



UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
CURSO 2018-2019

VR8D

Aplicando sonido 8D a las tecnologías de realidad
virtual

Autor

José Adrián Garrido Puertas

Director

Marcelino Cabrera Cuevas





VR8D

Aplicando sonido 8D a las tecnologías de realidad virtual

Autor

José Adrián Garrido Puertas

Director

Marcelino Cabrera Cuevas

VR8D: aplicando sonido 8D a las tecnologías de realidad virtual

José Adrián Garrido Puertas

Palabras clave: VR, sonido, 8D, Unity, cardboard, eco, posicionamiento, inmersión, occlusion culling, shaders

Resumen

En la actualidad, se pueden observar gran variedad de estudios y trabajos relacionados con el concepto de inmersión, buscando con ello facilitar la asimilación por parte de los usuarios de la información que el programa, terapia, estudio... proporcionan. Con este fin, este proyecto se dispone a investigar la integración en la realidad virtual de los algoritmos de sonido 8D para mejorar la inmersión del usuario en un entorno virtual. El trabajo presentado ha sido montado en Unity, desarrollándolo como una aplicación en Android. Las interacciones del usuario con el entorno virtual solo requieren la aplicación, un móvil con una cardbord y uno auriculares estéreo.

VR8D: applying 8D sound to virtual reality technologies

José Adrián Garrido Puertas

Keywords: Vr, sound, Unity, cardboard, eco, positioning, immersion, occlusion culling, sharers

Abstract

At present, a wide variety of studies and works related to the concept of immersion can be observed, seeking to facilitate the assimilation by users of the information that the program, therapy, study... provide. To this end, this project is preparing to investigate the integration into virtual reality of the 8D sound algorithms to improve user immersion in a virtual environment. The work presented has been assembled in unity, developing it as an application in android. The user's interactions with the virtual environment only require the application, a mobile with a cardboard and a stereo headset.

Yo, **José Adrián Garrido Puertas**, alumno de la titulación Grado en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 76438494H, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: José Adrián Garrido Puertas
Granada a 30 de Agosto de 2019 .

D. **Marcelino Cabrera Cuevas**, Profesor del Departamento Sistemas de Lenguajes y Sistemas Informáticos de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***VR8D: aplicando sonido 8D a las tecnologías de realidad virtual***, ha sido realizado bajo su supervisión por **José Adrián Garrido Puertas**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 30 de Agosto de 2019 .

Los directores:

Marcelino Cabrera Cuevas

Agradecimientos

Empiezo los agradecimientos con mi tutor Marcelino Cabrera Cuevas, sin el cuál nunca se me habría ocurrido buscar un tfg que plantee algo distinto a la típica interacción con un ordenador de teclado-ratón.

Quiero hacer hacer una mención especial a mis padres y hermana, ya que sin su constante apoyo no habría podido llevar a cabo mis estudios ni llevar a puerto el proyecto aquí presentado. Si estoy donde estoy es también gracias a vosotros.

Ahora quiero agradecer a mi familia escogida, empezando por Rubén Jiménez Martínez, Vanesa Rodríguez Rodríguez, Nathaniel Jiménez Rodríguez y Juan Manuel Olivero Maroto, cuya amistad ha sido un importante apoyo durante estos años; y continuando por José Pedro Cirre Mateos, David Galindo López, Alén Blanco Domínguez, Michaelle López Eudaric, Benjamín Alba Morales, Miguel Ángel Cano Mesa, Raúl Alberto Calderón López y David Padilla Montero, quienes son antiguos alumnos, grandes amigos y mejores personas.

Mención también para Juan Hernández García, quien presta su voz para la primera escena de la aplicación y quien es un inestimable amigo, y para José María Esteo Christopoulos, antiguo estudiante de bellas artes y gran amigo con el que he trabajado en unity anteriormente.

Agradecimientos también a aquellos profesores que se han dedicado a nosotros sus estudiantes, alentandonos a mantener la curiosidad sobre la profesión que hemos escogido; a mis compañeros de carrera, tanto los que han terminado como los que no, ya que hemos compartidos aulas, alegrías, estrés y sobre todos muchos cafés; y a la universidad, que aunque con el paso de los años sientes que te ha quitado mucho, al terminar te das cuentas que te ha dado mucho más.

Por último, quiero hacer una mención especial a Luis Castillo Vidal, que como profesor me animó inconscientemente a seguir adelante con la carrera durante una época peor de mi vida, demostrando que el que quiere puede.

Índice

1. Introducción	21
1.1. Contexto	21
1.2. Motivación	22
1.3. Objetivo	22
2. Especificaciones y Requisitos	25
2.1. Descripción del proyecto	25
2.1.1. Introducción	25
2.1.2. Equipo de desarrollo	25
2.1.3. Comparativa de motores gráficos	26
2.1.4. Decisión sobre el motor	30
2.2. Requisitos	30
2.2.1. Funcionales	30
2.2.2. No funcionales	31
2.2.2.1. Requisitos relacionados con el concepto de inmersión	31
2.2.2.2. Requisitos relacionados con el concepto de gamificación	32
3. Planificación General	34
3.1. Introducción	34
3.2. Planificación	34
3.3. Tareas	34
3.3.1. Etapa de análisis	35
3.3.2. Etapa de diseño	35
3.3.3. Etapas de Implementación, Testeo & Evolución	36
4. Análisis	39
4.1. Introducción	39
4.2. Estado del arte	39
4.2.1. Sonido 8D	39
4.2.2. Realidad Virtual	39
4.2.3. Gamificación	40
4.3. Tecnologías y herramientas	40
4.3.1. Lenguaje de programación C#	40
4.3.2. GitKraken y GitHub	41
4.3.3. Unity	41
4.3.4. Cardboard	41
5. Diseño	44
5.1. Visión global de la aplicación	44

5.2.	Escenas	44
5.2.1.	Init	44
5.2.1.1.	Componentes de Init	45
5.2.2.	EchoChamber	45
5.2.2.1.	Componentes de EchoChamber	47
5.2.3.	Forest	47
5.2.3.1.	Componentes de Forest	49
5.3.	Modificaciones del algoritmo 8D	50
6.	Implementación y control de versiones	53
6.1.	Introducción	53
6.2.	Control de versiones	53
6.2.1.	Cambios orgánicos en el planteamiento de la aplicación	54
6.3.	Implementación general: setUp de la aplicación	54
6.3.1.	Paquetes a descargar y assets gratuitos	55
6.4.	Jugador	57
6.4.1.	Controles de movimiento	57
6.4.2.	Control de cámara en VR	58
6.4.3.	Retícula	
6.4.3.1.	Interacción de la retícula	61
6.4.4.	Aplicar sonido 8D a un objeto	64
6.5.	Implementación por escenas	65
6.5.1.	Escena Init	65
6.5.1.1.	Cubo para interactuar	65
6.5.2.	Escena Echo Chamber	66
6.5.2.1.	Aplicar eco	66
6.5.2.2.	Cambiar en escena los materiales de la habitación	67
6.5.2.3.	Icosaedro el sonido	72
6.5.3.	Escena Forest	72
6.5.3.1.	Oclusion Culling	72
6.5.3.2.	Pájaros y su búsqueda	73
7.	Pruebas	76
7.1.	Introducción	76
7.2.	El caso de las retículas rebeldes	76
7.3.	La resina del bosque no me permite andar	77
7.4.	Mi código, mis normas	77
A.	Manual de Usuario	79
A.1.	Material necesario	79
A.2.	Acciones del jugador	79
A.3.	Objetivos en las pantallas	79
A.3.1.	Init	79
A.3.2.	EchoChamber	79

A.3.3. EchoChamber	80
B. Anecdotas	82
B.1. Biclope involuntario	82
C. GitFlow	84
C.1. ¿Qué es GitFlow?	84
C.2. ¿Cómo funciona? [20] [21]	84
C.2.1. Ramas Develop & Master	84
C.2.2. Ramas Feature	85
C.2.3. Ramas Release	85
C.2.4. Ramas Hotfix	86

Índice de figuras

3.1. Distribución del tiempo de desarrollo de la aplicación	34
5.1. Instrucciones de uso	45
5.2. Cubo	45
5.3. Icosaedro sonoro	46
5.4. Menú e instrucciones de la habitación	46
5.5. Menú para cambio de materiales	47
5.6. Menú en el bosque	48
5.7. Pajaros en el bosque	48
5.8. Comparativa entre shader de alta calidad y baja calidad	49
5.9. Percepción del algoritmo	50
5.10. Zonas del algoritmo	51
6.1. Parte superior del gitTree	53
6.2. Parte inferior del gitTree	54
6.3. Fantasy Forest	56
6.4. Hand Painted Forest Lite	56
6.5. Living Birds	57
6.6. Aspecto del Jugador en el árbol de escena	57
6.7. Cubo con interacciones	63
6.8. Script Resonace AudioSource y Resonance AudioListener en el Inspector . .	64
6.9. Resonance Audio Room ajustado a la habitación de la escena	66
6.10. Script Resonace Audio Room	67
6.11. Dropdown en la jerarquía de escena	68
6.12. Script de un dropdown	69
6.13. Menú de cambio de material	70
6.14. Parámetros aplicados en la aplicación para el culling	73
6.15. Script Resonace Audio Room	73
C.1. Ramas Develop y Master	84
C.2. Rama feature	85
C.3. Rama release	86

C.4. Rama hotfix	87
----------------------------	----

Índice de Códigos

1. PlayerWalk.cs	57
2. GvrReticleShader.shader	59
3. GVRButton.cs	61
4. ChangeSound.cs	65
5. Función changeMaterialLeft	71
6. Teleporter.cs	72
7. Ejemplo de cambio de audio para pájaro	73
8. función checkBird	74

1. Introducción

1.1. Contexto

El contexto en el que vivimos en la actualidad se presenta como un marco en constante cambio, requiriendo de esta forma que las personas sean capaces de adaptarse mejor y más rápidamente a una gran variedad de situaciones.

En el caso de aquellas personas que trabajan en el sector tecnológico esta situación siempre se ha dado, pero lo cierto es que los últimos años se ha incrementado, teniendo por ejemplo que aprender en el menor tiempo posible un lenguaje de programación, preparar una presentación para un proyecto...

Esto nos ha llevado a buscar nuevos conceptos y métodos para poder sobreponernos a una situación que es una gran fuente de estrés, y aunque se han planteado gran cantidad de soluciones que pasan desde algo tan simple como aprender a gestionar tu tiempo de forma eficiente, hasta métodos de aprendizaje alternativos, lo cierto es que la tecnología puede ayudar de una forma mucho más activa de lo que podría parecer a simple vista.

También hay que remarcar que cada vez se busca poder trabajar de forma remota, ya sea como en el caso de los desarrolladores que pueden estar en constante contacto con miembros del equipo en distintas partes del mundo, conferencias por parte de distintas personas, trabajos conjuntos de arte mediante internet o incluso llegando a realizarse operaciones a distancia [1]. Todo esto requiere que el servicio sea lo más rápido posible, pero cada vez más se exige también que las posibilidades que abarque aumenten de manera gradual.

Otro tema que también es interesante abordar es el videojuego, que ha demostrado ser una herramienta muy valiosa a la hora del aprendizaje, quedando patente en casos de gente que conoce a la perfección las diferentes etapas del videojuego al que dediquen en ese momento, llegando a saber ingentes cantidades de información de un mundo virtual, basado o no en el nuestro.

Hay muchos factores que facilitan la asimilación de información, como la pasión por el tema tratado o lo amena que la información se distribuya de modo que el usuario pueda consumirla en pequeñas dosis, pero un concepto que no empezó a tratarse hasta hace relativamente poco fue el de la *inmersión*, que a priori parece un concepto realmente potente que podría darnos una solución duradera.

Todo lo anteriormente mencionado pide una solución totalmente innovadora... o quizás no tan innovadora. Pensándolo fríamente hay buenas ideas en el pasado que quizás no se exploraron por la limitación de la tecnología de la época. Esto ya se está viendo con ideas como el *raytracing*, cuyo fue planteado en 1980 por Turner Whitted.

A mediados del siglo XX se empezaron a ver aparatos para visualizar fotografías en 3D

llamados *View-Master* o una patente de 1957 para unas gafas de realidad virtual [2], por no mencionar que uso de entornos completamente virtuales para entrenamientos militares en la aviación lleva con nosotros desde la década de los 70 [3], suceso que demostró que no es necesario realizar una práctica con un avión real para obtener los conocimientos requeridos.

Basándome en lo anteriormente expuesto, puedo decir que quizás el concepto de entorno virtual pueda ser de gran ayuda, pero está claro que por sí solo no termina de completar una idea que pueda ser funcional y a la vez no haya ya sido explotada.

1.2. Motivación

La motivación que me lleva a este trabajo no es otra que contribuir a lo anteriormente expuesto. La tecnología avanza inexorablemente y no para de incorporar conceptos de diferentes doctrinas, y es necesario que los desarrolladores dediquemos tiempo para poner ideas nuevas sobre la mesa, o al menos escoger ideas anteriores y explotar su verdadero potencial.

Ante esto, me plantee rescatar el concepto de realidad virtual y unirlo los conceptos de gamificación [4] [5] e inmersión en el entorno.

Ahora solo nos queda encontrar una forma de aplicarlos dentro de un entorno virtual, de forma que lo primero que necesitaríamos sería un objetivo para cumplir con la gamificación y alguna forma de aumentar la inmersión entre los distintos escenarios que pudiéramos encontrar en ella.

Lo primero que puede venir a la cabeza es mejorar los gráficos en la aplicación, pero lo cierto es que vamos a trabajar con un dispositivo móvil ya que dispositivos como Oculus Rift S cuestan alrededor de 250 euros [6].

Ante este mercado hay que plantear otra solución que no aumente excesivamente el procesado en la aplicación o requiera de equipos de elevado precio,

La solución por la que opté al final fue el sonido 8D. La idea de utilizar esta tecnología surgió a raíz de un video sobre ella por Jaime Altozano [7], y aunque parezca una tecnología revolucionaria, en realidad tiene ya un tiempo y una serie de problemas [8] que quizás sí podamos ser capaces de solucionar con la tecnología actual.

1.3. Objetivo

Ya tenemos las herramientas conceptuales, pero hay que detenerse un poco y analizar cuál o cuáles son los objetivos que se pretenden uniendo esta serie de tecnologías.

El primer objetivo por supuesto hacer la modificaciones pertinentes en el algoritmo de sonido 8D con la intención de mejorarlo e introducirlo dentro de una aplicación de realidad virtual.

Otro objetivo es observar el concepto de inmersión por parte del usuario al tener distintos focos de sonido que se sitúen a su alrededor, ya sean estáticos o tengan algún tipo de movimiento asignado.

Como último objetivo se encuentra la interacción auditiva por parte del usuario y como esta cambia la experiencia del usuario.

2. Especificaciones y Requisitos

2.1. Descripción del proyecto

2.1.1. Introducción

Esta sección va a hablar los requisitos principales para el desarrollo de una aplicación que haga uso de los algoritmos de 8D presentados en la actualidad y aplicarlos en un entorno de realidad virtual.

El caso presentado comparte similitudes con el desarrollo de un videojuego, las cuales son debidas a que la aplicación en cuestión es un entorno interactivo para poder testear el algoritmo de sonido 8D. Esto se desarrolla en los siguientes apartados.

