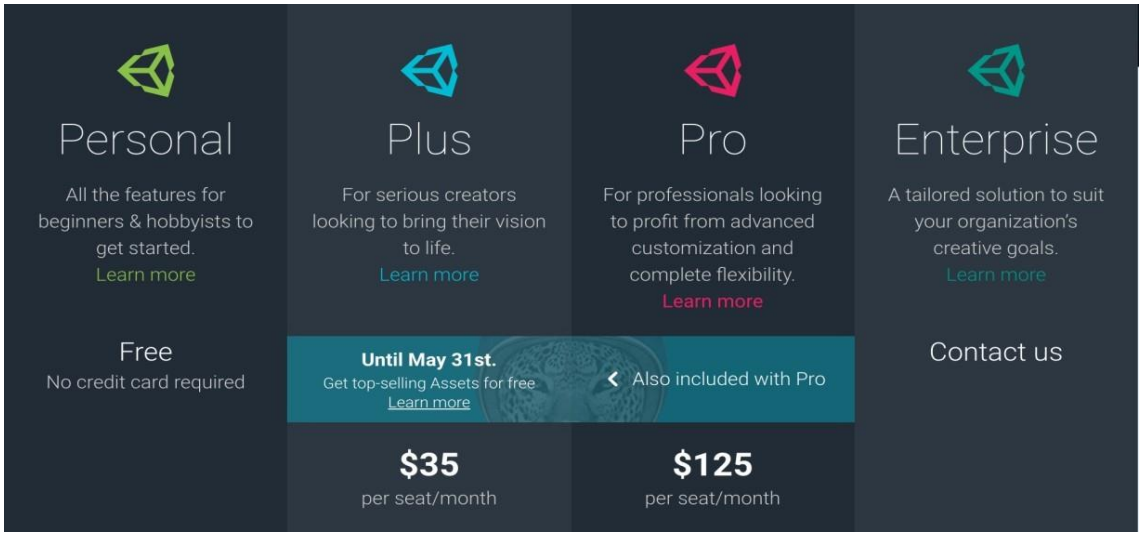


Figura 52: Licencias de Unity [82]

6.4. Netcodes - Unity Multijugador:

Casi todos los juegos multijugador deben tener en cuenta y resolver problemas inherentes a la red que afectan a la experiencia de juego, como la latencia, la pérdida de paquetes y la gestión de escenas. Los juegos resuelven estos problemas de distintas maneras.

Encontrar la solución adecuada depende del género del juego, la escala de sus jugadores y objetos en red, la competitividad y otros aspectos, como cuánto control se necesita sobre la capa de red. Distintos escenarios requieren soluciones de netcode diferentes.

6.4.1. Netcode:

Netcode es un término de alto nivel que muchos ingenieros utilizan para referirse a las estructuras diseñadas específicamente para facilitar la creación de ciertos aspectos de los juegos en red, como la sincronización de datos o la compensación de retrasos.

Una estructura totalmente en red contiene dos componentes esenciales:

- Una capa de transporte (base) que gestiona todo el tráfico y los paquetes enviados a/desde un cliente, host o servidor.
- Una capa de abstracción de nivel superior que simplifica las necesidades comunes de juego en red y se integra con las herramientas.

A menudo, una "solución de código de red" se refiere a la segunda capa de abstracción, ya que es aquí donde se implementan el juego en red y las optimizaciones. Las distintas soluciones de netcode tienen diferentes limitaciones y capacidades que pueden facilitar o dificultar la creación de tu experiencia multijugador. Es importante saber exactamente lo que quieres construir y evaluar tus opciones antes de empezar, para ayudar a reducir refactorizaciones que podrían ser costosas.

Figura 53: Netcodes

6.4.2. Mejores netcodes en el mercado:

Como se puede ver en la figura 54, hay muchos criterios para saber cual es el mejor netcode en el mercado. Para mi caso, el precio es el criterio más importante, luego viene la facilidad de uso con Unity, y por eso he elegido Photon Pun2.

	Stability/ support	Ease-of-use	Perfor-mance	Scalability	Feature breadth	Cost*	Customers recommend for
MLAPI	★★★★★	★★★★☆	★★★★★	★★★★☆	★★★★★	Free	Most client-server games for up to ~64 players that want a stable breadth of mid-level features
DarkRift 2	★★★★★	★★★☆☆	★★★★★	★★★★★	★★★☆☆	$100 for source	Games with high perf/ scale requirements that want to build on a stable LL layer
Photon PUN	★★★★☆	★★★★★	★★★☆☆	★☆☆☆☆	★★★★☆	$0.30/PCU	Simple and small (2–8 players) mesh-topology games
Photon Quantum 2.0	★★★★☆	★★★★☆	★★★★★	★★☆☆☆	★★★★☆	$1000/mo + $0.50/PCU	Games desiring deterministic roll-back, like MOBA games, for up to 32 players
Mirror	★★★★★	★★★★★	★★★☆☆	★★★★☆	★★★★☆	Free	Stable and proven client-server solution, loved best for its community and ease-of-use

Figura 54: Comparativa de los mejores netcodes [83]

6.4.3. Como elegir el netcode:

La figura 55 es una figura explicativa de como elegir el netcode perfecto para tu juego. Es el caso para los juegos más completos, más profesionales. Esta clasificación no toma en cuenta el precio, pero si la escalabilidad, que es el criterio más importante para los desarrolladores.

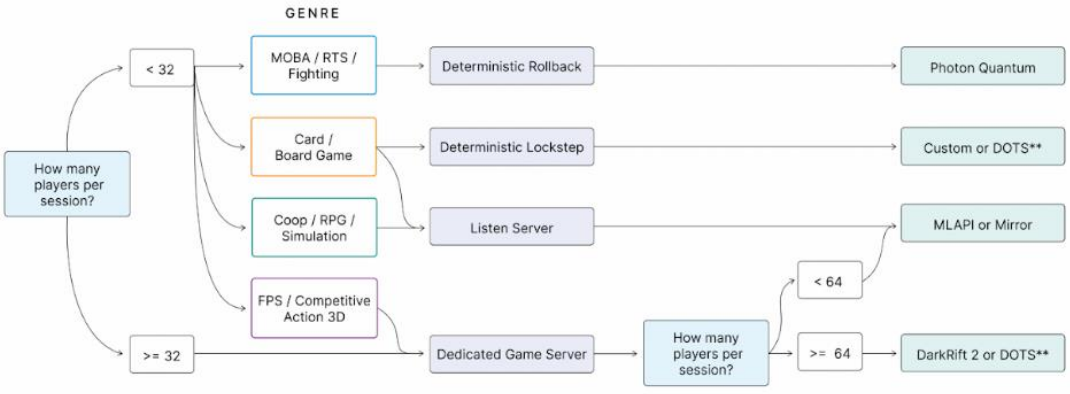


Figura 55: Como elegir la mejor solución [83]

6.5. Chuck:

ChucK es un lenguaje de programación para la síntesis de sonido y la creación musical en tiempo real. ChucK ofrece un modelo único de programación concurrente basado en el tiempo que es preciso y expresivo (lo llamamos fuertemente temporizado), tasas de control dinámicas y la posibilidad de añadir y modificar código sobre la marcha. Además, ChucK es compatible con MIDI, OpenSoundControl, dispositivos HID y audio multicanal. Es de código abierto y está disponible gratuitamente en macOS, Windows y Linux. Es divertido y fácil de aprender, y ofrece a compositores, investigadores e intérpretes una potente herramienta de programación para construir y experimentar con complejos programas de síntesis/análisis de audio y música interactiva en tiempo real.

Figura 56: Chuck [84]

6.6. Chuck con Unity - Chunity:

Chunity[85] (ChucK para Unity) es un entorno de programación para el diseño de juegos, instrumentos y experiencias audiovisuales interactivas. Adopta un enfoque basado en el sonido, que integra la programación de audio y la programación gráfica en el mismo flujo de trabajo, aprovechando las características de programación de audio en tiempo real del lenguaje de programación ChucK y el motor gráfico en tiempo real de Unity. En resumen, Chunity permite programar ChucK dentro del marco de desarrollo de juegos Unity y proporciona mecanismos de comunicación entre ChucK y C#/Unity. Chunity existe como un plugin de Unity, y se describió originalmente en un artículo de 2018[86] de Jack Atherton y Ge Wang.

Figura 57: Chunity

VI. Desarrollo:

Empezamos nuestro desarrollo presentando las herramientas software utilizada para llevar al cabo nuestra aplicación.

1. Herramientas y programas usados para el desarrollo de la aplicación:

1.1. Unity:

El motor Unity fue utilizado para el desarrollo del video juego en su versión 2019.2.12f1. Dentro de los tipos de licencia disponibles, se utilizó la versión personal (gratuita). Este software es el núcleo de la aplicación. Es el que une todas las otras herramientas, par a conseguir al final una aplicación completa. En este software se implementa la interfaz gráfica de la aplicación.

1.2. Visual Studio:

Se utilizo la versión 2022 del editor Visual Studio (Community) para la visualización de los scripts de la aplicación. Este IDE lo utilizamos como una herramienta de completado de código, estilo de código y depuración.

1.3. Chuck:

Se utilizo Chuck 1.4.1.0 para los scripts relacionados con la interacción musical. Chuck nos ofrece la posibilidad de leer archivos de audio con unos códigos propios, y sobre todo los archivos MIDI. Porqué en el caso de no utilizar Chuck, tendremos que utilizar un Asset de Unity, que habrá que descargar desde “Unity Assets Store”. Este asset se llama “MIDI Player ToolKit” o MPTK. En mi caso, no se ha podido instalar correctamente por la versión. Pero de todos modos, resulta mucho más fácil trabajar con Chuck.

1.4. C#:

Se utiliza el lenguaje C# para el desarrollo de los scripts. C# es un lenguaje de programación orientado a objetos moderno, utilizado en la mayoría de las aplicaciones Unity por su integración con el entorno de desarrollo.

Unity utiliza la versión 8 de C#.

1.5. Photon Pun2:

Photon PUN 2 se refiere a Photon Unity Networking 2, que es una herramienta de red gratis desarrollada por Exit Games. Photon PUN 2 es un conjunto de API (interfaces de programación de aplicaciones) diseñado para facilitar la implementación de funcionalidades de red en juegos desarrollados con Unity.

Photon PUN 2 nos permite pasar el juego a multijugador en tiempo real y conectar a los jugadores a través de la red. Proporciona una serie de características y funcionalidades esenciales, como la sincronización de objetos en tiempo real, la gestión de conexiones de red, la creación y unión de salas de juego, la comunicación entre jugadores y muchas otras capacidades relacionadas con la red.

1.6. Matlab:

MATLAB es un lenguaje y entorno de programación de alto nivel muy utilizado en los campos científico y de ingeniería. Sus siglas significan "MATrix LABoratory" y se desarrolló inicialmente para el cálculo numérico y la manipulación de matrices.

MATLAB proporciona una amplia colección de funciones integradas y cajas de herramientas que cubren una gran variedad de campos matemáticos y científicos. Estas funciones incorporadas permiten a los usuarios realizar cálculos complejos, resolver problemas matemáticos, manipular matrices y matrices y crear representaciones visuales de datos.

En nuestro proyecto, MATLAB se utiliza como una potente herramienta para trabajar con señales de audio y realizar diversas tareas relacionadas con el procesamiento y el análisis de audio. El MATLAB en nuestro caso se utilizará para:

- E/S de archivos de audio: MATLAB proporciona funciones para leer y escribir archivos de audio en varios formatos, como WAV, MP3 y FLAC. Esto permite cargar datos de audio en MATLAB para su posterior procesamiento o guardar el audio procesado de nuevo en un archivo.
- Procesado de señales de audio: MATLAB ofrece un amplio conjunto de funciones de procesado de señal que le permiten realizar tareas como compresión de audio, muestreo de audio, estiramiento del tiempo y síntesis de audio. Estas funciones proporcionan la flexibilidad y el control necesarios para algoritmos avanzados de procesamiento de audio

2. Arquitectura de la aplicación:

Nuestra aplicación se presenta en la arquitectura de la figura 58.
Como se puede ver, Empezamos con un procesamiento previo de los audios de la guitarra y de los drums. Eso es debido a que el procesamiento de audio en MATLAB es bastante más sencillo que utilizar otras librerías en Unity. En mi caso, como la versión utilizada es un poco antigua, la mayoría de esas librerías no están disponible para mi versión. Por ejemplo, hay un filtro en unity llamado “Audio Reverb Filter” que nos permite añadir un efecto de reverberación a un audio dentro de Unity.

Al utilizar ese filtro, habrá que importar los audios como componentes de la jerarquía del proyecto, que no es el caso aquí porqué Chuck se encargará de la carga de los audios.

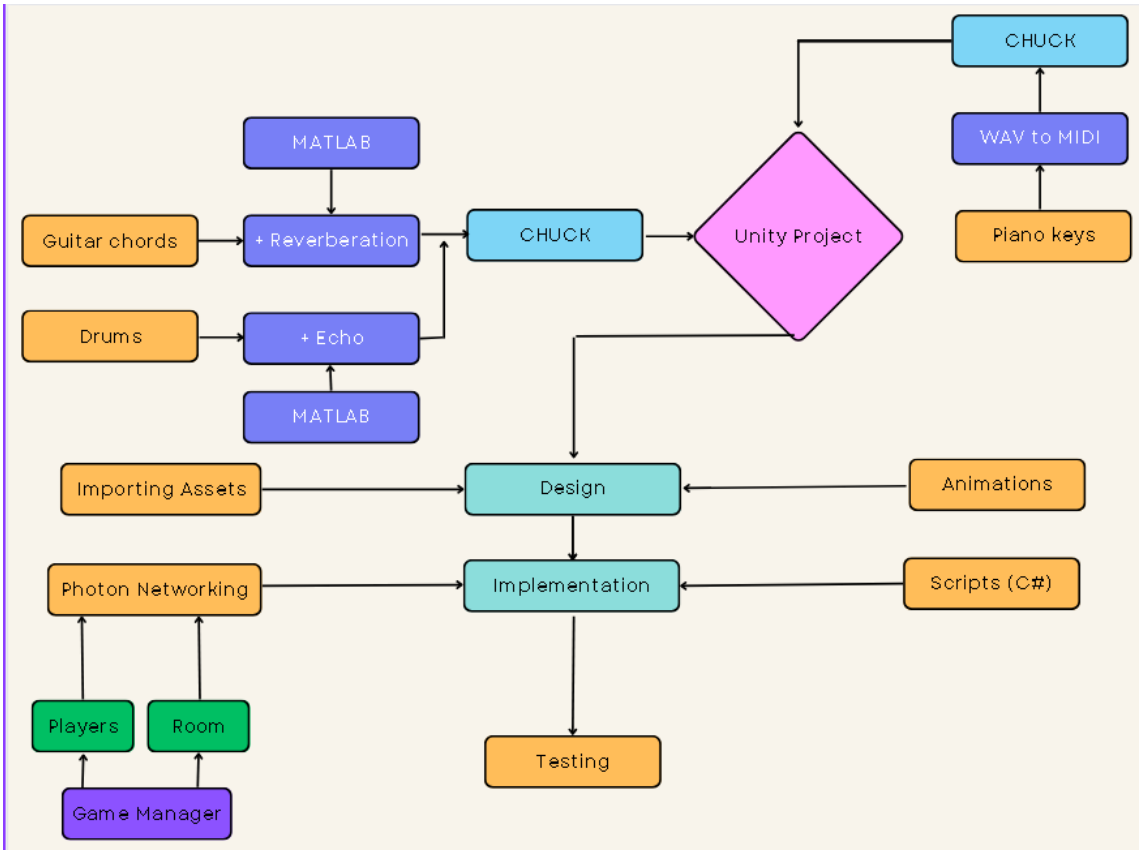


Figura 58: Arquitectura de la aplicación

Después del procesamiento de audio, se procederá a la carga de ellos utilizando Chuck, e importando todos los assets necesarios para la aplicación. Al tener toda la herramienta necesaria, nos ponemos a redactar los scripts y animaciones necesarias para la realización de las pruebas del escape room. Finalmente, antes de hacer las pruebas, se añadirá la parte de multijugador, con la ayuda de Photon. Todo el procedimiento vendrá más explicado en el próximo apartado.

3. Procedimiento:

En este capítulo, se va a desarrollar la explicación de como se ha realizado la aplicación, una vez expuestas las herramientas utilizadas.

3.1. Instalación del Software:

Previamente, antes de comenzar la implementación de la arquitectura del sistema, se debe instalar Unity. En mi caso, he instalado Unity 2019.2.12f1. Para ello, lo primero es instalar Unity Hub de la página principal de Unity https://unity.com/download .

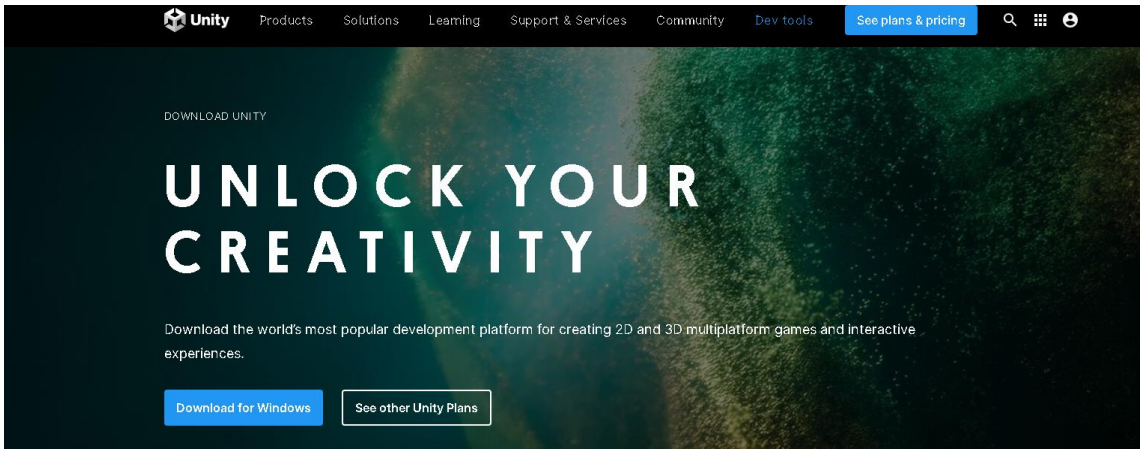


Figura 59: Descarga de Unity Hub

Luego, hay dos opciones:

- Descargar la versión de Unity directamente desde Unity Hub. Esta opción es limitada. Las versiones de Unity que se pueden descargar desde Unity Hub suelen ser las últimas disponibles.
- Descargar la versión que queremos de los archivos de Unity https://unity.com/es/releases/editor/archive , con la opción de instalarlo en Unity Hub.

