UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Loosy: un sintetizador digital en realidad mixta

Autor: Juan Riera Gomez
Tutor: Xavier Alamán Roldán

junio 2021

Juan Riera Gomez
Loosy: un sintetizador digital en realidad mixta

Juan Riera Gomez
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA - PRINTED IN SPAIN

Resumen

Las aplicaciones musicales en realidad mixta tienen todavía un largo camino que recorrer, en parte debido a las limitaciones tecnológicas que imposibilitan una medición precisa de la posición de las manos, junto con la ausencia de un cuerpos físicos palpables que sirvan de apoyo y referencia para los hologramas.

Loosy, el proyecto planteado en este trabajo, pretende resolver estos problemas mediante una interfaz que permite interpretar melodías con la mano derecha sin referencia visual o táctil al hacerlo de forma contínua en vez de discreta. La mano izquierda controla la armonía, que se elige pulsando sobre un panel virtual de grandes botones que el usuario puede agarrar y *pegar* a un objeto físico como una mesa o una pared, ofreciendo una referencia táctil para los cambios de armonía.

A pesar de no haberse podido probar en un dispositivo de realidad aumentada y de haberse desarrollado sólo la funcionalidad básica, Loosy ya se presenta como una aplicación divertida e intuitiva, incluso para alguien sin conocimientos musicales, y se espera completar su desarrollo para explotarla comercialmente en un futuro próximo.

Palabras clave

Sintetizador Digital, Realidad Mixta, Realidad Aumentada, Hololens 2, ChucK, Unreal Engine

Abstract

Music applications in mixed reality still have a wide room for improvement, partly because of the imposibility to obtain the precise location of the hands, combined with the inexistence of a phyisical body to be used as a physical reference.

Loosy, the project described in this document, pretends to solve these problems throught an interface that allows the user to play melodies with the right hand in a continous way, insted of discrete. On the left hand we have a chord keyboard, with big rectangular buttons that can be sticked to a phyisical object, like a table or a wall, offering a tactile reference for the harmony changes.

Even though we weren't able to test the application on a mixed reality headset, and even though we have only developed the basic functionality, Loosy already presents itself as a fun and intuitive app for musicians and for people with no musical background alike, and we expect to develop it further into commercial explotation.

Keywords

Digital Synthesizer, Mixed Reality, Augmented Reality, Hololens 2, ChucK, Unreal Engine

Índice

Listas

Lista de algoritmos

Lista de ecuaciones

Lista de figuras

Lista de tablas

1

Introducción

Loosy, el proyecto desarrollado a lo largo de este documento, se presenta de cara al usuario como un instrumento musical nuevo, que bebe de la tradición y estructura musical occidental y que tiene la especialidad de haber sido ideado para utilizarse en un dispositivo de realidad mixta.

Pretende solucionar los fallos de interfaz de otros sintetizadores en realidad virtual y aumentada, que al no dotar al usuario de referencias táctiles o visualmente precisas imposibilitan una correcta interpretación rítmica. Por ejemplo, si se simula un piano en realidad aumentada, el usuario al no poder apoyar los dedos sobre un objeto físico, no poder contrarrestar el margen de error del dispositivo encargado de localizar sus dedos, y tener que deducir la posición de las teclas en el espacio únicamente con la vista, no podrá tocar ninguna pieza con un ritmo mínimamente complejo. Loosy pretende solucionar parcialmente estos problemas.

Además, el diseño está orientado a ser intuitivo y gratificante para el usuario, incluso si este no tiene conocimientos musicales previos. Moviendo la mano derecha en el eje horizontal lateral, el usuario hace sonar distintas notas de una escala. Además podrá cambiar esta escala en tiempo real de entre un conjunto preseleccionado, que a su vez podrá editar desplegando un menú que aparece al mirarse la palma de la mano. Moviendo la mano hacia arriba o hacia abajo se controla la intensidad del sonido, y si acerca la mano al cuerpo lo suficiente dejará de producirse sonido. Al ser el cambio entre notas continuo y no discreto, dejamos de requerir de referencias visuales o táctiles estrictas.

Frente al usuario se muestra un tablero con una serie de botones prolongados hacia delante. Pulsando cada uno de esos botones con la mano izquierda el usuario puede hacer sonar un acorde, y alejando dicha mano de su cuerpo puede controlar la intensidad del mismo. El tablero se puede mover agarrándolo con la mano, y se puede pegar a un objeto físico real como una mesa o una pared. El tablero se pegará automáticamente a la superficie mas cercana, otorgando así referencias táctiles y visuales para los cambios de armonía.

En la práctica Loosy es una aplicación para Hololens 2, las gafas de realidad mixta de Microsoft, y lo único que podría tener de revolucionario es esta interfaz, que ha sido diseñada específicamente para realidad mixta. Desde un punto de vista más profundo, Loosy es un sintetizador digital con funcionalidad para hacer sonar escalas y acordes musicales.

Notación y aclareciones previas

- Cuando hablemos de síntesis de sonido se hablará recurrentemente de las propiedades del sonido, pero refiriéndonos a su significado en el contexto de la síntesis digital (ver [Roads et al., 1996], el capítulo 3 de [Howard and Angus, 2013] y las páginas 19-27 de [Rothstein, 1992]):
    - Amplitud: desplazamiento máximo en valor, no en tiempo. Una sonido en nuestro contexto no es una onda contínua como ocurre en un medio físico, sino que es una onda discretizada en intervalos de tiempo tan pequeños que el oído perciba el conjunto como un sonido continuo. Estos valores representan un desplazamiento del estado de reposo físico, y es ese desplazamiento el que llamamos amplitud. Son los valores del sonido en sí, producidos por ejemplo por un oscilador como explicamos a continuación.
    - Frecuencia: una característica que diferencia el sonido musical frente al resto de sonidos es la repetición de ondas sonoras, que hace que percibamos un tono y no un ruido. La frecuencia es el número de estas repeticiones por unidad de tiempo. Normalmente se mide en Hercios (Hz), que son el número de ciclos por segundo.
    - Una nota musical se puede definir de varias formas, pero lo que todas tienen en común es que una nota musical, en una afinación predefinida, tiene una frecuencia sonora asociada. En este documento esta frecuencia es la correspondiente con la afinación temperada del piano (ver [Olson, 1967] pgs 150-152), partiendo de la nota La inmediatamente superior al do central afinada a 440hz.
- El modelo clásico de sintetizador analógico se compone de varios bloques que generan o modifican el sonido. Los siguientes se mencionan en este documento (ver el capítulo 2 de [Vail, 2014]):
    - Oscilador: es el bloque básico de la mayoría de sintetizadores analógicos. Es un módulo que produce un voltaje oscilante con una determinada forma de onda y frecuencia. En este documento, dado que los osciladores son digitales, nos referimos a ellos como objetos que generan en el tiempo valores entre -1 y 1 siguiendo una onda y frecuencia concretas.
    - Filtro paso bajo (Low Pass Filter): módulo que recibe un sonido y lo transforma recortando frecuencias por encima de una determinada que llamaremos *de corte*. El corte no es generalmente un corte seco, suele parecerse mas a un descenso parabólico que comienza en la frecuencia de corte.
    - Reverb: módulo que produce el efecto de que el sonido que recibe se emite dentro de algún tipo de habitáculo, que puede ser tan pequeño como una caja de zapatos o tan grande como una catedral.
- Cuando hablemos de *ganancia* nos referiremos a un valor entre 0 y 1 que actúa como constante multiplicadora de la amplitud del sonido. Una ganancia de 0.5 reducirá la amplitud a la mitad, una ganancia de 1.0 dejará el sonido tal cual, y una amplitud de 0.0 convertirá el sonido en silencio.
- A lo largo de este documento se habla de realidad mixta, término que en este documento utilizaremos para referirnos a realidad aumentada en el contexto actual. En realidad, no hay un consenso en cuanto a las diferencias entre realidad aumentada y mixta. En este paper [Mann et al., 2018] se define la realidad mixta como un término que engloba todo el espectro intermedio entre realidad virtual y realidad, como si pudiéramos mezclar uno y otro en cualquier proporción, pasando por realidad aumentada (más cercano a la realidad) y realidad virtualizada (más cercano a la realidad virtual).
- El símbolo $\tilde{V}$ representa el vector unitario en la dirección del vector V.
- Cuando hablemos de ejes de coordenadas, será importante su orientación con respecto al usuario. Los ejes X e Y serán los ejes horizontales, siendo X el eje frontal e Y el eje lateral. El eje Z será por tanto el eje vertical. Esta es la disposición de ejes en Unreal Engine, una de las herramientas utilizadas en el trabajo.
- Las distancia entre distintas notas se le suele llamar intervalo musical. El intervalo más pequeño de la música occidental es el semitono, que es lo que se avanza cuando se sube un traste en la guitarra o se toca la siguiente tecla en el piano (incluyendo las teclas negras). Dos semitonos conforman un tono. (Ver el capítulo 9 de [Pilhofer

and Day, 2019])

- Los nombres de las notas se van repitiendo a lo largo del piano, cuando se sube por la escala de Do mayor, partimos de la nota Do, luego Re, Mi... hasta llegar a la nota Si. La siguiente nota es un Do de nuevo. Cuando se repite el nombre de la nota de esta manera se dice que se ha subido o avanzado una octava, que es el intervalo mas consonante que hay, tanto es así que dos notas separadas por una octava reciben el mismo nombre, aunque una suene más aguda que la otra. A oídos sin entrenamiento musical, ambas sonarán tremendamente *similares*. (Ver página 115 de [Pilhofer and Day, 2019])
- Una escala musical es una lista de intervalos que se repiten a lo largo del conjunto de semitonos de la afinación utilizada. Suelen abarcar una octava. La escala mayor por ejemplo sube con los intervalos tono, tono, semitono, tono, tono, tono, semitono, que son las distancias relativas entre las notas que componen la escala. Cuando se le pone una nota base es cuando la escala se sintetiza en notas concretas. Re mayor por ejemplo se compone de Re, Mi, Fa sostenido, Sol, La, Si, Do sostenido y Re. Y las distancias entre las notas son las encunciadas antes. Un acorde musical muy similar a priori, solo que el acorde está pensado para que sus notas suenen todas a la vez, mientras que en la escala en principio solo se escucha una, dando forma a una melodía. El concepto de escala se explora en el capítulo 3 de [Olson, 1967] y capítulo 7 de [Pilhofer and Day, 2019].
- Para hablar de una escala o un acorde siempre hablaremos de su nota raíz, que es la nota de la que parte, y su calidad (quality en inglés), que es lo que verdaderamente define la escala o el acorde, y que es lo que nos indica cómo se calcula el resto de notas a partir de la nota raíz. Do mayor por ejemplo, tiene Do como nota raíz y calidad mayor.
- El cifrado americano expresa las notas con nombres distintos a los europeos. En Estados Unidos se comienza desde la nota La, a la que se la llama “A”. Se sigue con “B” (si), “C” (do), “E” (mi), “F” (fa), “G” (sol) y “A” es La otra vez.
- Cuando hablemos de sostenido (#) estamos subiendo un semitono la nota raíz, y cuando hablemos de bemol (b) estaremos bajando un semitono. Por ejemplo Do# está un semitono por encima de la nota Do, y Reb está un semitono por debajo de la nota Re. Como Do y Re están a un tono de distancia, en nuestra afinación Do# y Reb tienen la misma frecuencia asociada, serían la misma tecla en el piano.
- Las notas midi (concepto introducido más adelante) avanzan en semitonos, cuando hablemos de distancia entre dos notas midi estamos hablando del número de semitonos entre esas dos notas. Entre un Mi y un Fa la distancia es de un semitono. Entre un Do y un Re, de dos (tono). (Ver páginas 22-23 de [Huber, 2007] y [Rothstein, 1992]).

2

Estado del arte

Analizamos aquí el contexto tecnológico que rodea al proyecto, que como vamos a ver, se vale de algunas tecnologías de absoluta vanguardia, en combinación con otras que son más antiguas.

2.1. *Smartglasses* de realidad mixta

Un dispositivo que cubre parcial o totalmente la visión con un cristal transparente sobre el que se proyectan imágenes, generalmente tienen además algún método para reconocer el espacio y su forma, e incluso para seguir los movimientos de las manos y ojos del usuario. Esta tecnología abre un espacio nuevo como pocas veces se ha visto en la historia del desarrollo tecnológico, desde aplicaciones en medicina y educación hasta nuevas formas de ocio.

Para gran parte del público, la primera implementación de esta idea fueron las Google Lens. Hoy parece que llegaron demasiado pronto, cuando la tecnología necesaria aún estaba en desarrollo y no había demanda suficiente. Por ello dejaron el proyecto de lado dos años después de iniciarlo (en 2015) y lo retomaron en 2017, para anunciar en 2019 una nueva versión de sus gafas. A pesar de esto, no son ni las únicas ni las mejores.

Figura 2.1: Gafas Google Glass.

2.1.1. Hololens 2

Hololens 2 son probablemente las gafas de realidad mixta más avanzadas del mercado. Fabricadas por Microsoft, cuentan con una serie de cámaras y sensores que reconocen la forma del espacio en el que se encuentran, así como la posición de las manos y de los ojos. Se anunciaron en 2019 como sucesoras de las Hololens 1, que ya fueron bastante revolucionarias. Funcionan como dispositivo autónomo, es decir, no requieren a priori de una unidad de procesamiento externa o de ningún dispositivo más allá de las propias gafas, y corren en el sistema operativo de Microsoft Windows 10.

Loosy se ha diseñado para usarse en este dispositivo, que aún está mejorándose. Tanto es así que durante la realización de este trabajo ha salido una actualización importante de sus herramientas de desarrollo, y se ha adaptado la implementación para utilizarlas. Estas herramientas, llamadas Mixed Reality Toolkit for Unreal Engine (MRTK) y OpenXR, no tienen aún una documentación extensa, y de hecho MRTK está en versión beta.

Algunas referencias interesantes relacionadas con este producto: [Noor, 2016], [Garon et al., 2016], [Hanna et al., 2018], [Bottino et al., 2017], [Kalantari and Rauschnabel, 2018]

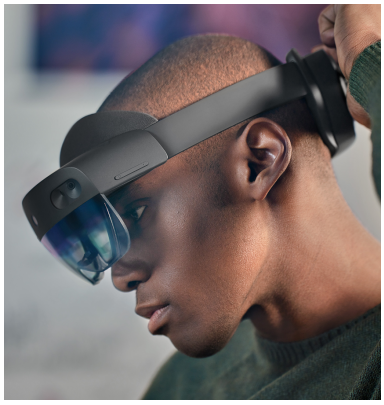


Figura 2.2: Gafas Hololens 2.

2.1.2. Otras opciones

La competencia de Microsoft ha desarrollado sus propios modelos de *smartglasses* tratando de solucionar problemas de las Hololens o buscando mejores características. Algunas, como las Iristick Z1 han tratado de solucionar la sensación de casco de las Hololens 2 aprovechando la capacidad de procesamiento de un smartphone, al que se conectan mediante un cable y una "unidad de bolsillo" que actúa como intermediaria. Otras como las o las Epson Moverio BT-200 no requieren de smartphone, pero sí de unidad de bolsillo, y otras como las Sony SmartEyeglass se conectan al smartphone por wifi o bluetooth. Ninguna tiene la autonomía de las Hololens 2.

Lo más parecido que hay en el mercado que hay en el mercado son las Magic Leap, que tienen una autonomía similar a las Hololens 2, pero que son inferiores al resto de niveles, tanto en calidad de imagen como en capacidad de procesamiento.