2.1.2. Equipo de desarrollo

Para proyectos de mayor envergadura, el desarrollo de una aplicación de estas características requiere un equipo conformado por los siguientes integrantes:

- Gestión/Producción:
 - CEO
 - Directores de proyecto
 - Productores
- Diseño:
 - Game designer
 - Level designer
- Arte y animación:
 - 2D artist:
 - Concept artist
 - Pixel artist
 - UI artist
 - 3D artist (modelaje, iluminación, texturización...):
 - Personajes
 - Escenario
- Animación:
 - Rigging
 - Animator
- Sonido:

- Ingeniero de sonido
- Compositor
- Programación:
 - Backend
 - Frontend
- Quality Assurance (QA):
 - Testers
 - Control de calidad

En este caso, el equipo está concentrado en una sola persona trabajando todos los aspectos, de forma que se recurre también a assets gratuitos para poder suplir las disciplinas de las que no se tiene conocimiento.

2.1.3. Comparativa de motores gráficos

Uno de los aspectos más importantes en el desarrollo de esta aplicación es contar con un framework 4 o un motor gráfico sobre el cual poder desarrollarla. A lo largo de los últimos años se han estandarizado una serie de productos que facilitan este desarrollo.

Por ello, se estudiarán diferentes opciones disponibles, teniendo en cuenta que no se pretende señalar todos y cada uno de los motores gráficos del mercado sino una selección de ellos de acuerdo a mis intenciones de abarcar el máximo espectro posible sin citar a todos.

A continuación, se señalan algunos casos:

▪ Source 2 Engine

Sucesor del motor gráfico Source propiedad de Valve, y motor de varios juegos famosos como Portal o Half Life. Estará disponible de forma pública y gratuita siempre y cuando se publique para la plataforma de Valve, Steam. Compatible con Vulkan (OpenGL) y usa un motor de físicas propio llamado Rubikon que sustituye a Havok.

Algunos de los juegos realizados con él son Counter-Strike: Global Offensive y Dota 2.

▪ Unity

Disponible para Windows, OS X y Linux, históricamente asociado a juegos de menor presupuesto, juegos indie y de móviles.

Dispone de varias versiones:

- Unity Personal: gratuita si no se sobrepasan los 100 mil dólares de ingresos
- Unity Plus: suscripción de 35 dólares al año, enfocado a desarrolladores móviles con ingresos menores a 200 mil dólares y que incluyen algunos servicios de Unity
- Unity Professional: suscripción de 125 dólares al mes, sin límite de ingresos, con todos los servicios de Unity
- Unity Enterprise
- Unity Educational

Las versiones Pro y Plus ofrecen soporte a la versión y acceso a todas las actualizaciones. También cabe señalar la Asset Store, lugar donde se concentran extensiones, herramientas y assets para los usuarios tanto gratuitos como de pago.

Algunos juegos hechos con Unity son Wasteland 2, Pillars of Eternity, Hearsthstone o Firewatch.

■ **CryEngine**

Desarrollado por Crytek y usado por primera vez en el juego Far Cry. La versión 5 utiliza de forma nativa DirectX12, Vulkan y soporte para VR.

Introdujo un nuevo modelo de licencias, el “paga lo que quieras”, 100 % royalty-free en la actualidad, para las plataformas Windows, Linux, PlayStation 4, Xbox One, Oculus Rift, HTC-Vive, Open-Source VR y PlayStation VR.

También posee un bazar, el “CRYENGINE Marketplace” donde los beneficios de las ventas son de un 70 % para el motor y el restante 30 % para el desarrollador del contenido.

Juegos que usan este motor: Saga Crysis, Ryse: Son of Rome, Aion Online (MMORPG online).

■ **Unreal Engine**

Sucesor de Unreal Development Kit (UDK), propiedad de Epic Games. Gratuito en la actualidad, aunque se paga a la empresa un 5 % de los beneficios cada trimestre a partir del momento en que el juego gane sus primeros 3000 dólares. Desarrollado en C++, para su uso en plataformas como Windows, OS X, Linux, iOS, Android, PlayStation 4, Xbox One, Nintendo Switch y navegadores (HTML5). También tiene soporte a Vulkan.

Al igual que Unity, tiene su propio bazar llamado “Unreal Engine Marketplace”, donde permite comprar y vender contenido (desde modelos, a tutoriales pasando por sonidos, efectos especiales, etcétera). También ha puesto en marcha un programa llamado “Unreal Dev Grants” con un presupuesto de cinco millones de dólares destinado a financiar a los desarrolladores que presenten proyectos innovadores usando el motor. Probablemente sea el motor más usado en la actualidad.

Juegos que usan este motor: DMC (Devil May Cry de Ninja Theory), saga Batman Arkham.

- **Snowdrop**

Motor propiedad de Ubisoft creado por Massive Entertainment (Ubisoft). Codificado en C++, tardó cinco años en ser desarrollado.

Su punto fuerte es la iluminación y el sistema de destrucción. Utiliza el motor de físicas Havok. Este motor está en esta lista porque a pesar de ser uno de los mejores del mercado no está disponible para todo el mundo: Sólo los equipos de desarrollo de Ubisoft tienen acceso a este motor.

Juegos: Tom Clancy's: The division, Mario + Rabbids Kingdom Battle, Skull & Bones.

- **Frostbite**

Desarrollado por DICE (Electronic Arts) y diseñado para ser un motor exclusivo de EA.

Inicialmente diseñado para hacer juegos en primera persona, ha ido evolucionando abrazando otros géneros. Codificado en C++, C#, Lua y IronPython. Las plataformas objetivo son PC, PlayStation 4 y Xbox One. Enfocado a permitir una mayor escala de interacciones multijugador en escenarios dinámicamente destruibles con condiciones atmosféricas cambiantes.

Como Snowdrop, es un motor propietario. Su presencia aquí es para poder comparar dos motores propietarios.

Juegos: Battlefield, Need for speed, Dragon Age, Mirror's Edge, FIFA, Star Wars: Battlefront

- **Amazon Lumberyard**

Motor propiedad de Amazon, gratuito y orientado a juegos AAA (grandes producciones).

Basado en CryEngine, está programado en C++ y en Lua. Tiene como características principales la integración con Amazon Web Services (AWS) y Twitch. El código fuente es completo y gratuito y no hay cuotas de suscripción ni requisitos económicos.

Solamente hay que pagar por los servicios de AWS que se utilicen (así es como sacan beneficios). Las plataformas objetivo son Windows, PlayStation 4, Xbox One, iOS, Android (con soporte limitado en estas dos últimas), Oculus Rift, HTC-Vive, OpenSource VR y PlayStation VR.

Juegos: Star Citizen

- **UbiArt framework**

Motor gráfico propiedad de Ubisoft y diseñado por el creador de Rayman, Michel Ancel. Totalmente centrado en la creación de proyectos 2D y 2.5D, responde a una búsqueda de la compañía de facilitar el desarrollo de juegos a un equipo pequeño de personas y con un presupuesto reducido.

Su punto fuerte es la facilidad para crear animaciones que da a los artistas y el aspecto artístico presente en los títulos desarrollados con este motor.

Juegos: Rayman Origins, Rayman Legends, Valiant Hearts, Child of light

- **GameMaker**

Propiedad de YoYo Games y diseñado para permitir a usuarios sin conocimientos de programación desarrollar juegos fácilmente. Contiene un lenguaje de programación de scripts, Game Maker Language (GML), para usuarios experimentados. Licencia EULA.

Juegos: Saga Hotline Miami, Undertale.

2.1.4. Decisión sobre el motor

A la hora de decir del motor, los factores que han influido son los siguientes:

- Mínimo coste de obtención de licencia
- Conocimiento previo.
- Portabilidad.
- Escalabilidad.

Hablando de costes, todos tiene la posibilidad de trabajar gratuitamente, pero sólo Unity y Unreal disponen de documentaciones para prácticamente todo lo que se nos pueda ocurrir durante el desarrollo, por lo que hago la discriminación a estos dos.

Respecto al conocimiento previo, en ambos tengo una larga experiencia, por lo que no es un factor realmente determinante para excluir definitivamente a ninguno de los dos, aunque con Unity si me he dedicado a trabajar en Game Jams, lo que hace que su interfaz se me antoje más cómoda en una situación de estrés.

Lo cierto es que con respecto a la escalabilidad, ambos estarían también empatados, pues ambos disponen de un gran abanico de módulos, plugins y assets externos. Ante esta situación solo puedo determinar que en la escalabilidad, ambos se encuentran igualados.

En caso en particular, donde la aplicación será para android, ambos tienen la capacidad de exportarla para este tipo de dispositivos.

Finalmente se ha optado por Unity, principalmente por mi anterior experiencia con él en situaciones de estrés trabajando en *Game Jams*.

2.2. Requisitos

El servicio que esta aplicación proporciona es simple, ya que se compone de tres entornos virtuales que por el que poder moverte e interaccionar, pero debemos tener en cuenta una serie de requisitos indispensables para el correcto funcionamiento de esta.

Dicho esto, nos disponemos a enumerar y describir dichos requisitos para una mejor comprensión por parte del lector de las intenciones de esta aplicación.

2.2.1. Funcionales

Aquí nos disponemos a hablar de todos aquellos requisitos que la aplicación debe proporcionar directamente. En nuestro caso, estos requisitos son:

- Entornos virtuales diferenciados entre sí

- Sonido preparado mediante los algoritmos 8D
- Interfaz adecuada a la situación planteada

Aunque sean pocos, es importante tener en cuenta que cada uno de ellos es imprescindible a la hora del correcto funcionamiento de la aplicación, y por consiguiente, no cumplir uno de ellos invalida la aplicación de forma automática. Así pues, desarrollarlos un adecuadamente.

Cuando se habla del *entorno virtual* se quiere hacer referencia al hecho de que necesitamos un entorno por el que el usuario pueda moverse y trabajar con los elementos dentro del entorno. Es importante remarcar que en este proyecto, los entornos van aumentando su complejidad gráfica e interna para así desarrollar una sensación de progresión en el usuario.

El sonido es imprescindible para el usuario, ya que proporciona el feedback que se requiere de la aplicación para poder interactuar con los elementos de cada escena. Si el sonido funciona mal, o directamente no funciona, esta aplicación puede considerarse un auténtico fracaso.

Al hablar de una interfaz adecuada, se hace referencia al hecho debe cumplir con su propósito y proporcionar feedback al usuario sobre qué se está haciendo con ella. Con respecto a la interfaz hay otros conceptos a tener en cuenta, pero ya entran dentro de los requisitos no funcionales, por lo que los desarrollaremos siguiente apartado,

2.2.2. No funcionales

Analizar los requisitos que la aplicación debe proporcionar de forma no directa puede ser un poco más problemático, porque dentro de ella se quieren integrar conceptos más abstractos.

Para hablar de estos requisitos, vamos a hacer una división sobre el objetivo que pretenden.

2.2.2.1. Requisitos relacionados con el concepto de inmersión

Esta sección plantea aquellos requisitos que busca hacer que la experiencia con la aplicación sea lo más inmersiva posible, de forma que el usuario pueda “olvidar” en su subconsciente que se encuentra en un entorno virtual.

Conseguir esto requiere varios factores a tener en cuenta, pero el primero sobre el que debemos hablar es la tasa de frames, la cual deberá ser estable dentro de las diferentes escenas. La estabilidad predominará sobre la cantidad de frames ya que se va a trabajar con un smartphone antiguo, y una gran cantidad de frames implicaría también reducir

el aspecto gráfico de la aplicación.

Otro aspecto a tener en cuenta son los tiempos de carga. El planteamiento de esta aplicación es tener unos tiempos de carga prácticamente inexistentes, de forma que el usuario no sienta una desconexión abisal entre las diferentes escenas.

Aquí se vuelve a hablar de los menús, ya que estos no deben estar integrados en el visor del usuario, si no que deben ser elementos que encontrremos por el entorno. Este planteamiento se debe a que la intención no es simular que el usuario lleva un casco con una interfaz, si no que se pretende simular que el usuario realmente se encuentra en las escenas presentadas. Un menú que interfiera con la visión rompería la ilusión de verse en otro lugar distinto a la habitación donde se está probando la aplicación.

El sonido 8D tiene también un valor añadido en los requisitos no funcionales, ya que este debe proporcionar la posición del emisor de una forma clara, de forma que el usuario sepa en todo momento dónde se sitúa el foco emisor. Esto será útil pues varios de los emisores que se encontrarán en la escena serán también elementos con los que poder interaccionar.

2.2.2.2. Requisitos relacionados con el concepto de gamificación

Se pretende que parte de las interacciones se produzcan surgidas de un interés relacionado con la gamificación. De esta forma, el usuario aprenderá a moverse y a interaccionar con los distintos elementos que componen la escena tomando como referencia los videojuegos.

3. Planificación General

3.1. Introducción

En esta sección se describe la planificación temporal del proyecto, así como el beneficio monetario que se espera de la aplicación.

3.2. Planificación

El desarrollo de este proyecto se inició seriamente en Marzo de 2019, pero debido a asuntos personales del autor, no se pudo dedicar todo el tiempo necesario desde el mismo inicio, lo que provocó que la implementación y pruebas se retrasaran en el marco temporal.

La planificación temporal de este proyecto se divide en una serie de etapas bien diferenciadas, mostradas en el siguiente diagrama de Gantt:



Figura 3.1: Distribución del tiempo de desarrollo de la aplicación

3.3. Tareas

Ahora pasamos a repasar las distintas tareas que podemos encontrar a lo largo de todo el periodo de desarrollo, Para facilitar la comprensión las agrupamos según las diferentes

etapas del desarrollo.

3.3.1. Etapa de análisis

Durante esta etapa se tuvo que analizar tanto las herramientas como los diferentes planteamientos para afrontar el problema al que nos enfrentamos. En concreto, gran parte del tiempo se dedicó a comprender tanto el algoritmo 8D como las diferentes implementaciones de la realidad virtual.

Hay que tener en cuenta también que es importante conocer cómo funciona tanto el 8D como la realidad virtual, pero los conceptos no se quedan solo ahí, ya que una parte fundamental es la inmersión dentro de la aplicación y cómo emplearla de forma natural.

Esto provocó un estudio exhaustivo de las diferentes formas de implementar un interfaz de usuario en realidad virtual, así como las mejores maneras de interactuar con ellas.

Quizás el análisis que más tiempo llevó fue el del algoritmo 8D, pues requería entender desde cero un concepto nuevo para mí, ya que al principio lo entendía de una manera totalmente errónea. A priori, este debe de trabajar desde la ubicación del usuario al foco, pero los cálculos y las implementaciones vistas me han demostrado que se hace un enfoque bidireccional, de forma que se pueda calcular el volumen del sonido de una forma más precisa.

También cabe destacar que desde el principio planteé la aplicación a dispositivos móviles, ya que cualquier otra opción implicaría inevitablemente una inversión de dinero que no podía asumir.

3.3.2. Etapa de diseño

Abordados ya los conceptos que debía tener claros, empecé con el planteamiento de la aplicación.

La primera idea fue hacer un entorno en un bosque donde escuchar el sonido de varios pájaros revoloteando, y de hecho este planteamiento es que lleva a la escena tres de la aplicación, pero me di cuenta que entonces dejaría muchos conceptos sin explorar, como el eco en una habitación o interactuar con un sujeto que se dirija a ti específicamente.

Este planteamiento fue el que me llevó a buscar nuevas ideas para las escenas, dando lugar a las escenas uno y dos, donde tendremos que buscar a un objetivo invisible y experimentar con un emisor de sonido en un entorno con eco respectivamente.

Parte de esta etapa la dediqué también a buscar diferentes assets que pudieran servir para completar los entornos en lo que se moverá el usuario, haciendo de esta forma que

sean más creíbles e inmersivos.