La descarga de Unity conlleva la descarga de Microsoft Visual Studio, donde se implementarán los scripts necesarios utilizando C#.

Otra cosa habrá que elegir es los SDK de Android o IOS dependiendo del objetivo de la aplicación.

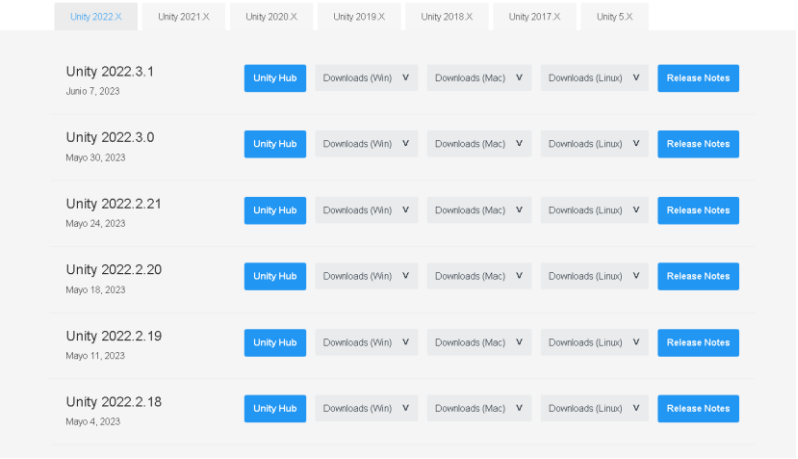


Figura 60: Descarga de Unity

Lo siguiente es descargar Chuck desde su página oficial https://chuck.stanford.edu/ .

Figura 61: Descarga de Chuck

El siguiente paso es instalar MATLAB de su página principal. Las versiones completas de MATLAB suelen ser caras, por eso las universidades suelen facilitar claves de acceso a sus estudiantes.

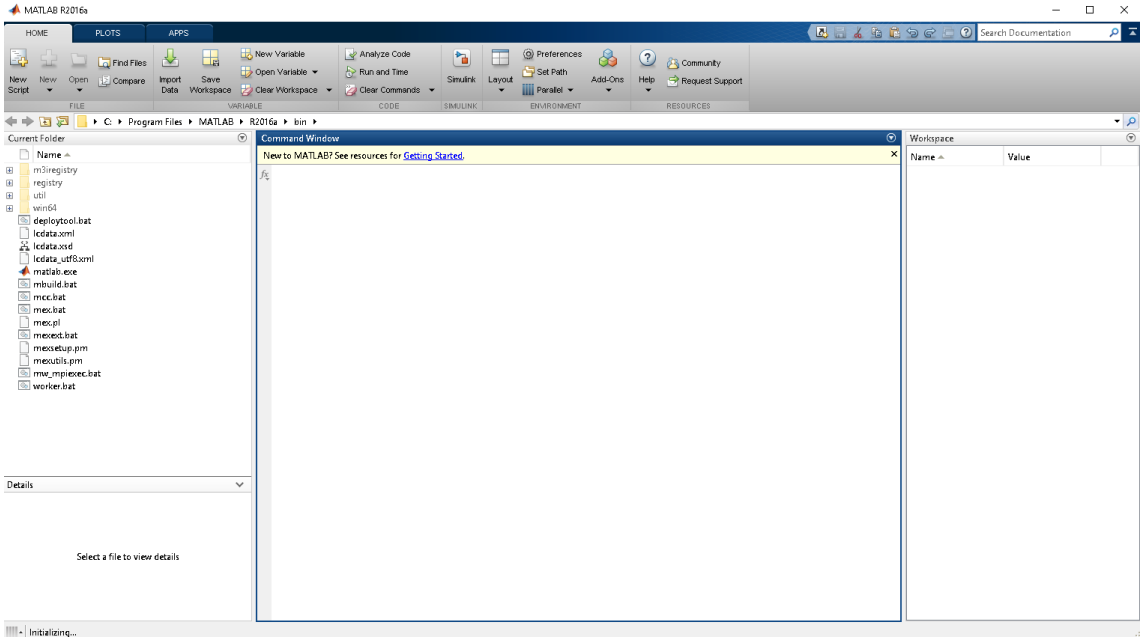


Figura 62: MATLAB

3.2. Preparación de los sonidos para el proyecto:

Como hemos comentado antes, en el primer paso de la arquitectura, se preparan los sonidos para su futura implementación en las próximas fases del proyecto:

- Para los sonidos que se utilizarán para cada tecla del piano, se han convertido los ficheros .wav, a ficheros .mid (utilizando herramientas Online gratuitas). El lenguaje Chuck nos permite leer los ficheros MIDI fácilmente (Ver Anexo A, 9.SoundMIDI.cs).
- Para los sonidos de las cuerdas de la guitarra, se ha implementado un código MATLAB para añadirles un efecto de reverberación. (ver Anexo A, 1.reverberacion.m).
- Para los sonidos de los drums, se ha implementado un código MATLAB para añadirles un efecto de eco. (ver Anexo A, 2. Eco.m).

3.3. Creación del proyecto en Unity y configuración inicial:

Después de tener todo instalado, se procede a la creación del proyecto “escape room” en Unity.

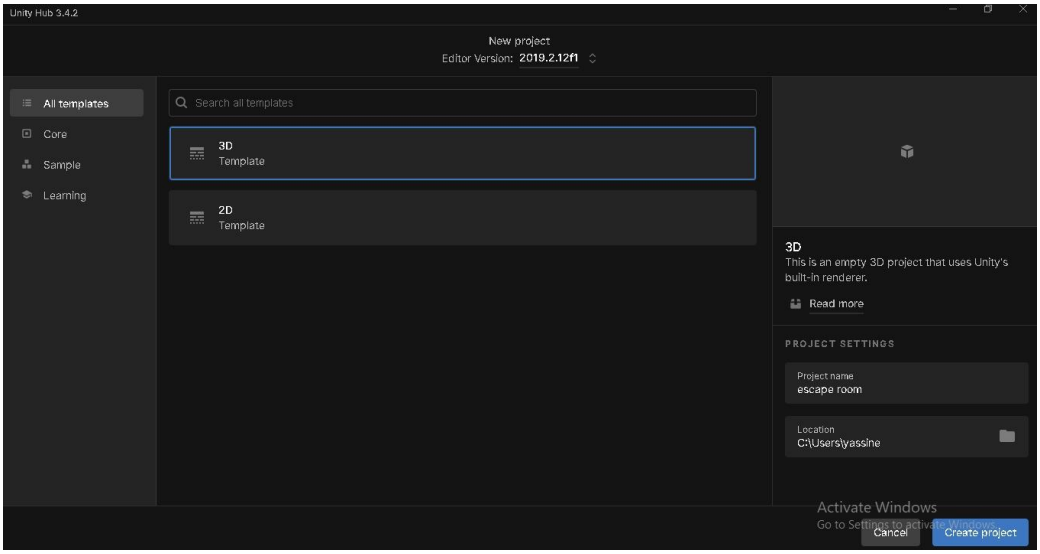


Figura 63: Creación del proyecto en Unity

Después de crear el proyecto de Unity, necesitamos descargar Photon Pun2 desde Unity Asset Store, e importarlo al proyecto. Hay dos versiones de Photon Pun2, elegimos la gratuita.

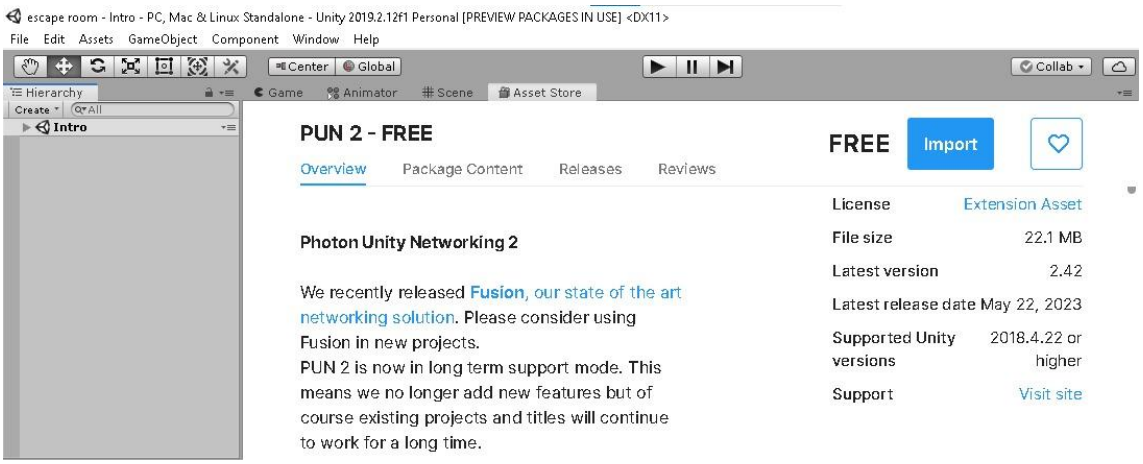


Figura 64: Importar Pun2

Lo primero es crear una cuenta en www.photonengine.com . Al descargar Photon, te pedirá primero el App Id de Pun. Luego se puede acceder al App Id de esta forma:

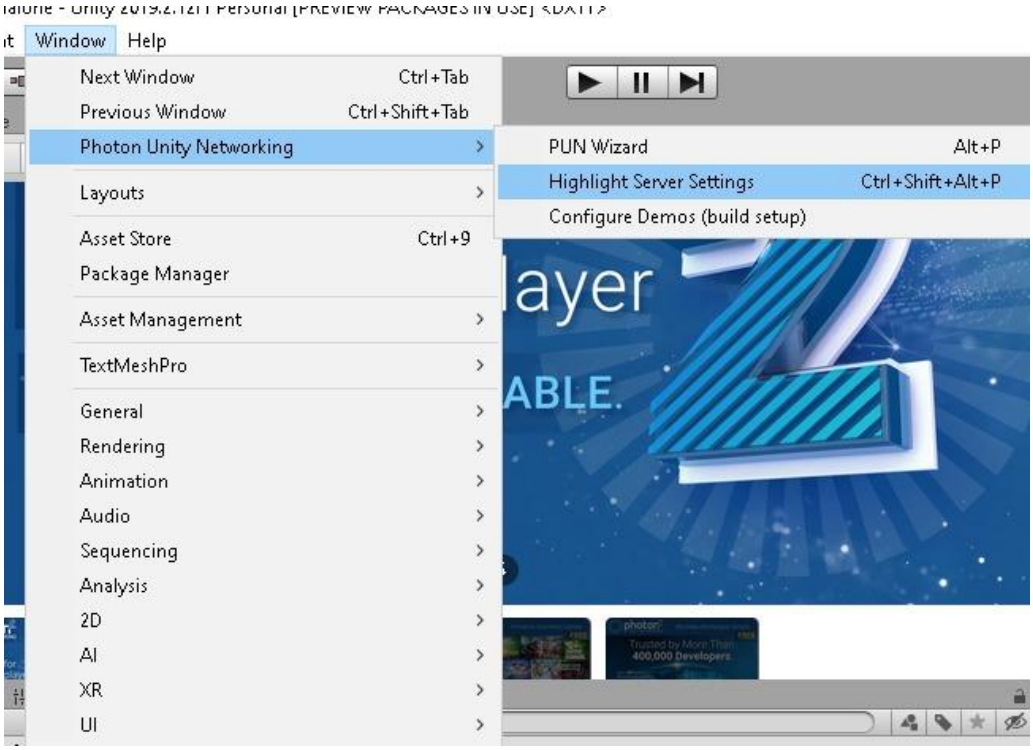


Figura 65: Acceder a la configuración del servidor de Photon

Antes de todo, hay que copiar el App ID y pegarlo en Unity.

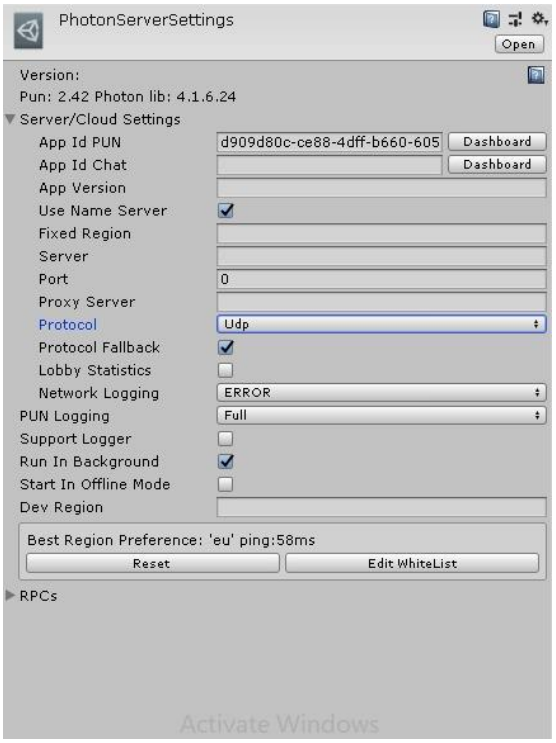


Figura 66: PhotonServerSettings en Unity

¿Dónde podemos encontrar nuestro App ID? Por defecto, al crear una cuenta en Photon Engine, se crea una aplicación “Pun” para multijugador y otra “Chat” para la implementación de un chat en el juego. Se pueden visualizar esas aplicaciones en el “Dashboard” de tu cuenta, donde se puede monitorizar el rendimiento de tu juego, cuantos usuarios conectados…

En mi caso, he utilizado solo la aplicación “Pun”, cambiando su nombre a “Multiplayer VR game”.

Figura 67: Aplicación Pun

Desde allí, se puede copiar el App ID y pasarlo a Unity. Luego, podemos cambiar los protocolos de transporte (en mi caso se utiliza UDP porque es más simple y rápido). Otra cosa muy importante es el tema de regiones.

Photon Cloud proporciona conectividad global para permitirle jugar con baja latencia en todo el mundo. Esto se consigue alojando servidores en varias regiones. Como las regiones disponibles pueden cambiar a lo largo de un proyecto, los clientes obtienen la lista actual de regiones de los servidores de nombres Photon. Cada región está completamente separada de las demás y consta de un Servidor Maestro (para matchmaking) y Servidores de Juego (que alojan salas). La mejor solución es dejar las secciones de “Dev Region” y “Fixed Region” vacías. En este caso, Photon automáticamente elige la mejor región (en nuestro caso será “eu”, de Europa).

Otra cosa importante para importar es el package “XR Interaction Toolkit” desde Window>Package Manager. En mi caso no lo voy a usar. Será necesario en los juegos de Realidad Virtual con el uso de Headsets.

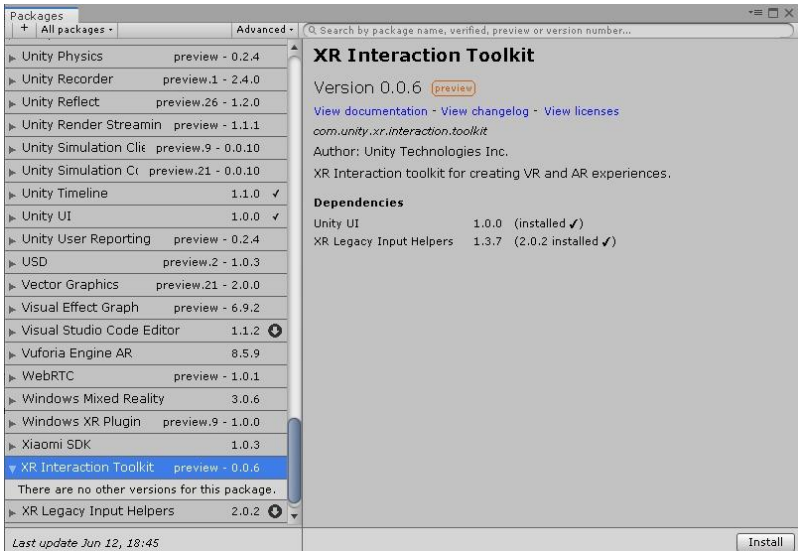


Figura 68: XR Interaction Toolkit

Este package nos permite el uso de una cámara VR. La aplicación de este proyecto no tiene una cámara VR, por la falta de un Headset (Oculus, OpenVR…). Pero el cambio de una cámara normal a una cámara de RV es muy sencillo. Solo hace falta crear en la jerarquía de cada escena una “Room-scale XR Rig”, y dentro de ese Rig pegar la cámara ordinaria que tenemos montada.

En mi caso, como mencioné antes, por falta de los dispositivos de realidad virtual, mi aplicación consiste en una experiencia de realidad virtual no inmersiva.

3.4. Creación de las escenas y la estructura de la aplicación:

Esta sección representa la justificación detrás de mi elección de esta versión de la plataforma Unity. Uno de los desafíos asociados con Unity es la compatibilidad entre sus diferentes versiones. Unity lanza más de 40 versiones al año, lo cual es beneficioso ya que cada versión aborda diversos errores o "bugs". Sin embargo, esto también conlleva la introducción de nuevas bibliotecas y funcionalidades. En consecuencia, existe ocasionalmente la falta de compatibilidad entre estas bibliotecas y las versiones más antiguas de Unity, incluso dentro del mismo año.

Después de la creación del proyecto, llega la parte de diseño. En esta parte, después de importar los audios, se procede a importar los assets y las animaciones en cada escena.

Para mi aplicación, he creado 6 escenas principales:

- Intro: Es la escena que aparece al ejecutar el juego. El jugador introducirá su NickName, y dará a Play para empezar a jugar e ir a la siguiente escena.