El mundo de la realidad virtual está también muy avanzado, como es el caso de las Oculus Quest 2. Está teniendo gran impulso porque en este caso sí existe la tecnología necesaria para que funcione bien, y porque se relacionan con el mundo de los videojuegos que es una de las mayores industrias del mundo.

2.2. Motores que asisten en el desarrollo en realidad mixta

Vamos a introducir brevemente los dos motores de videojuegos que se utilizan para desarrollar para Hololens 2. Hay una tercera alternativa que consiste en programar directamente de cero sin partir de un editor 3D, pero esa opción se descartó para este proyecto por resultar inabarcable.

2.2.1. Unreal Engine

Unreal engine es un motor de videojuegos basado en C++. Ampliamente conocido por juegos como Fornite, cuenta con un marco de librerías para facilitar el desarrollo en Hololens 2. El motor no para de avanzar y prueba de ello es que recientemente han sacado funcionalidad nueva para simular humanos de manera hiperrealista, y que durante el desarrollo de este proyecto sacaron una nueva versión para el motor (4.26) y ya han anunciado que van a sacar la versión 5.0.

Figura 2.3: Logo de Unreal Engine.

Una especialidad de Unreal Engine es que permite programar con un sistema inusual, pero que facilita el aprendizaje del motor cuando no se tienen conocimientos del mismo, llamado sistema de Blueprints. Un Blueprint es algo muy parecido a un fragmento de código, pero que se compone de *cajitas* interconectadas entre sí en lugar del sistema de texto en el que se basan los lenguajes de programación. El desarrollo de este proyecto en Unreal Engine se ha realizado utilizando este sistema. Referencias: [Sanders, 2016], [Karis and Games, 2013].

2.2.2. Unity

Unity es otro motor, en este caso basado en C#. Es probablemente el motor de videojuegos más famoso del mercado junto con Unreal Engine. Se ha utilizado en todo tipo de proyectos desde juegos indie hechos por equipos pequeños hasta juegos enormes como Hearthstone, que actualmente tiene 100 millones de jugadores. También introdujo librerías dedesarrollo para Hololens 2.

Referencia: [Halpern, 2019].

2.3. Programación de aplicaciones musicales

Las aplicaciones musicales son otro campo en expansión. Constantemente salen al mercado sintetizadores digitales nuevos, emulaciones de efectos analógicos como compresores o reverbs, emulaciones de instrumentos reales que cada año logran ser más realistas... Todo esto está mayoritariamente programado en C++, que facilita la programación musical porque requiere de mucha manipulación de bajo nivel. Sin embargo no es la única opción, ni de hecho la más especifica. En este proyecto hemos optado por otra herramienta creada para ocasiones como esta: ChucK.

ChucK es un lenguaje de programación orientado a la creación musical. Al ser el sonido un fenómeno que sucede en el tiempo y que tiene una estrecha relación con el mismo, ChucK dota al usuario de funciones en tiempo real con una precisión asombrosa. Funciona con una máquina virtual a la que se le añaden *Sporks*, que es el nombre que se le da a un hilo escrito en ChucK. Si se lanzan varios

Sporks a la vez se mantendrán coordinados en el tiempo.

En cuanto a sintaxis ChucK se parece mucho a un C++ con una funcionalidad más limitada en determinadas áreas como la orientación a objetos o la modularidad de programas en varios archivos. A cambio ChucK simula el funcionamiento de un sintetizador analógico modular (compuesto por módulos como osciladores o filtros que se interconectan) con una serie de tipos creados para tal fin. Un tipo podría ser un oscilador, que se conecta a un efecto de reverberación que se conecta a la salida de audio.

Algo muy especial de ChucK que podría sorprender al lector es el propio operador ChucK "=>". Sirve tanto para asignar variables (3 =>int a;), como para interconectar módulos musicales (oscilador =>filtro). En el código siguiente se presenta un ejemplo de este lenguaje.

Código 2.1: En esta figura se presenta un ejemplo de código ChucK en el que se declaran algunas variables de distintos tipos similares a otros lenguajes de programación, a las que se asignan algunos valores. A la de tipo string se le asigna la cadena "Hello World!" que se imprime. Después se declara un oscilador de diente de sierra (SawOsc), que se conecta a una reverb (Rev) y a la salida de audio del ordenador (dac) para finalmente asignarle una ganancia y una frecuencia. Por último esperamos un segundo, que es lo que dura el sonido. Después termina el programa y deja de sonar.

```
1  // Este es un fichero de ejemplo
2  int a;
3  0.2 => float b;
4  "Hello world!" => string c;
5  <<< c >>>; // Imprimimos Hello world!
6
7  SawOsc miOscilador => Rev miReverberacion => dac;
8  b => miOscilador.gain;
9  440 => miOscilador.freq;
10
11 1::second => now;
```

Referencias y tutoriales del lenguaje: [Wang, 2008], [Salazar et al., 2006], [Kapur et al., 2015].

2.4. Aplicaciones musicales en realidad mixta y realidad virtual

Presentamos a continuación algunas aplicaciones en realidad mixta y realidad virtual especialmente interesantes que demuestran el potencial de la tecnología.

- *Music Everywhere* es una aplicación para Hololens que sirve para aprender a tocar el piano. Constituye un método divertido para que además de aprender se pueda simular la experiencia de tocar con otros instrumentistas.
- *Exa* es una aplicación para realidad virtual que permite tocar sintetizadores a través de distintas interfaces virtuales, definitivamente en la línea de este proyecto en cuanto a que plantea nuevas disposiciones físicas para

producir el sonido, otras disposiciones son similares a instrumentos clásicos. Por ejemplo en una aparecen una serie de teclas dispuestas como las teclas de un piano en el aire.

- *Vinyl reality* permite hacer *djing* en realidad aumentada. El usuario tiene ante sí dos platos de vinilo que puede tocar y manipular.
- *Beat Saber* es un juego en realidad virtual en el que golpeas con dos sables láser a cajas al ritmo de la música.
- En [Corrêa et al., 2009] se habla de un sistema de musicoterapia asistido por realidad aumentada para rehabilitación de pacientes que han requerido de intervenciones cerebrales.
- Aplicaciones para interpretar música colaborativamente con otras personas [Poupyrev et al., 2000].
- La realidad mixta tiene muchísimo potencial en el área educativa. En [Molero et al., 2021], artículo publicado el mismo año que se presenta este trabajo, se describe un sistema para educación musical. Algo similar también se plantea en [Chow et al., 2013] y en [Rusiñol et al., 2018].
- *The music table* es un sistema de composición musical basado en tarjetas dispuestas sobre una mesa leídas por una cámara [Berry et al., 2003].
- Loosy, el proyecto presentado en este documento, podría algún día llegar a constituir un sistema de interpretación de música electrónica. Algo similar se plantea en [Poupyrev et al., 2001].

2.5. Sistemas de comunicación entre dispositivos orientados a música

Desde el inicio de la electrónica y la informática musical ha sido necesario comunicar parámetros y notas entre dispositivos. Presentamos a continuación los dos sistemas más utilizados.

2.5.1. OSC

OSC, siglas de Open Sound Control, es un protocolo de comunicación perfecto para expresar movimientos de parámetros de aplicaciones audiovisuales, y por ello se utiliza masivamente en programas de creación musical. El protocolo se basa en mensajes que tienen dos campos básicos:

- La dirección es un una cadena de caracteres, que contiene una serie de subcadenas separadas por el símbolo '/', muy similar a la forma típica de acceder a subcarpetas en sistemas operativos.
- El contenido es en esencia una lista de subcampos de lognitud y tipo variable. Los distintos subcampos se identifican con un índice y el tipo de cada uno de ellos.

Referencias: [Wright et al., 2017], [Freed and Schmeder, 2009].

2.5.2. Protocolo MIDI

Musical Instrument Digital Interface, es el sistema estándar de comunicación entre dispositivos musicales. Se creó en la década de los 80 y actualmente la gran mayoría de los instrumentos electrónicos como sintetizadores y teclados se comunican mediante este protocolo. Expresa el estado de diversos

parámetros musicales generalmente relacionados con la interpretación, así como las notas musicales en sí mismas. Además de utilizarse para expresar toda esta información en tiempo real, sirve también para almacenarla.

El protocolo en su primera versión se compone de mensajes estandarizados que en general son bastante sencillos pero de amplia variedad. Los valores se expresan en campos con rango 0-127 al utilizarse 8 bits y reservar el primero quedando 7 bits para datos, y los mensajes en lenguaje natural se traducen a frases como "Se pulsa la nota Do con una intensidad de 100". Otro dato importante y del que nos valdremos en este documento es que asocia a cada nota de la afinación temperada un número, siendo el 60 el do central y avanzando o retrocediendo por semitonos.

En enero del año anterior a la presentación de este trabajo se presentó la segunda versión del protocolo, elegida por unanimidad entre los fabricantes de instrumentos. MIDI 2.0 es compatible con la versión 1.0 y aún está pendiente de ser interiorizado por la industria. Trae grandes novedades como la introducción de valores de 16 bits frente a los campos de 8 bits mencionados antes.

Figura 2.4: Logo del protocolo MIDI 2.0.

Referencias: [Rothstein, 1992], [Huber, 2007], [Lehrman, 2020].

3

Definición del proyecto

Se explican a continuación los objetivos y limitaciones base del proyecto, con el alcance hasta el que se había planificado para este trabajo y las limitaciones con las que nos hemos encontrado.

3.1. Objetivos

Loosy es una interfaz musical en realidad mixta. Propone una experiencia similar a la de estar tocando un instrumento musical nuevo, inspirado por sintetizadores que ya existían antes como el Theremin y por interfaces de control mas modernas como el Novation Launchpad. La novedad aquí es este enfoque hacia la realidad mixta.

El sistema podría llegar a hacerse bastante personalizable si se sigue desarrollando, pero por ahora hay una función asignada a cada mano:

- La mano derecha hace sonar distintas notas dentro de una escala determinada al moverse en el eje horizontal lateral, produciendo un sonido más agudo al desplazarse hacia la derecha y más grave hacia la izquierda, como ocurriría en las teclas de un piano. El eje vertical controla la frecuencia de corte de un filtro paso bajo, que se traduce de cara al usuario como una mayor o menor intensidad del sonido al alterar el timbre de más suave y redondo a más agresivo y brillante. El eje horizontal frontal al usuario controla si está se esta *tocando* el instrumento o no. Cuando la mano se aleja unos 30 cm del cuerpo comienza a sonar para sonar con amplitud máxima de 35 cm en adelante.
- A su vez se puede cambiar la escala de la mano derecha haciendo uso de dos algoritmos distintos: rotar la mano con respecto al codo, y estirar los dedos índice y corazón. Los dos sistemas quedan diseñados e implementados. El primero se ha podido probar en un simulador, pero el segundo no debido a limitaciones del proyecto explicadas más adelante. Los dos algoritmos permiten al usuario elegir en tiempo real una de entre tres escalas musicales preseleccionadas.
- Si el usuario muestra la palma de su mano a su vista aparecerá un menú que le permitirá preseleccionar las escalas mencionadas antes.
- Frente a la mano izquierda se sitúan una serie de botones rectangulares, cada uno de los cuales sirve para seleccionar un acorde de acompañamiento distinto. La posición de la mano izquierda sobre el plano extendido del tablero de botones se utiliza para controlar la intensidad del acorde, en concreto modula la frecuencia de corte de un filtro.

  Este tablero además se puede *agarrar* y mover a una superficie real del entorno. El tablero se pegará a la superficie real más cercana, lo cual quiere decir que las gafas tratarán de ajustar la posición del tablero para

que quede lo más cerca posible a una superficie que detecte en el entorno del usuario, como podría ser una mesa o una pared. Esta funcionalidad provee al usuario de una referencia táctil y visual clara para los cambios de armonía.

3.2. Restricciones y suposiciones

Loosy en el estado en el que se plantea en este trabajo no pretende ser una aplicación terminada y lista para el usuario final. Pretende ser intuitiva una vez está lanzada y funcionando, pero la configuración, instalación y ejecución no están pensadas ahora mismo para un usuario sin conocimientos informáticos, que no sea capaz de editar los ficheros de configuración, o que no sea capaz de lanzar los dos procesos que componen la aplicación en orden: el controlador se debe iniciar después que el sintetizador. Esto se verá más claro más adelante. Así pues, la principal suposición es que el usuario que va a utilizar la aplicación tiene los conocimientos necesarios para utilizarla.

Sería sencillo desarrollar menús visuales que el usuario pueda utilizar en lugar de los ficheros de configuración y se deja como trabajo futuro más allá del alcance de este proyecto. También queda pendiente la realización de una documentación propiamente dicha en forma de manual de usuario.

El proyecto se ha visto restringido por las limitaciones de tiempo del propio trabajo y por dificultades burocráticas para conseguir unas gafas en las que probar el proyecto, que se ha desarrollado exclusivamente con un simulador de las mismas. Dicho simulador solo permite mover las manos en los tres ejes, y hacer algún gesto como rotar las manos torpemente, de manera que funcionalidades como la de estirar los dedos no se han podido llegar a probar, y otras como la rotación de la mano se han podido probar dificultosamente. Mencionar que tanto el tutor como el alumno contaban con obtener una copia de las gafas para el desarrollo de este proyecto al principio, pero la compra se fue retrasando hasta llegar a este punto.

A esto se ha sumado que las herramientas de desarrollo para Hololens 2, al ser tan novedosas, tienen una documentación muy limitada. Los recursos disponibles en internet son muy básicos y posiblemente el más útil fuera una retransmisión en directo que publicó Microsoft en la plataforma Youtube y que quedó grabada, en la que mostraban algunas de estas herramientas. Además el proyecto se ha desarrollado durante la pandemia del virus COVID-19 de forma telemática, en un equipo con una capacidad computacional muy limitada.

4

Diseño del proyecto

4.1. Diseño funcional

El proyecto consta de dos bloques: *SynthController* y *LoosySynth*. El primero es el módulo controlador, que se ejecuta en las propias gafas Hololens 2, y su función principal es la de obtener todos los datos necesarios para **controlar** el sonido. El segundo es el módulo sintetizador, que se encarga de recibir esos datos y **producir el sonido** propiamente dicho. Actualmente se ejecuta también en las propias gafas, pero el diseño está preparado para que en un futuro no sea así.

4.1.1. Obtención de los datos posicionales de las manos en SynthController

Para ello se encarga de seguir el movimiento de las manos del usuario, ordenarlo como corresponda y enviarlo a *LoosySynth*. Sean los siguientes vectores de tres coordenadas que representan las posiciones en el espacio de los puntos importantes de las manos:

- P_d, P_i los centros de las palmas de la mano derecha e izquierda.
- C_p, C_i, C_k la punta, falange intermedia y puño (knuckle) del dedo corazón de la mano derecha.
- U_k el puño del pulgar de la mano derecha.
- M_k el puño del meñique de la mano derecha
- I_p, I_i, I_k la punta, falange intermedia y puño del dedo índice de la mano derecha.

Otro punto importante es la posición de las gafas, que denominaremos G, y la dirección en la que está mirando el usuario, de la cual solo nos interesa su proyección en el plano XY (el plano horizontal) que llamaremos G_x. Rotando este vector sobre el eje Z 90° obtenemos otro vector del plano XY perpendicular a G_x que llamaremos G_y.