Por último me plantee cómo explicar las instrucciones de uso, de forma que para evitar ventanas emergentes que romperían la inmersión, o complejos monólogos que abrumaran al usuario, decidí explicarme con letras grandes en una de las paredes, ya que se añadiría otro elemento a las escenas, haciendo juego con los menús que pudiéramos encontrar en ellas.

3.3.3. Etapas de Implementación, Testeo & Evolución

Estas tres etapas se presentan de forma casi simultánea durante este desarrollo, por lo que considero imprescindible abordarlas también de esta forma.

Empecé implementando el bosque, donde lo principal era:

- Crear un terreno irregular para dar una sensación más realista
- Utilizar un algoritmo procedural ya implementado para situar árboles e hierba
- Situar los elementos que podamos encontrar extra, como las rocas o las estatuas que se encuentran en el bosque

Lo siguiente fue trabajar con el asset “living birds” para disponer los diferentes pájaros por la zona de forma aleatoria y automática, así como implementar en ellos el 8D.

Cuando esto se llevó a cabo, pase a desarrollarla habitación con eco, que implicó también varias tareas:

- Crear el entorno
- Disponer la zona de eco
- Desarrollar el control del usuario
- Desarrollar los menús para las diferentes interacciones que se quisieran aplicar a los objetos de la zona

Para el entorno utilice un prefab de Google VR, y delimité la zona de eco, donde inserte un icosaedro que emite la canción libre de derechos *“If i had a chicken”*. Esto permitió testear el eco de la zona de una manera más sencilla.

Ahora tocaba dedicar un tiempo al usuario, por lo que empecé trabajando con la retícula que sería la forma de interaccionar con el entorno, aunque se presentó un problema a la hora de implementar una barra de carga para que el usuario supiera que se estaba interactuando con algo. Este problema se expone en el apartado (7.2).

Ahora tocaba desarrollar un método para poder mover al usuario por la escena, por lo que dispuse un control para hacerlo por mando. Esta implementación acabó siendo desechada ya que requería un periférico extra, dando lugar al control implementado de mantener pulsado para avanzar en la dirección hacia la que mire el usuario.

Lo siguiente fue implementar un método para que al interactuar con el icosaedro, éste se teletransporte a otro lugar dentro de los límites de las escena.

La última implementación en esta etapa fue un menú para poder avanzar al bosque o cerrar la aplicación, menú que también se utilizará en el bosque más adelante para volver a esta habitación o cerrar la aplicación.

La primera escena fue paradójicamente la última con la que empecé, aunque el diseño era mucho más sencillo. Me planteé el hecho de que en esta aplicación el gran protagonista es el sonido, por lo que una vez implemente un espacio sencillo compuesto por un plano que hace de suelo y muros invisibles que limitan la posibilidad de moverse, cree un cubo que llama al usuario. Fue entonces cuando me di cuenta de que sería más interesante si el cubo fuera invisible y solo apareciera cuando por fin sea encontrado y se interactúe con él, indicándose mediante otro audio que te manda a otra escena, en este caso la habitación del eco.

Lo siguiente fue optimizar el bosque, ya que la carga poligonal era excesiva y el móvil no era capaz de mantenerla abierta. Con este fin, dediqué tiempo a reducir y modificar los shaders, así como también dedicar tiempo a implementar “Oclusion Culling”. De esta forma, el bosque pasó de ser un posible descarte a ser probablemente la zona más interesante de la aplicación.

Con todo esto ya implementado, añadí al bosque unos muros invisibles para delimitar la zona de movimiento y se lo enseñé mi tutor para el TFG, con quién estuve hablando sobre las posibilidades, y me recomendó que intentara implementar algún objetivo en el bosque, así como alguna manera de interaccionar con el eco de la habitación.

Esto me llevó a implementar interacciones con solo algunos pájaros, así como un indicador encima del menú que nos presentase cuales efectivamente habíamos conseguido localizar.

En el caso del eco, simplemente implementé una serie de menús formados por los dropdown de Unity, de forma que pudiéramos cambiar el material que forma cada pared de la habitación, así como suelo y techo. Esta implementación también presentó un problema que detallo en el apartado (7.4).

4. Análisis

4.1. Introducción

4.2. Estado del arte

Aquí se van a analizar varios aspectos sobre el proyecto.

4.2.1. Sonido 8D

El audio 8D es la sensación de escuchar los sonidos a través de unos cascos en ángulos de trescientos sesenta grados, parecido a la realidad virtual.

Este término que ha resurgido gracias a las redes sociales, dista de ser nuevo, ya que antiguamente era conocido como ambisonic, binaural o simplemente sonido 3D.

Como estrategia de marketing, algunos músicos sugieren que la música danza alrededor de quien la escucha, provocando que muchos de los temas actuales, y no tan actuales, adquieran este sistema.

A pesar de los avances, la sensación real y envolvente está muy lejos de la realidad, ya que muchos elementos no se tiene en cuenta a la hora de determinar el posicionamiento del objeto. Por ejemplo, la transición entre delante y detrás del usuario pueden por momentos ser indistinguibles, ya que no se tienen de forma alguna en cuenta la orientación de las orejas, o el hecho de que si el usuario no se mueve, el cerebro humano no es capaz de distinguir entre una posición delante o detrás, asignando por defecto una posición delantera hasta que la fuente del sonido o el receptor se muevan, permitiéndose entonces que el cerebro si sitúe en el espacio virtual.

Ante este problema un desarrollador puede verse abrumado al principio, pero la solución a ambos problemas es más simple de lo que podría parecer, ya que en el caso de la orientación de las orejas es aplicar un coeficiente de división en los sonidos situados detrás del usuario, que se basa en el índice de absorción de sonido de la piel humana. En este caso se ha utilizado el de la goma para simular la piel. El segundo problema no tiene una solución software a simple vista, pero teniendo en cuenta que el usuario interacciona con el entorno moviendo la cabeza y avanzando por la escena de forma que cambia su posición y orientación con respecto al foco, este problema se maquilla por la propia inmersión de la aplicación.

4.2.2. Realidad Virtual

Cuando hablamos de realidad virtual, hablamos de un entorno de apariencia real generado mediante tecnología informática que da la sensación de inmersión al usuario.

Algunos centros educativos lo están usando para comprobar su viabilidad [18], sobre todo en casos de personas con dificultades en el aprendizaje, aunque presenta una serie de inconvenientes, como el coste o el espacio físico necesario para el usuario. A pesar de ello, se tienen grandes expectativas sobre su utilidad.

Es importante añadir que en los últimos años hemos sufrido un boom con la aparición de la VR en el mundo del videojuego de una forma serie, sin embargo el precio de los equipos para los usuarios hace que los desarrolladores no innoven en este campo, por lo que si no se vuelven más económicos, algunos ^{expertos} afirman que podría desaparecer.

Esta afirmación parece muy alarmista, ya que la VR no se utiliza solo en el ámbito del videojuego y se han hecho grandes avances para otros campos.

4.2.3. Gamificación

Se como el uso de diseños, elementos y características de juegos en contextos totalmente ajenos a estos. Socialmente hablando, especialmente a aquellos asiduos a las redes sociales, con los que comparten elementos como la lealtad del usuario, los logros o el reclutamiento a eventos con métodos atractivos y divertidos, dándoles una sensación de control y motivando su uso.

Respecto a la educación, muchos han adoptado este estilo de enseñanza aunque no todos están de acuerdo. Su uso sigue expandiéndose cada vez más.

En 2017, el Gobierno chino puso en marcha un enorme proyecto piloto en el ámbito social [5], basado en una especie de gamificación. Con ello, se cambió la magnitud y escala de lo que entendíamos por ‘gamificación’, de tal manera que conectó la actividad online de sus ciudadanos con su estrategia social a gran escala, incorporándose a un sistema de medición de conducta individual con ‘premios’ y ‘castigos’, a partir de un sistema de puntuación llamada originalmente ‘crédito social’ que en origen, tenía propósitos comerciales relacionados con recompensas o incentivos.

Esto no sería posible sin dos factores, la colaboración de grandes empresas chinas basadas en internet (como Alibaba, Baidu o Tencent) y las nuevas leyes de ciberseguridad chinas, que dan cobertura legal al acceso completo a casi todos los datos personales.

4.3. Tecnologías y herramientas

4.3.1. Lenguaje de programación C#

C sharp [17] es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET.

Al ser un lenguaje orientado a objetos, admite los conceptos de encapsulación, herencia y polimorfismo. Todas las variables y métodos, incluido el método Main, el punto de entrada de la aplicación, se encapsulan dentro de las definiciones de clase. Una clase puede heredar directamente de una clase primaria, pero puede implementar cualquier número de interfaces. Los métodos que invalidan los métodos virtuales en una clase primaria requieren la palabra clave override como una manera de evitar redefiniciones accidentales. En C#, un struct es como una clase sencilla; es un tipo asignado en la pila que puede implementar interfaces pero que no admite herencia.

Escoger este lenguaje facilita el trabajo con Unity, motivo por el cuál es el seleccionado.

4.3.2. GitKraken y GitHub

El uso de un repositorio y control de versiones es algo fundamental para llevar un control del trabajo en la aplicación, así como poder volver a versiones anteriores del desarrollo en caso de ser necesario.

GitKraken [15] es una herramienta que tiene absolutamente todas las funcionalidades que se pueden llegar a querer en una herramienta para control de versiones con un rendimiento que deja en muy mal lugar al resto de herramientas (como SourceTree o Tortoise Git).

Es la única herramienta que tiene versión de pago con su licencia PRO (49\$/año), sólo necesitarás esta licencia para un uso profesional o no comercial, pudiendo realizar tus proyectos personales con la versión gratuita sin ningún tipo de problemas, ya que las funcionalidades son las mismas.

En cualquier caso, con la licencia de estudiante gratuita de GitHub se puede tener acceso a todas las funcionalidades, tanto de GitHub como de GitKraken.

4.3.3. Unity

Ha sido la opción para el desarrollo de la app. Ya hablamos de él en el apartado *2.1.3. Comparativa de motores gráficos*.

4.3.4. Cardboard

Google Cardboard es una plataforma de realidad virtual (VR) desarrollada por Google sobre la base de cartón plegable, de allí su nombre, que funciona a partir de montar un teléfono móvil inteligente con Android o IOS [16].

Los lentes están para dar la sensación de profundidad. Los campos de visión para el ojo izquierdo y derecho están delimitados por una franja de cartón separatoria en el centro de las gafas. Los cristales crean un efecto lupa, así que es bastante importante que nuestro teléfono tenga una concentración más bien alta de píxeles por pulgada para usar Google Carboard.

Actualmente google lo vende como experiencias en VR a bajo costo.

5. Diseño

5.1. Visión global de la aplicación

La aplicación consta de tres escenas:

- Init: donde la aplicación empieza.
- EchoChamber: donde se testeó el efecto del eco.
- Forest: donde se prueba el efecto del 8D con varios focos de sonido.

Antes de empezar a hablar de cada uno de los elementos particulares de cada escena, es imperativo hablar de los elementos que son comunes en todas ellas. De este modo, en toda escena vamos a encontrar una serie de elementos que son imprescindibles:

- Foco de luz: configurado como luz direccional
- RealPlayer: contiene la cámara y la retícula y hace la función de representarnos en el mundo virtual
- GvrEditorEmulator: encargado de simular con el ratón el movimiento en modo VR
- GvrEventSystem: que se encarga de activar el input de colisión entre el rayo de detección que se emite desde la cámara por la retícula y los objetos con los que podemos interaccionar
- SceneManager: se encarga de gestionar los eventos de salir de la aplicación y cambiar de escena item GvrControllerMain: se utiliza para determinar controles más allá del giro de cámara y el click, pero al final este elemento no se llegó a utilizar

Ahora se pasa a hablar de las particularidades de cada escena.

5.2. Escenas

5.2.1. Init

Esta escena pretende poner en situación al usuario, indicando con un texto sobre una pared invisible como moverse y que es lo que tiene que hacer en ella.

El objetivo será tan simple como encontrar el foco de sonido que lo está llamando. La voz para el foco ha sido aportada por Juan Hernández García, persona que menciono en los agradecimientos.

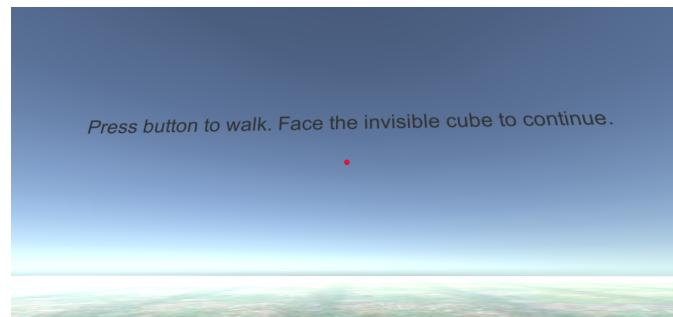


Figura 5.1: Instrucciones de uso

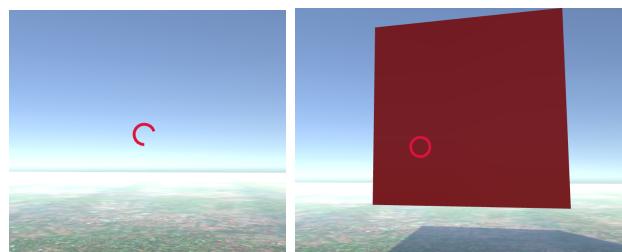


Figura 5.2: Cubo

5.2.1.1. Componentes de Init

Dicho todo esto, toca hablar de cada uno de los elementos no comunes de la escena. De este modo, quedan por comentar los siguientes elementos:

- Cube: sujeto que se vuelve visible al encontrar su posición siguiendo el sonido de su llamada
- Plane: suelo de la escena
- Canvas: contiene un texto en que van unas pequeñas instrucciones
- Walls: Se forma con cuatro prismas invisibles que actuarán como muros invisibles para delimitar la zona de movimiento

5.2.2. EchoChamber

Esta habitación tiene como objetivo mostrar el efecto del eco ante un foco de sonido, en este caso será un icosaedro que toca la tonadilla libre de derechos *If i had a chicken*, típica canción de taberna del oeste. Este icosaedro se tele transportará a otro lugar de la habitación cuando interactuemos con él.

Además, se encontrarán dos menús, unos con dos botones y otro con varios desplegables.

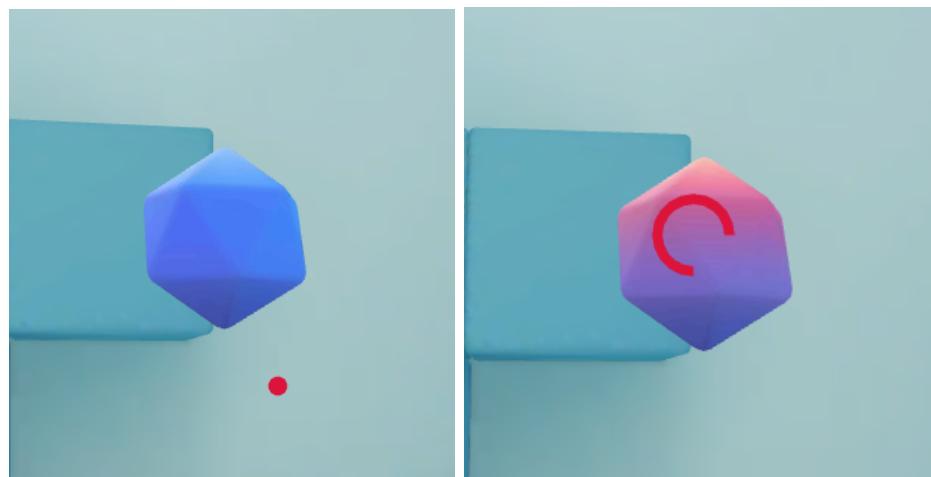


Figura 5.3: Icosaedro sonoro

Mediante el menú de dos botones se podrá salir de la aplicación o avanzar a la última escena. Además este menú contará con un texto que indique la utilidad de esta escena.