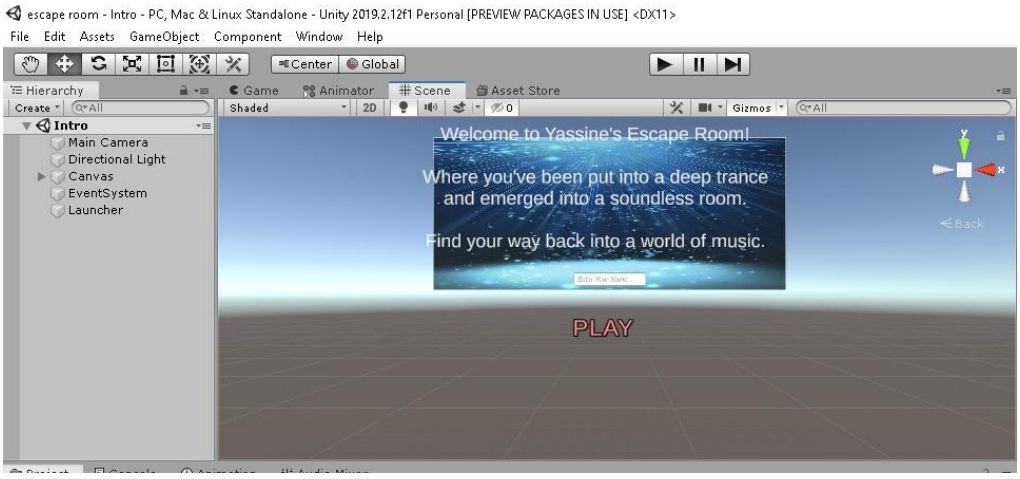


Figura 69: Intro Scene

Para montar esta escena, se crea un panel como backgroud, luego se le puede añadir el texto que queremos mostrar allí (TextMeshPro). El “Name InputField” es lo que se utiliza para identificar cada jugador (Se utiliza para Photon). El “PlayButton” es el botón que utiliza para empezar el juego.

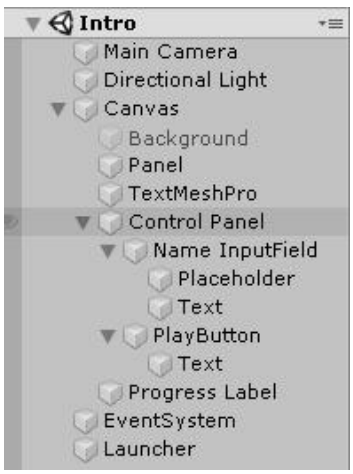


Figura 70: Jerarquía de la escena

El elemento “Launcher” solo contiene el script principal de esta escena (Ver Anexo A, 3.Launcher.cs).

En ese script, se implementan las funciones necesarias para empezar el juego:

- Connect(): Es la función asignada al Botón “PlayButton”. Cuando se pulsa el botón, se conecta al servidor de Photon.
- OnConnectedToMaster(): Cuando se conecta al servidor, se activa esa función. En mi caso la utilizo para crear o entrar a una sala con nombre “Room Yassine”.
- OnDisconnected(): Cuando se desconecta un jugador, la variable booleana “IsConnecting” vuelve a false, para que pueda conectarse otra vez.
- OnJoinedRoom(): Cuando el jugador ya se une a la sala creado. Es cuando se pasa a la siguiente escena “EscapeScene”.
- OnPlayerEnteredRoom(Player newPlayer): Cuando se une otro jugador a la sala.

Todas estas funciones son funciones predefinidas o Callbacks de Photon. Por eso, se utilizan regiones para limitar cada parte del código y asignarla a la región donde aplican.

- EscapeScene: Es el Lobby del juego, es la escena más importante. Esta escena tiene todos los instrumentos o fases del juego, todas las claves para superar esos pasos… En esta escena podemos encontrar:
    - Un piano: Acercando al piano, nos lleva a la primera prueba del escape room.
    - Una guitarra: Acercando a la guitarra, nos lleva a la segunda prueba del escape room.
    - Unos drums: Acercando a los drums, nos lleva a la tercera y última prueba del escape room.

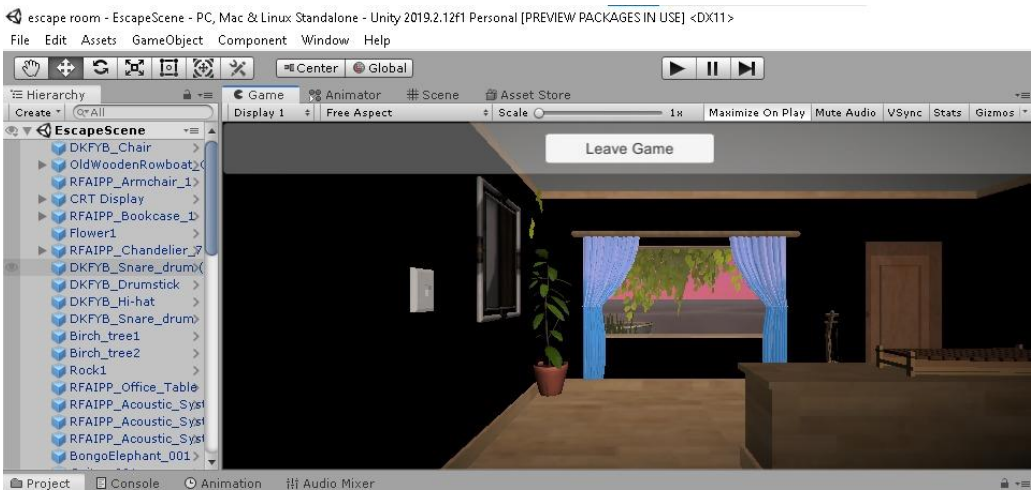


Figura 71: EscapeScene

Esta escena es la más importante porqué es la que más tiene assets. La mayoría de los assets están importados de otros proyectos que miré por Internet, otros vienen de Unity Asset Store.

Esta escena tiene el GameManager. Este elemento de la jerarquía de la escena solo se utiliza para tener el script principal de Photon en esta escena (ver Anexo A, 4.GameManager.cs). Este script solo tiene las funciones Callback de Photon que estén relacionadas con el botón “Leave Game” para salir del juego y volver a la escena principal.

Otro elemento importante de esta escena es “The Chuck”. Este elemento tendrá el script principal de Chuck para poder llamarlo en cuando necesitamos lanzar un archivo de audio… Chuck nos permite leer un archivo de audio fácilmente, dentro del script de C# (Ver Anexo A, 5. RoomSound.cs). Este elemento estará en todas las escenas donde haya sonido en el fondo.

El resto de los elementos, son assets que se activarán o no (se activarán las animaciones correspondientes) dependiendo del estado del juego.

Por ejemplo, En “ArtRoom”, hay un tablero “Paintings” que se utiliza para saber que notas habrá que tocar en el piano. Cuando se acerca el jugador, aparece un texto “note: remember this…”.

Luego, otros elementos importantes son los “Cube SCENE SWITCH” en frente de cada instrumento. Estos cubes se utilizan para que, si el jugador toca uno de ellos, se ejecuta

el script correspondiente, necesarios para ir a las escenas de cada elemento. (ver Anexo A, 7. SceneSwitchPiano.cs, 10. SceneSwitchGuitar.cs y 14.SceneSwitchDrums.cs).

Por ejemplo, en el caso del piano:

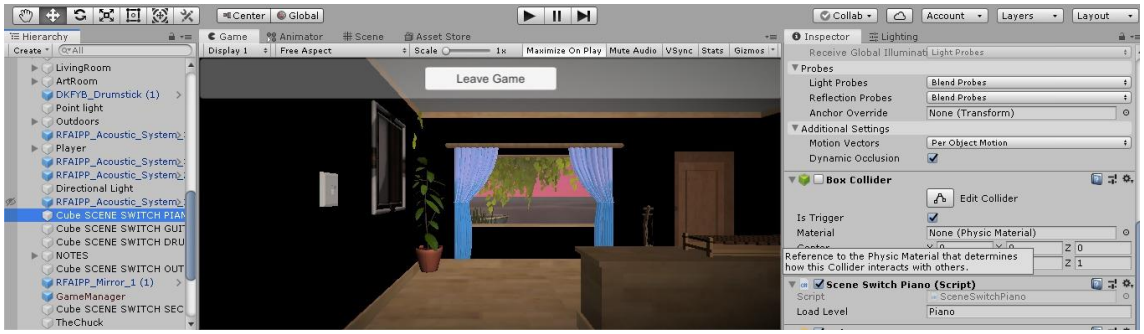


Figura 72: Cube SCENE SWITCH PIANO

- Piano: Tercera escena del juego. En esta escena solo hay las teclas del piano.

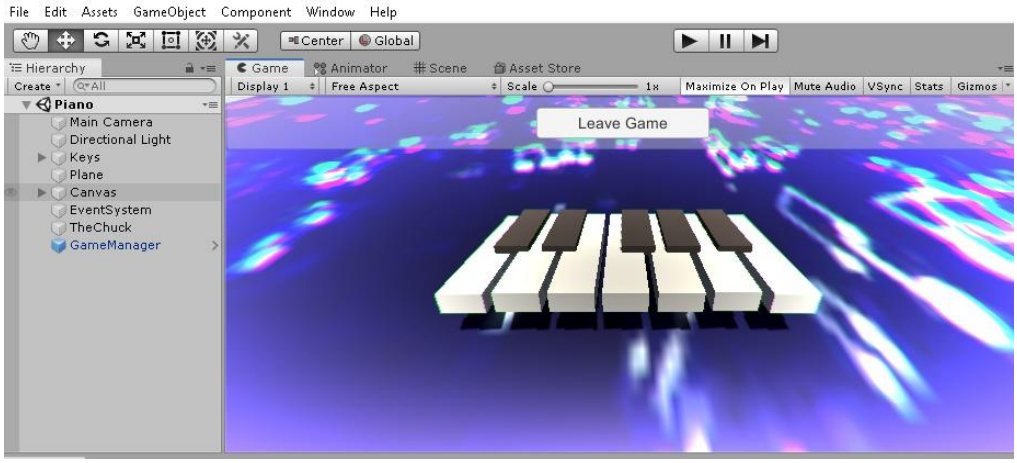


Figura 73: Piano scene

Esta escena tiene su elemento “GameManager” igual que la escena “EscapeScene”. Se utiliza para poder salir del juego. Cuando un elemento se utiliza en varias escenas, se guarda como prefab en los assets, para facilitar el trabajo, y por eso aparece en color azul.

Cada tecla tendrá su elemento en la jerarquía de la escena, para poder referenciarlo dentro del script (ver Anexo A: 8.PressKeys.cs).

Cuando se tocan las teclas correctas, aparecerá un texto “COMPLETE” y se volverá a la escena “EscapeScene”. Lo que se nota ahora es que al sonido de fondo de la Escape scene se le ha sumado un sonido de Piano

- Guitar: Cuarta Escena del juego. Para esta escena, quería cambiar el tipo de juego. Como no estoy utilizando una cámara de realidad virtual, no tengo muchas posibilidades. Por lo que he decidido que en esta parte "regalada" del juego, habrá que tocar las teclas correctas con el teclado.

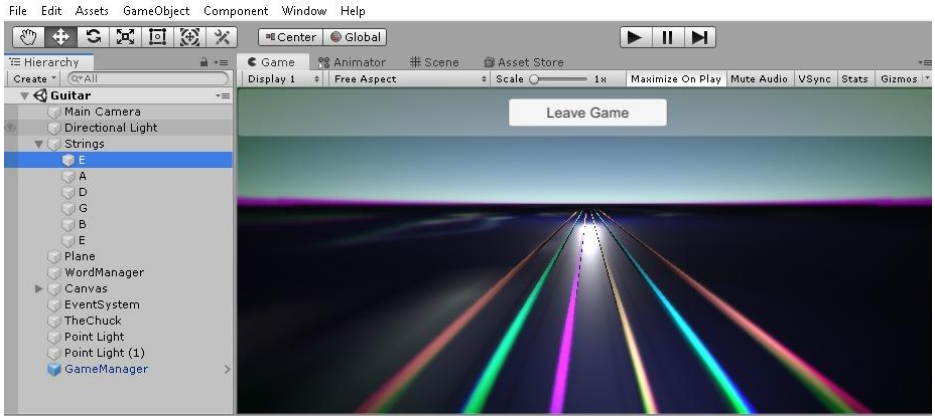


Figura 74: Guitar Scene

Para esta parte, he utilizado el sistema de partículas de Unity. El Sistema de Partículas en Unity es un robusto sistema de efectos de partículas donde puedes simular líquidos en movimiento, humo, nubes, llamas, hechizos mágicos, y un montón de otros efectos. En este caso, lo he utilizado para cada cuerda de la guitarra. Este sistema nos permite hacer animaciones para cada cuerda. (ver Anexo A, 11. ParticleA.cs, 12. ParticleB.cs, 13.ParticleD.cs, 14. ParticleE.cs y 15. ParticleG.cs).

Al principio de este juego, aparecerá 3 palabras que hay que seguir pulsando con el teclado. Cada vez que se toca una tecla, se desaparecerá la letra, hasta tocar todas las teclas necesarias.

Para llevar al cabo esta parte, se hicieron los scripts necesarios. (ver Anexo A, 17.Word.cs, 18.WordInput.cs, 19.WordDisplay.cs,20.WordManager.cs). Básicamente, estos scripts son muy sencillos, el Manager se utiliza para preparar las palabras iniciales, para detectar las teclas que se tocan utilizando el script "WordInput.cs" y reproducir el sonido correspondiente, y detectar cuando se gana (aparecerá la palabra "COMPLETE"). El script "Word.cs" se utiliza para controlar las teclas que toca el jugador, cada vez se tecla la letra correcta, la borra de la pantalla llamando al script "WordDisplay.cs". Después de ganar y volver a la escape scene, se puede notar que al sonido de fondo de la EscapeScene, se le ha sumando un sonido de guitarra.

- Drums: Quinta escena del proyecto. Cambiamos el tipo de juego otra vez. Esta vez, el juego se llama “Simon Says”. El jugador tiene que repetir una secuencia random que hace el juego.

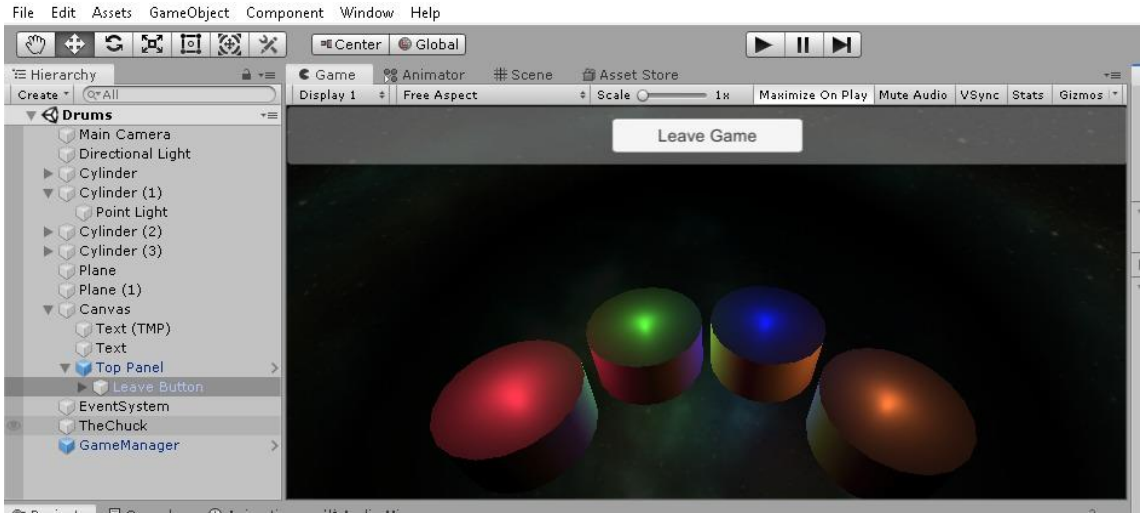


Figura 75: Drums Scene

Unos elementos más en esta escena son los Point Light, que se encienden cuando se toca cada drum. Para ello, se utiliza una animación que se llama “Point Light”. Las animaciones en Unity son diagramas que te permiten animar:

- ❖ La posición, rotación y escala de GameObject.
- ❖ Propiedades de los componentes tales como el color de un material, la intensidad de una luz, el volumen de un sonido
- ❖ Propiedades dentro de sus propios scripts, incluyendo variables tipo float, int, Vector y boolean
- ❖ La sincronización de llamadas de funciones dentro de sus propios scripts.

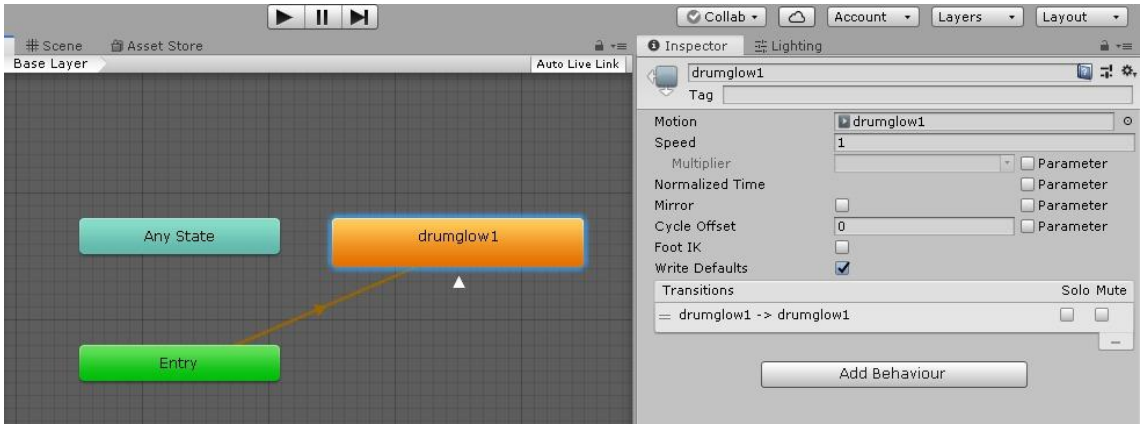


Figura 76: Point Light Animation

En esta escena, se utiliza también el script “GameManager.cs”, con el botón de “Leave Game” para poder salir del juego. Luego tiene también su elemento Chuck para poder utilizar las funcionalidades de Chuck en los scripts (ver Anexo A, 22. DrumsSound.cs).

Luego este juego el script principal de esta escena es el controlador (ver Anexo A, 23.DrumsGameController.cs). Este script crea en cada ronda un array de enteros random (entre 0 y 3), que son los IDs de cada uno de los drums. Y de allí se hace la secuencia random. Luego utiliza otro script (ver Anexo A, 24. ButtonB.cs) que puede leer lo drums que toca el usuario. El controlador luego crea un array con lo que ha tocado el usuario y hace una comparación entre los dos arrays. En el caso de que se se hacen 3 rondas correctas, se pasa el juego y aparece la palabra “COMPLETE”.