Esto nos permite construir un nuevo sistema de coordenadas con origen en G y ejes $\langle G_x, G_y, Z\rangle$, y al traducir las posiciones de todos los puntos mencionados antes a este nuevo sistema lo que nos queda es la posición relativa de todos esos puntos con respecto a la posición de la cabeza, teniendo en cuenta hacia dónde miran las gafas en el plano XY, pero obviando la inclinación de la cabeza. A

partir de aquí nos referiremos a los puntos en este nuevo sistema de coordenadas. Obtenemos ahora para la mano derecha:

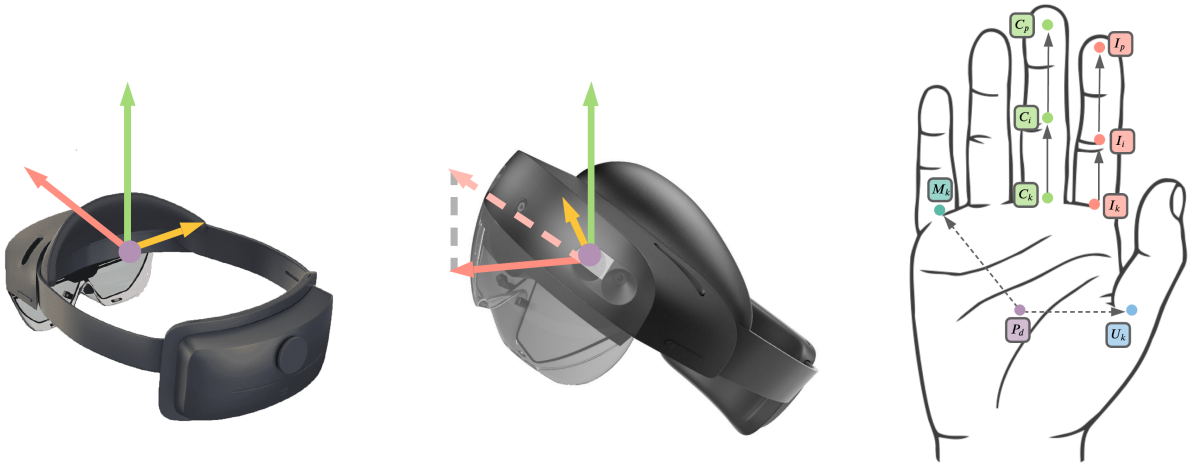


Figura 4.1: En la izquierda y centro tenemos el sistema de coordenadas con respecto a las gafas $\langle G_x, G_y, G_z \rangle$, en colores rojo, amarillo y verde respectivamente, con el centro de coordenadas en lila. En la figura central se ilustra cómo la inclinación de la cabeza no afecta a los ejes dibujando además el vector frontal de las gafas en rojo rayado.
A la derecha un diagrama de los puntos importantes de la mano derecha. Los vectores rayados son los utilizados para medir la rotación de la mano, y los lisos para medir cómo de estirados están los dedos.

- El parámetro $A_i = \left(\frac{I_p - I_i}{||I_p - I_i||}\right) \cdot \left(\frac{I_i - I_k}{||I_i - I_k||}\right)$ mide cómo de alineados están los vectores *puño* $\rightarrow$ *falange intermedia* y *falange intermedia* $\rightarrow$ *punta* del dedo índice, es decir, mide si el dedo índice está estirado o no, siendo $A_i = 1$ completamente estirado y $A_i \simeq 0$ una posición contraída.
- De la misma manera, $A_c = \left(\frac{C_p - C_i}{||C_p - C_i||}\right) \cdot \left(\frac{C_i - C_k}{||C_i - C_k||}\right)$ mide cómo de estirado está el dedo corazón.
- El vector $K_x = C_k - P_d$ es el que une el centro de la palma con el puño del dedo corazón. Proyectándolo sobre el eje XY y rotándolo 90° con respecto al eje Z obtenemos un segundo vector perpendicular al primero que llamamos K_y, y haciendo el producto vectorial entre estos dos vectores obtenemos un tercero perpendicular a los dos $K_z = K_x \times K_y$. Este nuevo sistema de coordenadas $\left\langle \tilde{K_x}, \tilde{K_y}, \tilde{K_z} \right\rangle$ es el sistema de coordenadas unitario relativo a la mano ignorando la rotación de la misma con respecto al codo y su inclinación con respecto a $\tilde{K_y}$ (recordar que $\tilde{V}$ es V unitario). Realizando la proyección: $R = \left(\frac{U_k - P_d}{||U_k - P_d||}\right) \cdot \tilde{K_z} - \left(\frac{M_k - P_d}{||M_k - P_d||}\right) \cdot \tilde{K_z}$ obtenemos un valor real que mide cómo de rotada está la mano en el sentido de las agujas del reloj, siendo el 0 la posición horizontal.
- Estos valores que hemos calculado permiten al usuario cambiar de escala musical en directo con dos algoritmos distintos. Como se explica en otros apartados estos dos algoritmos quedan desarrollados y listos para probarse, ya que no se han podido realizar experimentos para ver cual es más intuitivo para el usuario por limitaciones técnicas.
    - El primer algoritmo consiste en mirar si el usuario tiene estirados uno, dos o ninguno de los dedos índice y corazón.
    - El segundo algoritmo consiste en dividir el intervalo entre -1 y 1 en 3 subintervalos, de manera que la mano siempre esté en uno de ellos.

Ambos algoritmos devuelven uno de tres valores posibles que llamaremos fases. Cada fase se asocia con una escala musical, y el conjunto de las 3 escalas musicales asociadas a cada fase puede ser editado por el usuario girando la palma de la mano derecha hacia las gafas. Cuando hace eso, se despliega un menú para cambiar las escalas seleccionadas.

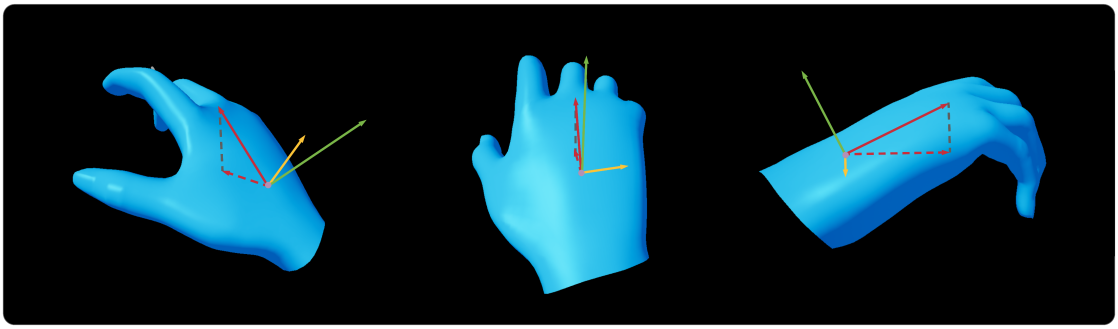


Figura 4.2: Tres posiciones distintas de la mano derecha, con el sistema $\langle K_x, K_y, K_z \rangle$.

Por último se calcula la proyección de la mano izquierda sobre el plano del teclado de acordes extendido a todo el espacio y se mira cómo de avanzada está sobre el eje horizontal frontal dentro del propio teclado. A esta cantidad entre 0 y 1 la llamaremos A_v. Por ejemplo, si la mano está en el borde inferior del teclado de acordes (el más cercano al usuario si el teclado está en la posición horizontal frente al usuario), el valor será 0, y si está en el borde superior (el más alejado del usuario si el teclado está en posición horizontal), tendrá valor 1.

4.1.2. LoosySynth

Este módulo sintetizador, que recibe los datos anteriores y genera el sonido corresponiente. Para ello sabe en qué escala tiene que sonar y traduce las posiciones de las manos a los sistemas de medida correspondientes para que los movimientos sean intuitivos. En concreto se realizan las siguientes operaciones:

- P_{dz}, que es la altura relativa a la que se encuentra la palma de la mano derecha, se traduce a frecuencias de filtro mediante la siguiente fórmula:

$$\text{freq} = \max\left\{|P_{dz} + 50|^2, 0\right.$$

  De manera que el punto más bajo son 50 cm por debajo de la cabeza, y desde ahí va creciendo cuadráticamente. Estimé que el punto más alto al que puede llegar la mayoría de las personas con su mano derecha son los 50 cm por encima de su propia cabeza, llegando a un máximo de 100 kHz de frecuencia de filtro. A esta frecuencia el filtro se anula al estar muy por encima del espectro auditivo humano, que llega como hasta los 20 kHz.

- P_{dy}, que es el desplazamiento lateral relativo de la mano derecha se traduce a notación midi mediante la siguiente fórmula:

$$m_1 = \frac{P_{dy}}{10} + 60$$

  De manera que teniendo la mano en el plano XZ (el plano vertical-frontal), la nota midi es 60, que es el Do central, y cada 10 cm que se mueve la mano a la derecha o izquierda se cambia de nota. Una vez en midi se

mira cuántas notas se han avanzado desde el do central hacia arriba o hacia abajo, y ese número de notas es el que se "avanza" en la escala que tengamos seleccionada. Por ejemplo si tenemos una escala con las notas Do, Re, Mi, Fa y Sol, y la mano está en una posición que se corresponde con la nota $m_1 = 63$ (que en midi riguroso representaría la nota Mi bemol) a través del cálculo anterior, la nota que se hace sonar m_2 en este caso es la cuarta de la escala, que es un Fa.

Como la mano no está en posiciones discretas siempre va a estar entre dos notas, así que comenzamos calculando m_2 mediante interpolación lineal. Pero esto no nos interesa, ya que casi todas las notas sonarían desafinadas. Lo que nos interesa es que cada nota tenga un ancho en el que la nota suene razonablemente afinada, así seguidamente convertimos esta interpolación a no lineal:

$$m_3 = \left(\frac{2\,(m_2 - t)}{d}\right)^9 \cdot \frac{d}{2} + t \tag{4.1}$$

Donde d es la distancia entre las notas de la escala que conforman los extremos del intervalo en el que se halla m_2 y t (nota *target*) es la nota más cercana de la escala a m_2. Ahora sí, la nota que hacemos sonar para la mano derecha es m_3.

Esta función es la representada en la figura 4.3, en la que se observa la interpolación no lineal, la curva suave entre notas, el ancho de cada escalón en el que la nota suena afinada y cómo distintas escalas tienen escalones de distinta altura.

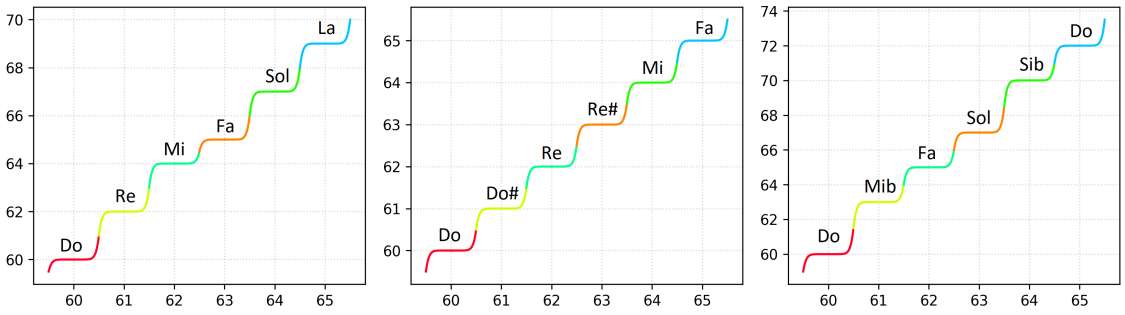


Figura 4.3: Resultado de la ecuación (4.1), que convierte la nota midi calculada a partir del desplazamiento de la mano (desplazamiento natural hacia la derecha$/10 + 60$) a una nota de la escala con una interpolación no lineal. El eje X es m_2 y el eje Y es m_3. De izquierda a derecha son las escalas: Do mayor, Do cromática y Do pentatónica menor. Cada cambio de color es un cambio de nota *target*.

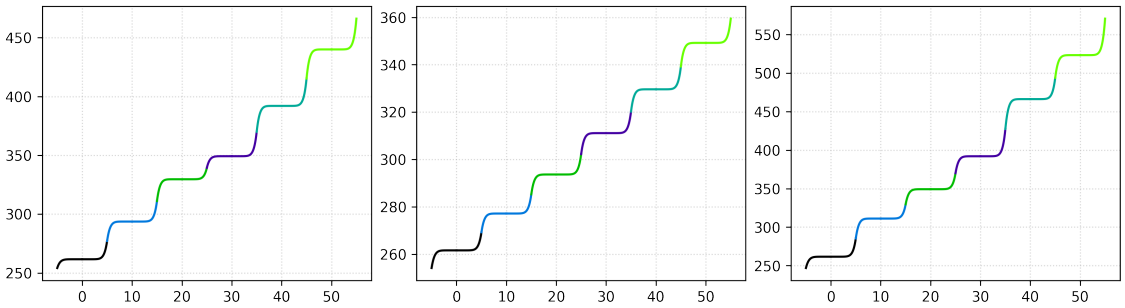


Figura 4.4: Variante de la figura 4.3 pero representando todo el proceso, es decir, el eje X son los centímetros a la derecha que se ha desplazado la mano con respecto a la cabeza, y el eje Y la frecuencia que suena. Son las misma escalas que en la figura 4.3 y por tanto las mismas notas.

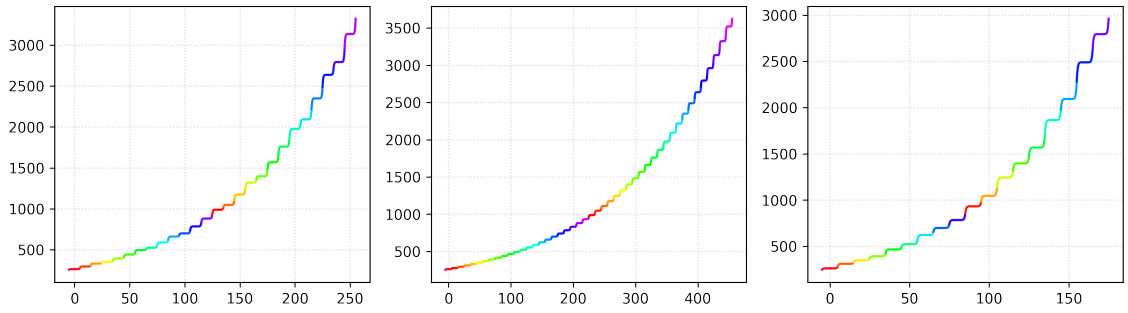


Figura 4.5: Si se continuara la gráfica 4.4 a lo largo de 4 octavas obtenemos esta figura, en la que se observa claramente el crecimiento exponencial de las frecuencias de las notas, correspondiendose con el crecimiento exponencial de las frecuencias de las notas del piano.

- La coordenada X (frontal horizontal) relativa de la mano izquierda sobre el tablero de acordes que habíamos llamado A_v se traduce a frecuencias de apertura de filtro mediante la siguiente formula:

$$\text{freq} = A_v \cdot 5000$$

De manera que el valor máximo que puede alcanzar es 5 kHz.

4.2. Diseño técnico

Presentamos a continuación algunos diagramas de flujo y de clases que simbolizan la estructura de clases y de datos que componen Loosy. Como hemos mencionado anteriormente Loosy tiene dos piezas clave: *SynthController* y *LoosySynth*. El primero actúa como controlador del segundo, que es el sintetizador propiamente dicho. Dicho control se basa en los datos obtenidos de las manos como hemos explicado en el apartado anterior, y se realiza a través de tres tipos de mensajes OSC: de inicialización, de control y de posición.