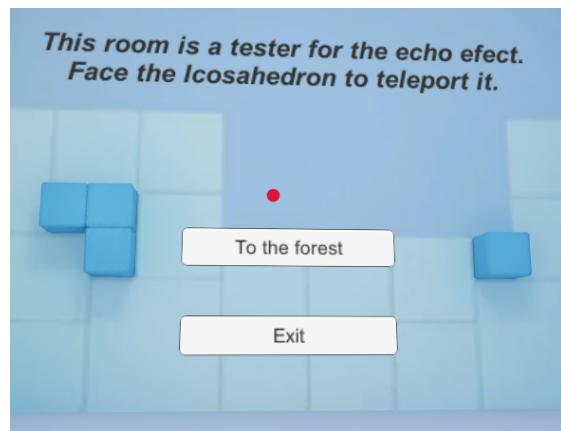


Figura 5.4: Menú e instrucciones de la habitación

El menú de desplegables se utiliza para cambiar el tipo de material del que se componen las superficies de la habitación.

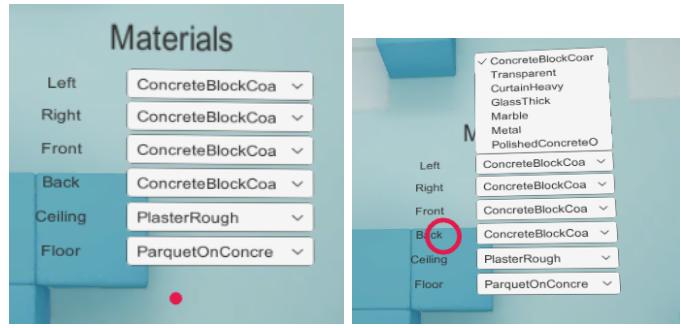


Figura 5.5: Menú para cambio de materiales

5.2.2.1. Componentes de EchoChamber

Los elementos no comunes de la escena EchoChamber determinarán el comportamiento de la misma:

- ResonanceAudioRoom: delimita la zona cúbica donde el eco puede trabajar. Si se sale de esta delimitación el eco desaparece
- CubeRoom: en esta escena se tiene una escena cúbica que delimita el espacio de interacción
- Menu: compuestos por dos botones (uno para avanzar de escena y otro para salir de la aplicación) y un texto que explica cómo funciona la escena.
- Icosahedron: es el punto de sonido de la escena, y al interaccionar con él se teletransportará a un lugar aleatorio de la escena para poder observar cómo esto afecta al eco de la habitación
- Canvas: este canvas tiene dentro una serie de dropdown que interaccionan directamente con ResonanceAudioRoom para modificar los diferentes materiales que componen sus caras.

En particular ResonanceAudioRoom, Icosahedron y Canvasson especialmente importantes, pues son los que permiten trabajar con el eco en esta habitación, pero no adelantemos acontecimientos.

5.2.3. Forest

Esta escena pretende representar un pequeño bosque lleno de pájaros que van cantando, volando por las cercanías.

Dispone de un menú para volver a la escena EchoChamber o salir de la aplicación.



Figura 5.6: Menú en el bosque

En la parte superior del menú se encuentra un banner con los cinco pájaros dentro de la selección que se introduzcan en la aplicación. Cuando se visualice al tipo de pájaro objetivo, se marcará en ese banner.



Figura 5.7: Pajaros en el bosque

Debido a la carga que tendrá esta escena, es importante cambiar shaders y aplicar técnicas de optimización para evitar que la tasa de frames caiga demasiado.

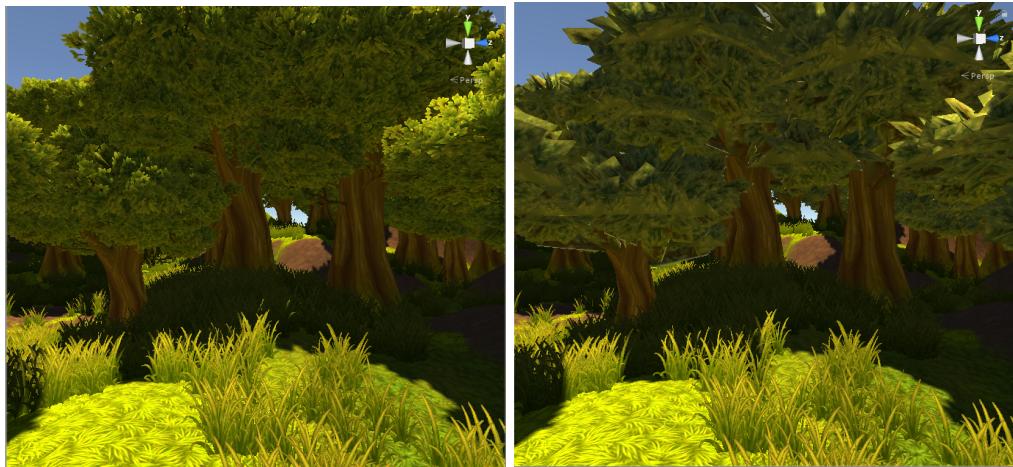


Figura 5.8: Comparativa entre shader de alta calidad y baja calidad

5.2.3.1. Componentes de Forest

Forest es con diferencia la escena con más carga de elementos de todo el proyecto, por lo que voy a diferenciar entre tipos de elementos.

Primero debemos hablar de los elementos que tiene que ver con la topografía de las escena:

- Terrain: terreno irregular que crearemos utilizando las herramientas de Unity. Aprovechando el asset Fantasy Forest, le agregaremos proceduralmente hierba y árboles
- Rock: Utilizaremos los dos tipos de rocas del asset Hand Painted Forest para dar algo de personalidad a nuestro bosque
- Statue: utilizaremos los modelados de estatuas del asset anteriormente nombrado para darle más personalidad al bosque

Ahora pasamos a ver los elementos asociados al comportamiento de los pájaros:

- `_livingBirdsController`: se encarga de lanzar los distintos tipos de pájaros para que interactúen por el bosque
- `lc_perchTarget`: hace de objetivo donde los pájaros aterrizan, siempre que este no esté en el suelo
- `lb_GroundTarget`: determina un lugar de aterrizaje para los pájaros en el suelo

Una aclaración importante es que `lc_perchTarget` y `lb_GroundTarget` afectan de manera distinta al árbol de animaciones de los pájaros.

Forest posee, al igual que Init, un conjunto de muros invisibles llamado Wall que determina la zona por la que el jugador puede moverse.

El último tipo de elemento que queda por remarcar es un menú con las mismas interacciones que tenía el de EchoChamber, pero con una particularidad, posee un banner encima suyo que determina si hemos visto o no los pájaros en el postrado.

5.3. Modificaciones del algoritmo 8D

El algoritmo 8D clásico, a día del inicio de la parte práctica de este proyecto, presenta un problema fundamental a la hora de trabajar con un sonido al que se puede determinar la posición de su foco, y es que si el foco se mueve, la transición entre estar delante o detrás del usuario no se realiza de una forma correcta, dando la impresión de que el foco solo se está moviendo delante del usuario.

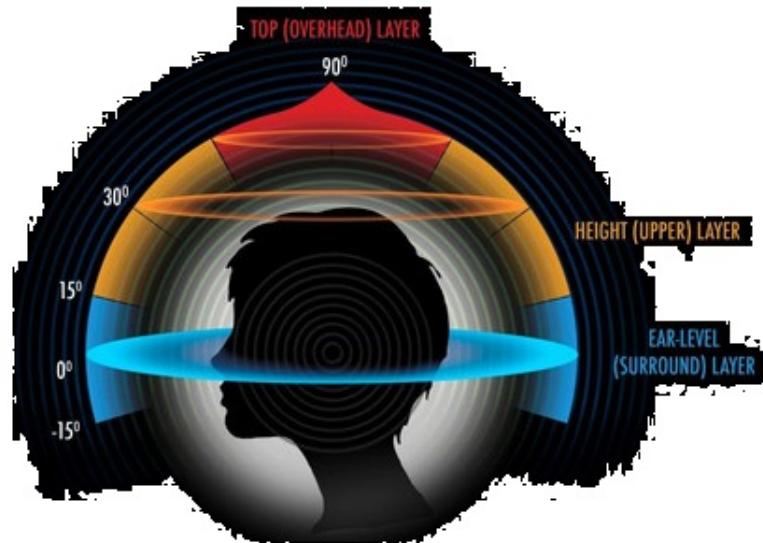


Figura 5.9: Percepción del algoritmo

Este problema se presenta, pues no se aplica un factor de reducción del sonido que tenga en cuenta dos factores determinantes a la hora del cálculo, que el ser humano tiene orejas que están orientadas, y que además estas poseen un factor de absorción del sonido, de forma que no deben bloquearlo totalmente, si no reducirlo en función de este factor.

Para determinar esto y aplicarlo al algoritmo con el que nos encontramos, solo debemos tener en cuenta que ahora no tenemos un solo vector de orientación para la cámara a la hora determinar la posición de los sonidos en la parte trasera de la cabeza del usuario,

si no que debemos hacer una discriminación los cálculos, sabiendo que la orientación de la oreja afecta a la percepción del sonido, de forma que el cálculos de los vectores que determinan la llegada debe variar aplicando así una reducción extra en la intensidad con la que se percibe.

Ahora toca hablar sobre la reducción de sonido que plantea la oreja no por su orientación, si no por su materia.

Los cambios efectuados por esto se aplican los últimos, ya que solo afecta a la zona trasera de la cabeza. Teniendo en cuenta el factor de absorción de sonido de la piel humana, que en este caso se ha equiparado al de la goma, se divide el resultado si se encuentra el foco en la zona designada.

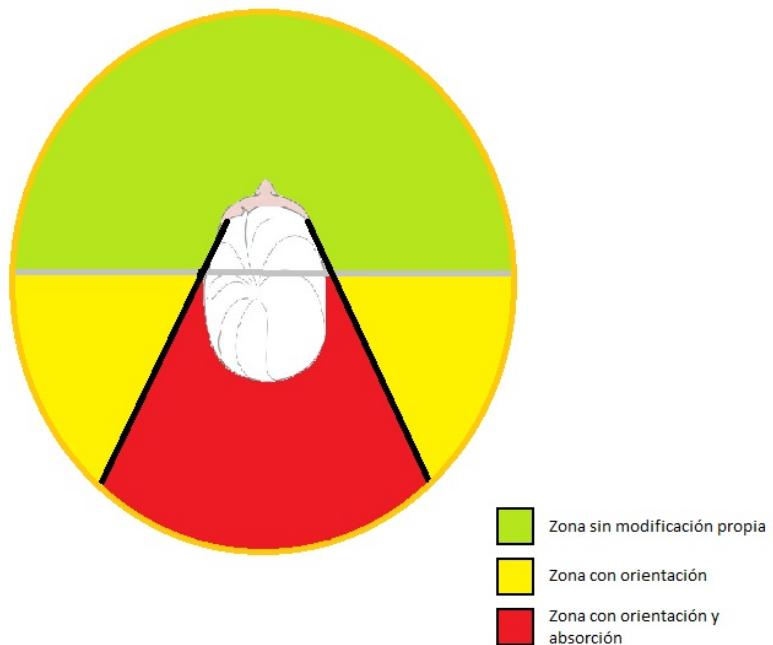


Figura 5.10: Zonas del algoritmo

6. Implementación y control de versiones

6.1. Introducción

En esta parte se presentan las diferentes implementaciones necesarias, así como un control de versiones de la aplicación en la que se desmenuzan las diferentes versiones.

Llevar un control de versiones es esencial como desarrollador, por lo que para este fin se utiliza la página *github*, y como herramienta secundaria para la gestión del repositorio, la herramienta *GitKraken*. Para el desarrollo de esta aplicación se ha seguido el paradigma de *GitFlow* a la hora del control de versiones con el repositorio.

6.2. Control de versiones

A continuación, se presenta el árbol de desarrollo de la aplicación en las que se ve un resumen de las modificaciones:

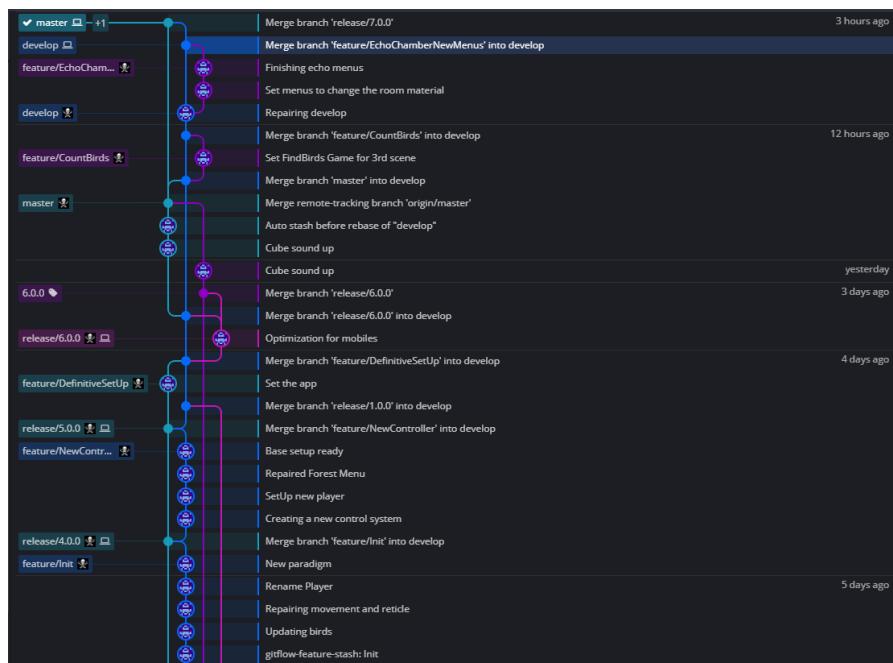


Figura 6.1: Parte superior del gitTree

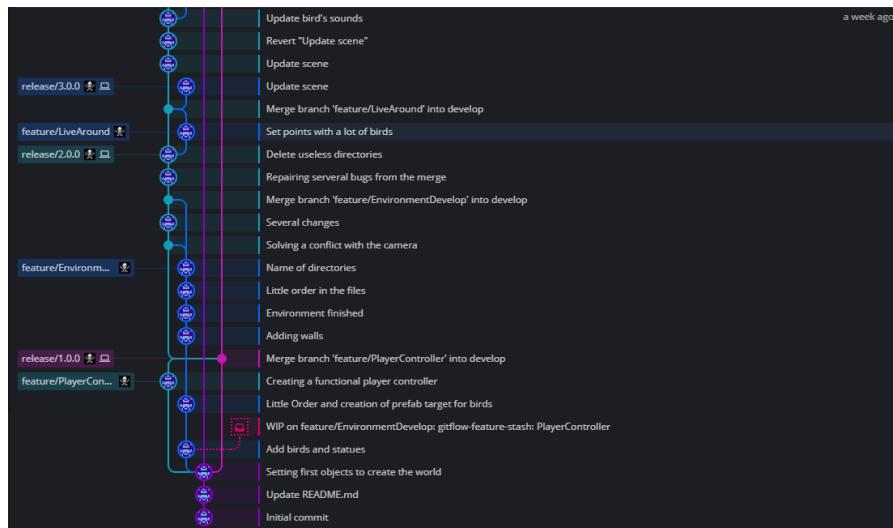


Figura 6.2: Parte inferior del gitTree

6.2.1. Cambios orgánicos en el planteamiento de la aplicación

El desarrollo de una aplicación no es ni mucho menos algo estático que sigue estrictamente los pasos definidos al inicio de éste durante la planificación. Un buen desarrollo debe saber adaptarse a requisitos que pudieron no tenerse en un principio.