Una cosa que comentar en esta escena, es que cuando se exporta el juego como una extensión “.exe”, en la segunda ronda los point lights empiezan a saltar sin ningún comando previo, y allí se complica más para el jugador. En ese caso, no se puede ver la secuencia random correctamente. Es un error que no podía arreglar. De todos modos, cuando se ejecuta el juego en Unity, no pasa ese problema.

Cuando se termina esta fase, se vuelve otra vez a la escena “EscapeScene”. Otra vez, se nota que se suma un sonido de drums al sonido de fondo de la Escena principal. Lo que se puede notar ahora también es que las notas musicales en frente del armario de libros se empiezan a mover (ver AnexoA, 25. ActivateNotes.cs). Eso significa que ya has acabado el juego. Lo que hay que hacer ahora es acercarse al armario para poder pasar a la habitación secreta, y recoger la llave.

Figura 77: Notas de la escena “EscapeScene”

- Secret Scene: Esta es la última escena. Esta escena solo tiene notas musicales celebrando la victoria, junto con la llave en medio.

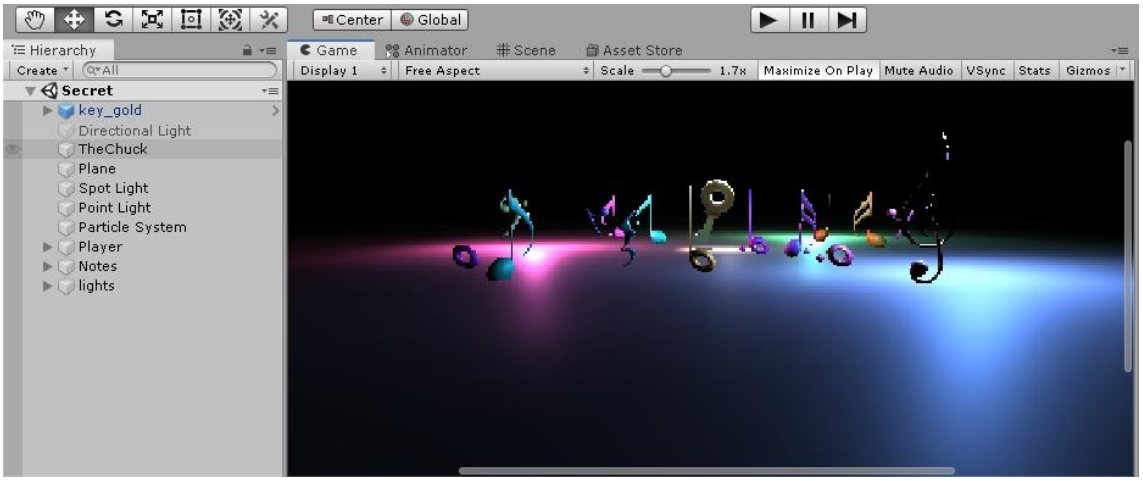


Figura 78: Secret Scene

Solo hace falta recoger la llave para salir de esta escena. Tiene su elemento Chuck para poder escuchar los sonidos de fondo.

Cuando se sale de esta escena, volvemos otra vez al "EscapeScene", donde ahora se cambian los colores de los muros. Cuando pulsas con el ratón sobre la puerta (ver Anexo A, 26. DoorController.cs y 27.Door.js), se abre usando la llave que se acaba de recoger. Y ya se puede salir.

Como Mapa del juego, se puede enseñar la siguiente figura:

Figura 79: Mapa del juego

3.5. Photon Networking:

Hemos establecido las etapas del juego para un jugador. Ahora nos toca el procedimiento para la conexión de dos o más otros jugadores.

La primera parte de Photon está explicada en los últimos dos apartados, porqué tiene que ir sincronizada con la creación del proyecto en Unity. En esa parte, vimos como descargar e importar Photon Pun2. Luego en la escena de inicio vimos que el botón Play tiene su propio script (ver Anexo A, 3.Launcher.cs). En cuanto se pulsa el botón Play, podemos ver los siguientes Logs.

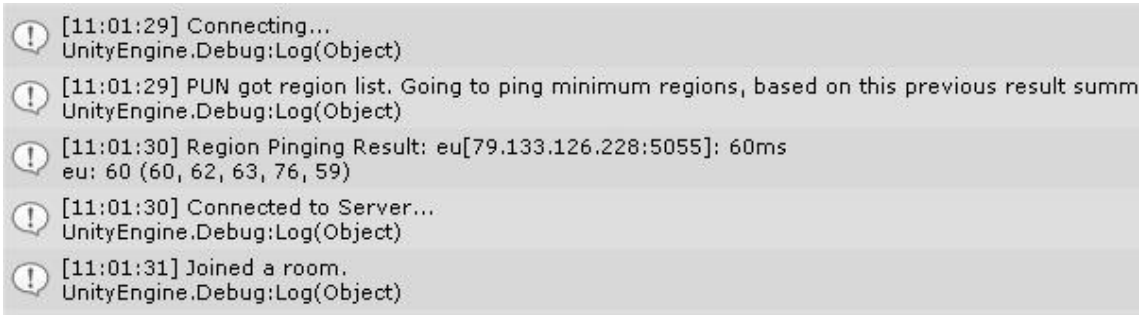


Figura 80: Logs de Conexión

Luego en cuanto se conecte otro jugador, aparece el log siguiente.

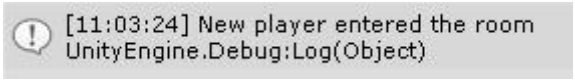


Figura 81: Log de un jugador nuevo

Ese Log es el que aparece en el script del GameManager.cs (ver Anexo A, 4).

Hasta ahora, podemos ver que ya se conectan los jugadores en la misma sala.

Lo siguiente que hay que hacer es crear un avatar para identificar a cada jugador, y para que cada jugador puede ver al otro.

Para eso, he creado un GameObject nuevo con nombre “Network Player” (se puede crear en cualquier escena). Luego pasamos ese objeto a la carpeta Resources (Es como la carpeta Prefab pero para Photon). En cuanto tengamos el objeto en la carpeta Resources, se puede borrar de la escena. Mi avatar tendrá una cabeza (sphere), y luego dos manos.

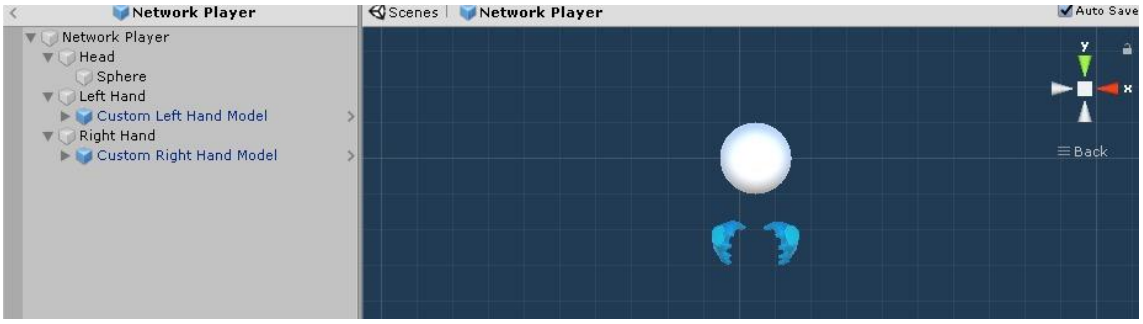


Figura 82: Network Player

En cuanto tengamos el avatar, procedemos a manipularlo para conseguir lo que necesitamos. Lo primero que hice es crear un script nuevo (ver Anexo A, 28.NetworkPlayerSpawner) y añadirlo como componente al GameManager.

Ese script nos permite inicializar el Avatar cuando el jugador entra al salón, y quitarlo del medio cuando el jugador sale del salón.

Luego, lo que hay que fijar es la posición del avatar con la cámara, y seguirla todo el rato.

Para eso, añadimos las componentes necesarias al prefab “Network Player”.

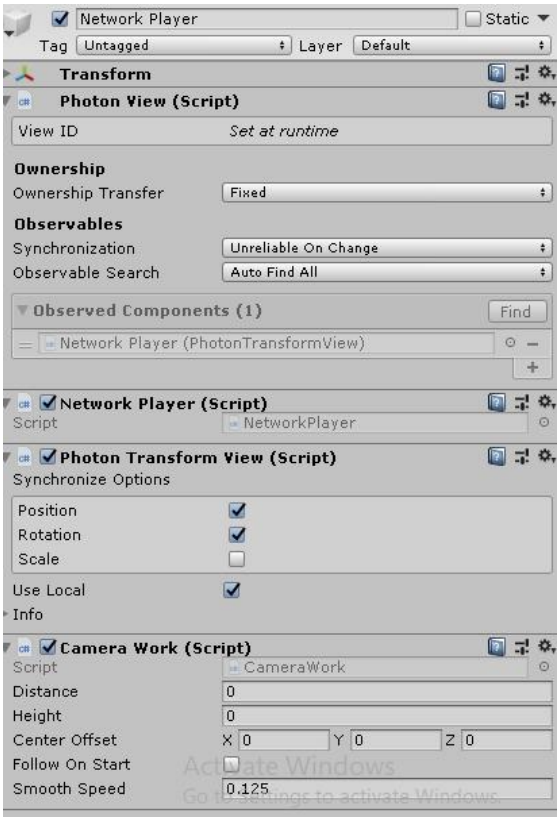


Figura 83: Network Player Components

Primero, el jugador necesita un componente PhotonView. La componente PhotonView se utiliza en el framework de Photon Networking para permitir la sincronización de objetos y variables en un entorno de juego multijugador en tiempo real y, lo más importante de todo, permite que los cambios realizados en un objeto sean transmitidos a todos los demás clientes en la partida.

Con el PhotonView, cada jugador se conecta con un ID. Lo podemos ver en el log siguiente.

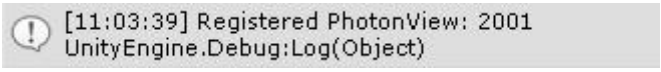


Figura 84: Log PhotonView

Luego, el jugador tendrá su script propio (ver Anexo A, 29.NetworkPlayer). Este script sirve primero para asignar cada jugador a su propio avatar local, y luego para seguir solo ese avatar. Eso se puede controlar con el PhotonView local.

Lo siguiente es añadir el componente Photon Transform View. Esto sirve para sincronizar el movimiento (la posición) de los jugadores en las escenas.

Y últimamente, que es la parte que me parecía más complicada, que es la Camera Work. Es un script que se puede consultar en la carpeta de los Assets (CameraWork.cs). En el caso de que el avatar tenga su propia cámara, habrá que crear ese script y montarlo con el avatar. El script puede servir para cámaras en primera y en tercera persona. En mi caso, eso no será necesario, porqué la cámara de la escena ya está montada. Lo que se ha hecho es añadir al script NetworkPlayer.cs unas funciones para que el avatar siga la cámara.

Este fue el desafío que encontré y no pude superar en la implementación del componente de multijugador, debido a las restricciones de tiempo. Al conectar diferentes jugadores, se utilizó la misma cámara principal, pero los jugadores no podían ver a los demás. Durante las etapas del juego, el primer jugador conectado se convierte en el anfitrión ("master") y, al cambiar a otra escena, todos los jugadores le siguen. Sin embargo, cuando un jugador que no es el anfitrión se acerca a un instrumento y se cambia a otra escena, los demás jugadores no le siguen. El desafío principal en esta situación radica en la necesidad de distinguir a cada usuario de manera adecuada, más que en otros aspectos técnicos.

VII. PRUEBAS:

Para esta parte, he preparado una hoja de evaluación del juego (ver Anexo B).

1. Proceso del juego:

Figura 85: proceso del juego

2. Prueba del creador del juego:

Una vez explicado el desarrollo de la aplicación, se han hecho varias pruebas para comprobar su funcionamiento. Las pruebas realizadas son las siguientes:

1. Prueba del botón “Leave Game” en cada escena.
2. Prueba de tocar teclas erróneas del piano.
3. Prueba de tocar cuerdas erróneas de la guitarra.
4. Prueba de tocar drums erróneos.
5. Prueba de hacer el juego completo.
6. Prueba de conexión de dos o más jugadores.
7. Prueba del máster liderando las fases del juego.
8. Prueba de otro jugador liderando las fases del juego.

Todas estas pruebas se pueden ver en un vídeo mediante el siguiente link:

https://drive.google.com/drive/u/0/folders/1BSNC3T-THbbCwdRJYzAdBr_lCZdfCaSp

Por otro lado, se preparó una autoevaluación del juego, que se puede ver en el Anexo C.

3. Prueba de otros usuarios:

3.1. Prueba del Usuario A (un ingeniero técnico de telecomunicaciones):

Los resultados de esta prueba se pueden ver en el anexo D.

3.2. Prueba del Usuario B (un músico):

Los resultados de esta prueba se pueden ver en el anexo E.

VIII. RESULTADOS Y CONCLUSION:

1. RESULTADOS:

Recordamos cuáles eran los objetivos principales de este proyecto de fin de grado. En primer lugar, además de realizar una revisión bibliográfica y estudiar las técnicas de realidad virtual relacionadas con la aplicación, se debía explorar los mecanismos que permiten incorporar una interacción musical en el juego. Para lograrlo, se estudiaron las aplicaciones existentes en el mercado. Al mismo tiempo, era necesario analizar las herramientas de hardware y software disponibles para desarrollar la aplicación. Todo esto se encuentra detallado en el marco teórico de la memoria del proyecto.

Una vez estudiado y preparado el anteproyecto, el objetivo más importante era el desarrollo de un Escape Room en Realidad Virtual con interacción musical. Este diseño se basó en las herramientas estudiadas previamente. Por último, era necesario comprobar el correcto funcionamiento de la aplicación.

Habiendo mencionado los objetivos de este proyecto, considero que los resultados obtenidos con la aplicación creada han sido satisfactorios. Durante el desarrollo de esta aplicación, se aprendió a planificar y crear aplicaciones en Unity, a integrar otros softwares o SDKs como Chuck para la gestión y presentación de contenidos multimedia, a utilizar el procesamiento de sonido en MATLAB como se aprendió en la carrera y a implementar el multijugador en los juegos de Unity utilizando Photon. Sin embargo, esta última parte no fue tan satisfactoria debido a limitaciones de tiempo, conocimiento y recursos disponibles en Internet.

Centrándonos en los resultados obtenidos, se ha logrado crear una aplicación funcional capaz de reproducir notas musicales de diferentes instrumentos en formato .wav y .mid (MIDI), y sincronizarlas con las pruebas del Escape Room. Lo mejor de todo es que se ha conseguido una interfaz muy simple y clara para el jugador.

Para lograr este trabajo, realicé varios intentos, revisé numerosos proyectos en Internet y pasé horas y horas viendo tutoriales en YouTube. Al final, lo más importante es que aprendí varias técnicas, tecnologías, ideas y ampliaciones que se pueden aplicar al proyecto.

Es cierto que, aunque se logró desarrollar la aplicación, todavía presenta algunas fallas. Una de ellas es que los puntos de luz de los tambores no funcionan correctamente cuando la aplicación se exporta como un archivo .exe. Este fue un problema que me costó aceptar, ya que intenté solucionarlo de varias formas, pero sin éxito.

Otra parte que no está completamente funcional es la del multijugador, la cual no fue incluida en el juego inicialmente. Sin embargo, siempre encontré más interesantes los juegos con capacidad multijugador y deseaba aprender cómo implementarlos. Así comenzó mi aventura. Digo "aventura" porque los SDK de multijugador son un amplio universo en sí mismos. Cada SDK tiene su propia arquitectura de red y servidores. Elegí Photon porque su versión gratuita está bastante avanzada y ofrece muchas opciones. El único inconveniente fue que empecé a incorporar el código de Pun2 cuando ya tenía casi desarrollada toda la aplicación. Esto es algo que generalmente no se hace en los juegos, ya que Photon tiene sus propias funciones que incluso se utilizan para temas de Unity que no tienen nada que ver con el multijugador, como, por ejemplo, el cambio de una escena a otra.

Al final, he logrado que se connecten los jugadores a la misma sala, detectando la conexión de ambos. Luego también he logrado que los dos juegan en la misma sala. El problema es que cada jugador no puede ver al otro. Es un problema que no sabía cómo solucionarlo. Seguiré a la búsqueda de soluciones, para poder lograrlo.

Para terminar, según las evaluaciones de los Usuarios A y B, se puede ver que el juego le faltan muchas cosas técnicas y no técnicas (Se proponen ampliaciones y mejoras en el apartado 3.).

2. Conclusión:

Después de este extenso proceso de exploración en dichos ámbitos, me he percatado de lo mucho que aún queda por aprender, y agradezco la oportunidad que se me brindó al seleccionarme para llevar a cabo este proyecto. Esta experiencia me ha permitido continuar desarrollando los fundamentos básicos adquiridos en campos que están experimentando un crecimiento significativo en la actualidad, y que incluso podrían abrirme puertas laborales en el futuro.

Debo admitir que no fue sencillo llevar a cabo este proyecto. En primer lugar, enfrenté dificultades debido a mi limitado conocimiento e inexperiencia en los campos de estudio de la realidad virtual y el multijugador. Además, la situación excepcional en la que me encuentro, estudiando el máster y trabajando, ha generado una inestabilidad y un caos parcial en mi vida, lo cual también contribuyó a los desafíos que enfrenté.

Ha sido todo un desafío lograr una aplicación funcional que cumpla con los requisitos establecidos desde el inicio de su desarrollo. Asimismo, ha sido un reto redactar una memoria que satisfaga todas las pautas indicadas. Fue un trabajo arduo, dedicando horas y horas a la redacción hasta obtener un texto comprensible y coherente que aborde cada aspecto del proyecto de manera consistente.

En mi opinión personal, considero que todo el esfuerzo invertido valió la pena y que he crecido a medida que encontraba soluciones para cada problema que se presentaba en el camino.