Los mensajes de inicialización sirven para que el controlador le explique al sintetizador cual es el conjunto de escalas y acordes que va a estar utilizando. Los de control sirven para comunicar un cambio de acorde o de escala, y los de posición sirven para transmitir datos de la posición de las manos que tienen que estar actualizando variables del sintetizador constantemente. A su vez, al estar el sintetizador escrito en ChucK funciona con una máquina virtual que reproduce el sonido. Toda esta estructura se puede observar en la figura 4.6.

4.2.1. *SynthController*

Para decidir qué escalas y acordes utilizar, y cuáles serán los que tendrá el usuario por defecto en el tablero de acordes y el menú de la mano derecha, el controlador lee dos archivos llamados *Scales.xml*

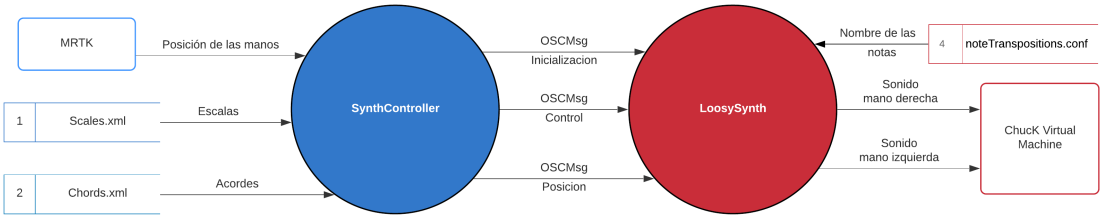


Figura 4.6: Diagrama de flujo de nivel 0. Este es un diagrama básico que expresa la estructura general del proyecto, con los dos bloques fundamentales, sus entradas y sus salidas

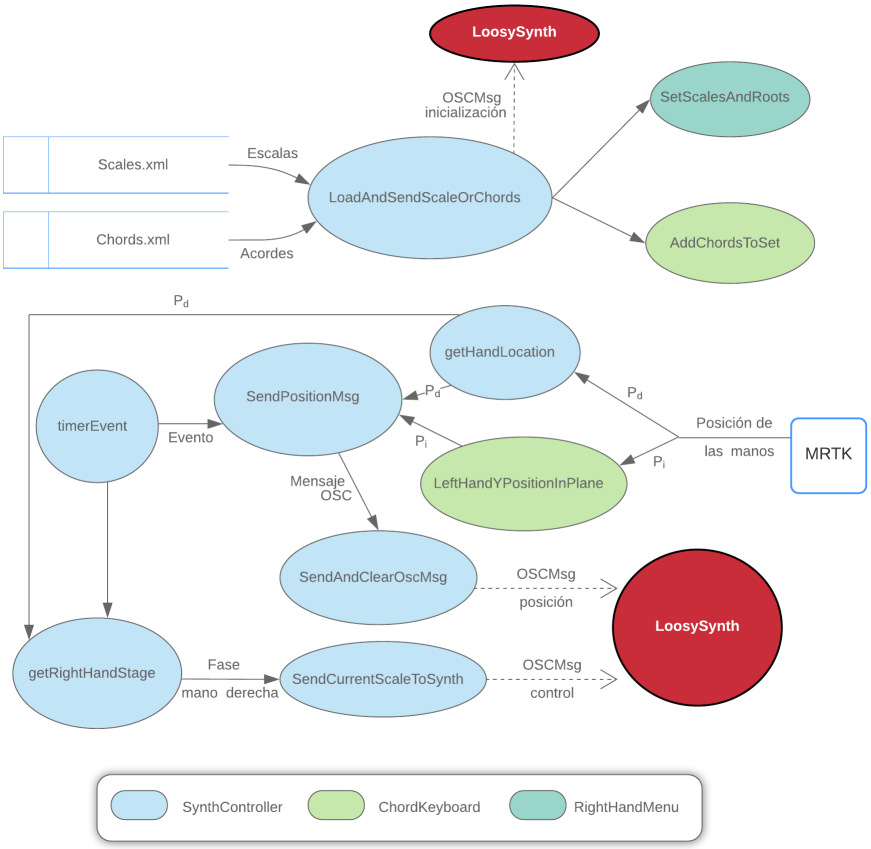


Figura 4.7: Diagrama de flujo de nivel 1 con las funciones de inicialización, cambio de escala y envío de paquetes de posición por parte del controlador.

y *Chords.xml*, que se detallan en la sección 5.2.1. Este proceso se observa en la mitad superior de la figura 4.7.

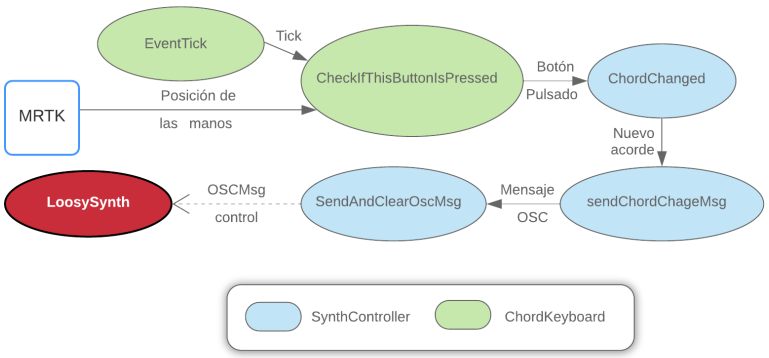


Figura 4.8: Diagrama de flujo con las funciones de cambio de acorde por parte del controlador.

Una vez completado este proceso de inicialización se crea un evento asociado a un *timer* de Unreal Engine, que nos permite ejecutar el evento cada 0,02 segundos. Cada vez que esto ocurre, se desencadena el flujo de la mitad inferior de la figura 4.7: se localiza la posición de las manos y se calculan los datos necesarios que se envían al sintetizador en un mensaje de posición, se comprueba si la mano ha rotado lo suficiente como para cambiar de escala y si es así se envía otro paquete OSC de control comunicando el cambio.

La figura 4.8 ilustra el flujo de datos que ocurre en cada tick (unidad mínima de tiempo en Unreal Engine), momento en el que se comprueban todos los botones del tablero para ver si la mano izquierda está pulsando alguno de ellos. Que lo esté pulsando quiere decir que la distancia sea mínima, y si es así, se llama a la función *chordChanged* de *SynthController* y si el acorde pulsado es distinto del anterior acorde que estaba sonando se envía al sintetizador en un mensaje de control.

La estructura de objetos del controlador se puede ver en la figura 4.9, que es un diagrama de clases de este módulo. Aquí se aclara un poco qué forma tiene exactamente el menú de la mano derecha que hemos mencionado en varias ocasiones: funciona con tres *sliders*, que el usuario puede arrastrar de forma no contínua por las distintas escalas disponibles (las especificadas en scales.xml). Cada slider selecciona una escala para cada una de las tres fases de rotación en las que se puede hallar la mano derecha. Todo esto se controla y almacena desde el objeto *RightHandMenu*.

A su vez, *ChordKeyboard* se encarga del tablero de acordes, además de contenerlos se encarga de comprobar si se está pulsando algún botón, y si se da el caso de comunicárselo a *SynthController* que actúa en esta estructura como objeto padre.

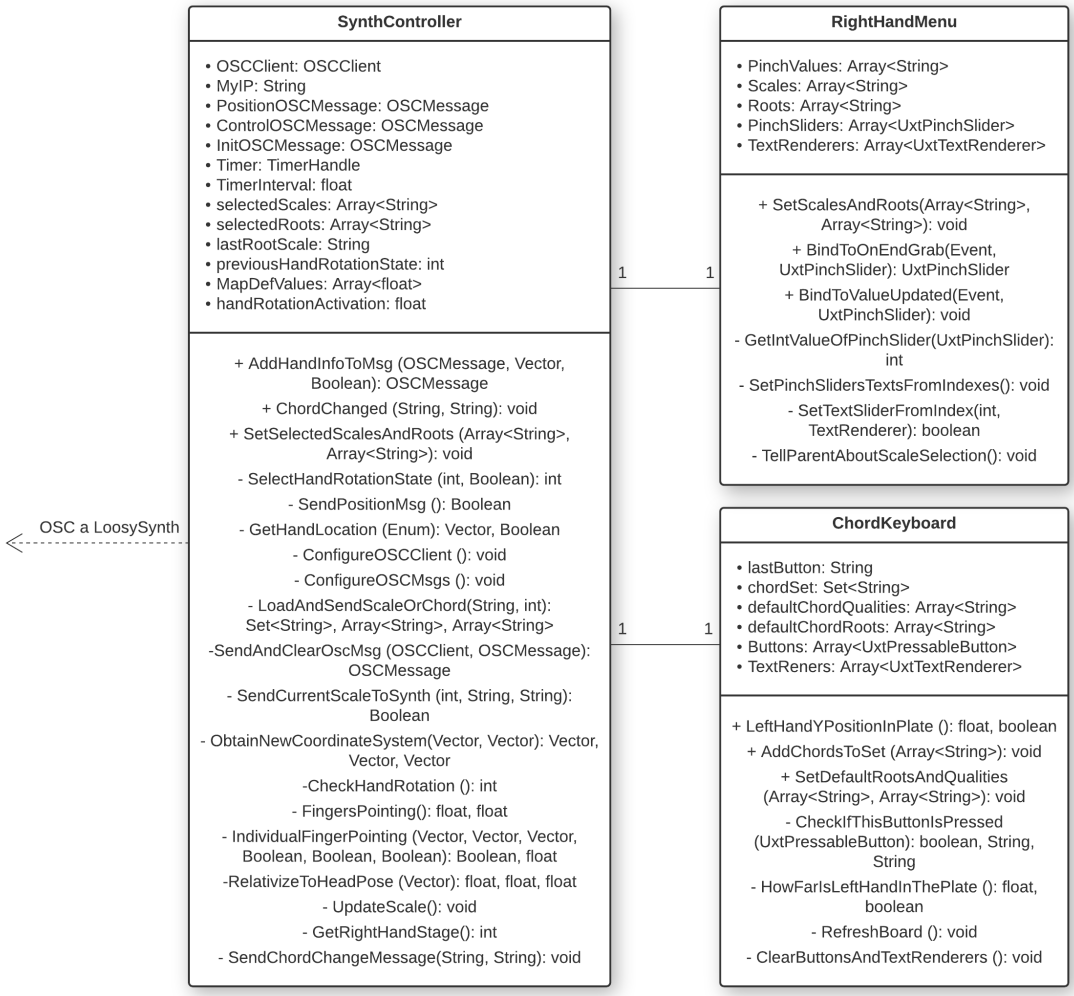


Figura 4.9: Diagrama de clases del controlador

4.2.2. *LoosySynth*

El módulo sintetizador, creados e inicializados sus objetos y estructuras que aparecen en la figura 4.10, se queda a la espera de recibir mensajes. De por sí, el sintetizador comienza vacío, sin ninguna escala ni acorde.

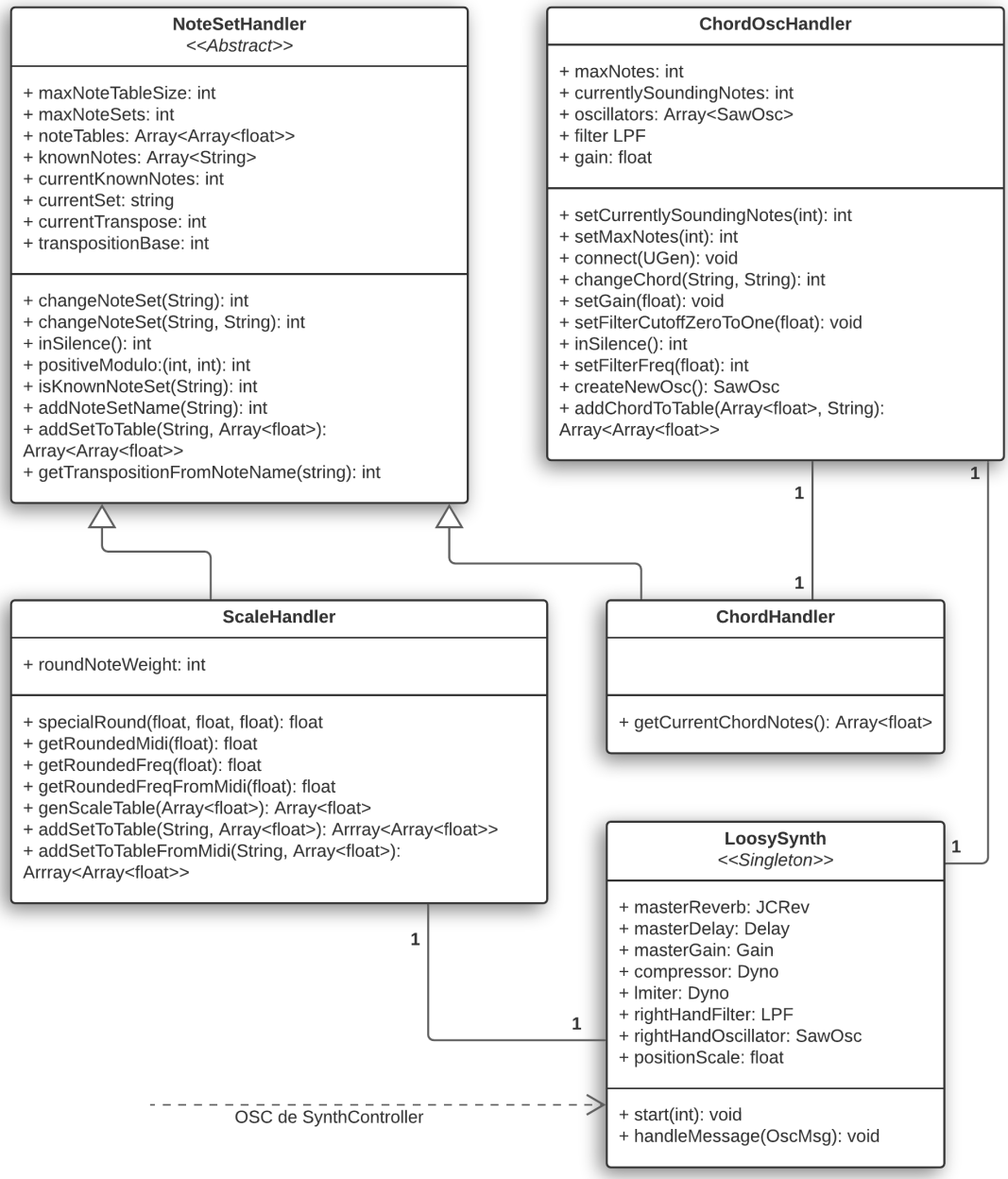


Figura 4.10: Diagrama de clases del sintetizador

En éste diagrama de clases se aprecia el concepto que se ha explorado a lo largo de este documento de que las escalas y acordes musicales son en realidad conjuntos de notas. En el diseño se ha utilizado esto para crear una estructura de herencias, en la que el objeto padre es un conjunto de

notas (*NoteSetHandler*) y las escalas y acordes son casos particulares de conjuntos de notas.