Este es el caso de la aplicación aquí presentada, de forma que me dispongo a enumerar unos cuantos cambios que surgieron a lo largo del desarrollo y que merecen ser resaltados por encima de los demás:

- En la versión 5.0.0, desecho un control del movimiento del personaje basado en mando por un control basado únicamente en los que el cardboard nos presenta, pasando el botón de pantalla a haber que el personaje se mueva hacia adelante, y las interacciones con elementos de las escenas presentadas con un cargador de tiempo. Este cambio surge ante el planteamiento de facilitar el uso para el usuario, al no depender de otro dispositivo externo, en este caso un mando.
- En la versión 6.0.0 la retícula de carga pasa a ser la propia retícula puntero que utiliza. Esto se realiza con la idea de evitar que la superposición de ambas retículas pueda marear al usaron cuando la de carga entra en escena.
- En la versión 6.0.0 se aplica *Occlusion Culling* en la escena del bosque. Esto se hace para reducir su carga y así maximizar los frames de la escena.

6.3. Implementación general: `setUp` de la aplicación

El trabajo de esta aplicación se ha desarrollado con Unity 2018.4.6f1. Esto se ha hecho así debido a que el hub de Unity determinaba que esta era la última versión estable de

la aplicación en el momento del inicio de la programación de este proyecto.

Deberemos configurar también la sdk para que Unity pueda trabajar. Normalmente si ya se tiene la sdk de android instalada, Unity la reconocerá sin necesidad de añadirla manualmente.

Debemos tener en cuenta que, en ".Edit>Project Settings", debemos activar la casilla *Virtual Reality Supported* de XR Settings para android en el "Player", y fijar en la pestaña *Other Settings* el *Minimum API Level* a nivel 19.

6.3.1. Paquetes a descargar y assets gratuitos

Los paquetes que necesitaremos para poder utilizar la tecnología necesaria en Unity son:

- ResonanceAudioForUnity_<versión_deseada>.unitypackage [13]
- GoogleVRForUnity_<versión_deseada>.unitypackage [11]

El paquete GoogleVR es un set de utilidades desarrolladas por google para facilitar el desarrollo de una aplicación con realidad virtual que además tiene versión para varios motores.

El paquete Resonance es un set de herramientas con los algoritmos pertinentes para poder simular los distintos aspectos de un sonido envolvente, con distintos efecto, percepción de profundidad (conocido como sonido 3D) y sonido 8D.

En la documentación hay un enlace para poder acceder a todas las características aplicadas a Unity para ambos paquetes [10] [12].

Para instalar un paquete descargado, solo habrá que introducir el paquete manualmente desde la pestaña *Assets>Import Package>Custom Package*.

Por otra parte, los assets gratuitos sacados de la store de Unity son los siguientes:

- Fantasy Forest
- Hand Painted Forest Lite
- Living Birds

Fantasy Forest ha sido utilizado para extraer el modelaje de los árboles y la textura 2D para la hierba. La aplicación de los árboles en un terreno, así como de la hierba se hace siguiendo el procedimiento standard de Unity.

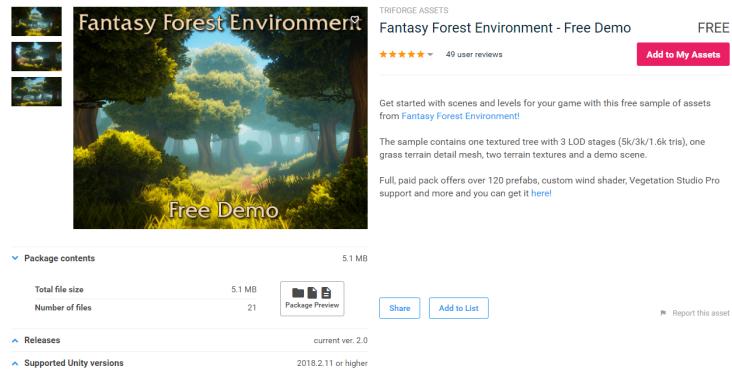


Figura 6.3: Fantasy Forest

Hand Painted Forest Lite ha sido descargado exclusivamente por el modelado de las estatuas, las cuáles han sido integradas al terreno en el que el jugador se sitúa.

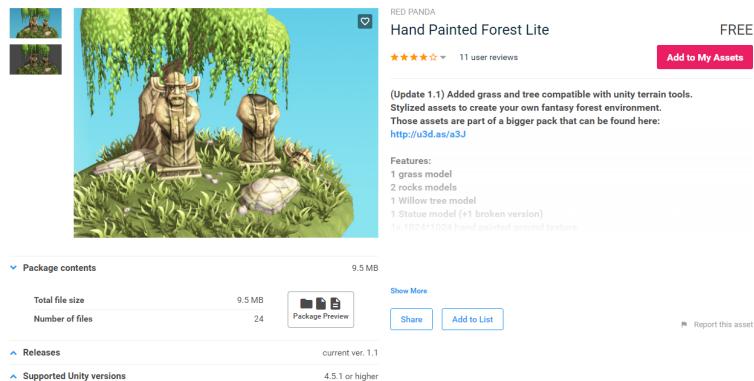


Figura 6.4: Hand Painted Forest Lite

Living Birds posee los modelados y animaciones de los pájaros, además de un sistema para hacer que los pájaros se generarán automáticamente para simular un entorno orgánico. En el momento en el que se utilizó este asset, lo único implementado fueron los modelajes con sus animaciones, ya que la mayoría de script asociados a la generación automática estaban vacíos, o directamente no existían, por lo que parte del desarrollo se dedicó a rehacer gran parte de este asset.



Figura 6.5: Living Birds

6.4. Jugador

El jugador estará formado por un GameObject llamado *RealPlayer* que contendrá la "MainCamera". Se introducirá como hijo de ésta el prefab *GvrReticlePointer* que pondrá en el jugador la retícula que utilizaremos para las interacciones. Hay una serie de cambios en la retícula que se explican en el siguiente subapartado.



Figura 6.6: Aspecto del Jugador en el árbol de escena

6.4.1. Controles de movimiento

Se ha implementado para el movimiento que cuando se hace click en la pantalla, el jugador se mueve. Esto se consigue añadiendo un script de C# a *RealPlayer* que contendrá el siguiente código:

Código 1: PlayerWalk.cs

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

public class PlayerWalk : MonoBehaviour
{
    public int playerSpeed;
```

```

// Update is called once per frame
void Update()
{
    if (Input.GetButton("Fire1"))
    {
        transform.position = transform.position + Camera.main.transform
            .forward * playerSpeed * Time.deltaTime;
    }
}

```

6.4.2. Control de cámara en VR

Simplemente se añadirá al árbol de escena el prefab de GoogleVR *GvrEditorEmulator*, que en ordenador proporciona una vista normal mientras que al instalar la apk en un dispositivo, este mostrará la pantalla dividida y adaptada para ser utilizada en un dispositivo de visionado como las *cardboard*.

El control simulado en el PC se utiliza de la siguiente manera:

- *Alt + move* mouse para rotar sobre uno mismo y cambiar la dirección a la que se orienta el jugador
- *Ctrl + move* mouse para rotar sobre el eje que sale desde el jugador hacia adelante

No voy a entrar en detalles con el código del objeto en cuestión ya que no es código propio.

6.4.3. Retícula

La retícula que viene por defecto en unity no será necesaria en el caso presentado, pues es mejor y más eficiente utilizar el script *GvrPointerPhysicsRaycaster.cs*, adjunto con el paquete de GoogleVR.

La retícula es el medio para interactuar con el medio, pero no interesa que se active automáticamente, si no que será preciso que haya un tiempo de espera para que el usuario decida si se va a llevar a cabo esta interacción o si por el contrario desea cancelarla.

Con motivo de ello, se aprovechará la capacidad de la retícula de google de ampliarse en una interacción (efecto producido con el prefab *GvrEventSystem*.) y se modificará el ángulo mostrado en pantalla de ésta el cual se irá modificando, convirtiéndola así en una barra de carga circular. Para esto se requieren las siguientes modificaciones en el shader [19] de Google:

- Añadir la propiedad Angle
- Añadir el float que la representará

- Sustituir la función *vert* para para que tenga en cuenta esta nueva propiedad en el shader

A continuación se adjunta el resultado final para ver como debe quedar el código:

Código 2: GvrReticleShader.shader

```
// Copyright 2015 Google Inc. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
// http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

// Trololo

Shader "GoogleVR/ Reticle" {
Properties {
    _Color ("Color", Color) = ( 1, 1, 1, 1 )
    _InnerDiameter ("InnerDiameter", Range(0, 10.0)) = 1.5
    _Angle("Angle", Range(0, 360)) = 180 // Propiedad con la que trabajaremos
        en el shader
    _OuterDiameter ("OuterDiameter", Range(0.00872665, 10.0)) = 2.0
    _DistanceInMeters ("DistanceInMeters", Range(0.0, 100.0)) = 2.0
}

SubShader {
    Tags { "Queue"="Overlay" "IgnoreProjector"="True" "RenderType"="Transparent"
        "}
    Pass {
        Blend SrcAlpha OneMinusSrcAlpha, OneMinusDstAlpha One
        AlphaTest Off
        Cull Back
        Lighting Off
        ZWrite Off
        ZTest Always

        Fog { Mode Off }
        CGPROGRAM
}

#pragma vertex vert
#pragma fragment frag

#include "UnityCG.cginc"

uniform float4 _Color;
```

```

uniform float _InnerDiameter;
uniform float _OuterDiameter;
uniform float _DistanceInMeters;
uniform float _Angle; // Variable con la que trabajar

struct vertexInput {
    float4 vertex : POSITION;
};

struct fragmentInput{
    float4 position : SV_POSITION;
};

/* fragmentInput vert(vertexInput i) {
float scale = lerp(_OuterDiameter, _InnerDiameter, i.vertex.z);

float3 vert_out = float3(i.vertex.x * scale, i.vertex.y * scale,
    _DistanceInMeters);

fragmentInput o;
o.position = UnityObjectToClipPos (vert_out);
return o;
}
*/
}

// Nueva version del metodo vert
// que tiene en cuenta el angulo a la hora del dibujado
fragmentInput vert(vertexInput i) {
    fragmentInput o;
    if (_DistanceInMeters < 20) {
        // 180* = 2.9
        // limit = - (a - 180) * 2.9/180
        float limit = -(_Angle - 180) * 0.0161111111;
        float3 vert_out = float3(0, 0, _DistanceInMeters);
        float a = -atan2(i.vertex.x, i.vertex.y);
        if (a >= limit) {
            float scale = lerp(_OuterDiameter, _InnerDiameter,
                i.vertex.z);
            vert_out = float3(i.vertex.x * scale, i.vertex.y *
                scale, _DistanceInMeters);
        }
        o.position = UnityObjectToClipPos(vert_out);
    }
    return o;
}

fixed4 frag(fragmentInput i) : SV_Target {
fixed4 ret = _Color;
return ret;
}

ENDCG
}

```

```
}
```

6.4.3.1. Interacción de la retícula

Ahora entra en juego como conseguir la interacción con el objeto. Para ello se requieren dos pasos muy importantes:

- Añadir al objeto un *EventTrigger* que trabajará cuando el puntero entre dentro del área que ocupa el objeto y cuando salga.
- Añadir un script al objeto que en este caso se ha llamado *GVRButton.cs*.

Lo primero debe hacerse para determinar las funciones GvrOn y GvrOff que determinarán el comportamiento ante la entrada y la salida del objeto. El segundo determina la acción que se llevará a cabo a partir de que termine el evento de entrada, que en este caso que la barra de carga termine de llenarse.

Se adjuntan seguidamente es aspecto de un objeto que posee estas cualidades y el código asociado a *GVRButton.cs*.

Código 3: GVRButton.cs

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

public class GVRButton : MonoBehaviour
{
    GameObject pointer;
    public UnityEvent GVRClick;
    public float totalTime = 2;
    bool gvrStatus;
    public float gvrTimer;
    Renderer rend;
    float angle;
    bool counted;

    private void Start()
    {
        pointer = GameObject.FindWithTag("gvrpointer"); // Busqueda
        // del puntero con el que se va a trabajar
        rend = pointer.GetComponent<Renderer>(); // Extraccion del renderer del puntero
        counted = false; // Filtro para el conteo de parajulos en el bosque
    }
}
```

```

}

// Update is called once per frame
void Update()
{
    if (gvrStatus)
    {
        gvrTimer += Time.deltaTime;           // Calculo de la
        fraccion de tiempo de la carga en la que se esta trabajando
        angle = gvrTimer / totalTime * 360;   // Calculo de la
        fraccion de arco a dibujar
        rend.material.SetFloat("_Angle", angle); // Cambiar la
        variable angulo de la reticula
    }

    if (gvrTimer > totalTime)
    {
        GVRClick.Invoke();                // Efectuar
        un click
        gvrStatus = false;             //
        Determinar que se ha terminado la ejecucion
        GvrOff();
    }
}

// Metodo de inicio de interaccion de la reticula
public void GvrOn()
{
    gvrStatus = true;
}

// Metodo de finalizacion de interaccion de la reticula
public void GvrOff()
{
    gvrStatus = false;
    gvrTimer = 0;
    rend.material.SetFloat("_Angle", 360f);
}

// Metodo para marcar los pajaros vistos en el cartel del bosque
public void checkBird(string bird)
{
    if (!counted)
    {
        GameObject.FindWithTag(bird).GetComponent<Text>().text = "OK";
        counted = true;
    }
}
}