3. Ampliaciones y mejoras:

A lo largo del desarrollo del proyecto, se fueron identificando distintas ampliaciones disponibles:

3.1. Más instrumentos y más sonido:

El juego es bastante simple de mi punto de vista. Por lo que una de las funciones a mejorar en el sistema es aumentar la variedad de instrumentos y sonidos, añadiendo algunos instrumentos que simularán instrumentos de vientos… Esto conlleva el desarrollo de nuevas pruebas, que pueden ser incluso más complicadas que las que están ahora. Por ejemplo, el jugador tendrá que reproducir las notas que escuchan en el fondo de la aplicación…

3.2. Procesamiento de sonido dentro de Unity:

Esto es una cosa que quería lograr muchas veces en mi proyecto. Pero como estoy utilizando una versión un poco antigua, no era capaz de descargar muchas librerías y assets del Asset Store de Unity.

3.3. Mejorar la parte del Multijugador:

Tenía mucha ilusión para lograr esta parte, pero al final era más complicado que lo que pensaba. Es un tema que voy a seguir averiguando. Lo que se puede ampliar en esta parte es ampliar la sala para que puedan jugar más jugadores. Se puede dar la posibilidad para que cada jugador puede superar alguna prueba, y la prueba superada por un jugador está superada por todos. Otra mejora es proporcionar al jugador la opción de elegir du avatar, de elegir la cámara (primera o tercera persona) …

IX. BIBLIOGRAFÍA:

[1] F. Manes, «¿Cómo afectan las nuevas tecnologías a nuestro cerebro?» El País, 28 de Diciembre 2015.

[2] Redacción, «Advierten sobre la importancia que tiene que los chicos sigan escribiendo a mano» El Día, 1 de Junio 2016.

[3] C. Barreno, «ESFINGE. Apuntes para un pensamiento diferente» Septiembre 2011. [En línea].

[4] «Ciclo de sobreexpectación» [En línea]. Disponible: https://es.wikipedia.org/wiki/Ciclo_de_sobreexpectaci%C3%B3n

[5] «virtualtravelog» [En línea]. Disponible: http://www.virtualtravelog.net/wp/wp-content/gallery/2011-04-hype-cycles/Gartner-Hype-Cycle-2010.gif

[6] « http://na2.www.gartner.com/imagesrv/newsroom/images/hype-cycle-pr.png;wada20fd4bd7891509 » [En línea].

[7] «Final Project: A Musical Escape Room» [En línea]. Disponible: https://ccrma.stanford.edu/~vsc/256a/finalproject/

[8] «Inventor Virtual Reality» [En línea]. Disponible : http://www.mortonheilig.com/InventorVR.html

[9] «Sensorama patent» 28 08 1962. [En línea]. Disponible : http://www.mortonheilig.com/SensoramaPatent.pdf

[10] «Telesphere Mask Patent». [En línea]. Disponible : http://www.mortonheilig.com/TelesphereMask.pdf

[11] «Sensorama». [En línea]. Disponible : http://proyectoidis.org/sensorama/

[12] «Arcade,» [En línea]. Disponible: https://es.wikipedia.org/wiki/Arcade

[13] «Sensorama simulator» [En línea]. Disponible: https://www.engadget.com/2014/02/16/morton-heiligs-sensorama-simulator/

[14] «Telesphere» [En línea]. Disponible: https://en.wikipedia.org/wiki/Telesphere

[15] «HMD history and objectives of inventions». [En línea]. Disponible: https://glassdevelopment.wordpress.com/2014/04/17/hmd-history-and-objectives-of-inventions

[16] «Head mounted display» [En línea]. Disponible : https://en.wikipedia.org/wiki/Head-mounted_display

[17] «Virtual Reality History» [En línea]. Disponible: https://www.vrs.org.uk/virtual-reality/history.html

[18] «Ivan Sutherland» 2016. [En línea]. Disponible: https://es.wikipedia.org/wiki/Ivan_Sutherland

[19] I. E. Sutherland, «The Ultimate Display»

[20] «Augmented Reality: "The Ultimate Display" by Ivan Sutherland, 1965» 2009. [En línea]. Disponible: https://www.wired.com/2009/09/augmented-reality-the-ultimate-display-by-ivan-sutherland-1965/

[21] «Processing power compared» [En línea]. Disponible: http://pages.experts-exchange.com/processing-power-compared/

[22] «La espada de damocles ivan sutherland» 2011. [En línea]. Disponible: http://rvjaac.blogspot.com.es/2011/08/la-espada-de-damocles-ivan-sutherland.html

[23] «Jaron Lanier» [En línea]. Disponible: https://es.wikipedia.org/wiki/Jaron_Lanier

[24] J. N. M. Luna, «2.1.6 La diversificación de la tecnología» de REALIDAD VIRTUAL

[25] «5dt data glove 14 ultra 2» [En línea]. Disponible: https://www.vrealities.com/products/data-gloves/5dt-data-glove-14-ultra-2

[26] «VPL Research» [En línea]. Disponible: https://en.wikipedia.org/wiki/VPL_Research

[27] «Virtuality (gaming)» [En línea]. Disponible: https://en.wikipedia.org/wiki/Virtuality_(gaming)

[28] «Cave Automatic Virtual Environment» [En línea]. Disponible: https://es.wikipedia.org/wiki/Cave_Automatic_Virtual_Environment

[29] «Sega VR» 2016. [En línea]. Disponible: http://segaretro.org/Sega_VR

[30] «Virtual Boy» [En línea]. Disponible: https://es.wikipedia.org/wiki/Virtual_Boy

[31] «[Up nalisis] Analisis a la virtual boy» 2015. [En línea]. Disponible: http://www.3djuegos.com/comunidad-foros/tema/30646333/0/up-nalisis-analisis-a-la-virtual-boy/

[32] «Oculus Rift» 2016. [En línea]. Disponible: https://www3.oculus.com/en-us/rift/

[33] «kickstarter» [En línea]. Disponible: https://www.kickstarter.com/

[34] «Facebook compra Oculus Rift por 2.000 millones de dólares» Europapress, 25/03/2014

[35] J. PENALVA, «Nuevo Project Morpheus: Sony se va a 2016 y sólo para PS4 para pelear en la realidad virtual» 04 03 2015. [En línea]. Disponible: https://www.xataka.com/videojuegos/nuevo-project-morpheus-sony-se-va-a-2016-y-solo-para-ps4-para-pelear-en-la-realidad-virtual

[36] «Fove» [En línea]. Disponible: https://en.wikipedia.org/wiki/Fove

[37] A. Sabán, «Apple compra Metaio, un startup de realidad aumentada» [En línea]. Disponible: https://hipertextual.com/2015/05/apple-compra-metaio-una-startup-de-realidad-aumentada

[38] «cardboard google» [En línea]. Disponible: https://vr.google.com/intl/es_es/cardboard/

[39] «Get cardboard» [En línea]. Disponible: https://vr.google.com/intl/es_es/cardboard/get-cardboard/

[40] «Google Cardboard» [En línea]. Disponible: https://es.wikipedia.org/wiki/Google_Cardboard

[41] «Samsung Gear VR» 2016. [En línea]. Disponible: http://www.samsung.com/global/galaxy/gear-vr/

[42] «Amazon, Samsung Gear VR» [En línea]. Disponible: https://www.amazon.com/Samsung-Gear-VR-Virtual-Warranty/dp/B016OFYGXQ

[43] «Diodo orgánico de emisión de luz» [En línea]. Disponible: https://es.wikipedia.org/wiki/Diodo_org%C3%A1nico_de_emisi%C3%B3n_de_luz

[44] A. Sánchez, «PlayStation VR» 2016. [En línea]. Disponible: https://hipertextual.com/analisis/playstation-vr

[45] «PlayStation VR Sony» [En línea]. Disponible: https://www.playstation.com/es-es/explore/playstation-vr/

[46] «PlayStation VR2 Sony» [En línea]. Disponible: https://www.playstation.com/es-es/ps-vr2/?emcid=pa-co-422218&gclid=CjwKCAjw__ihBhADEiwAXEazJoWQTBxKloDOxfmrEgab

WenDaX7w_2f4lL6YsZwo1JRUPwML7c0XxBoC-sIQAvD_BwE&gclsrc=aw.ds

[47] «Oculus Rift» 2016. [En línea]. Disponible: http://mundo-virtual.com/gafas-realidad-virtual/oculus-rift/

[48] J. MATURANA, «HTC Vive, análisis: esto sí que es realidad virtual interactiva» [En línea]. Disponible: https://www.xataka.com/analisis/htc-vive-analisis-esto-si-que-es-realidad-virtual-interactiva

[49] «Oculus Rift» [En línea]. Disponible: https://www.oculus.com/rift/

[50] «HTC Vive» [En línea]. Disponible: https://www.vive.com/eu/

[51] «Oculus Quest» [En línea]. Disponible: https://www.oculus.com/experiences/quest/

[52] «Oculus Quest 2» [En línea]. Disponible: https://www.meta.com/quest/products/quest-2/

[53] «Oculus Quest 3: All we know so far» [En línea]. Disponible: https://www.techradar.com/news/oculus-quest-3-price-release-date-specs

[54] «Khronos Announces VR Standards Initiative» . [En línea]. Disponible: https://www.khronos.org/news/press/khronos-announces-vr-standards-initiative

[55] «steampowered» [En línea]. Disponible: https://support.steampowered.com/kb_article.php?ref=5254-FJKZ-7829&l=spanish

[56] «proyectofreak» [En línea]. Disponible: http://proyectofreak.com/pw/plataforma-de-realidad-virtual-para-videojuegos/

[57] «Escapismo» [En línea]. Disponible: https://es.wikipedia.org/wiki/Escapismo

[58] «Cubick room escape» 2016. [En línea]. Disponible: http://cubickroomescape.es

[59] «Lever Escape Room» 2016. [En línea]. Disponible: http://www.leverescaperoom.com

[60] «MAD Escape Room» 2016. [En línea]. Disponible: http://madescaperoom.com

[61] «Parapark» [En línea]. Disponible: http://parapark.es/map

[62] «La linea del tiempo, iPlay.» 2016. [En línea]. Disponible: http://iplaytimeline.com/room-escape/

[63] «Mihály Csíkszentmihályi» [En línea]. Disponible: https://es.wikipedia.org/wiki/Mih%C3%A1ly_Cs%C3%ADkszentmih%C3%A1lyi

[64] Cubick Room Escape, «Origen Ideologico Room Escape» 2016. [En línea]. Disponible: http://cubickroomescape.es/origen-ideologico-room-escape/

[65] «http://parapark.es/map» [En línea]. Disponible: http://parapark.es/map

[66] «Origen geográfico salas escapismo» [En línea]. Disponible: http://cubickroomescape.es/origen-geografico-salas-escapismo/

[67] Chris Walden, « http://www.japanator.com/japanatour-odaiba-22091.phtml » [En línea]. Disponible: http://www.japanator.com/japanatour-odaiba-22091.phtml

[68] Mad Escape Room, «Cual es el origen de los juegos de escape» 2016. [En línea]. Disponible: http://madescaperoom.com/cual-es-el-origen-de-los-juegos-de-escape/

[69] «World of escapes» [En línea]. Disponible: https://worldofescapes.com/

[70] A. Davis, «Virtual Reality, Real Profits: 11 Great Stocks To Play The Coming VR/AR Boom» Forbes

[71] C. Neiger, «Virtual Reality Companies: Best and Worst Investments» The Motly Fool

[72] «La realidad virtual se asienta en el mercado del videojuego,» [En línea]. Disponible: www.ClaveLocal.com

[73] «Escape Room VR» [En línea]. Disponible: http://www.escroomvr.com/

[74] A. Sánchez, «Viajes al espacio por 15 euros,» El Periódico

[75] Jason Gregory. (2009). Game Engine Architecture. Taylor and Francis Group, LLC.

[76] «Unreal Engine» [En línea]. Disponible: https://www.unrealengine.com

[77] «Unity Engine» [En línea]. Disponible: https://unity3d.com

[78] «Cry Engine» [En línea]. Disponible: https://www.cryengine.com

[79] GDC. (2017). ESTADO DE LA INDUSTRIA DEL JUEGO 2017 . GDC.

[80] «Unity Asset Store» [En línea]. Disponible: https://www.assetstore.unity3d.com/en/

[81] «Unity Services» [En línea]. Disponible: https://unity3d.com/services

[82] «Unity Store» [En línea]. Disponible: https://store.unity.com/

[83] «Multiplayer VR Development in Unity » [En línea]. Disponible: https://www.vrwiki.cs.brown.edu/vr-development-software/unity3d/multiplayer-vr-development-in-unity

[84] «Chuck» [En línea]. Disponible: https://chuck.stanford.edu/

[85] «Chunity» [En línea]. Disponible: https://chuck.stanford.edu/chunity/

[86] Jack Atherton and Ge Wang. Chunity: Integrated Audiovisual Programming in Unity. Proceedings of the 2018 Conference on New Interfaces for Musical Expression, 2018.

ANEXO A: código de la aplicación:

Se dispone de todo el código fuente generado para la aplicación. Hay muchos códigos importados que no se especificaran en este apartado.

1. reverberacion.m:

```
%REVERBERACIÓN
%y[n]=x[n]+A*y[n-D]

%señal monocanal
[x,fs]=audioread('Gchord.wav');
sound(x,fs);
col=x(:,1);
%generamos los ceros
c=zeros(length(x),1);
tamx=length(x);
D=0.1;
A=0.7;
TD=round(fs*D);

for   i= 1:length(col)
    if i<=TD
        c(i)=col(i);
    else
         c(i)= col(i)+A.*(col(i-TD));
    end
end
y=[c c];
sound(y,fs);
audiowrite('reverGchord.wav',y,fs);
```

2. eco.m:

```
%ECO
%y[n]=x[n]+ax(n-D)

%señal monocanal
[x,fs]=audioread('hat.wav');
D=0.3;%retardo
A=0.2;%atenuacion
m=zeros(fs*D,1);
X1=x(:,1);
a=[m;X1];
b=[X1;m];
z=b+(A*a);%en el parentesis añadimos retardo a la señal original
sound(z,fs);
audiowrite('ecohat.wav',z,fs);
```

3. Launcher.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Photon.Pun;
using Photon.Realtime;

namespace Com.MyCompany.MyGame
{
    public class Launcher : MonoBehaviourPunCallbacks
    {
        #region Private Serializable Fields

        /// <summary>
        /// The maximum number of players per room. When a room is full, it
can't be joined by new players, and so new room will be created.
        /// </summary>
        [Tooltip("The maximum number of players per room. When a room is full,
it can't be joined by new players, and so new room will be created")]
        [SerializeField]
        private byte maxPlayersPerRoom = 5;

        #endregion

        #region Private Fields

        /// <summary>
        /// This client's version number. Users are separated from each other
by gameVersion (which allows you to make breaking changes).
        /// </summary>
        string gameVersion = "1";
        bool isConnecting;

        [Tooltip("The Ui Panel to let the user enter name, connect and play")]
        [SerializeField]
        private GameObject controlPanel;
        [Tooltip("The UI Label to inform the user that the connection is in
progress")]
        [SerializeField]
        private GameObject progressLabel;

        #endregion

        #region MonoBehaviour CallBacks

        /// <summary>
        /// MonoBehaviour method called on GameObject by Unity during early
initialization phase.
        /// </summary>
        void Awake()
        {
            // #Critical
            // this makes sure we can use PhotonNetwork.LoadLevel() on the
master client and all clients in the same room sync their level automatically
            PhotonNetwork.AutomaticallySyncScene = true;
        }
```

```
        /// <summary>
        /// MonoBehaviour method called on GameObject by Unity during
initialization phase.
        /// </summary>
        void Start()
        {

            progressLabel.SetActive(false);
            controlPanel.SetActive(true);

        }

        #endregion


        #region Public Methods

        /// <summary>
        /// Start the connection process.
        /// - If already connected, we attempt joining an existing room
        /// - if not yet connected, Connect this application instance to Photon
Cloud Network
        /// </summary>
        public void Connect()
        {

            progressLabel.SetActive(true);
            controlPanel.SetActive(false);
            // #Critical, we must first and foremost connect to Photon Online
Server.
            Debug.Log("Connecting...");
            isConnecting = PhotonNetwork.ConnectUsingSettings();
            PhotonNetwork.GameVersion = gameVersion;

        }

        #endregion
        #region MonoBehaviourPunCallbacks Callbacks

        public override void OnConnectedToMaster()
        {
            if (isConnecting)
            {
                Debug.Log("Connected to Server...");
                base.OnConnectedToMaster();
                // #Critical: The first we try to do is to join a potential
existing room. If there is, good, else, we'll be called back with
OnJoinRandomFailed()
                RoomOptions roomOptions = new RoomOptions();
                roomOptions.MaxPlayers = maxPlayersPerRoom;
                roomOptions.IsVisible = true;
                roomOptions.IsOpen = true;
                PhotonNetwork.JoinOrCreateRoom("Room Yassine", roomOptions,
TypedLobby.Default);

                isConnecting = false;
            }
        }
```

```
        public override void OnDisconnected(DisconnectCause cause)
        {

            progressLabel.SetActive(false);
            controlPanel.SetActive(true);

            isConnecting = false;

            Debug.LogWarningFormat("Player Disconnected");
        }

        public override void OnJoinedRoom()
        {
            Debug.Log("Joined a room.");
            PhotonNetwork.LoadLevel("EscapeScene");


        }

        public override void OnPlayerEnteredRoom(Player newPlayer)
        {
            Debug.Log("New player entered the room");
            base.OnPlayerEnteredRoom(newPlayer);
        }

        #endregion



    }
}
```

4. GameManager.cs:

```
using System;
using System.Collections;

using UnityEngine;
using UnityEngine.SceneManagement;

using Photon.Pun;
using Photon.Realtime;

namespace Com.MyCompany.MyGame
{
    public class GameManager : MonoBehaviourPunCallbacks
    {

        #region Photon Callbacks

        /// <summary>
        /// Called when the local player left the room. We need to load the
launcher scene.
        /// </summary>
        public override void OnLeftRoom()
        {
            Debug.Log("Left the room!");
            PhotonNetwork.LoadLevel("Intro");
        }

        #endregion

        #region Public Methods

        public void LeaveRoom()
        {
            PhotonNetwork.LeaveRoom();
            PhotonNetwork.Disconnect();
        }

        public override void OnPlayerEnteredRoom(Player newPlayer)
        {
            Debug.Log("New player entered the room");
            base.OnPlayerEnteredRoom(newPlayer);
        }