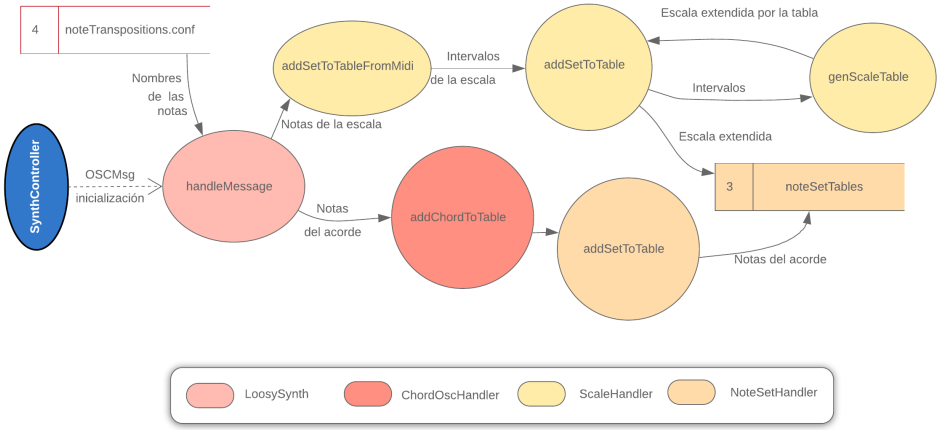


Figura 4.11: Diagrama de flujo con las funciones de inicialización por parte del sintetizador.

ChordOscHandler es el módulo encargado de gestionar los osciladores que harán sonar las notas de los acordes, la ganancia de cada uno, y el conjunto de acordes (*ChordHandler*). Por su parte, el objeto singleton *LoosySynth* actúa como objeto padre de los demás y es el que recibe y gestiona todos los paquetes que le llegan. Las funcionalidades mas importantes se han ilustrado con diagramas de flujo como la figura 4.11, que explora lo que ocurre cuando llegan mensajes de inicialización.

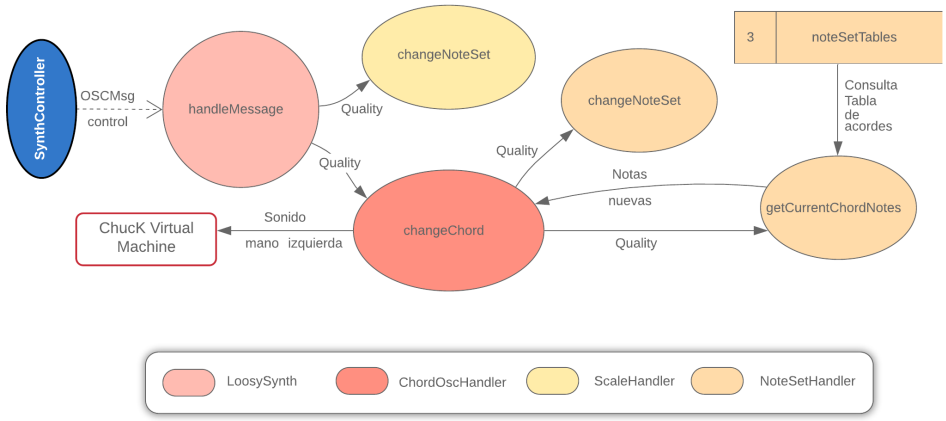


Figura 4.12: Diagrama de flujo con las funciones de control por parte del sintetizador. Notar que *quality* se refiere a la calidad del acorde o escala: si es mayor, menor etc.

En esencia, se obtienen los datos necesarios de dichos mensajes, se transfieren a *ChordOscHandler* y *ScaleHandler* para que almacenen los conjuntos de notas que va a manejar el controlador. Aparece además un fichero llamado “noteTranspositions.conf” que sirve para que el sintetizador sepa los

nombres de las notas en lenguaje natural. Se detallará en el siguiente capítulo.

La figura 4.12 pretende ilustrar el flujo de datos que ocurre cuando llega un paquete de control. En caso de ser un cambio de escala es sencillo, porque simplemente se actualiza una variable. En el caso de un cambio de acorde hay un grado añadido de complejidad, porque hay que recalcular cuántas notas van a estar sonando a la vez, reajustar las ganancias para que no suba o baje de volumen en exceso, obtener las notas en cuestión y asignar cada una a un oscilador distinto.

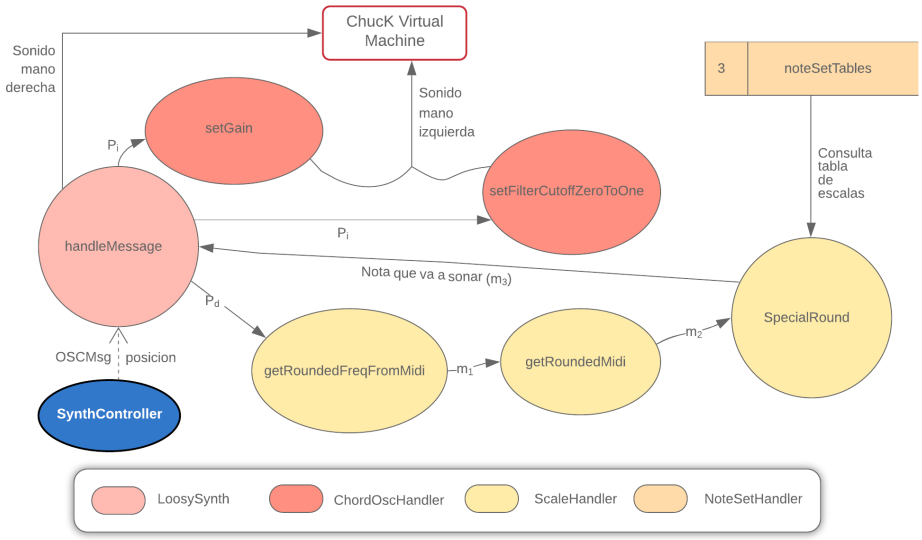


Figura 4.13: Diagrama de flujo con las funciones asociadas a los mensajes de posición por parte del sintetizador.

Para terminar tenemos en la figura 4.13 donde se ilustra el movimiento de datos cuando el sintetizador recibe un paquete de posición. A saber, se reajustan las ganancias y frecuencias de corte de los filtros de la melodía y los acordes como se ha explicado en la sección anterior, y se ajustan también la nota que suena en la melodía.

4.3. Planificación del proyecto

El proyecto comienza cuando el alumno se pone en contacto con el tutor al comenzar el verano con la idea de crear algún tipo de aplicación musical por determinar, a lo que el tutor propone desarrollar una aplicación en realidad mixta aprovechando que la universidad le había concedido un proyecto de investigación con unas gafas Hololens 2. Por dificultades burocráticas no se han conseguido las gafas, pero el proyecto ha seguido adelante.

Se comienza con una lluvia de ideas y un estudio de las herramientas disponibles para desarrollar

Figura 4.14: Diagrama de Gantt

el proyecto, para seguidamente tomar decisiones preliminares en cuanto a qué herramientas aprender a utilizar, y qué caminos tomar. Esta actividad empieza en verano, antes del comienzo del curso y se alarga hasta la mitad del primer cuatrimestre. Se ha estimado que esta actividad conllevó unas 25 horas.

Con el comienzo de las clases el alumno se apunta a un curso online de Unreal Engine, a ver tutoriales y a continuar estudiando qué librerías se pueden utilizar para hacer sonar el proyecto que se estaba diseñando. Es entonces cuando aparece la idea de utilizar ChucK, lenguaje por el que el alumno ya había desarrollado algo de interés. Con esta idea comienza otro curso online para aprender este lenguaje. Se estima que el aprendizaje de Unreal Engine constituyó unas 30 horas, y el aprendizaje de ChucK 25 horas. Unreal Engine fue una tarea más ardua por el nuevo paradigma de programación en Blueprints, y porque el entorno de Unreal Engine puede ser muy restrictivo si no se está acostumbrado. Además el alumno rara vez había tenido contacto con herramientas de diseño 3D. ChucK tampoco fué fácil y conllevó un riesgo tomar la decisión de utilizarlo, ya que al principio no estaba claro si el sistema de mensajes por OSC iba a resultar funcional o no.

Paralelamente a esta actividad se profundiza en el diseño, y las ideas pasan a convertirse en experimentos que sirven para comprobar su viabilidad y su grado de dificultad. Poco a poco se van logrando pequeños éxitos, el primero de ellos ocurre en octubre cuando se logra que moviendo la mano en el eje horizontal se produzca un sonido, o en enero cuando se logra que lleguen los primeros

mensajes a ChucK satisfactoriamente. Se estima que esta fase de diseño, que estuvo llena de prueba y error dura hasta mediados de f ebrero y constituyó unas 50 horas.

Con el comienzo del segundo cuatrimestre se da el pistoletazo de salida al desarrollo de la aplicación en el estado en el que se encuentra ahora, comenzando por mejorar el sistema de paquetes (unas 40 horas) y el sistema de síntesis de sonido (unas 45 horas). Cuando esto empieza a funcionar se comienza con la implementación del sistema de escalas, que también tuvo prueba y error, y que se valió mucho del sorprendente descubrimiento de que ChucK, con lo básico que es para algunas cosas tiene arrays asociativos, que son muy similares a diccionarios. Esto solucionó muchos problemas. Para el desarrollo de esta funcionalidad se estima una dedicación de 45 horas.

Por último solo faltaba hacer sonar acordes. Con la experiencia que ya tenía el alumno con las herramientas utilizadas, y las estructuras que ya había creado fue bastante más sencillo que enfrentarse al problema de las escalas. El último paso fue la redacción de esta memoria. A cada una de estas actividades se le han destinado unas 35 horas.

Estos cálculos aproximados suman un total de unas 330 horas aproximadamente. El total de esta actividad de ilustra en el diagrama de Gantt de la figura 4.14.

5

Implementación y experimentación

Ampliamos en este capítulo detalles de la implementación que han quedado sin explicar en el capítulo de diseño y que se han considerado importantes, posiblemente entrando en tecnicismos en contraste con la abstracción de la que gozaba el capítulo anterior. Además demostramos el funcionamiento de la aplicación y exponemos algunos datos generales del proyecto.

5.1. Librerías y herramientas utilizadas

Se exponen aquí las librerías y herramientas que han hecho posible el proyecto, y qué hemos utilizado exactamente de ellas.

- **Unreal Engine 4.26**: motor de videojuegos que ha sido la base de todo el desarrollo del controlador. En un inicio se eligió este frente a Unity basado en C#, que era la otra opción buena, porque Unreal está basado en C++, lo cual permite que se programe en dicho lenguaje dentro del motor.
  Esto es beneficioso para nuestro proyecto porque a priori C++ tiene más capacidades de programación musical y una enorme cantidad de aplicaciones musicales se desarrollan en C++. Esto también podría haber sido útil si se hubiera querido importar alguna librería musical en C++, pero no ha sido necesario. Además, no siempre estuvo claro que la idea de hacer dos módulos separados comunicados por OSC fuera a funcionar tan bien.
- **Plugins de Unreal Engine**: Además se han utilizado plugins para extender la funcionalidad del motor:
  - **Mixed Reality Toolkit** y **OpenXR**: son las dos herramientas creadas para desarrollar en realidad mixta, y en particular para Hololens 2. Gracias a ellas la funcionalidad de que se pueda agarrar el tablero de acordes, para después soltarlo y que se pegue a una superficie, fue tan sencillo como añadir un componente. De estas librerías también se han sacado los *pinchSliders* del menú de la mano derecha, la funcionalidad de que aparezca el menú al mostrar la palma de la mano a las gafas y parte de la funcionalidad de los botones del tablero de la mano izquierda, así como la posibilidade de simular la aplicación.
  - **BlueprintUML:** Parser de UML que se ha utilizado para leer los ficheros de configuración UML.
  - **OSC**: Extensión para generar y enviar los mensajes OSC.
- **ChucK**: se eligió este lenguaje, que hemos introducido en el capítulo 2, para facilitar la programación musical y tener a mano la potencia de este lenguaje. No existe prácticamente nada igual, así que la alternativa era programar el sintetizador en C++ desde cero.
- **MiniAudicle**: es un pequeño IDE para desarrollar en ChucK, cuenta con un editor de texto que colorea el código, lanzamiento de la máquina virtual en la que ejecutar el código y una consola para recibir feedback.

- **LucidChart** y **Overleaf**: son las herramentas utilizadas para la creación de este documento. La primera ha sido la mejor web para hacer gráficos on-line que he encontrado, y la segunda es un potente editor de TEX online.
- **Microsoft Teams**: herramienta que se ha utilizado para comunicarse con el tutor.

5.2. Detalles de la implementación que no han quedado explicados anteriormente

En el capítulo 4 no ha quedado explicado todo. Aunque parte del código está en los apéndices, ampliamos a continuación algunos detalles que se han considerado importantes.

5.2.1. Ficheros de configuración

¿De donde salen las escalas y los acordes que llevamos mencionando durante todo el documento? De estos ficheros de configuración. Los dos más importantes son *Scales.xml* y *Chords.XML*. Los dos tienen la misma estructura y de hecho son leídos por la misma función. Ambos se basan en la idea de que tanto las escalas como los acordes son conjuntos de notas. Por eso, bajo el nodo raíz llamado *noteSetsFile*, que significa "fichero de conjuntos de notas", hay una serie de nodos *noteSet* (conjunto de notas) que tienen un nombre (*name*) y unas notas (*notes*). Las notas se expresan en numeración midi, donde el 60 es el Do central, 61 Do sostenido, 62 Re, 64 Re sostendido, 65 Mi y así sucesivamente. Dichas notas están separadas por comas y son floats, permitiendo potencialmente la presencia de escalas microtonales.

- En el caso de Scales.xml, el nodo *notes* representa las notas que conforman la escala incluyendo la primera y la última, que coincide con la primera de la siguiente repetición de la escala. Es decir, la escala de Do mayor se compone de las notas Do, Re, Mi, Fa, Sol, La, Si y Do. Este último Do técnicamente forma parte de la octava siguiente, pero aquí se utiliza para marcar dónde empieza la siguiente repetición de la escala.
  Esto puede ser útil cuando la escala no se repite en la octava. Esto ocurre por ejemplo con la escala cromática, que es la que contiene todas las teclas del piano. En este caso con escribir una nota y la siguiente (por ejemplo, Do y Do sostenido) ya es suficiente para inferir las demás repitiendo este patrón, subiendo y bajando de semitono en semitono. En el caso de la escala mayor, el patrón de intervalos entre notas es tono, tono, semitono, tono, tono, tono, semitono. Se observa que son siete intervalos, de manera que hasta que no se han subido ocho notas no se repite el patrón y por eso hay que escribir la octava entera. Se asume que, dado que las escalas son independientes de la nota raíz y lo importante son intervalos entre las notas, las escalas estarán expresadas en Do.
  Además hay otro nodo especial bajo el nodo raíz llamado *defs* que expresa qué escalas estarán entre las disponibles en el menú de la mano derecha. Cada escala se codifica con un nodo *defaultSet* que contiene una nota raíz (*root*) y un nombre de escala (*quality*). De esta manera por ejemplo se puede elegir Do mayor, siendo Do la raíz y mayor la *quality*.
- En Chords.xml, *notes* son las notas que conforman el acorde simple y llanamente, sin transportarse a otras octavas ni nada similar. El nodo que acabamos de introducir llamado *defs* incluye los acordes que aparecerán en el tablero de acordes.

Código 5.1: Ejemplo del archivo de configuración Chords.xml del módulo controlador. Contiene dos acordes *major* y *minor*. Para cada imp tiene las notas correspondientes. Además la segunda mitad serán los acordes qeu aparecerán en el tablero de acordes: *Cmajor* y *Dminor*, que son Do mayor y Re menor.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <noteSetsFile>
3
4      <noteSet>
5        <name>major</name>
6        <notes>48,52,55</notes>
7      </noteSet>
8
9      <noteSet>
10       <name>minor</name>
11       <notes>48,51,55</notes>
12     </noteSet>
13
14     <defs>
15         <defaultSet>
16             <root>
17                 C
18             </root>
19             <quality>
20                 major
21             </quality>
22         </defaultSet>
23
24         <defaultSet>
25             <root>
26                 D
27             </root>
28             <quality>
29                 minor
30             </quality>
31         </defaultSet>
32
33     </defs>
34 </noteSetsFile>
```

Código 5.2: Ejemplo del archivo de configuración Scales.xml del módulo controlador. Cuenta con dos escalas: mayor y cromática. Y con tres escalas para el menú: Do mayor, Do# Blues y Si cromática.