```

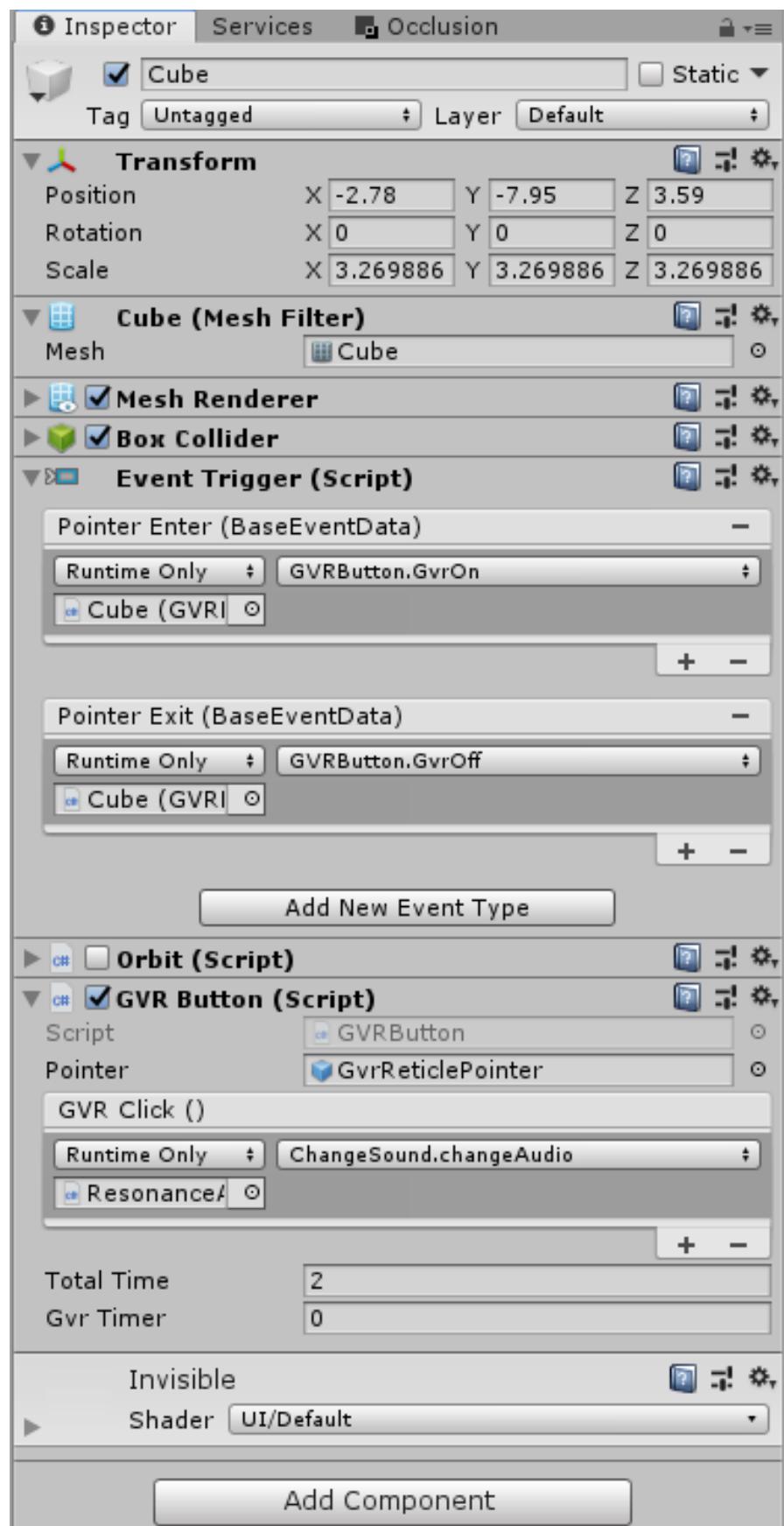


Figura 6.7: Cubo con interacciones

En el código se muestra como acceder al shader y modificar el ángulo que creamos en el apartado anterior dentro de la función update.

6.4.4. Aplicar sonido 8D a un objeto

Esta parte se soluciona rápidamente añadiendo los dos siguientes elementos al objeto emisor de sonido y a la mainCamera:

- El script *ResonanceAudioListener.cs* a la main camera de la escena.
- El prefab *Resonance AudioSource* al objeto que hará de emisor.

Es importante añadir el audio que queremos que se reproduzca en la propiedad *AudioSource* para que los scripts puedan trabajar.

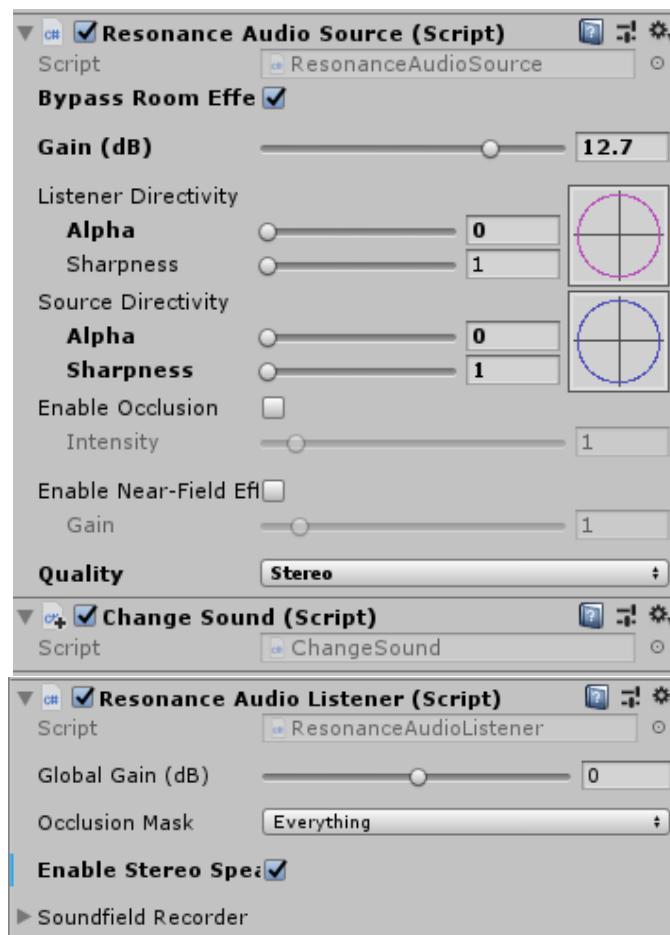


Figura 6.8: Script Resonace AudioSource y Resonance Audio Listener en el Inspector

6.5. Implementación por escenas

Ahora analizaremos los detalles más importantes implementados por escena.

6.5.1. Escena Init

6.5.1.1. Cubo para interactuar

Ya se ha hablado de el cubo de la primera escena que interactúa con el usuario cuando lo mira, pero tiene varias acciones que realizar en el momento de la interacción:

- Cambiar el audio que se está reproduciendo durante la ejecución.
- Cambiar la escena en la que se encuentra el jugador.
- Cambiar el material del cubo para que se haga visible.

La primera acción requiere que por nuestra parte creemos un script que active la subrutina que cambie en ejecución el audio reproducido. La segunda acción requiere que cuando se termine de reproducir el nuevo audio se cargue la siguiente escena. El siguiente código muestra cómo hacer ambas cosas desde la subrutina WaitFinish:

Código 4: ChangeSound.cs

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

public class ChangeSound : MonoBehaviour
{
    AudioSource myaudio;
    Material mymaterial;
    Renderer rend;

    public void changeAudio()
    {
        mymaterial = Resources.Load<Material>("RedMat");           // Busca el
        // material por el que se va a cambiar
        rend = GetComponentInParent<Renderer>();                  // Se
        // obtiene el renderer del parent
        rend.enabled = true;                                         // Nos aseguramos de que este activo
        // rend.sharedMaterial = mymaterial;                          // Se
        // Se cambia el material
        StartCoroutine(WaitFinish());                                // Se llama
        // a la subrutina que cambia el audio durante la ejecucion
    }
}
```

```

// Subrutina que espera a que un audio termine de reproducirse
// completamente para avanzar a la siguiente escena
IEnumerator WaitFinish()
{
    myaudio = GetComponent< AudioSource >(); // Localiza
    // el componente AudioSource de la fuente de sonido
    myaudio.clip = Resources.Load< AudioClip >("porfinteevo3"); // Cambia
    // el audio reproducido por el foco de sonido
    myaudio.Play(); // Reproduce el nuevo audio
    yield return new WaitForSeconds(myaudio.clip.length); // Espera
    // la longitud del audio reproducido
    SceneManager.LoadScene("EcoChamber"); // Carga de
    // la nueva escena
}

```

En el mismo código se incluye dentro de la función changeAudio el método para poder cambiar el material de un GameObject.

6.5.2. Escena Echo Chamber

6.5.2.1. Aplicar eco

Dentro del GameObject que define la habitación se debe añadir el prefab ResonanceAudioRoom, modificando el tamaño de este para que se ajuste al tamaño de la habitación.

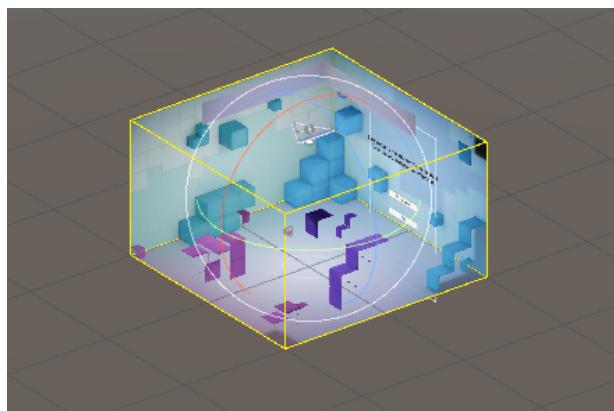


Figura 6.9: ResonanceAudioRoom ajustado a la habitación de la escena

Desde el script asociado al prefab, se pueden modificar los materiales que componen las diferentes caras del cubo que delimita el espacio que dispondrá de eco, así como las características del eco en cuestión (reflectividad, tiempo, ganancia en DB, brillo).

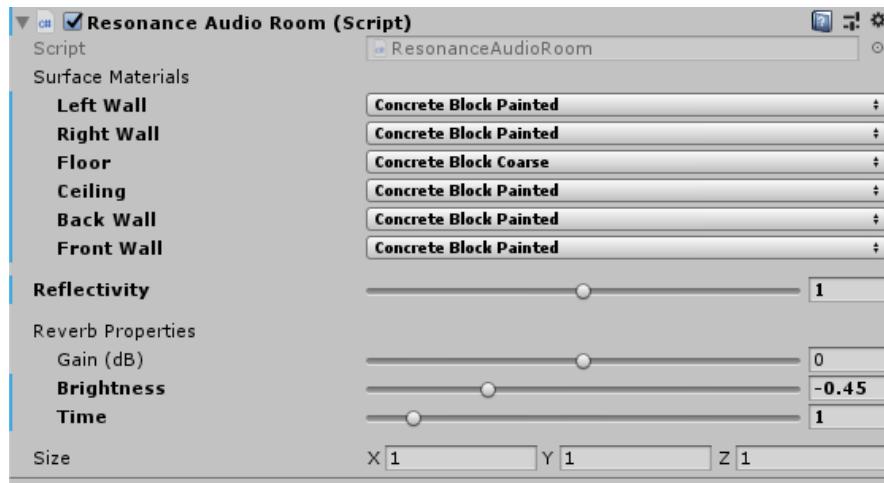


Figura 6.10: Script ResonaceAudioRoom

6.5.2.2. Cambiar en escena los materiales de la habitación

En la aplicación se encuentran varios materiales implementados para los diferentes lados del cubo que delimita la zona. Para simplificarlo, en esta aplicación solo se han implementado una pequeña porción para demostrar un ejemplo de cambio de material dinámico de estos.

La implementación de *dropdown* ha sido llevada a cabo mediante el objeto suministrado por Unity. Debido a que no se dispone de una versión de pago para el desarrollo de esta aplicación, no se pueden tocar internamente el script que lo pone en funcionamiento, pero hay total libertad a la hora de añadir los elementos que se requieran en la implementación.

Siguiendo ese razonamiento, durante este desarrollo se han añadido un *eventTrigger* y el script anteriormente mencionado *GVRButton*, haciendo la configuración requerida para poder interactuar con los elementos internos del dropdown.

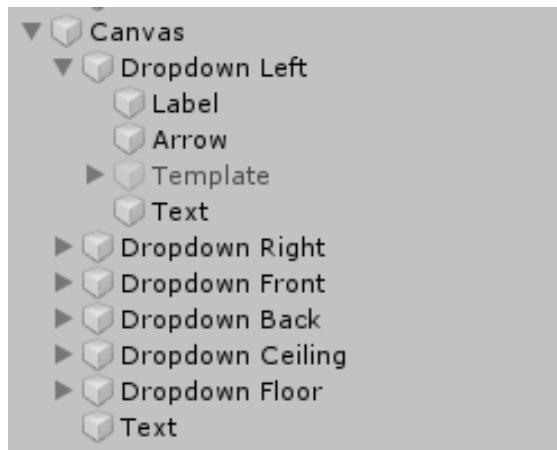


Figura 6.11: Dropdown en la jerarquía de escena

Como se puede apreciar, no hay problema a la hora de definir los elementos del desplegable, a los que se les asignará por defecto un valor entero en el orden en el que son añadidos.

Para poder acceder al valor asignado, en la zona de *OnValueChange* se ha añadido al elemento *ResonanceAudioRoom* como *GameObject*, y se llama a una de las funciones que se montaran dentro de su propio código llamadas *changeMaterialRight*, *changeMaterialLeft*, *changeMaterialFront*, *changeMaterialBack*, *changeMaterialCeiling* y *changeMaterialFloor*.

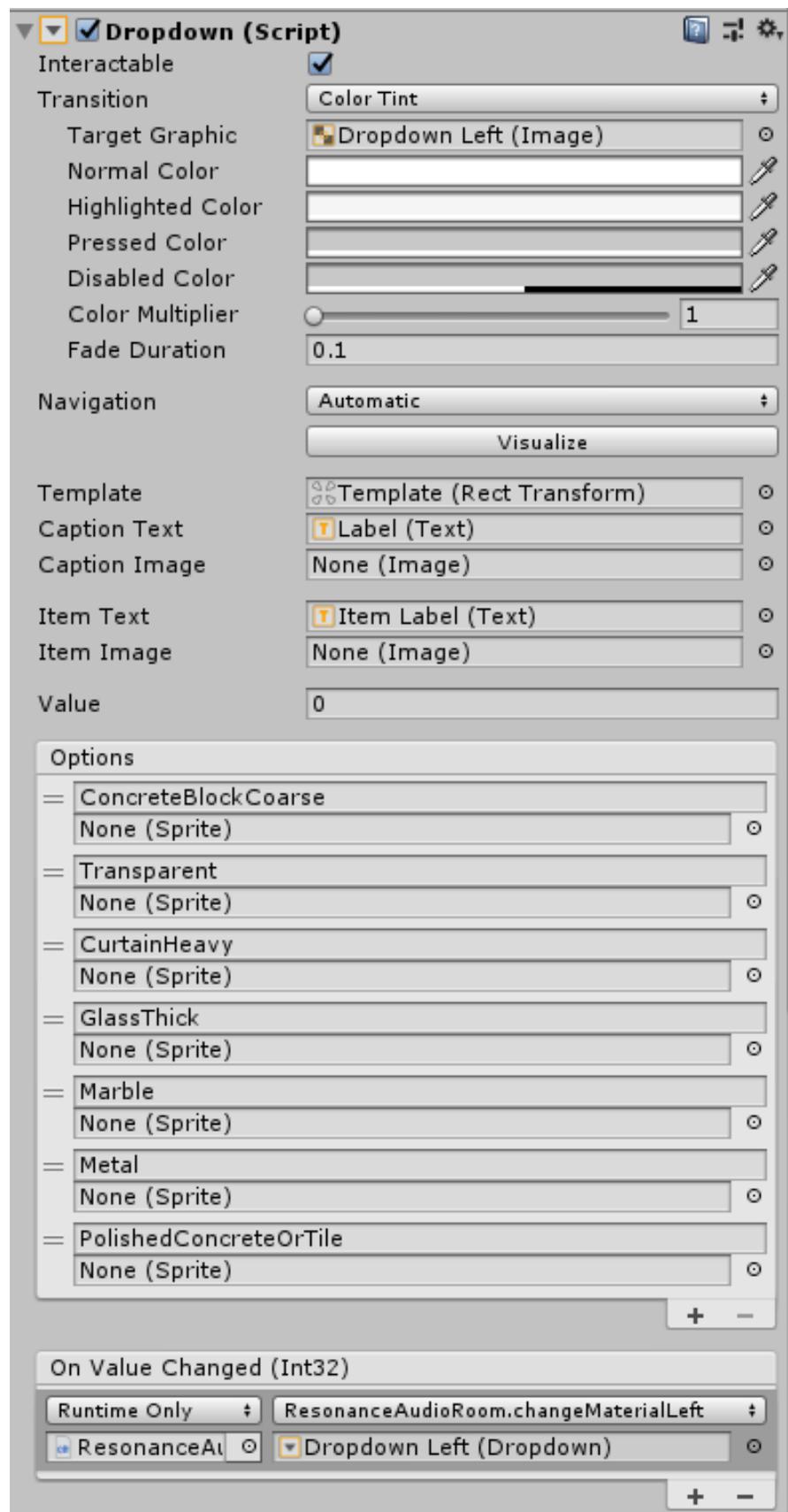


Figura 6.12: Script de un dropdown

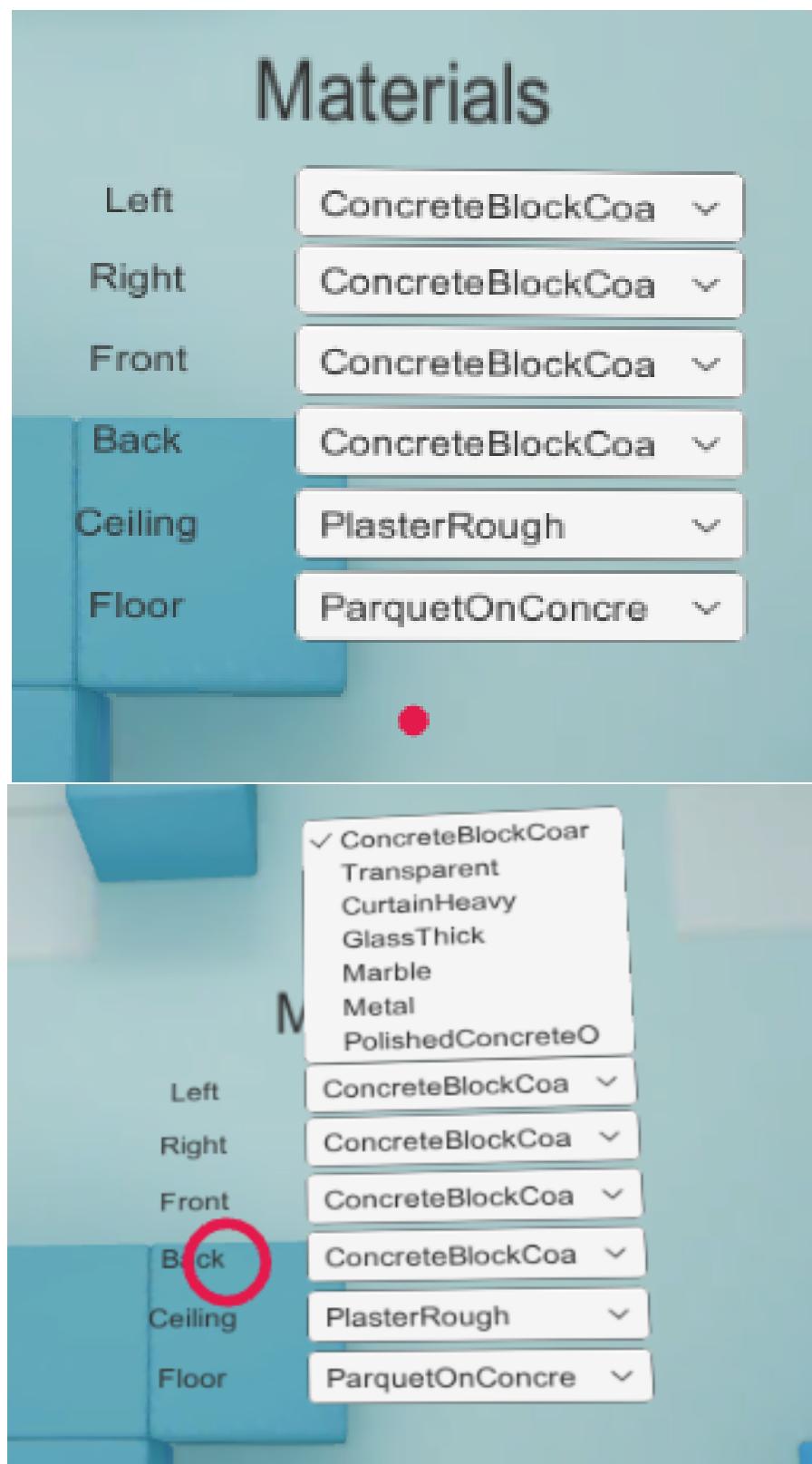


Figura 6.13: Menú de cambio de material

La implementación de las seis funciones mencionadas es prácticamente la misma, ya que solo se varía la referencia al lado sobre el que se van a hacer las modificaciones.

Código 5: Función changeMaterialLeft