        #endregion
    }
}
```

5. RoomSound.cs:

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

public class RoomSound : MonoBehaviour
{

    ChuckSubInstance chuckRoom;


    // Start is called before the first frame update
    void Start()
    {
        chuckRoom = GetComponent<ChuckSubInstance>();

        FindObjectOfType<RoomSound>().playSynth();

         if (WordManager.guitarDone == true)
        {
            //play pluck.wav or guitar.wav
            FindObjectOfType<RoomSound>().playGuitar();
        }

        if (PressKey.pianoDone == true)
        {
            //play pianotexture.wav
            FindObjectOfType<RoomSound>().playPiano();
        }

        if (DrumGameController.drumsDone == true)
        {
            //play drums.wav
            FindObjectOfType<RoomSound>().playDrums();
        }
     }

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

    }

    public void playSynth()
    {
        Debug.Log("SYNTH PLAYING");
        chuckRoom.RunCode( (@"
            SndBuf buf => dac;
            0.04 => buf.gain;
            me.dir() + ""synthloop.wav"" => string synth;

            synth => buf.read;
            buf.samples() => buf.pos;

            while( true )
            {
                0 => buf.pos;
                buf.length() => now;
            }
```

```
        "));
    }

    public void playPiano()
    {
        Debug.Log("PIANO PLAYING");
        chuckRoom.RunCode(( @"

            SndBuf buf => dac;
            0.05 => buf.gain;
            me.dir() + ""pianotexture.wav"" => string piano;

            piano => buf.read;
            buf.samples() => buf.pos;

            while( true )
            {
                0 => buf.pos;
                buf.length() => now;
            }

        "));
    }

    public void playGuitar()
    {
        Debug.Log("GUITAR PLAYING");
        chuckRoom.RunCode(( @"
            SndBuf buf => dac;
            0.5 => buf.gain;
            me.dir() + ""pluck.wav"" => string pluck;

            pluck => buf.read;
            buf.samples() => buf.pos;

            while( true )
            {
                0 => buf.pos;
                buf.length() => now;
            }

        "));
    }

    public void playDrums()
    {
        Debug.Log("DRUMS PLAYING");
        chuckRoom.RunCode(( @"
            SndBuf buf => dac;
            0.04 => buf.gain;
            me.dir() + ""drums.wav"" => string drums;

            drums => buf.read;
            buf.samples() => buf.pos;

            while( true )
            {
                0 => buf.pos;
                buf.length() => now;
            }

        "));
    }
}
```

6. MoveOnActive.js:

```
var downPos:Vector3;
var downRot:Quaternion;
private var upPos:Vector3;
private var upRot:Quaternion;
var moveSpeed:float;
var rotateSpeed:float;
private var isFalse:boolean=false;

function Start(){
    upPos=transform.position;
    upRot=transform.rotation;
    //downRot=Quaternion.LookRotation(downRot2,Vector3.up);
}
function Activate () {
    isFalse=!isFalse;
    GetComponent.<AudioSource>().Play();
}

function Update () {
    if(isFalse){

transform.position=Vector3.MoveTowards(transform.position,downPos,moveSpeed*Time.deltaTime);
        transform.rotation=Quaternion.RotateTowards(transform.rotation,downRot,rotateSpeed);
    }else{

transform.position=Vector3.MoveTowards(transform.position,upPos,moveSpeed*Time.deltaTime);
        transform.rotation=Quaternion.RotateTowards(transform.rotation,upRot,rotateSpeed);
    }
}
```

7. SceneSwitchPiano.cs:

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

using UnityEngine.SceneManagement;


public class SceneSwitchPiano : MonoBehaviour
{
    [SerializeField] private string loadLevel;
    // Start is called before the first frame update
    void Start()
    {

    }

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

    }

    void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            Debug.Log("initiaitng");
            Initiate.Fade("Piano",Color.white,2.0f);
            //SceneManager.LoadScene(loadLevel);
        }
    }
}
```

8. PressKey.cs:

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

using UnityEngine.UI;

public class PressKey : MonoBehaviour
{
    public float force = 2;

    public List<string> pressed;

    List<string> notes = new List<string>(new string[] {"G",  "C", "A", "F", "E",
"C", "D"});
    //List<string> pressed = new List<string>(new string[] {});


    public static bool pianoDone = false;

    public Text winText;

    // Start is called before the first frame update
    void Start()
    {
        // List<string> notes = new List<string>(new string[] {"G",  "C", "A", "F",
"E", "C", "D"});
        // List<string> pressed = new List<string>(new string[] {});
        winText.text = "";

    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            RaycastHit hit;
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

            if (Physics.Raycast(ray, out hit, 100.0f ))
            {
                if (hit.transform != null)
                {

                    Rigidbody rb;
                    //assign rb, and perform check
                    //ie. if hit.transform is null, rb is null, otherwise its a
rigidbody
                    if (rb = hit.transform.GetComponent<Rigidbody>())
                    {
                        PrintName(hit.transform.gameObject);
                        Press(rb);
                        //Debug.Log(GetName(hit.transform.gameObject));

                        //PLAY SOUNDS
                        if (GetName(hit.transform.gameObject) == "C")
                        {
```

```
                        //
                         Debug.Log("calling playC");
                        // GetComponent<ChuckSubInstance>().RunCode( (@"
                        //     fun void playC()
                        //     {
                        //         SndBuf buf => dac;
                        //         0.075 => buf.gain;
                        //         me.dir() + ""C.wav"" => string C;

                        //         C => buf.read;
                        //         2::second => now;
                        //     }
                        //     "));
                        FindObjectOfType<Sound>().playC();
                    }
                    else if (GetName(hit.transform.gameObject) == "D")
                    {
                        FindObjectOfType<Sound>().playD();
                    }
                    else if (GetName(hit.transform.gameObject) == "E")
                    {
                        FindObjectOfType<Sound>().playE();
                    }
                    else if (GetName(hit.transform.gameObject) == "F")
                    {
                        FindObjectOfType<Sound>().playF();
                    }
                    else if (GetName(hit.transform.gameObject) == "G")
                    {
                        FindObjectOfType<Sound>().playG();
                    }
                    else if (GetName(hit.transform.gameObject) == "A")
                    {
                        FindObjectOfType<Sound>().playA();
                    }
                    else if (GetName(hit.transform.gameObject) == "B")
                    {
                        FindObjectOfType<Sound>().playB();
                    }

                    pressed.Add(GetName(hit.transform.gameObject));
                    Debug.Log("hi" + pressed[0]);//Debug.Log(pressed);

                }
            }
        }
    }
    if (pressed.Count == 7 && notes[notes.Count - 1] == pressed[pressed.Count - 1])
    {
        StartCoroutine(Win());
    }
    // if (notes.SequenceEqual(pressed))
    // {
    //     Initiate.Fade("EscapeScene",Color.white,2.0f);
    // }
}
```

```
    IEnumerator Win()
    {
            Debug.Log("WIN CALLED");
            pianoDone = true;
            yield return new WaitForSeconds(1f);
            winText.text = "COMPLETE";
            yield return new WaitForSeconds(2f);
            Initiate.Fade("EscapeScene",Color.white,2.0f);
            winText.text = "";
    }


    private void PrintName(GameObject go)
    {
        print(go.name);
    }
    private string GetName(GameObject go)
    {
        return go.name;
    }

    private void Press(Rigidbody rig)
    {
        rig.AddForce(0, force, 0, ForceMode.Impulse);
    }


}
```

9. SoundMIDI.cs:

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

public class Sound : MonoBehaviour
{
    ChuckSubInstance chuckSound;
    // Start is called before the first frame update
    void Start()
    {
        chuckSound = GetComponent<ChuckSubInstance>();
    }

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

    }

    public void playC()
    {
        Debug.Log("playC called!!!!!!!!");
        chuckSound.RunCode(@"

        // path
        NRev reverb => dac;
        // just a bit of reverb
        0.025 => reverb.mix;

        // the MIDI file in object
        MidiFileIn min;
        // the MIDI message shuttle
        MidiMsg msg;

        // the filename
        string filename;
        // if no command line arguments provided
        if( me.args() == 0 ) me.dir() + ""C.mid"" => filename;
        // else use that filename
        else me.arg(0) => filename;

        // open the file; exit on error
        if (!min.open(filename)) me.exit();

        // print out
        cherr <= ""----------"" <= IO.newline();
        cherr <= ""MIDI file: "" <= IO.newline()
              <= "" |- "" <= filename <= IO.newline()
        // print
        cherr <= ""----------"" <= IO.newline();
        cherr <= ""playing..."" <= IO.newline();
        cherr <= ""----------"" <= IO.newline();

        // flag
        int done;
```

```
        // entry point for each shred assigned to a track
        fun void doTrack( int track, float speed )
        {
            // hack polyphony
            Wurley s[4];
            // for each voice
             for( int i; i < s.size(); i++ )
             {
                // set gain
                .5 => s[i].gain;
                // connect to output
                s[i] => reverb;
            }
            // voice number for quick polyphony
            int v;

            // read through all MIDI messages on track
            while( min.read( msg, track ) )
            {
                // this means no more MIDI events at current time; advance time
                if( msg.when > 0::second )
                    msg.when * speed => now; // speed of 1 is nominal

                    // catch NOTEON messages (lower nibble == 0x90)
                if( (msg.data1 & 0xF0) == 0x90 && msg.data2 > 0 && msg.data3 > 0 )
                {
                    // get the pitch and convert to frequencey; set
                    msg.data2 => Std.mtof => s[v].freq;
                    // velocity data; note on
                    msg.data3/127.0 => s[v].noteOn;
                    // cycle the voices
                    (v+1)%s.size() => v;

                    // log
                    cherr <= ""NOTE ON track: "" <= track <= "" pitch: "" <=
msg.data2 <="" velocity: "" <= msg.data3 <= IO.newline();
                }
                // other messages
                else
                {
                    // log
                    // cherr <= ""----EVENT (unhandled) track:"" <= track <= ""
type:"" <= (msg.data1&0xF0)
                    //      <= "" data2:"" <= msg.data2 <= "" data3:"" <= msg.data3
<= IO.newline();
                }
            }

            // done with track
            done++;
         }

        ");


    }
```

```
    public void playD()
    {
        Debug.Log("playD called!!!!!!!!");
        chuckSound.RunCode( (@"

        // path
        NRev reverb => dac;
        // just a bit of reverb
        0.025 => reverb.mix;

        // the MIDI file in object
        MidiFileIn min;
        // the MIDI message shuttle
        MidiMsg msg;

        // the filename
        string filename;
        // if no command line arguments provided
        if( me.args() == 0 ) me.dir() + ""D.mid"" => filename;
        // else use that filename
        else me.arg(0) => filename;

        // open the file; exit on error
        if (!min.open(filename)) me.exit();

        // print out
        cherr <= ""----------"" <= IO.newline();
        cherr <= ""MIDI file: "" <= IO.newline()
              <= "" |- "" <= filename <= IO.newline()
        // print
        cherr <= ""----------"" <= IO.newline();
        cherr <= ""playing..."" <= IO.newline();
        cherr <= ""----------"" <= IO.newline();

        // flag
        int done;
        // entry point for each shred assigned to a track
        fun void doTrack( int track, float speed )
        {
            // hack polyphony
            Wurley s[4];
            // for each voice
             for( int i; i < s.size(); i++ )
             {
                // set gain
                .5 => s[i].gain;
                // connect to output
                s[i] => reverb;
            }
            // voice number for quick polyphony
            int v;

            // read through all MIDI messages on track
            while( min.read( msg, track ) )
            {
                // this means no more MIDI events at current time; advance time
                if( msg.when > 0::second )
                    msg.when * speed => now; // speed of 1 is nominal
```

```
                    // catch NOTEON messages (lower nibble == 0x90)
                if( (msg.data1 & 0xF0) == 0x90 && msg.data2 > 0 && msg.data3 > 0 )
                {
                    // get the pitch and convert to frequencey; set
                    msg.data2 => Std.mtof => s[v].freq;
                    // velocity data; note on
                    msg.data3/127.0 => s[v].noteOn;
                    // cycle the voices
                    (v+1)%s.size() => v;

                    // log
                    cherr <= ""NOTE ON track: "" <= track <= "" pitch: "" <=
msg.data2 <="" velocity: "" <= msg.data3 <= IO.newline();
                }
                // other messages
                else
                {
                    // log
                    // cherr <= ""----EVENT (unhandled) track:"" <= track <= ""
type:"" <= (msg.data1&0xF0)
                    //      <= "" data2:"" <= msg.data2 <= "" data3:"" <=
msg.data3 <= IO.newline();
                }
            }

            // done with track
            done++;
         }

        "));
    }

    public void playE()
    {
        Debug.Log("playE called!!!!!!!!");
        chuckSound.RunCode( (@"

        // path
        NRev reverb => dac;
        // just a bit of reverb
        0.025 => reverb.mix;

        // the MIDI file in object
        MidiFileIn min;
        // the MIDI message shuttle
        MidiMsg msg;
        // the filename
        string filename;
        // if no command line arguments provided
        if( me.args() == 0 ) me.dir() + ""E.mid"" => filename;
        // else use that filename
        else me.arg(0) => filename;

        // open the file; exit on error
        if (!min.open(filename)) me.exit();

        // print out
        cherr <= ""----------"" <= IO.newline();
        cherr <= ""MIDI file: "" <= IO.newline()
              <= "" |- "" <= filename <= IO.newline()
```

```
        // print
        cherr <= ""----------"" <= IO.newline();
        cherr <= ""playing..."" <= IO.newline();
        cherr <= ""----------"" <= IO.newline();

        // flag
        int done;

        // entry point for each shred assigned to a track
        fun void doTrack( int track, float speed )
        {
            // hack polyphony
            Wurley s[4];
            // for each voice
             for( int i; i < s.size(); i++ )
             {
                // set gain
                .5 => s[i].gain;
                // connect to output
                s[i] => reverb;
            }
            // voice number for quick polyphony
            int v;

            // read through all MIDI messages on track
            while( min.read( msg, track ) )
            {
                // this means no more MIDI events at current time; advance time
                if( msg.when > 0::second )
                    msg.when * speed => now; // speed of 1 is nominal

                    // catch NOTEON messages (lower nibble == 0x90)
                if( (msg.data1 & 0xF0) == 0x90 && msg.data2 > 0 && msg.data3 > 0 )
                {
                    // get the pitch and convert to frequencey; set
                    msg.data2 => Std.mtof => s[v].freq;
                    // velocity data; note on
                    msg.data3/127.0 => s[v].noteOn;
                    // cycle the voices
                    (v+1)%s.size() => v;

                    // log
                    cherr <= ""NOTE ON track: "" <= track <= "" pitch: "" <=
msg.data2 <="" velocity: "" <= msg.data3 <= IO.newline();
                }
                // other messages
                else
                {
                    // log
                    // cherr <= ""----EVENT (unhandled) track:"" <= track <= ""
type:"" <= (msg.data1&0xF0)
                    //      <= "" data2:"" <= msg.data2 <= "" data3:"" <=
msg.data3 <= IO.newline();
                }
            }

            // done with track
            done++;
         }

        "));
    }
```

```
public void playF()
{
    Debug.Log("playF called!!!!!!!!");
    chuckSound.RunCode( (@"

    // path
    NRev reverb => dac;
    // just a bit of reverb
    0.025 => reverb.mix;

    // the MIDI file in object
    MidiFileIn min;
    // the MIDI message shuttle
    MidiMsg msg;

    // the filename
    string filename;
    // if no command line arguments provided
    if( me.args() == 0 ) me.dir() + ""F.mid"" => filename;
    // else use that filename
    else me.arg(0) => filename;

    // open the file; exit on error
    if (!min.open(filename)) me.exit();

    // print out
    cherr <= ""----------"" <= IO.newline();
    cherr <= ""MIDI file: "" <= IO.newline()
          <= "" |- "" <= filename <= IO.newline()
    // print
    cherr <= ""----------"" <= IO.newline();
    cherr <= ""playing..."" <= IO.newline();
    cherr <= ""----------"" <= IO.newline();

    // flag
    int done;

    // entry point for each shred assigned to a track
    fun void doTrack( int track, float speed )
    {
        // hack polyphony
        Wurley s[4];
        // for each voice
         for( int i; i < s.size(); i++ )
         {
            // set gain
            .5 => s[i].gain;
            // connect to output
            s[i] => reverb;
        }
        // voice number for quick polyphony
        int v;

        // read through all MIDI messages on track
        while( min.read( msg, track ) )
        {
            // this means no more MIDI events at current time; advance time
            if( msg.when > 0::second )
                msg.when * speed => now; // speed of 1 is nominal
```

```
                    // catch NOTEON messages (lower nibble == 0x90)
                if( (msg.data1 & 0xF0) == 0x90 && msg.data2 > 0 && msg.data3 > 0 )
                {
                    // get the pitch and convert to frequencey; set
                    msg.data2 => Std.mtof => s[v].freq;
                    // velocity data; note on
                    msg.data3/127.0 => s[v].noteOn;
                    // cycle the voices
                    (v+1)%s.size() => v;

                    // log
                    cherr <= ""NOTE ON track: "" <= track <= "" pitch: "" <=
msg.data2 <="" velocity: "" <= msg.data3 <= IO.newline();
                }
                // other messages
                else
                {
                    // log
                    // cherr <= ""----EVENT (unhandled) track:"" <= track <= ""
type:"" <= (msg.data1&0xF0)
                    //      <= "" data2:"" <= msg.data2 <= "" data3:"" <=
msg.data3 <= IO.newline();
                }
            }

            // done with track
            done++;
         }

        "));
    }

    public void playG()
    {
        Debug.Log("playG called!!!!!!!!");
        chuckSound.RunCode( (@"

        // path
        NRev reverb => dac;
        // just a bit of reverb
        0.025 => reverb.mix;

        // the MIDI file in object
        MidiFileIn min;
        // the MIDI message shuttle
        MidiMsg msg;

        // the filename
        string filename;
        // if no command line arguments provided
        if( me.args() == 0 ) me.dir() + ""G.mid"" => filename;
        // else use that filename
        else me.arg(0) => filename;
        // open the file; exit on error
        if (!min.open(filename)) me.exit();

        // print out
        cherr <= ""----------"" <= IO.newline();
        cherr <= ""MIDI file: "" <= IO.newline()
              <= "" |- "" <= filename <= IO.newline()
```

```
        // print
         cherr <= ""----------"" <= IO.newline();
         cherr <= ""playing..."" <= IO.newline();
         cherr <= ""----------"" <= IO.newline();