```
<?xml version="1.0" encoding="UTF-8"?>
<noteSetsFile>

    <noteSet>
      <name>Major</name>
      <notes>60,62,64,65,67,69,71,72</notes>
    </noteSet>

    <noteSet>
        <name>Chromatic</name>
        <notes>60,61</notes>
    </noteSet>

    <defs>
        <defaultSet>
            <root>
                C
            </root>
            <quality>
                Major
            </quality>
        </defaultSet>

        <defaultSet>
            <root>
                C#
            </root>
            <quality>
                Blues
            </quality>
        </defaultSet>

        <defaultSet>
            <root>
                B
            </root>
            <quality>
                Chromatic
            </quality>
        </defaultSet>

    </defs>
</noteSetsFile>
```

Estos dos ficheros que hemos introducido están pensados para que puedan ser editados por un usuario que sepa XML y conozca la estructura que se acaba de exponer. Son leídos por el controlador que acto seguido le comunica al módulo sintetizador la información del fichero, de manera que si en un futuro se desarrolla la funcionalidad de tener varios controladores conectados al mismo sintetizador cada controlador puede proponer sus propias escalas y acordes, y el sintetizador las conocería todas.

Pero, cuando el controlador le comunica al sintetizador que tiene que sonar en Fa mayor, ¿cómo sabe trasladar lo que ha aprendido en Do a Fa? Transportándolo. Si conozco la escala de Do mayor, la escala de Fa mayor será igual pero sumando dos tonos y medio a todas las notas. Cuánto hay que transportar para cada nombre de nota es una información que bien podría estar incrustada en el código porque el sistema occidental de notación musical no parece ir a cambiar pronto, pero por si acaso se ha sacado a un fichero llamado *noteTranspositions.conf*, que tiene en la primera fila el string “noteName transposition”, y en las filas siguientes el nombre de la nota y su transposición asociada separados por un espacio. La primera línea pretende ser indicativa de la estructura del documento, pero realmente no está pensado para ser editado.

Código 5.3: Ejemplo del archivo de configuración noteTranspositions.conf del módulo sintetizador