```
...
public ResonanceAudioRoomManager.SurfaceMaterial rightWall =
    ResonanceAudioRoomManager.SurfaceMaterial.ConcreteBlockCoarse;
...
public void changeMaterialLeft(Dropdown mine) {
    int value = mine.value; // Se obtiene el valor que tiene la opcion
                            // seleccionada del dropdown

    switch (value) // Dependiendo del valor obtenido
        se el material con el que se cambiara
    {
        case 0:
            leftWall = ResonanceAudioRoomManager.SurfaceMaterial.
                ConcreteBlockCoarse;
            break;
        case 1:
            leftWall = ResonanceAudioRoomManager.SurfaceMaterial.
                Transparent;
            break;
        case 2:
            leftWall = ResonanceAudioRoomManager.SurfaceMaterial.
                CurtainHeavy;
            break;
        case 3:
            leftWall = ResonanceAudioRoomManager.SurfaceMaterial.
                GlassThick;
            break;
        case 4:
            leftWall = ResonanceAudioRoomManager.SurfaceMaterial.Marble
                ;
            break;
        case 5:
            leftWall = ResonanceAudioRoomManager.SurfaceMaterial.Metal;
            break;
        case 6:
            leftWall = ResonanceAudioRoomManager.SurfaceMaterial.
                PolishedConcreteOrTile;
            break;
        default:
            break;
    }
}
```

6.5.2.3. Icosaedro el sonido

Respecto al icosaedro solo queda añadir la función que lo hace teleportarse cuando el evento entra en acción. Esta función se recoge en el siguiente código:

Código 6: Teleporter.cs

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

public class Teleporter : MonoBehaviour
{
    Vector3 destination; // Almacenara la posicion random
    RectTransform rt;

    // Se cambia la posicion del icosaedro por una posicion random
    public void randomPlace()
    {
        destination = new Vector3(Random.Range(-8, 8), Random.Range(1, 8),
            Random.Range(-8, 8));
        rt = GetComponent<RectTransform>();
        rt.transform.localPosition = destination;
    }
}
```

6.5.3. Escena Forest

6.5.3.1. Occlusion Culling

Occlusion Culling [14] es una característica que desactiva el renderizado de objetos cuando actualmente no estén visibles por la cámara puesto que están oscurecidos (occluded) por otros objetos. Esto no sucede automáticamente en gráficas computacionales 3D ya que la mayoría de veces los objetos que están más lejos de la cámara son dibujados primero y los objetos más cercanos son dibujados encima de estos (esto se llama “overdraw”). El Occlusion Culling es diferente del Frustum Culling, ya que este solamente desactiva los renderers para objetos que están fuera del área visible de la cámara, pero no desactiva nada oculto de la vista por overdraw.

Para que el culling funcione, los objetos que se deseen ver afectados por él deben tener la casilla Static activada. De esta forma evitamos que objetos que se mueven se vean afectados por la desaparición en el dibujado.

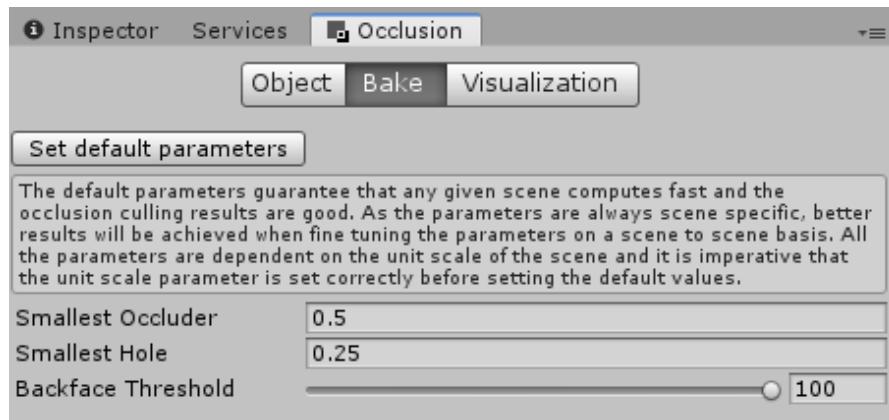


Figura 6.14: Parámetros aplicados en la aplicación para el culling

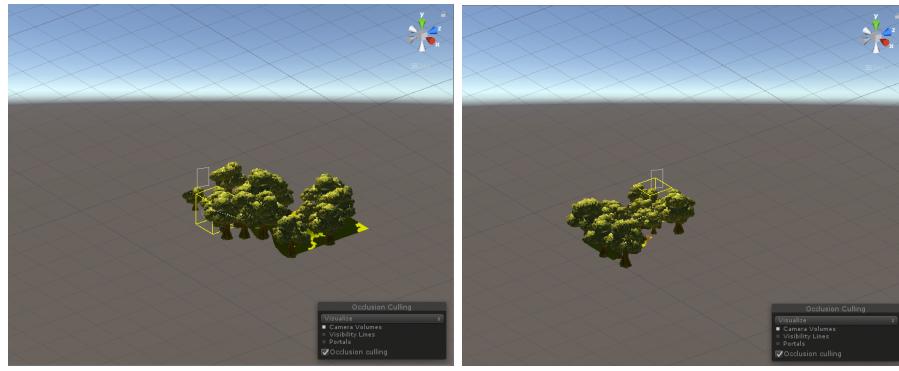


Figura 6.15: Script ResonaceAudioRoom

6.5.3.2. Pájaros y su búsqueda

El sonido en los pájaros se genera de la misma forma que en cualquier elemento con sonido (el cubo o el icosaedro), aunque presentan una pequeña diferencia. Donde encontramos dentro del script *lb_Bird.cs* una reproducción de uno de los cuatro audios que componen los sonidos del pájaro, se debe tener en cuenta que ahora el pájaro no contará con un AudioSource propio, si no que ese componente se encontrará dentro del objeto hijo Resonance AudioSource que se le añadirá.

Para poder acceder al componente de este objeto hijo, se deberá cambiar la linea que hace el play por lo siguiente:

Código 7: Ejemplo de cambio de audio para pájaro

```
this.transform.Find("Resonance AudioSource").gameObject.GetComponent<
```

Deben hacerse cuatro cambios en el script *lb_Bird.cs*.

Lo último a tener en cuenta en esta escena, es cambiar el shader de los materiales que componen la escena a *Movile/Diffuse*. De esta forma se garantiza que el shader está optimizado para trabajar en un móvil.

Ahora toca la implementación para poder interactuar con los distintos pájaros que aparecen en la escena, de forma que cuando interactuemos con dentro de la lista *Most Wanted!!* situada encima del menú en el bosque, ésta chequee a ese ave en particular y la marque con un *OK*.

Con ese fin se ha escrito dentro del script GVRButton la función *checkBird*, a la cual se le pasa un string, el cual será el tag del texto que deberemos cambiar de vacío a ".OK".

Código 8: función *checkBird*

```
// Metodo para marcar los pajaros vistos en el cartel del bosque
public void checkBird(string bird)
{
    if (!counted)
    {
        GameObject.FindGameObjectWithTag(bird).GetComponent<Text>().text = "OK";
        counted = true;
    }
}
```

Es importante recordar que hay que modificar el prefab de cada pájaro que se quiera convertir en interactivo mediante los pasos que ya se explicaron con anterioridad.

7. Pruebas

7.1. Introducción

Como es evidente, a lo largo de un desarrollo, por pequeño que sea, aparecen problemas que el desarrollador debe afrontar utilizando su conocimiento sobre la materia, sus dotes deductivas y de búsqueda. Planteó la situación de esta manera ya que la mayoría de los errores que se pueden encontrar trabajando con un motor de videojuegos como es Unity no vienen determinados por la compilación. Muchas veces se duplica un objeto o simplemente hay un bug asociado a una parte del motor que no responde correctamente con los paquetes que estemos utilizando para el proyecto.

Normalmente los problemas de incompatibilidad vienen dados por utilizar una versión del motor muy antigua, o utilizar un asset que no está completo, pero es importante informarse bien con qué se está trabajando y cuáles son los requisitos necesarios para su correcto funcionamiento,

Dicho esto, es tarea programador buscar una solución que arregle el problema encontrado, aunque durante el desarrollo de esta aplicación he llegado a la conclusión se que en varias ocasiones es más importante dejar atrás una idea que no avanza correctamente y replantear el problema.

A continuación se presentan un par de casos que aparecieron durante este desarrollo.

7.2. El caso de las retículas rebeldes

Siguiendo un tutorial de youtube [9], se intentó hacer un temporizador para la interacciones de la retícula con los objetos.

El resultado era satisfactorio dentro del entorno en el PC, pero los problemas comenzaron al pasar la aplicación al dispositivo móvil, ya que la retícula de carga no se centraba correctamente, provocando una sensación de mareo al no saber sobre que reticula centrarse.

La solución por la que se optó al descubrir que todo era producido por un bug interno producido por incompatibilidades entre la versión de Unity y la del paquete GoogleVR no fue igualar versiones, ya que hubiera implicado rehacer prácticamente todo el proyecto, si no que se optó por transformar la retícula base y convertirla en un temporizador cuando fuese necesario. Esto implicó, como se explica en el apartado 6.5.3. *Retícula*, realizar una serie de modificaciones en el shader de google para el material de la retícula.

7.3. La resina del bosque no me permite andar

La gran carga gráfica del bosque no solo lagueaba la aplicación, si no que además inutilizaba la retícula y el movimiento del personaje, mientras que el movimiento de cámara seguía en funcionamiento.

La solución pasó por reducir el detalle de los modelados, pero esto no termino de solucionar el problema, por lo que se optó por el culling, así como cambiar todo los shaders a los menos pesados con los que contaba Unity.

Esta solución ha hecho que el bosque sea jugable por fin, pero si se nota que va un poco más lento que las otras dos escenas, cuya carga poligonal es mucho menor.

7.4. Mi código, mis normas

Aquí nos encontramos con el problema de la encapsulación del código, que ha provocado que las interacciones con los *dropdown* se vuelvan más problemáticas.

Debido a que no se puede entrar en el script del dropdown para poder hacer modificaciones, no se puede acceder a la lista de elementos de forma sencilla, y mucho menos modificar la interacción con estos elementos, de forma que el click de andar y el de presionar sobre esos elementos se solapa en lugar de poder hacer la interacción con un temporizador y la retícula (como en los otros casos).

La solución definitiva será montar un dropdown personal, pero ante la falta de tiempo se ha optado por deshabilitar el control de movimiento cuando estamos interaccionando con los dropdown, de forma que al terminar la interacción podamos seguir moviéndonos.