         // flag
         int done;

         // entry point for each shred assigned to a track
         fun void doTrack( int track, float speed )
         {
             // hack polyphony
             Wurley s[4];
             // for each voice
              for( int i; i < s.size(); i++ )
              {
                 // set gain
                 .5 => s[i].gain;
                 // connect to output
                 s[i] => reverb;
             }
             // voice number for quick polyphony
             int v;

             // read through all MIDI messages on track
             while( min.read( msg, track ) )
             {
                 // this means no more MIDI events at current time; advance time
                 if( msg.when > 0::second )
                     msg.when * speed => now; // speed of 1 is nominal

                     // catch NOTEON messages (lower nibble == 0x90)
                 if( (msg.data1 & 0xF0) == 0x90 && msg.data2 > 0 && msg.data3 > 0 )
                 {
                     // get the pitch and convert to frequencey; set
                     msg.data2 => Std.mtof => s[v].freq;
                     // velocity data; note on
                     msg.data3/127.0 => s[v].noteOn;
                     // cycle the voices
                     (v+1)%s.size() => v;

                     // log
                     cherr <= ""NOTE ON track: "" <= track <= "" pitch: "" <=
msg.data2 <="" velocity: "" <= msg.data3 <= IO.newline();
                 }
                 // other messages
                 else
                 {
                     // log
                     // cherr <= ""----EVENT (unhandled) track:"" <= track <= ""
type:"" <= (msg.data1&0xF0)
                     //      <= "" data2:"" <= msg.data2 <= "" data3:"" <=
msg.data3 <= IO.newline();
                 }
             }

             // done with track
             done++;
          }

         "));
    }
```

```
    public void playA()
    {
        Debug.Log("playA called!!!!!!!!");
        chuckSound.RunCode( (@"

        // path
        NRev reverb => dac;
        // just a bit of reverb
        0.025 => reverb.mix;

        // the MIDI file in object
        MidiFileIn min;
        // the MIDI message shuttle
        MidiMsg msg;

        // the filename
        string filename;
        // if no command line arguments provided
        if( me.args() == 0 ) me.dir() + ""A.mid"" => filename;
        // else use that filename
        else me.arg(0) => filename;

        // open the file; exit on error
        if (!min.open(filename)) me.exit();

        // print out
        cherr <= ""----------"" <= IO.newline();
        cherr <= ""MIDI file: "" <= IO.newline()
              <= "" |- "" <= filename <= IO.newline()
        // print
        cherr <= ""----------"" <= IO.newline();
        cherr <= ""playing..."" <= IO.newline();
        cherr <= ""----------"" <= IO.newline();

        // flag
        int done;

        // entry point for each shred assigned to a track
        fun void doTrack( int track, float speed )
        {
            // hack polyphony
            Wurley s[4];
            // for each voice
             for( int i; i < s.size(); i++ )
             {
                // set gain
                .5 => s[i].gain;
                // connect to output
                s[i] => reverb;
            }
            // voice number for quick polyphony
            int v;

            // read through all MIDI messages on track
            while( min.read( msg, track ) )
            {
                // this means no more MIDI events at current time; advance time
                if( msg.when > 0::second )
                    msg.when * speed => now; // speed of 1 is nominal
```

```
                    // catch NOTEON messages (lower nibble == 0x90)
                if( (msg.data1 & 0xF0) == 0x90 && msg.data2 > 0 && msg.data3 > 0 )
                {
                    // get the pitch and convert to frequencey; set
                    msg.data2 => Std.mtof => s[v].freq;
                    // velocity data; note on
                    msg.data3/127.0 => s[v].noteOn;
                    // cycle the voices
                    (v+1)%s.size() => v;

                    // log
                    cherr <= ""NOTE ON track: "" <= track <= "" pitch: "" <=
msg.data2 <="" velocity: "" <= msg.data3 <= IO.newline();
                }
                // other messages
                else
                {
                    // log
                    // cherr <= ""----EVENT (unhandled) track:"" <= track <= ""
type:"" <= (msg.data1&0xF0)
                    //      <= "" data2:"" <= msg.data2 <= "" data3:"" <=
msg.data3 <= IO.newline();
                }
            }

            // done with track
            done++;
         }

        "));
    }

    public void playB()
    {
        Debug.Log("playB called!!!!!!!!");
        chuckSound.RunCode( (@"

        // path
        NRev reverb => dac;
        // just a bit of reverb
        0.025 => reverb.mix;

        // the MIDI file in object
        MidiFileIn min;
        // the MIDI message shuttle
        MidiMsg msg;

        // the filename
        string filename;
        // if no command line arguments provided
        if( me.args() == 0 ) me.dir() + ""B.mid"" => filename;
        // else use that filename
        else me.arg(0) => filename;

        // open the file; exit on error
        if (!min.open(filename)) me.exit();

        // print out
        cherr <= ""----------"" <= IO.newline();
        cherr <= ""MIDI file: "" <= IO.newline()
              <= "" |- "" <= filename <= IO.newline()
```

```
        // print
        cherr <= ""----------"" <= IO.newline();
        cherr <= ""playing..."" <= IO.newline();
        cherr <= ""----------"" <= IO.newline();

        // flag
        int done;

        // entry point for each shred assigned to a track
        fun void doTrack( int track, float speed )
        {
            // hack polyphony
            Wurley s[4];
            // for each voice
             for( int i; i < s.size(); i++ )
             {
                // set gain
                .5 => s[i].gain;
                // connect to output
                s[i] => reverb;
            }
            // voice number for quick polyphony
            int v;

            // read through all MIDI messages on track
            while( min.read( msg, track ) )
            {
                // this means no more MIDI events at current time; advance time
                if( msg.when > 0::second )
                    msg.when * speed => now; // speed of 1 is nominal

                    // catch NOTEON messages (lower nibble == 0x90)
                if( (msg.data1 & 0xF0) == 0x90 && msg.data2 > 0 && msg.data3 > 0 )
                {
                    // get the pitch and convert to frequencey; set
                    msg.data2 => Std.mtof => s[v].freq;
                    // velocity data; note on
                    msg.data3/127.0 => s[v].noteOn;
                    // cycle the voices
                    (v+1)%s.size() => v;

                    // log
                    cherr <= ""NOTE ON track: "" <= track <= "" pitch: "" <=
msg.data2 <="" velocity: "" <= msg.data3 <= IO.newline();
                }
                // other messages
                else
                {
                    // log
                    // cherr <= ""----EVENT (unhandled) track:"" <= track <= ""
type:"" <= (msg.data1&0xF0)
                    //      <= "" data2:"" <= msg.data2 <= "" data3:"" <=
msg.data3 <= IO.newline();
                }
            }

            // done with track
            done++;
         }

        "));
    }
}
```

10. SceneSwitchGuitar.cs:

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

using UnityEngine.SceneManagement;


public class SceneSwitchGuitar : MonoBehaviour
{
    [SerializeField] private string loadLevel;
    // Start is called before the first frame update
    void Start()
    {

    }

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

    }

    void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            Debug.Log("initiaitng");
            Initiate.Fade("Guitar",Color.white,2.0f);
            //SceneManager.LoadScene(loadLevel);
        }
    }
}
```

11. ParticleA.cs:

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

public class ParticleA : MonoBehaviour
{
    public ParticleSystem p;

    void Start ()
    {
        p.emissionRate = 0.0f;
    }

    void Update ()
    {
        if (Input.GetKeyDown(KeyCode.A))
        {
            p.emissionRate = 150.0f;
        }

        else
        {
            p.emissionRate = 0.0f;
        }
    }


}
```

12. ParticleB.cs:

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

public class ParticleB : MonoBehaviour
{
    public ParticleSystem p;

    void Start ()
    {
        p.emissionRate = 0.0f;
    }

    void Update ()
    {
        if (Input.GetKeyDown(KeyCode.B))
        {
            p.emissionRate = 150.0f;
        }

        else
        {
            p.emissionRate = 0.0f;
        }
    }


}
```

13. ParticleD.cs:

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

public class ParticleD : MonoBehaviour
{
    public ParticleSystem p;

    void Start ()
    {
        p.emissionRate = 0.0f;
    }

    void Update ()
    {
        if (Input.GetKeyDown(KeyCode.D))
        {
            p.emissionRate = 150.0f;
        }

        else
        {
            p.emissionRate = 0.0f;
        }
    }


}
```

14. ParticleE.cs:

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

public class ParticleE : MonoBehaviour
{
    public ParticleSystem p;

    void Start ()
    {
        p.emissionRate = 0.0f;
    }

    void Update ()
    {
        if (Input.GetKeyDown(KeyCode.E))
        {
            p.emissionRate = 150.0f;
        }

        else
        {
            p.emissionRate = 0.0f;
        }
    }


}
```

15. ParticleG.cs:

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

public class ParticleG : MonoBehaviour
{
    public ParticleSystem p;

    void Start ()
    {
        p.emissionRate = 0.0f;
    }

    void Update ()
    {
        if (Input.GetKeyDown(KeyCode.G))
        {
            p.emissionRate = 150.0f;
        }

        else
        {
            p.emissionRate = 0.0f;
        }
    }


}
```

16. GuitarSounds.cs:

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

public class GuitarSound : MonoBehaviour
{
    ChuckSubInstance chuckGuitar;
    // Start is called before the first frame update
    void Start()
    {
        chuckGuitar = GetComponent<ChuckSubInstance>();
    }

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

    }

    public void playEchord()
    {
        Debug.Log("playEchord called!!!!!!!!");
        chuckGuitar.RunCode( @"

            SndBuf buf => dac;
            0.035 => buf.gain;
            me.dir() + ""reverEchord.wav"" => string Echord;

            Echord => buf.read;
            buf.samples() => buf.pos;

            0 => buf.pos;
            buf.length() => now;

        ");
    }

    public void playAchord()
    {
        Debug.Log("playAchord called!!!!!!!!");
        chuckGuitar.RunCode( @"

            SndBuf buf => dac;
            0.035 => buf.gain;
            me.dir() + ""reverAchord.wav"" => string Achord;

            Achord => buf.read;
            buf.samples() => buf.pos;

            0 => buf.pos;
            buf.length() => now;

        ");
    }
```

```
    public void playDchord()
    {
        Debug.Log("playDchord called!!!!!!!!");
        chuckGuitar.RunCode( @"

            SndBuf buf => dac;
            0.035 => buf.gain;
            me.dir() + ""reverDchord.wav"" => string Dchord;

            Dchord => buf.read;
            buf.samples() => buf.pos;

            0 => buf.pos;
            buf.length() => now;

        ");
    }

    public void playGchord()
    {
        Debug.Log("playGchord called!!!!!!!!");
        chuckGuitar.RunCode( @"

            SndBuf buf => dac;
            0.035 => buf.gain;
            me.dir() + ""reverGchord.wav"" => string Gchord;

            Gchord => buf.read;
            buf.samples() => buf.pos;

            0 => buf.pos;
            buf.length() => now;

        ");
    }

    public void playBchord()
    {
        Debug.Log("playBchord called!!!!!!!!");
        chuckGuitar.RunCode( @"

            SndBuf buf => dac;
            0.035 => buf.gain;
            me.dir() + ""reverBchord.wav"" => string Bchord;

            Bchord => buf.read;
            buf.samples() => buf.pos;

            0 => buf.pos;
            buf.length() => now;

        ");
    }



}
```

17. Word.cs:

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

[System.Serializable]
public class Word
{
    public string word;
    private int typeIndex;

    WordDisplay display;

    public Word(string _word, WordDisplay _display)
    {
        word = _word;
        typeIndex = 0;

        display = _display;
        display.SetWord(word);
    }

    public char GetNextLetter()
    {
        return word[typeIndex];
    }

    public void TypeLetter()
    {
        typeIndex++;
        //remove letter on screen
        display.RemoveLetter();
    }

    public bool WordTyped()
    {
        bool wordTyped = (typeIndex >= word.Length);
        if (wordTyped)
        {
            // Remove word on screen
            display.RemoveWord();
        }
        return wordTyped;
    }
    // Start is called before the first frame update
    void Start()
    {

    }

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

    }
}
```

18. WordInput.cs:

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

public class WordInput : MonoBehaviour
{
    public WordManager wordManager;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        foreach (char letter in Input.inputString)
        {
            wordManager.TypeLetter(letter);
            Debug.Log("PRESSED" + letter);

            // if (letter.Equals("e"))
            // {
            //     Debug.Log("if letter equals e....play sound!");
            //     FindObjectOfType<GuitarSound>().playEchord();
            // }
            // else if (letter.Equals("a"))
            // {
            //     FindObjectOfType<GuitarSound>().playAchord();
            // }
            // else if (letter.Equals("d"))
            // {
            //     FindObjectOfType<GuitarSound>().playDchord();
            // }
            // else if (letter.Equals("g"))
            // {
            //     FindObjectOfType<GuitarSound>().playGchord();
            // }
            // else if (letter.Equals("b"))
            // {
            //     FindObjectOfType<GuitarSound>().playBchord();
            // }

        }
    }
}
```

19. WordDisplay.cs:

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

public class WordDisplay : MonoBehaviour
{
    public Text text;

    public void SetWord(string word)
    {
        text.text = word;
    }

    public void RemoveLetter()
    {
        text.text = text.text.Remove(0, 1);
        text.color = Color.red;
    }

    public void RemoveWord()
    {
        Destroy(gameObject);
    }
    // Start is called before the first frame update
    void Start()
    {

    }

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

    }
}
```

20. WordManager.cs:

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



public class WordManager : MonoBehaviour
{
    public List<Word> words;
    private bool hasActiveWord;
    private Word activeWord;

    public Text winText;

    public WordSpawner wordSpawner;

    public static bool guitarDone = false;


    // Start is called before the first frame update
    void Start()
    {
        AddWord();
        //AddWord();
        winText.text = "";
    }

    public void AddWord()
    {

        //WordDisplay wordDisplay = wordSpawner.SpawnWord();
        Word word1 = new Word("adgebdb", wordSpawner.SpawnWord());
        Word word2 = new Word("daegeda", wordSpawner.SpawnWord());
        Word word3 = new Word("gbdba", wordSpawner.SpawnWord());
        Debug.Log(word1.word);

        words.Add(word1);
        words.Add(word2);
        words.Add(word3);
        //words.Add(word2);
    }
    public void TypeLetter(char letter)
    {
        Debug.Log("letter: " + letter);
        Debug.Log("letter type: " + letter.GetType().ToString());
            if (Input.GetKeyDown(KeyCode.E))
            {
                Debug.Log("if letter equals e....play sound!");
                FindObjectOfType<GuitarSound>().playEchord();
            }
            else if (Input.GetKeyDown(KeyCode.A))
            {
                FindObjectOfType<GuitarSound>().playAchord();
            }
```

```
            else if (Input.GetKeyDown(KeyCode.D))
            {
                FindObjectOfType<GuitarSound>().playDchord();
            }
            else if (Input.GetKeyDown(KeyCode.G))
            {
                FindObjectOfType<GuitarSound>().playGchord();
            }
            else if (Input.GetKeyDown(KeyCode.B))
            {
                FindObjectOfType<GuitarSound>().playBchord();
            }
            else
            {
                Debug.Log("uh oh");
            }


        if (hasActiveWord)
        {
            //remove from word
            if (activeWord.GetNextLetter() == letter)
            {
                activeWord.TypeLetter();
            }
        }
        else
        {
            foreach(Word word in words)
                {
                    if (word.GetNextLetter() == letter)
                    {
                        activeWord = word;
                        hasActiveWord = true;
                        word.TypeLetter();
                        break;
                    }
                }
        }

        if (hasActiveWord && activeWord.WordTyped())
        {
            hasActiveWord = false;
            words.Remove(activeWord);
        }


    }
    // Update is called once per frame
    void Update()
    {
        if (words.Count == 0)
        {
            StartCoroutine(Win());


        }
    }
```

```
    IEnumerator Win()
    {
            Debug.Log("WIN CALLED");
            winText.text = "COMPLETE";
            guitarDone = true;
            yield return new WaitForSeconds(3f);
            Initiate.Fade("EscapeScene",Color.white,2.0f);
            winText.text = "";
    }
}
```

21. SceneSwitchDrums:

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

using UnityEngine.SceneManagement;


public class SceneSwitchDrums : MonoBehaviour
{
    [SerializeField] private string loadLevel;
    // Start is called before the first frame update
    void Start()
    {

    }

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

    }

    void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            Debug.Log("initiaitng");
            Initiate.Fade("Drums",Color.white,2.0f);
            //SceneManager.LoadScene(loadLevel);
        }
    }
}
```

22. DrumSound.cs:

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

public class DrumSound : MonoBehaviour
{
    ChuckSubInstance chuckDrum;
    // Start is called before the first frame update
    void Start()
    {
        chuckDrum = GetComponent<ChuckSubInstance>();
    }

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

    }

    public void playHat()
    {
        Debug.Log("playHat called!!!!!!!!");
        chuckDrum.RunCode( @"

            SndBuf buf => dac;
            0.075 => buf.gain;
            me.dir() + ""ecohat.wav"" => string hat;

            hat => buf.read;
            buf.samples() => buf.pos;

            0 => buf.pos;
            buf.length() => now;

        ");
    }

    public void playSnare()
    {
        Debug.Log("playSnare called!!!!!!!!");
        chuckDrum.RunCode( @"

            SndBuf buf => dac;
            0.075 => buf.gain;
            me.dir() + ""ecosnare.wav"" => string snare;

            snare => buf.read;
            5::second => now;

        ");
    }
```

```
    public void playKick()
    {
        Debug.Log("playKick called!!!!!!!!");
        chuckDrum.RunCode( @"

            SndBuf buf => dac;
            0.075 => buf.gain;
            me.dir() + ""ecokick.wav"" => string kick;

            kick => buf.read;
            5::second => now;

        ");
    }

    public void playCymbal()
    {
        Debug.Log("playCymbal called!!!!!!!!");
        chuckDrum.RunCode( @"
            SndBuf buf => dac;
            0.075 => buf.gain;
            me.dir() + ""ecocymbal.wav"" => string cymbal;

            cymbal => buf.read;
            5::second => now;

        ");
    }
}
```

23. DrumGameController.cs:

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

using UnityEngine.UI;

public class DrumGameController : MonoBehaviour
{
    public static DrumGameController Instance;
    public ButtonB[] btns;

    static int simonMax;
    static float simonTime;

    static List<int> userList, simonList;

    public static bool simonIsSaying;
    int success = 0;
    public Text winText;
    public static bool drumsDone = false;