```
1 noteName␣Transposition␣
2 A␣-3␣
3 B␣-1␣
4 C␣0␣
5 D␣2␣
6 E␣4␣
7 F␣5␣
8 G␣7␣
```

5.2.2. Estructuras de almacenamiento y generación de escalas y acordes musicales

La información que ha leído el controlador de Scales.xml y Chords.xml se pasa por OSC al módulo sintetizador y éste se encarga de procesarlo y almacenarlo.

En el caso de las escalas, el objeto ScaleHandler mide la distancia entre las notas de la escala que acaba de recibir e infiere la estructura de intervalos que conforma la escala. Lo siguiente que hace es extrapolar esa estructura por una tabla de 127 valores, tantos como notas midi, y guarda dicha tabla en un diccionario con clave el nombre de la escala. El valor 60, cercano al centro de la tabla, es el punto de partida y siempre aparecerá el valor 60 en este campo. De esta manera, la tabla de la escala mayor tendrá las notas midi Do, Re, Mi, Fa, Sol, La, Si, Do, Re, Mi... hacia arriba, y hacia abajo, Do, Si, La, Sol... Notar que incluso si la escala estaba codificada en otro tono en el fichero Scales.xml, se guarda en Do para poder transponer luego correctamente.

Con esto es fácil mirar cuantas decenas de centímetros se ha movido la mano hacia la derecha, buscar el valor entero más cercano y mirar en la tabla partiendo del valor 60. Con estos datos ya se pueden aplicar las fórmulas de la sección 4.1.2 y producir el sonido correspondiente.

En cuanto a los acordes, el almacenamiento es similar, con la salvedad de que no se extrapola al resto de la tabla, simplemente se guarda en la tabla las notas que correspondan, y para hacerlas sonar simplemente se transponen y se asignan a osciladores distintos.

5.2.3. Estructura de mensajes OSC

No se ha aclarado todavía la estructura de mensajes entre controlador y sintetizador. Como se mencionó en la sección 2.5.1 los mensajes de OSC son bastante flexibles, ya que el campo del contenido consiste en una lista variable de campos de distintos tipos. En nuestra estructura de mensajes se aprovecha esta flexibilidad. El primer campo será un entero que indicará el tipo de mensaje:

- Mensajes de posición (tipo 0): son mensajes en los que se codifica la posición de las manos, en particular se envía:
  - Un entero de validez para la mano izquierda que toma el valor 1 si la mano está localizada por las gafas y 0 si no. Si vale 0 será como si el usuario hubiera retirado la mano para dejar de producir sonido.
  - A_v que recordemos que es un valor entre 0 y 1 que mide cómo de adentrada está la mano izquierda en el plano extendido del tablero de acordes.
  - Otro entero de validez para la mano derecha, con la misma funcionalidad.
  - Una variante de P_{d_x} en el sistema de coordenadas respecto a G que recordemos que es la proyección del vector que une la mano derecha y las gafas sobre el eje horizontal frontal. Decimos una variante, porque el valor que se envía en el mensaje vale 0 P_{d_x} es menor que 30 cm, vale 1 cuando P_{d_x} es mayor que 35 cm y se interpola linealmente los valores intermedios. Este valor controla directamente la amplitud del oscilador de la mano derecha.
  - P_{d_y} en el sistema con origen en G que recordemos que es el desplazamiento lateral con respecto a la cabeza en centímetros. Este valor controla la nota que está sonando.
  - P_{d_z} en el sistema G que es la altura a la que está la mano derecha con respecto a la cabeza.
- Mensajes de control (tipo 1): Se encargan por ahora de los cambios de acorde y de escala. Después del entero de tipo, llevan un entero que indica si estamos ante un cambio de escala (valor 0) o un cambio de acorde (valor 1), seguido de dos strings, que contienen la nota raíz y el nombre de la escala o el acorde. Por ejemplo cambiar la escala de do (que en cifrado americano se expresa como C) mayor (en inglés major) contendría en sus campos: 1, 0, “C”, “major”.
- Mensajes de inicialización (tipo 2): Sirven para comunicar al sintetizador la presencia de una escala o un acorde nuevos. Llevan primero un entero para expresar si es una escala (valor 0) o un acorde (valor 2, el valor 1 queda reservado para funcionalidad añadida en el futuro) como los mensajes de control, seguido de un string con el nombre de la escala o el acorde en cuestión, seguido de tantos enteros como notas sean necesarias. Dichas notas están en notación midi.

5.3. Estructura de osciladores, filtros y efectos en ChucK.

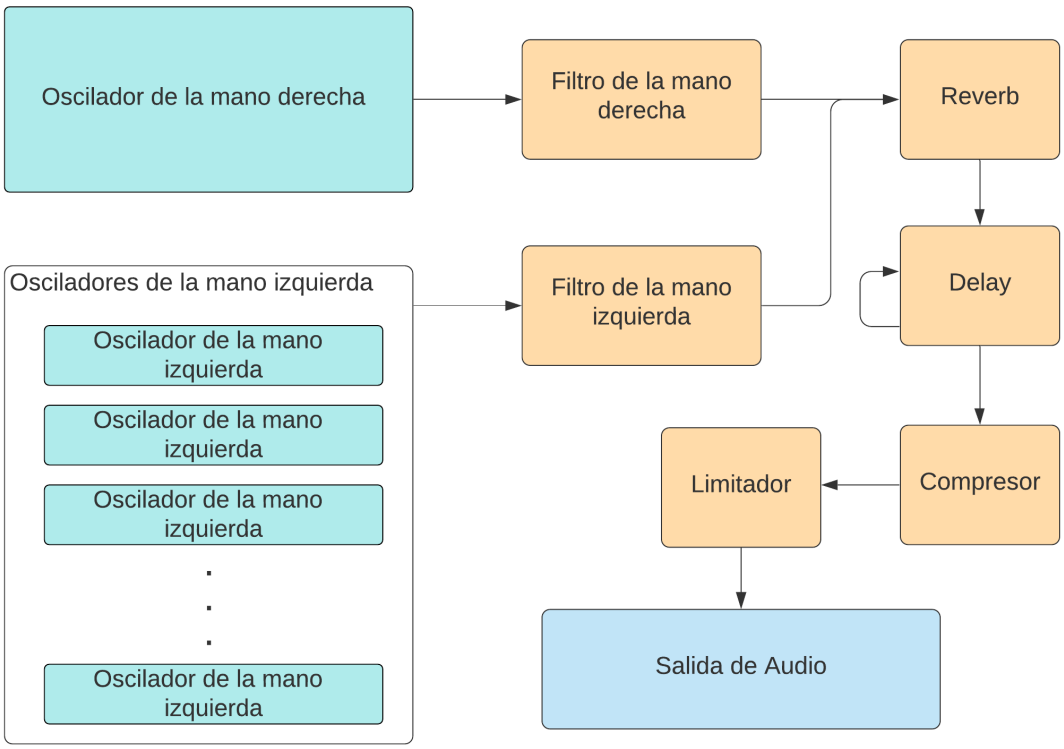


Figura 5.1: Diagrama del flujo del sonido dentro del sintetizador.

Veamos como están interconectados los distintos elementos que componen el sintetizador (ver la sección de notación y aclaraciones previas). Por orden de conexión en la cadena, tenemos para la mano derecha:

- Un oscilador de diente de sierra (SawOsc), que se dice que tiene tonos similares a los instrumentos de cuerda. Realmente es un oscilador en el que la onda es una recta que desciende del 1 al -1. Cuando vuelve a empezar, vuelve a empezar desde el 1, dando una imagen en conjunto de diente de sierra.
- Un filtro paso bajo (LPF) que permite al usuario contorlar el brillo del sonido, que se percibe como intensdidad del mismo, al subir o bajar la mano.
- Una reverb, que da efecto de reverberación producida por un espacio, en este caso bastante grande como una iglesia. Esto aporta naturalidad al sonido, lo embellece y lo suaviza.
- Un efecto delay, que los humanos estamos acostumbrados a percibirlo como eco. Consiste en la repetición del sonido de entrada con un cierto retraso. Se retroalimenta para que el eco continúe.
- Un compresor, para que si el sonido generado excede la amplitud 0.5, se empiece a comprimir la amplitud para que nunca superemos 1.
- Un limitador, para que si superamos la amplitud 0.9 a pesar del compresor, se comprima de manera más extrema. Esto es muy importante porque si en algún momento se acumula reverberación y delay junto con los acordes y la melodía sonando, y el conjunto llega a tener valores superiores en módulo a 1, el sonido se empezaría a distorsionar produciendo un efecto desagradable. Por eso, la mayoría de los elementos que acabamos de mencionar controlan el volumen teniendo ganancias por debajo del 0,5.

El caso de los acordes tiene algo más de complejidad:

- Para los acordes tenemos un array de osciladores de diente de sierra, y tenemos un entero almacenado que nos dice cuandas voces hay sonando. La ganancia de cada oscilador se ajusta automáticamente en base a dicho entero. Cuando está sonando un acorde de 3 voces, cada voz tendrá un tercio de la amplitud, y las voces restantes tienen ganancia 0.
- Los osciladores anteriores están todos conectados al mismo filtro paso bajo, que nuevamente controla el brillo del sonido. La frecuencia de corte de este filtro es precisamente lo que se controla cuando se mueve la mano izquierda hacia delante en el tablero de acordes.
- El filtro se conecta a la reverb mencionada antes, y a partir de aquí la señal se mezcla con la de la mano derecha.

Todo este flujo queda expresado en el diagrama de la figura 5.1.

5.4. Demostración de funcionamiento y pruebas realizadas

La aplicación se ha probado en distintas condiciones con el simulador, sobre todo la funcionalidad del sintetizador que era la que teníamos más posibilidades de probar. Estas pruebas han incluido introducir numerosas escalas, varias escalas con el mimso nombre, utilizar sostenidos y bemoles tras las notas, distintas transposiciones, distintas escalas y acordes etc. y ha pasado todas las pruebas.

El controlador se ha probado en la medida de lo posible y parece comportarse bien en todos los casos. El único momento en el que se comporta de forma inusual es cuando se agarra el teclado de acordes y se intenta pegar a una superficie virtual, que cuesta un poco y ocurren algunos *glitches*. Suponemos que esto se debe a que la librería está aún en versión beta, pero la ausencia de documentación nos impide profundizar más.

La figura 5.2 es lo primero que se ve desde las gafas cuando inicia el controlador: el tablero de acordes frente al usuario, con los acordes por defecto del archivo de configuración. A la distancia a la que está la mano en este momento en el simulador, la mano derecha ya está sonando, si se acerca al cuerpo dejará de sonar, si se mueve en el resto de direcciones tendrá el efecto esperado.

Cuando el usuario acerque cualquiera de las manos a los botones estos reaccionarán resaltándose como en la figura 5.3, pero únicamente cambiarán de acorde con la mano izquierda. Con cualquiera de las manos, agarrando, aunque sea desde lejos, el teclado por una parte en la que no haya botones, podrá moverlo con libertad y arrastrarlo, dentro de lo que permite la tecnología tal y como está ahora por las superficies del espacio en el que se encuentre el usuario como se muestra en la figura para dejarlo pegado a la superficie que elija, quedando como en la figura 5.4.

Una vez soltado, el tablero quedará pegado como en la figura 5.5

Al rotar la mano derecha hacia las gafas, y mostrar claramente la palma de la mano, aparecerá el ya famoso menú de la mano derecha, ilustrado en la figura 5.6. Con la mano izquierda se pueden

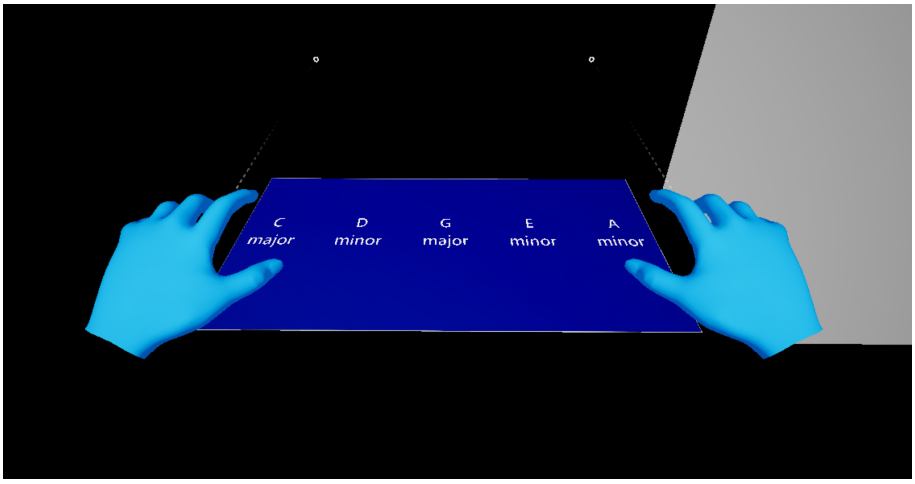


Figura 5.2: Vista básica desde la que empieza el usuario

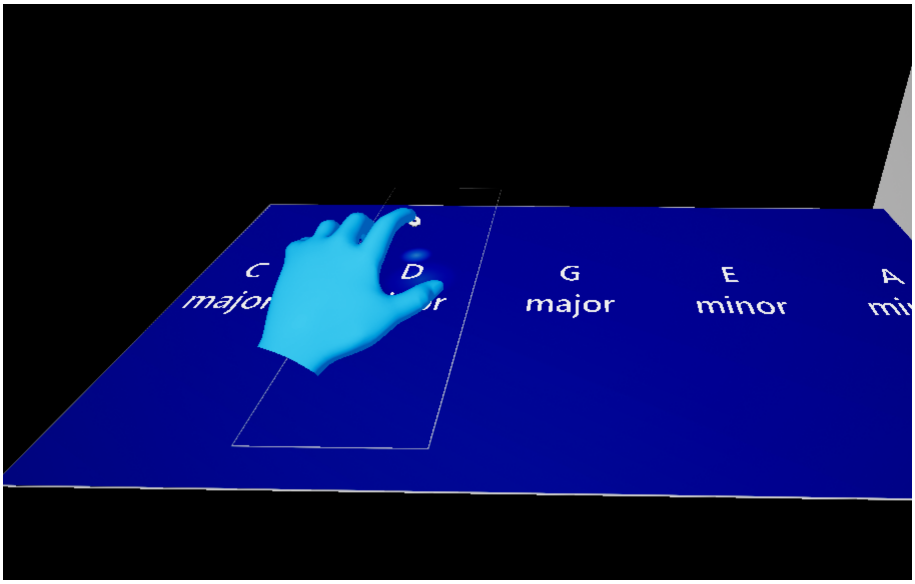


Figura 5.3: Cuando se acerca la mano a los botones, estos se resaltan.

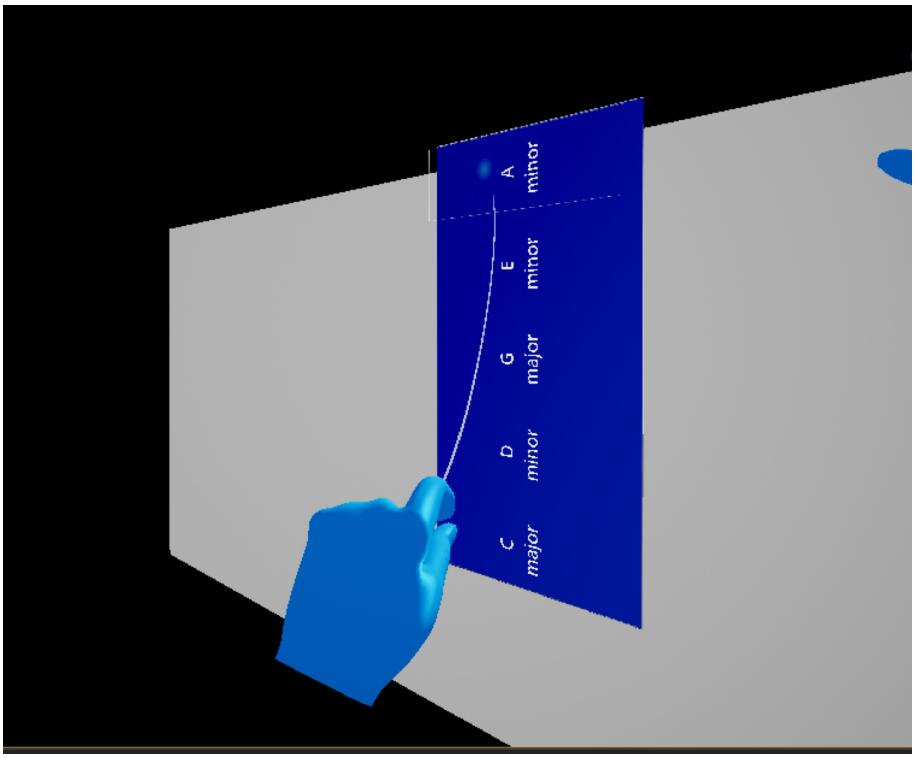


Figura 5.4: Agarrando tablero

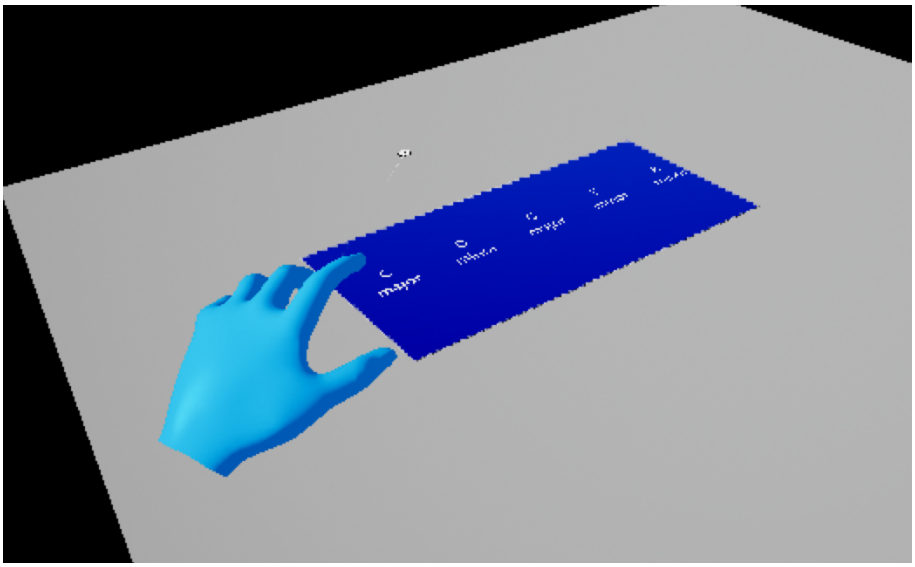


Figura 5.5: Tablero de acordes pegado a una superficie virtual

manipular los sliders como en la figura 5.7. Cambian el texto mientras se mueven y que cuando se sueltan tienen un efecto en el sonido si la mano está en el estado correspondiente.

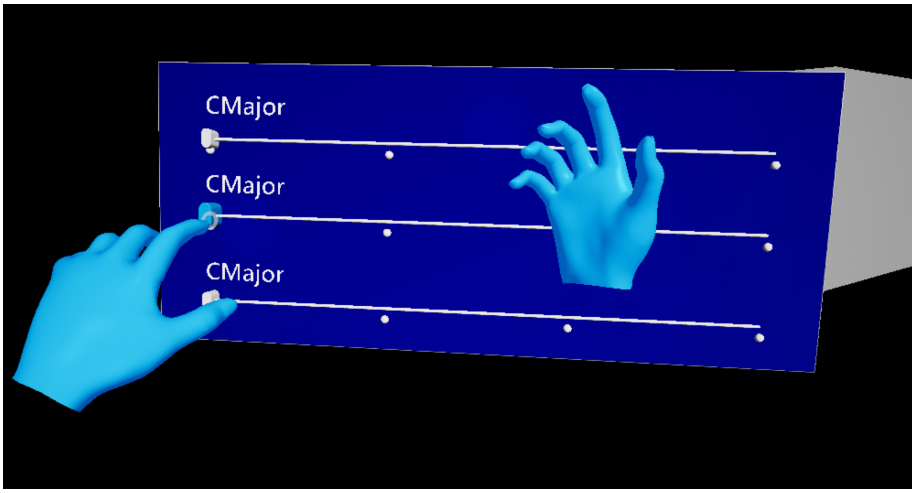


Figura 5.6: Menú mano derecha

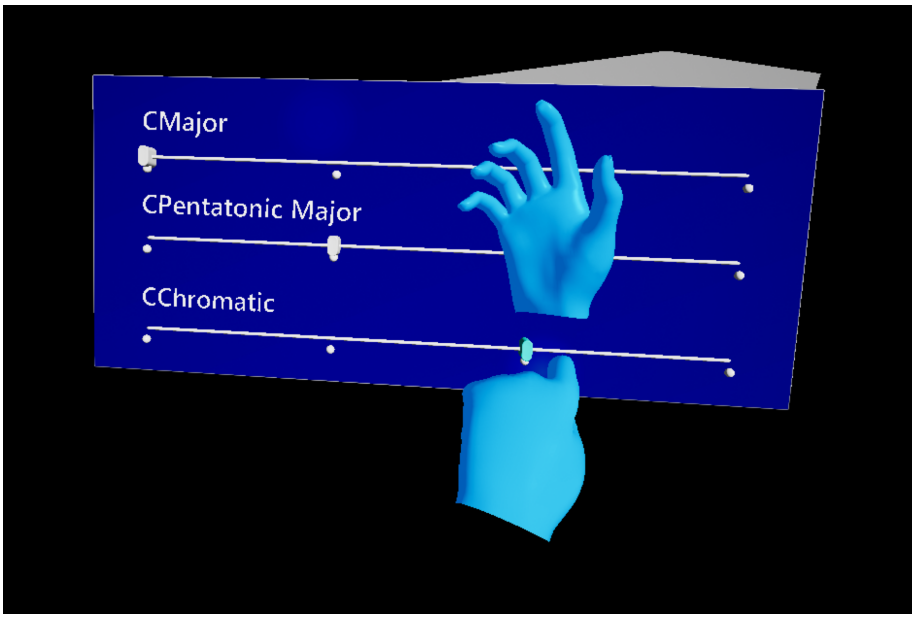


Figura 5.7: *Sliders* de la interfaz siendo agarrados

En el siguiente enlace se puede ver y sobre todo oír una demostración de las funciones principales del proyecto: https://youtu.be/SsYbiApdPus

5.5. Datos estadísticos del proyecto

Se exponen a continuación algunos datos generales del proyecto sobre los números de lineas, funciones y variables de cada uno de los módulos, para ilustrar la magnitud del proyecto en su desarrollo

actual.

- El submódulo del controlador *ChordKeyboard* tiene 105 nodos en sus blueprints (que se corresponden aproximadamente con el mismo número de líneas de código equivalentes), 6 variables y 7 funciones.
- El submódulo del controlador *RighHandMenu* tiene 114 nodos en sus blueprints, 5 variables y 7 funciones.
- El módulo padre del controlador *SynthController* tiene 277 nodos en sus blueprints, 13 variables y 19 funciones.
- El submódulo del sintetizador *NoteSetHanlder* tiene 212 líneas de código, 8 variables y 8 funciones.
- El submódulo del sintetizador *ChordOscHanlder* tiene 153 líneas de código, 5 variables y 10 funciones.
- El submódulo del sintetizador *ScaleHanlder* tiene 184 líneas de código, una variable y 7 funciones.
- El submódulo del sintetizador *ChordHandler* tiene 28 líneas de código, 0 variables y una función.
- El módulo principal del sintetizador *LoosySynth* tiene 171 líneas de código, 8 variables y dos funciones.
- Por último el fichero padre que llama a todo el resto del sintetizador llamado *main.ck* tiene 21 líneas de código.

Esto hace un total de 1265 líneas de código sin contar con ficheros de configuración, que actualmente tienen 158 líneas, de las que 149 son XML. Además hay 46 variables y 61 funciones.

6

Conclusiones y posibles desarrollos futuros

Tal y como está ahora, Loosy resulta un experimento interesante que permite vislumbrar el potencial artístico de las aplicaciones en realidad mixta. Pienso que tecnologías como las empleadas aquí abren la puerta, no solo a nuevas técnicas e interfaces para hacer música o dibujar en 3D, sino a formas nuevas de arte que ahora no somos capaces de imaginar. Nadie se imaginaba el cine como arte cuando los hermanos Lumiere proyectaron sus primeras películas, podríamos estar ante una revolución de magnitud similar.

Con un manual decente y una serie de conocimientos previos, Loosy podría incluso ser una experiencia divertida para algún informático con inquietudes musicales, pero con algo más de desarrollo podría convertirse en una aplicación con potencial comercial. Algunas de las derivas que podría tomar dicho desarrollo son las siguientes:

- Un conjunto extensivo de pruebas con las gafas y con distintas personas con el fin de testear a fondo toda la funcionalidad ya implementada y ajustar los parámetros para que resulte intuitivo y fácil para la mayor cantidad de personas y complexiones corporales posibles.
- Hacer desaparecer la necesidad de editar los ficheros de configuración, y pasar su funcionalidad a menús intuitivos para el usuario, tratando de que con unos mínimos conocimientos musicales se pueda utilizar y personalizar la aplicación con el mayor grado de libertad posible creando menús como los siguientes:
    - Un menú para crear escalas nuevas, que permita seleccionar qué notas de la octava forman parte de la escala.
    - Otro menú similar para crear acordes nuevos.
    - Otro más para personalizar un listado de escalas y tonalidades favoritas, que serán las que se podrán seleccionar con los *sliders* de la mano derecha. (Por ejemplo elegir que el listado contenga las escalas de Do mayor, Re menor y Mi pentatónica mayor).
- Introducir la funcionalidad de estirar y contraer los dedos de la mano derecha, posiblemente aprovechándola para asociar a cada gesto una función distinta. Una posibilidad sería elegir una armonización de la melodía que se esté interpretando (por terceras, quintas o sextas) o cambiar de instrumento (simular un saxofón, piano o violín).
- Crear funcionalidad para interpretar con percusiones en lugar de un tablero de acordes. Esto sería especialmente interesante porque las limitaciones generalizadas entre los instrumentos musicales en realidad virtual impiden una interpretación rítmica perfecta, pero con el tablero que se apoya sobre una superficie real podríamos estar más cerca de solucionar este problema.
- Actualmente los dos módulos fundamentales de la aplicación se comunican valiéndose del protocolo OSC, que en principio está pensado también para funcionar a través de internet. Esta decisión de diseño se tomó pensando

que en algún momento posterior a este trabajo los dos módulos (controlador y sintetizador) podrían estar corriendo en máquinas distintas, conectadas entre sí a través de una red común. Con un módulo de descubrimiento en cada extremo esto sería muy fácil de implementar tal y como está el diseño actualmente. Es más, el diseño está preparado para que se implemente la funcionalidad de que varios controladores se conecten al mismo sintetizador, pudiendo varias personas distintas con Hololens, posiblemente separadas entre sí físicamente, tocar música en un mismo dispositivo.

- Se podrían añadir además nuevos gestos, o incluso gestos personalizados con acciones personalizables también, pero este objetivo está más lejos que los anteriores. De la misma manera se podría mejorar la accesibilidad de la aplicación, integrando por ejemplo una opción para zurdos que intercambie las funciones actuales de las manos

Así pues, la aplicación tiene amplias posibilidades de desarrollo que además podrían no quedarse en mera especulación: hay planes de continuar con el proyecto e intentar una explotación comercial del mismo. Dichos planes incluyen reunir a un equipo más grande de desarrolladores e intentar completar poco a poco los puntos expuestos anteriormente. Si todo esto se cumple y sale bien, Loosy podría pasar de ilustrar las puertas de la interpretación musical en realidad mixta a abrir algunas de ellas. ¡Quién sabe!

Bibliografía

[Berry et al., 2003] Berry, R., Makino, M., Hikawa, N., and Suzuki, M. (2003). The augmented composer project: The music table. In *The Second IEEE and ACM International Symposium on Mixed and Augmented Reality, 2003. Proceedings.*, pages 338–339. IEEE.

[Bottino et al., 2017] Bottino, A. G., García, A. M., and Occhipinti, E. (2017). *Holomuseum: A prototype of interactive exhibition with Mixed Reality glasses HoloLens*. PhD thesis, Master Dissertation, Polytechnic university of Valencia.

[Chow et al., 2013] Chow, J., Feng, H., Amor, R., and Wünsche, B. C. (2013). Music education using augmented reality with a head mounted display. In *Proceedings of the Fourteenth Australasian User Interface Conference-Volume 139*, pages 73–79.

[Corrêa et al., 2009] Corrêa, A. G. D., Ficheman, I. K., do Nascimento, M., and de Deus Lopes, R. (2009). Computer assisted music therapy: A case study of an augmented reality musical system for children with cerebral palsy rehabilitation. In *2009 Ninth IEEE International Conference on Advanced Learning Technologies*, pages 218–220. IEEE.

[Freed and Schmeder, 2009] Freed, A. and Schmeder, A. (2009). Features and future of open sound control version 1.1 for nime. In *NIME*, volume 4, page 2009.

[Garon et al., 2016] Garon, M., Boulet, P.-O., Doiron, J.-P., Beaulieu, L., and Lalonde, J.-F. (2016). Real-time high resolution 3d data on the hololens. In *2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR-Adjunct)*, pages 189–191. IEEE.

[Halpern, 2019] Halpern, J. (2019). *Introduction to Unity*, pages 13–30. Apress, Berkeley, CA.

[Hanna et al., 2018] Hanna, M. G., Ahmed, I., Nine, J., Prajapati, S., and Pantanowitz, L. (2018). Augmented reality technology using microsoft hololens in anatomic pathology. *Archives of pathology & laboratory medicine*, 142(5):638–644.

[Howard and Angus, 2013] Howard, D. and Angus, J. (2013). *Acoustics and psychoacoustics*. Routledge.

[Huber, 2007] Huber, D. M. (2007). *The MIDI manual: a practical guide to MIDI in the project studio*. Taylor & Francis.

[Kalantari and Rauschnabel, 2018] Kalantari, M. and Rauschnabel, P. (2018). Exploring the early adopters of augmented reality smart glasses: The case of microsoft hololens. In *Augmented reality and virtual reality*, pages 229–245. Springer.

[Kapur et al., 2015] Kapur, A., Cook, P., Salazar, S., and Wang, G. (2015). *Programming for Musicians and Digital Artists*. Manning Publications Co.

[Karis and Games, 2013] Karis, B. and Games, E. (2013). Real shading in unreal engine 4. *Proc. Physically Based Shading Theory Practice*, 4:3.

[Lehrman, 2020] Lehrman, P. D. (2020). Midi 2.0: Promises and challenges. In *Music Encoding Conference 2020*.

[Mann et al., 2018] Mann, S., Furness, T., Yuan, Y., Iorio, J., and Wang, Z. (2018). All reality: Virtual, augmented, mixed (x), mediated (x, y), and multimediated reality. *arXiv preprint arXiv:1804.08386*.

[Molero et al., 2021] Molero, D., Schez-Sobrino, S., Vallejo, D., Glez-Morcillo, C., and Albusac, J. (2021). A novel approach to learning music and piano based on mixed reality and gamification. *Multimedia Tools and Applications*, 80(1):165–186.

[Noor, 2016] Noor, A. K. (2016). The hololens revolution. *Mechanical Engineering*, 138(10):30–35.

[Olson, 1967] Olson, H. F. (1967). *Music, physics and engineering*, volume 1769. Courier Corporation.

[Pilhofer and Day, 2019] Pilhofer, M. and Day, H. (2019). *Music theory for dummies*. John Wiley & Sons.

[Poupyrev et al., 2001] Poupyrev, I., Berry, R., Billinghurst, M., Kato, H., Nakao, K., Baldwin, L., Kurumisawa, J., et al. (2001). Augmented reality interface for electronic music performance. In *Proceedings of HCI*, pages 805–808.

[Poupyrev et al., 2000] Poupyrev, I., Berry, R., Kurumisawa, J., Nakao, K., Billinghurst, M., Airola, C., Kato, H., Yonezawa, T., and Baldwin, L. (2000). Augmented groove: Collaborative jamming in augmented reality. In *ACM SIGGRAPH 2000 Conference Abstracts and Applications*, volume 17, page 77.

[Roads et al., 1996] Roads, C., Strawn, J., et al. (1996). *The computer music tutorial*. MIT press.

[Rothstein, 1992] Rothstein, J. (1992). *MIDI: A comprehensive introduction*, volume 7. AR Editions, Inc.

[Rusiñol et al., 2018] Rusiñol, M., Chazalon, J., and Diaz-Chito, K. (2018). Augmented songbook: an augmented reality educational application for raising music awareness. *Multimedia Tools and Applications*, 77(11):13773–13798.

[Salazar et al., 2006] Salazar, S., Wang, G., and Cook, P. R. (2006). miniaudicle and chuck shell: New interfaces for chuck development and performance. In *ICMC*.

[Sanders, 2016] Sanders, A. (2016). *An introduction to Unreal engine 4*. CRC Press.

[Vail, 2014] Vail, M. (2014). *The synthesizer: a comprehensive guide to understanding, programming, playing, and recording the ultimate electronic music instrument*. Oxford University Press.

[Wang, 2008] Wang, G. (2008). *The ChucK audio programming language.“A strongly-timed and on-the-fly environ/mentality”*. Princeton University.

[Wright et al., 2017] Wright, M., Freed, A., and Momeni, A. (2017). 2003: Opensound control: State of the art 2003. *A NIME Reader*, pages 125–145.

Apéndices

A

Código ChucK

A continuación están trozos importantes de los ficheros en lenguaje ChucK. Estos ficheros conforman el módulo del sintetizador, en las referencias se puede encontrar algún tutorial del lenguaje, pero en esencia se parece a C++. La gran especialidad es el operador chuck "=>", que simboliza a la vez la asignación de variables y la interconexión de módulos de sintetizador. El código completo puede encontrarse en la subcarpeta *LoosySynth* del siguiente repositorio:

https://github.com/juan260/Loosy

Código A.1: Fichero *main.ck*, que es el fichero raíz y el que hay que ejecutar los demás. Primero importa todas las librerías que hacen falta y ejecutar LoosySynth