A. Manual de Usuario

A.1. Material necesario

A parte de la aplicación descargada, el usuario debe disponer de los siguientes elementos:

- Una cardboard o similar
- Unos cascos estéreo
- Un móvil con sistema mínimo *android 4.4 Kit Kat*

A.2. Acciones del jugador

Las acciones que el usuario puede desarrollar son las siguientes;

- Andar hacia adelante: presionar el botón de la cardboard.
- Girar la cámara: girar sobre uno mismo para que el acelerómetro determine hacia dónde miras.
- Interaccionar con un elemento: si se puede interaccionar, al apuntar la retícula hacia él esta cambiará a una barra de carga que, al terminar, desata la interacción asociada.

A.3. Objetivos en las pantallas

A.3.1. Init

Esta es la primera escena de la aplicación y está pensada para acostumbrarse a la interfaz de usuario.

La tarea es bien sencilla en este caso: encontrar al ser invisible que nos está llamando desde algún lugar.

Cuando por fin lo encontramos, nos mandará a otra escena después de despedirse.

A.3.2. EchoChamber

Aparecemos en una habitación cúbica después de despedirnos del amigo invisible donde suena una tonadilla alegre.

En esta zona hay tres particularidades:

- Un icosaedro
- Un menú para cerrar la aplicación o avanzar a la siguiente zona

- Un menú que indica cambiar el material

¿No hay mucho eco aquí? El usuario deberá interaccionar con los distintos elementos y comprobar qué efectos tienen sus acciones en ese entorno.

A.3.3. EchoChamber

Esta ya es la última zona, un pequeño bosque con distintas aves. Prueba a acercarte e interaccionar con algunas, quizás interaccionar con ellas modifique el cartel en el que se lee "Most Wanted!!".

B. Anecdotas

B.1. Biclope involuntario

Este problema es algo que ocurre a todo los desarrolladores, pero es interesante comentarlo para que quede constancia de que el error humano existe y a veces es necesario descansar y continuar con el trabajo más adelante.

El problema que se presentó fue que en algunas escenas, la retícula no interaccionaba con los elementos dispuestos para ello. Esto fue un problema serio en su momento, pues parecía que tocaría rehacer gran parte del trabajo, pero después de un descanso, se percibió que la retícula estaba duplicada en la escena, lo que provocaba que no interaccionasen correctamente. Una vez eliminada la sobrante, todo funcionó como la seda.

C. GitFlow

C.1. ¿Qué es GitFlow?

Gitflow es un diseño de flujo de trabajo Git que se publicó por primera vez y se hizo popular por *Vincent Driessens* en *nvie*. El flujo de trabajo de Gitflow define un modelo de ramificación estricto diseñado en torno al lanzamiento del proyecto. Esto proporciona un marco robusto para gestionar proyectos más grandes.

Gitflow es ideal para proyectos que tienen un ciclo de lanzamiento programado, pues este flujo de trabajo no agrega nuevos conceptos o comandos más allá de lo que se requiere para el Flujo de trabajo de la rama de funciones, si no que asigna roles muy específicos a diferentes ramas y define cómo y cuándo deben interactuar. Además de las ramas de características, utiliza ramas individuales para preparar, mantener y grabar lanzamientos. Por supuesto, también puede aprovechar todos los beneficios del flujo de trabajo de Branch Branch: solicitudes de extracción, experimentos aislados y una colaboración más eficiente.

C.2. ¿Cómo funciona? [20] [21]

C.2.1. Ramas Develop & Master

En lugar de una sola rama, este flujo de trabajo usa dos ramas para registrar el historial del proyecto.

La rama *master* almacena el historial de lanzamiento oficial, y la rama *develop* sirve como una rama de integración de características. También es conveniente etiquetar todos los commits en master con un número de versión.

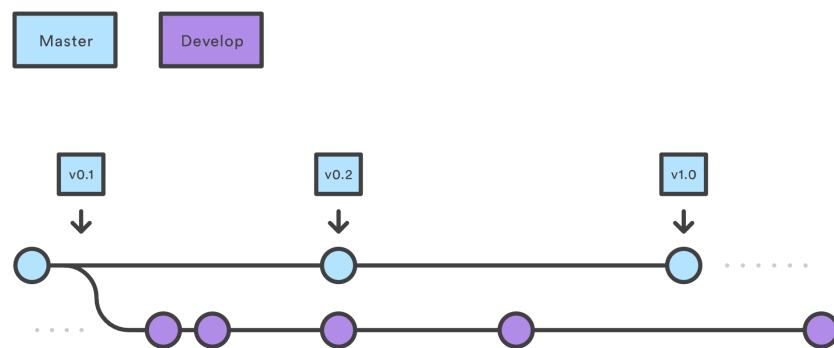


Figura C.1: Ramas Develop y Master

C.2.2. Ramas Feature

Cada nueva feature debe residir en su propia rama, que se puede enviar al repositorio central como respaldo/colaboración.

En lugar de bifurcarse de master, las features usan el desarrollo como su rama principal, de forma que cuando se completa una característica, se fusiona nuevamente en develop. Las características nunca deberían interactuar directamente con el maestro.

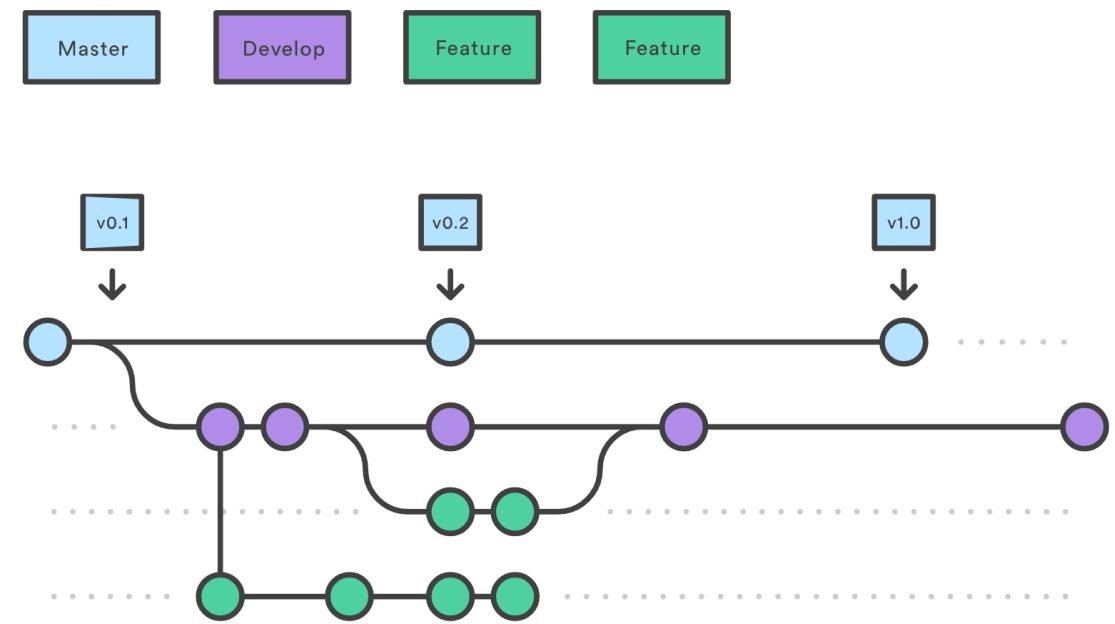


Figura C.2: Rama feature

C.2.3. Ramas Release

Una vez que el desarrollo ha adquirido suficientes características para un lanzamiento, bifurca una rama de release fuera del desarrollo. La creación de esta rama inicia el siguiente ciclo de lanzamiento, por lo que no se pueden agregar nuevas características después de este punto. Solo las correcciones de errores, la generación de documentación y otras tareas orientadas a la versión deben ir en esta rama.

Una vez que está listo para enviar, la release se fusiona en master y se etiqueta con un número de versión. Además, debe fusionarse nuevamente en el desarrollo, que puede haber progresado desde que se inició el lanzamiento.

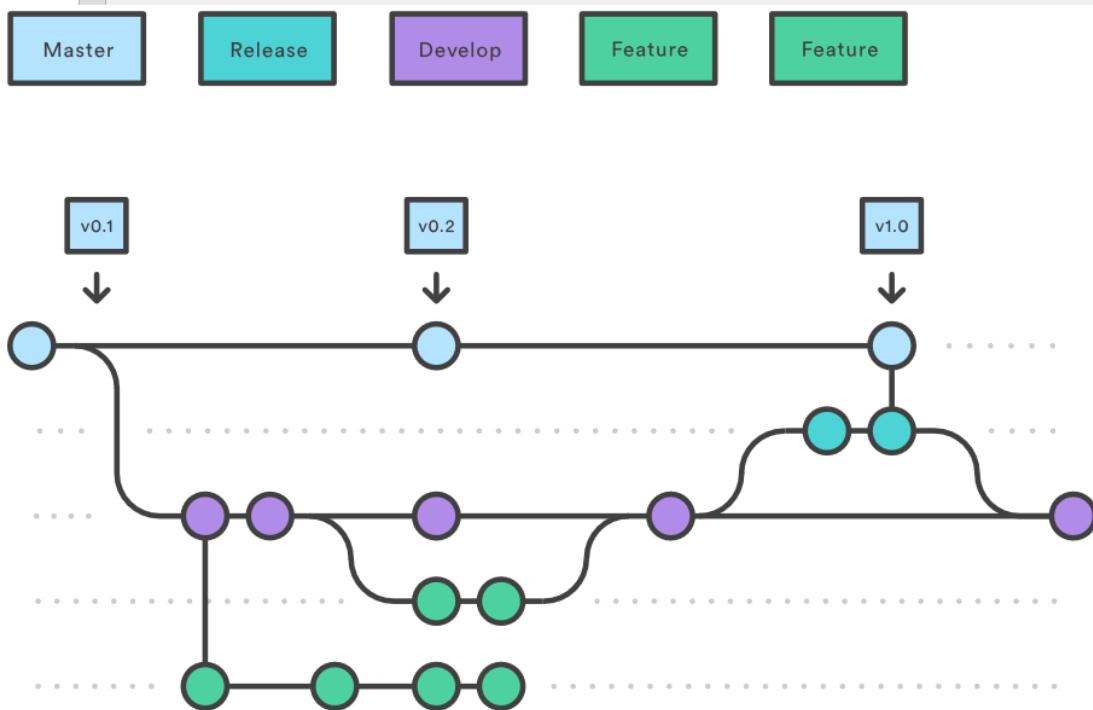


Figura C.3: Rama release

C.2.4. Ramas Hotfix

Las ramas de mantenimiento o "hotfix" se utilizan para parchear rápidamente las versiones de producción.

Las ramificaciones de revisión son muy parecidas a las ramificaciones de lanzamiento y ramificaciones de características, excepto que se basan en master en lugar de develop.

Esta es la única rama que debe bifurcarse directamente de master, y tan pronto como se complete la corrección, debe fusionarse tanto en master como en develop (o en la rama de la versión actual), y master debe etiquetarse con un número de versión actualizado.

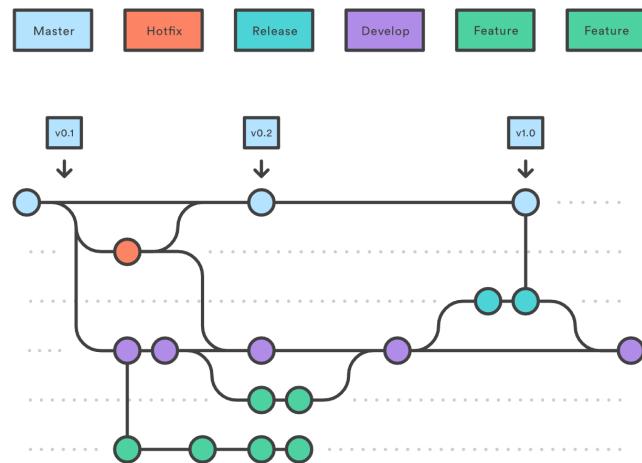


Figura C.4: Rama hotfix

Referencias

- [1] LOS CIRUJANOS QUE OPERAN A CIENTOS DE KILÓMETROS DE DISTANCIA *BBC*.
https://www.bbc.com/mundo/noticias/2014/05/140520_vert_fut_salud_cirujano_a_distancia_gtg
- [2] FORGOTTEN GENIUS: THE MAN WHO MADE A WORKING VR MACHINE IN 1957
techradar.
<https://www.techradar.com/news/wearables/forgotten-genius-the-man-who-made-a-working-2>
- [3] VIRTUAL REALITY: HISTORY *NCSA Illinois*.
<https://web.archive.org/web/20150821054144/http://archive.ncsa.illinois.edu/Cyberia/VETopLevels/VR.History.html>
- [4] GAMIFICACIÓN: EL APRENDIZAJE DIVERTIDO *Educativa*.
<https://www.educativa.com/blog-articulos/gamificacion-el-aprendizaje-divertido/>
- [5] ‘GAMIFICACIÓN’ DE LA CONDUCTA CIUDADANA EN CHINA *La Razón*.
<https://innovadores.larazon.es/es/not/gamificacion-de-la-conducta-ciudadana-en-china>
- [6] OCULUS RIFT S *Oculus*.
<https://www.oculus.com/rift-s/>
- [7] QUÉ ES LA MÚSICA 8D Y POR QUÉ SE HA HECHO VIRAL *Jaimé Altozano*.
<https://www.youtube.com/watch?v=e6Ekz7ZDV-w>
- [8] SONIDO 8D *Verdades y mentiras de la nueva forma de escuchar música que te 'hackea' el cerebro*.
https://www.yasss.es/sabiduria-pop/audio-8d-que-es-hackea-cerebro_0_2643900156.html
- [9] TUTORIAL RETÍCULA *Unity VR Tutorial - Gaze Timer Interaction + Teleport*.
<https://www.youtube.com/watch?v=bmMaVTV8UqY>
- [10] LIBRERÍA GOOGLEVR *Unity*.
<https://developers.google.com/vr/develop/unity/get-started-android>
- [11] DESCARGA GOOGLEVR *GoogleVR*.
<https://github.com/googlevr/gvr-unity-sdk/releases>
- [12] LIBRERÍA RESONANCE EN UNITY *Unity*.
<https://resonance-audio.github.io/resonance-audio/develop/unity/getting-started>
- [13] DESCARGA RESONANCE *Resonance*.
<https://github.com/resonance-audio/resonance-audio-unity-sdk/releases>

- [14] OCCLUSION CULLING *Unity*.
<https://docs.unity3d.com/es/current/Manual/OcclusionCulling.html>
- [15] GITKRAKEN *Información sobre GitKraken*.
<https://support.gitkraken.com>
- [16] CARDBOARD *Información sobre las cardboard*.
https://vr.google.com/intl/es_es/cardboard/
- [17] GUÍA DE C# *Microsoft*.
<https://docs.microsoft.com/es-es/dotnet/csharp/>
- [18] VR EN LAS AULAS *La realidad virtual en las aulas: ¿Realidad o virtual?*.
<https://www.educaciontrespuntocero.com/noticias/realidad-virtual-aulas-educacion/68851.html>
- [19] TUTORIAL SHADERS *Unity*.
<https://docs.unity3d.com/es/current/Manual/Shaders.html>
- [20] GITKRAKEN *Control de Versiones: ¿Por qué GitKraken?*.
<https://medium.com/@sergupe6/control-de-versiones-por-qu%C3%A9-gitkraken-ee1f30b4a18f>
- [21] TUTORIAL GITFLOW *Atlassian*.
<https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>
- [22] RAYTRACING *Introduction to Ray Tracing: a Simple Method for Creating 3D Images*.
<https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-ray-tracing?url=3d-basic-rendering/introduction-to-ray-tracing>