    // Start is called before the first frame update
    void Start()
    {
        Instance = this;

        simonMax = 4;
        simonTime = 0.5f;

        winText.text = "";

        StartCoroutine(SimonSays());

    }
    public void PlayerAction(ButtonB b)
    {
        //Debug.Log("b's id: " + b.id);
        userList.Add(b.id);
        Debug.Log("userlist: " + userList[userList.Count - 1]);
        Debug.Log("simonList: " + simonList[userList.Count - 1]);
        if (userList[userList.Count - 1] != simonList[userList.Count - 1])
        {
            Start();
            Debug.Log("Lose");
        }
        else if (userList.Count == simonList.Count)
        {
            Debug.Log("Next Level");

            StartCoroutine(SimonSays());
            success += 1;
            if (success == 3)
            {
                StartCoroutine(Win());
                //Initiate.Fade("EscapeScene",Color.white,2.0f);
            }
        }
    }
```

```
    IEnumerator Win()
    {
            Debug.Log("WIN CALLED");
            winText.text = "COMPLETE";

            drumsDone = true;
            Debug.Log(drumsDone);
            yield return new WaitForSeconds(2f);
            Initiate.Fade("EscapeScene",Color.white,2.0f);
            winText.text = "";
    }

    IEnumerator SimonSays()
    {
        Debug.Log("Prepare");
        yield return new WaitForSeconds(3);

        simonIsSaying = true;
        userList = new List<int>();
        simonList = new List<int>();


        for (int round = 0; round < 1; round++)
        {
            for (int i = 0; i < simonMax; i++)
            {
                int rand = Random.Range(0, 4);
                simonList.Add(rand+2);
                btns[rand].Action();
                yield return new WaitForSeconds(simonTime);
            }

            simonTime -= 0.015f;
            //simonMax++;
            simonIsSaying = false;


        }



    }
}
```

24. ButtonB.cs:

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

public class ButtonB : MonoBehaviour
{
    public int id;
    public Animator anim;


    // Start is called before the first frame update
    void Start()
    {
        id = transform.GetSiblingIndex();
    }


    void OnMouseDown()
    {
        if (!DrumGameController.simonIsSaying)
        {
            Action();
            Debug.Log("this id: " + this.id);
            DrumGameController.Instance.PlayerAction(this);

            if (this.id == 2)
            {
                FindObjectOfType<DrumSound>().playHat();
            }
            else if (this.id == 3)
            {
                FindObjectOfType<DrumSound>().playKick();
            }
            else if (this.id == 4)
            {
                FindObjectOfType<DrumSound>().playSnare();
            }
            else if (this.id == 5)
            {
                FindObjectOfType<DrumSound>().playCymbal();
            }
        }
    }


    public void Action()
    {
        anim.enabled = true;
        anim.SetTrigger("glow");

    }

}
```

25. ActivateNotes.cs

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

public class ActivateNotes : MonoBehaviour
{
    public GameObject notes;
    public static bool pianoDone = false;
    public static bool drumsDone = false;
    public static bool guitarDone = false;

    public static bool canOpenDoor = false;
    // Start is called before the first frame update
    void Start()
    {
        notes.SetActive(false);
        //FindObjectOfType<RoomSound>().playSynth();
    }

    // Update is called once per frame
    void Update()
    {
        //if (WordManager.guitarDone == true)
        if ( WordManager.guitarDone == true && PressKey.pianoDone == true &&
DrumGameController.drumsDone == true)

        {
            notes.SetActive(true);
            canOpenDoor = true;
        }
    }
}
```

26. DoorController.cs:

```
using UnityEngine;
using System.Collections;

public class DoorController : MonoBehaviour {

    public Animator animator;

    public bool doorOpen = false;


public static bool doorOpenPlayOcean = false;

    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            RaycastHit hit;
            // if the door is closed but you can now open it, then open it
            if (doorOpen == false && ActivateNotes.canOpenDoor == true)
            {
                if (Physics.Raycast(ray, out hit))
                {
                    if (hit.collider.tag == "Door")
                    {
                        open();
                    }
                }

             }
            //if door is open
            else if (doorOpen == true)
            {

                if (Physics.Raycast(ray, out hit))
                {
                     if (hit.collider.tag == "Door")
                    {
                        close();
                    }
                }

            }
        }
        //OPEN DOOR
    public void open()
    {
        Debug.Log("Setting Trigger");
        animator.SetTrigger("OpenDoor");
        doorOpen = true;

        doorOpenPlayOcean = true;
        Debug.Log("Opening door");
        Debug.Log("doorOpenplayocean is " + doorOpenPlayOcean + "and doorOpen
is " + doorOpen);
    }
 }
```

```
    //CLOSE DOOR
    public void close()
    {
        animator.SetTrigger("CloseDoor");
        doorOpen = false;


        doorOpenPlayOcean = false;
        Debug.Log("doorOpenplayocean is " + doorOpenPlayOcean + "and doorOpen is "
+ doorOpen);
    }
}
```

27. Door.js:

```
var inventoryScript:Inventory;
var theDoor:Transform;
var isOpen:boolean=false;
private var startRot:Quaternion;
var endRot:Quaternion;
var rotateSpeed:float=10;
var lockObj:GameObject;

function Start(){
    startRot=theDoor.rotation;
}
function Activate(){
    if(inventoryScript.items[0]==1){//We have the key
        isOpen=!isOpen;
        GetComponent.<AudioSource>().Play();
    }else{
        lockObj.GetComponent.<AudioSource>().Play();
    }

}

function Update () {
    if(isOpen){
        theDoor.rotation=Quaternion.RotateTowards(theDoor.rotation,endRot,rotateSpeed);
    }else{
        theDoor.rotation=Quaternion.RotateTowards(theDoor.rotation,startRot,rotateSpeed);
    }
}
```

28. NetworkPlayerSpawner.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Photon.Pun;
public class NetworkPlayerSpawner : MonoBehaviourPunCallbacks
{

    private GameObject SpawnedPlayerPrefab;

    public override void OnJoinedRoom()
    {
        base.OnJoinedRoom();
        SpawnedPlayerPrefab = PhotonNetwork.Instantiate("Network Player",
transform.position, transform.rotation);

    }

    public override void OnLeftRoom()
    {
        base.OnLeftRoom();
        PhotonNetwork.Destroy(SpawnedPlayerPrefab);
    }
}
```

29. NetworkPlayer.cs

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

using Photon.Pun;

namespace Com.MyCompany.MyGame
{
    public class NetworkPlayer : MonoBehaviourPunCallbacks
    {
        #region Public Fields
        [Tooltip("The local player instance. Use this to know if the local player
is represented in the Scene")]
        public static GameObject LocalPlayerInstance;
        #endregion

        // Start is called before the first frame update
        #region MonoBehaviour CallBacks

        private void Awake()
        {
            // used in GameManager.cs: we keep track of the localPlayer instance
to prevent instantiation when levels are synchronized
            if (photonView.IsMine)
            {
                NetworkPlayer.LocalPlayerInstance = this.gameObject;
            }
            // #Critical
            // we flag as don't destroy on load so that instance survives level
synchronization, thus giving a seamless experience when levels load.
            DontDestroyOnLoad(this.gameObject);
        }
        void Start()
        {
            Camera mainCamera = Camera.main;
            mainCamera.enabled = false;
            // Asigna la cámara a la vista del avatar
            PhotonView photonView =
LocalPlayerInstance.GetComponent<PhotonView>();
            photonView.ObservedComponents[0] = mainCamera;
        }

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

        }
        #endregion

    }
}
```

ANEXO B

Hoja de evaluación del juego

Preparada por

Mohamed Yassine Abcha

What age range do you think this game is suitable for (circle one):

3 – 6 6 – 9 9 – 12 12 – adult adult

How many minutes did it take you to finish the game?

Under 10 10-20 21-40 41-60 61-90 Over 90 mins.

If this was an abstract strategy game, would it be better if it had a theme? ... ... Yes No
If this was a themed game, did you like the theme? ... ... .Yes No
If this was a themed game, does the theme fit the play (mechanisms)?... ... .Yes No
Is there an unfair advantage depending on whether you move first or last?... ... .Yes No

Rate these on a continuum; circle one number in each area:

Complexity:

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
very simple average very complex

Game Instructions/Rules:

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
very simple average very complex

Luck vs. Skill:

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
pure luck half luck, half skill all skill

Uniqueness / Game Mechanics (How different was this game from other games?):

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
Not much different – Very different

Playing time (Was the game too short, too long or just right?):

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
Too short Just right Too long

Appearance (How much did you like the graphics/illustrations?):

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
Did not like – Loved

Materials (How much did you like the materials and/or game pieces):

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
Did not like Average Loved

Game Idea (Concept) or Theme:

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
Boring or weak OK Terrific

Interest (How much did you like this game?):

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
Hated it It was OK Loved it

Repeat Play (How often will you play this game?):

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
never again now & then a lot

Interaction (How much did the game play cause you to interact with other players?):

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
Never Only on my turn All the time

Waiting time with nothing to do (How much waiting between your turns?):

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
Very little Normal amount Too much

Game Options: Are there not enough options for what you can do on each turn, too many options (too many choices) or just the right amount?

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
not enough just right too many

Game pieces (size): Were the pieces too small, too big, or just the right size?

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
too small just right too big

Text size: Was the text on the board, cards or instructions too small or just right?

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
too small just right too big

Do you have any specific complaints, or precise suggestions that you feel would make the game better, especially in terms of the movement of the pieces, the balance and the object of the game?

ANEXO C

AUTOEVALUACIÓN

What age range do you think this game is suitable for (circle one):

3 – 6 6 – 9 9 – 12 **12 – adult** adult

How many minutes did it take you to finish the game?

Under 10 10-20 21-40 41-60 61-90 Over 90 mins.

If this was an abstract strategy game, would it be better if it had a theme? ... ... Yes No
If this was a themed game, did you like the theme? ... ... **Yes** No
If this was a themed game, does the theme fit the play (mechanisms)?... ... ... **Yes** No
Is there an unfair advantage depending on whether you move first or last?... ... .Yes **No**

Rate these on a continuum; circle one number in each area:

Complexity:

0 – 1 – 2 – 3 – **4** – 5 – 6 – 7 – 8 – 9 – 10
very simple average very complex

Game Instructions/Rules:

0 – 1 – 2 – **3** – 4 – 5 – 6 – 7 – 8 – 9 – 10
very simple average very complex

Luck vs. Skill:

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – **8** – 9 – 10
pure luck half luck, half skill all skill

Uniqueness / Game Mechanics (How different was this game from other games?):

0 – 1 – 2 – 3 – 4 – **5** – 6 – 7 – 8 – 9 – 10
Not much different – Very different

Playing time (Was the game too short, too long or just right?):

0 – 1 – 2 – **3** – 4 – 5 – 6 – 7 – 8 – 9 – 10
Too short Just right Too long

Appearance (How much did you like the graphics/illustrations?):

0 – 1 – 2 – 3 – 4 – 5 – **6** – 7 – 8 – 9 – 10
Did not like – Loved

Materials (How much did you like the materials and/or game pieces):

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – **8** – 9 – 10
Did not like Average Loved

Game Idea (Concept) or Theme:

0 – 1 – 2 – 3 – 4 – **5** – 6 – 7 – 8 – 9 – 10
Boring or weak OK Terrific

Interest (How much did you like this game?):

0 – 1 – 2 – 3 – **4** – 5 – 6 – 7 – 8 – 9 – 10
Hated it It was OK Loved it

Repeat Play (How often will you play this game?):

0 – 1 – **2** – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
never again now & then a lot

Interaction (How much did the game play cause you to interact with other players?):

0 – 1 – **2** – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
Never Only on my turn All the time

Waiting time with nothing to do (How much waiting between your turns?):

0 – 1 – **2** – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
Very little Normal amount Too much

Game Options: Are there not enough options for what you can do on each turn, too many options (too many choices) or just the right amount?

0 – 1 – 2 – 3 – **4** – 5 – 6 – 7 – 8 – 9 – 10
not enough just right too many

Game pieces (size): Were the pieces too small, too big, or just the right size?

0 – 1 – 2 – 3 – 4 – **5** – 6 – 7 – 8 – 9 – 10
too small just right too big

Text size: Was the text on the board, cards or instructions too small or just right?

0 – 1 – 2 – 3 – 4 – **5** – 6 – 7 – 8 – 9 – 10
too small just right too big

Do you have any specific complaints, or precise suggestions that you feel would make the game better, especially in terms of the movement of the pieces, the balance and the object of the game?

The game can be a lot better with some changes.

ANEXO D

Prueba del Usuario A (un ingeniero técnico de telecomunicaciones)

What age range do you think this game is suitable for (circle one):

3 – 6 **6 – 9** 9 – 12 12 – adult adult

How many minutes did it take you to finish the game?

Under 10 10-20 21-40 41-60 61-90 Over 90 mins.

If this was an abstract strategy game, would it be better if it had a theme? ……Yes **No**
If this was a themed game, did you like the theme? ……Yes **No**
If this was a themed game, does the theme fit the play (mechanisms)?……..**Yes** No
Is there an unfair advantage depending on whether you move first or last?……Yes **No**

Rate these on a continuum; circle one number in each area:

Complexity:

0 – 1 – 2 – 3 – 4 – **5** – 6 – 7 – 8 – 9 – 10
very simple average very complex

Game Instructions/Rules:

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
very simple average very complex

Luck vs. Skill:

0 – 1 – 2 – 3 – 4 – 5 – 6 – **7** – 8 – 9 – 10
pure luck half luck, half skill all skill

Uniqueness / Game Mechanics (How different was this game from other games?):

0 – 1 – 2 – 3 – 4 – **5** – 6 – 7 – 8 – 9 – 10
Not much different – Very different

Playing time (Was the game too short, too long or just right?):

0 – **1** – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
Too short Just right Too long

Appearance (How much did you like the graphics/illustrations?):

0 – 1 – 2 – 3 – 4 – **5** – 6 – 7 – 8 – 9 – 10
Did not like – Loved

Materials (How much did you like the materials and/or game pieces):

0 – 1 – 2 – 3 – 4 – **5** – 6 – 7 – 8 – 9 – 10
Did not like Average Loved

Game Idea (Concept) or Theme:

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
Boring or weak OK Terrific

Interest (How much did you like this game?):

0 – 1 – **2** – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
Hated it It was OK Loved it

Repeat Play (How often will you play this game?):

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
never again now & then a lot

Interaction (How much did the game play cause you to interact with other players?):

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
Never Only on my turn All the time

Waiting time with nothing to do (How much waiting between your turns?):

0 – 1 – 2 – 3 – 4 – **5** – 6 – 7 – 8 – 9 – 10
Very little Normal amount Too much

Game Options: Are there not enough options for what you can do on each turn, too many options (too many choices) or just the right amount?

0 – **1** – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
not enough just right too many

Game pieces (size): Were the pieces too small, too big, or just the right size?

0 – 1 – 2 – 3 – 4 – **5** – 6 – 7 – 8 – 9 – 10
too small just right too big

Text size: Was the text on the board, cards or instructions too small or just right?

0 – 1 – 2 – 3 – 4 – **5** – 6 – 7 – 8 – 9 – 10
too small just right too big

Do you have any specific complaints, or precise suggestions that you feel would make the game better, especially in terms of the movement of the pieces, the balance and the object of the game?

- Hay que tener conocimiento de las notas musicales para jugar el juego.
- El juego no está mal, le falta más dificultad.

ANEXO E

Prueba del Usuario B (un músico)

What age range do you think this game is suitable for (circle one):

3 – 6 6 – 9 **9 – 12** 12 – adult adult

How many minutes did it take you to finish the game?

Under 10 10-20 21-40 41-60 61-90 Over 90 mins.

If this was an abstract strategy game, would it be better if it had a theme? ......Yes No
If this was a themed game, did you like the theme? ....**Yes** No
If this was a themed game, does the theme fit the play (mechanisms)?.........**Yes** No
Is there an unfair advantage depending on whether you move first or last?......Yes No

Rate these on a continuum; circle one number in each area:

Complexity:

0 – **1** – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
very simple average very complex

Game Instructions/Rules:

0 – **1** – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
very simple average very complex

Luck vs. Skill:

0 – 1 – 2 – 3 – 4 – **5** – 6 – 7 – 8 – 9 – 10
pure luck half luck, half skill all skill

Uniqueness / Game Mechanics (How different was this game from other games?):

0 – 1 – 2 – 3 – 4 – **5** – 6 – 7 – 8 – 9 – 10
Not much different – Very different

Playing time (Was the game too short, too long or just right?):

0 – 1 – **2** – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
Too short Just right Too long

Appearance (How much did you like the graphics/illustrations?):

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – **8** – 9 – 10
Did not like – Loved

Materials (How much did you like the materials and/or game pieces):

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – **8** – 9 – 10
Did not like Average Loved

Game Idea (Concept) or Theme:

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – **8** – 9 – 10
Boring or weak OK Terrific

Interest (How much did you like this game?):

0 – 1 – 2 – 3 – 4 – 5 – **6** – 7 – 8 – 9 – 10
Hated it It was OK Loved it

Repeat Play (How often will you play this game?):

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
never again now & then a lot

Interaction (How much did the game play cause you to interact with other players?):

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
Never Only on my turn All the time

Waiting time with nothing to do (How much waiting between your turns?):

0 – 1 – 2 – 3 – 4 – **5** – 6 – 7 – 8 – 9 – 10
Very little Normal amount Too much

Game Options: Are there not enough options for what you can do on each turn, too many options (too many choices) or just the right amount?

0 – 1 – **2** – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10
not enough just right too many

Game pieces (size): Were the pieces too small, too big, or just the right size?

0 – 1 – 2 – 3 – 4 – **5** – 6 – 7 – 8 – 9 – 10
too small just right too big

Text size: Was the text on the board, cards or instructions too small or just right?

0 – 1 – 2 – 3 – 4 – **5** – 6 – 7 – 8 – 9 – 10
too small just right too big

Do you have any specific complaints, or precise suggestions that you feel would make the game better, especially in terms of the movement of the pieces, the balance and the object of the game?

- Game can be better. But it's acceptable for a student.
- Piano scene should be better. Because if you do a wrong piano key, you have to leave the scene and enter again.
- Guitar scene is too simple.