```
0  //␣Fichero:␣main.ck␣
1  //␣Autor:␣Juan␣Riera␣Gomez␣
2
3  //␣Este␣fichero␣es␣el␣fichero␣raiz␣y␣es␣el␣primero␣al␣que␣se␣llama␣para␣cargar␣
4  //␣todos␣los␣demas.␣Carga␣las␣librerias␣necesarias␣y␣luego␣ejecuta␣LoosySynth.ck.␣
5
6
7
8  //␣Creamos␣un␣array␣con␣todos␣los␣ficheros␣que␣tiene␣que␣abrir␣en␣orden␣
9  ["NoteSetHandler",␣"ChordHandler",␣"ScaleHandler",␣"ChordOscHandler",␣"LoosySynth"]␣@=>␣string␣
       libs[];␣
10
11 //␣Abre␣los␣ficheros␣de␣uno␣en␣uno␣
12 for(0␣=>␣int i;␣i<␣libs.cap();␣i++){␣
13
14     //␣Si␣uno␣no␣lo␣encuentra␣o␣falla␣
15     if(Machine.add(me.dir()+"/"+libs[i]+".ck")==0){␣
16         //␣Informamos␣al␣usuario␣
17         <<<␣libs[i]␣+␣"␣not␣found␣or␣already␣added!"␣>>>;␣
18     }␣
19 }␣
```

Código A.2: Parte I del fichero *LoosySynth.ck*, que se encarga de recibir mensajes, albergar los objetos ScaleHandler y chordOscHandler, gestionar el oscilador y filtro de la melodía y manejar los paquetes. Hasta la definición de handleMessage, ocurre la inicialización del objeto. HandleMessage es la función que maneja los paquetes que llegan y decide queé hacer con esa información.

```
36  class LoosySynth{␣
37      OscMsg␣msg;␣//␣Mensaje␣OSC␣
38      OscIn␣oin;␣//␣Cliente␣OSC␣para␣recibir␣mensajes␣
39
40      //␣Conexion␣de␣la␣cadena␣de␣seniales␣
41      JCRev␣masterReverb␣=>␣Delay␣masterDelay␣=>␣Dyno␣compressor␣=>␣Dyno␣limiter␣=>␣Gain␣
            masterGain;␣
42
43      //␣Ademas␣del␣delay,␣metemos␣senial␣directa␣
44      masterReverb␣=>␣masterGain;␣
45      LPF␣rightHandFilter␣=>␣masterReverb;␣
46      SawOsc␣rightHandOscillator␣=>␣rightHandFilter;␣
47      0.2␣=>␣masterGain.gain;␣//␣Bajamos␣volumenes␣
48      0.2␣=>␣rightHandOscillator.gain;␣
49      0.5␣=>␣masterReverb.gain;␣
50      0.6␣=>␣masterDelay.gain;␣
51
52      //␣Ajustamos␣parametros␣de␣los␣efectos␣
53      .5␣=>␣masterReverb.mix;␣
54      .6::second␣=>␣masterDelay.max␣=>␣masterDelay.delay;␣
55      //␣Retroalimentacion␣del␣delay␣
56      masterDelay␣=>␣Gain␣delayFeedback␣=>␣masterDelay;␣
57      0.4␣=>␣delayFeedback.gain;␣
58      compressor.compress();␣
59      limiter.limit();␣
60
61      //␣Creacion␣de␣los␣manejadores␣necesarios␣
62      ScaleHandler␣scaleHandler;␣
63      ChordOscHandler␣chordOscHandler;␣
64      chordOscHandler.connect(masterReverb);␣
65
66      //␣Cuantos␣centimetros␣hay␣que␣avanzar␣para␣cambiar␣de␣nota␣
67      10.0␣=>␣float positionScale;␣
68
69      //␣Nota␣do␣central␣para␣el␣oscilador,␣aunque␣va␣a␣empezar␣en␣silencio␣
70      Std.mtof(60)␣=>␣rightHandOscillator.freq;␣
71
72
73  //␣Funcion␣de␣manejo␣de␣paquetes.␣Recibe␣un␣paquete␣y␣decide␣que␣hacer␣
74  //␣con␣esa␣informacion,␣a␣que␣funciones␣llamar...␣
75  //␣Entradas:␣El␣mensaje␣OSC␣
76
77      fun␣void handleMessage(OscMsg␣msg){␣
78          if(msg.typetag.charAt(0)␣==␣'i'){␣
79                  if(msg.getInt(0)␣==␣POSITIONMSG␣&&␣msg.typetag␣==␣"iifffff"){␣
80                      //␣Position␣message␣
```

Código A.3: Parte II del fichero *LoosySynth.ck*, continuamos con el cuerpo de la función handleMessage.

```
81                      if (msg.getInt(1) == 0 || msg.getFloat(2)==0.0) {
82                          // Si es un mensaje invalido o silencioso
83                          chordOscHandler.setGain(0.0);
84
85                      } else {
86                          // Ajustamos la ganancia y el filtro de los acordes
87                          chordOscHandler.setGain(Math.pow(msg.getFloat(2),1/7));
88                          chordOscHandler.setFilterCutoffZeroToOne(
89                          msg.getFloat(2));
90                      }
91
92                      if (msg.getInt(3) == 0 || scaleHandler.inSilence()){
93                          // Si es un mensaje invalido o no hay escala
94                          // seleccionada nos quedamos en silencio
95                          0.0 => rightHandOscillator.gain;
96                      } else {
97                          // Ajustamos la ganancia, la nota y
98                          // el filtro de la mano derecha
99                          msg.getFloat(4)*0.6 => rightHandOscillator.gain;
100                         scaleHandler.getRoundedFreqFromMidi(
101                         msg.getFloat(5)/positionScale+60) =>
102                         rightHandOscillator.freq;
103
104                         Math.pow(Math.fabs(msg.getFloat(6)+50.0),
105                         2) => rightHandFilter.freq;
106                     }
107                 }
108                 // Control message
109                 else if(msg.getInt(0) == CONTROLMSG){
110                   if(msg.typetag == "iiss"){
111                     if(msg.getInt(1) == SCALECHANGE){
112
113                         //Change scale
114                         scaleHandler.changeNoteSet(msg.getString(2),
115                         msg.getString(3));
116                     } else if(msg.getInt(1) == CHORDCHANGE){
117                         //Change chord
118
119                         chordOscHandler.changeChord(msg.getString(2),msg.getString(3));
120                     }
121                   }
122                 }
123                 // Initialization message
124             else if(msg.getInt(0)==INITMSG && msg.typetag.charAt(1)=='i'){
```

Código A.4: Parte III del fichero *LoosySynth.ck*, Final del cuerpo de handleMessage, y función start, que es la que inicia verdaderamente el sintetizador la ponerlo a escuchar un puerto.

```
125
126                     //␣Scale␣initialization␣
127                      if␣(msg.getInt(1)␣==␣NEWSCALE␣&&␣msg.typetag.charAt(2)=='s'){␣
128                              extractFloatsFromOscMsg(msg)␣@=>␣float␣extractedFloats[];␣
129                              scaleHandler.addSetToTableFromMidi(msg.getString(2),␣extractedFloats);␣
130
131
132
133                     }␣else␣if(msg.getInt(1)␣==␣NEWCHORD␣&&␣msg.typetag.charAt(2)=='s')␣{␣
134                         //␣Chord␣initialization␣
135                         extractFloatsFromOscMsg(msg)␣@=>␣float␣extractedFloats[];␣
136                         chordOscHandler.addChordToTable(extractedFloats,␣msg.getString(2));␣
137                     }␣
138                 }␣
139
140
141             }␣
142         }␣
143
144 //␣Funcion␣para␣arrancar␣el␣sintetizador,␣solo␣recibe␣el␣puerto␣
145 //␣del␣que␣debe␣esperar␣mensajes,␣y␣la␣propia␣funcion␣se␣pone␣a␣escuchar␣
146 //␣en␣bucle.␣
147 //␣Entrada:␣el␣puerto␣del␣que␣recibir␣paquetes␣
148
149     fun␣void␣start(int␣port){␣
150
151         port␣=>␣oin.port;␣
152         oin.listenAll();␣
153         masterGain␣=>␣dac;␣
154         while(true)␣
155         {␣
156             oin␣=>now;␣
157             while(oin.recv(msg))␣
158             {␣
159               handleMessage(msg);␣
160
161             }␣
162             0.005::second␣+=>␣now;␣
163
164
165         }␣
166     }␣
167 }␣
```

Código A.5: Fragmento del fichero *NoteSetHandler.ck*, que es la clase abstracta que representa el concepto de conjunto de notas del que heredan las los acordes y las escalas. Lo interesante a observar aquí es que cambiar el conjunto de notas (cambiar de escala o de acorde) es simplemente cambiar el contenido de un string, que actúa de índice en la tabla de conjuntos de notas, de manera que una vez cambiado las siguientes consultas devolverán el nuevo conjunto de notas. Otra observación interesante es que la función inSilence() lo que hace es comprobar si está seleccionado el conjunto de notas por defecto, que actúa como conjunto vacío "Silence". En la práctica está lleno de notas inaudiblemente graves para el oído humano por si a algún programador se le olvida llamar a esta función.

```
58  //␣Funcion␣que␣se␣encarga␣de␣cambiar␣el␣conjunto␣de␣notas␣actual.␣
59  //␣
60  //␣Entradas:␣un␣string␣con␣el␣nombre␣del␣conjunto␣al␣que␣se␣quiere␣cambiar␣
61  //␣Salidas:␣1␣si␣es␣un␣conjunto␣conocido,␣-1␣si␣es␣un␣conjunto␣desconocido␣
62  //␣al␣que␣por␣tanto␣no␣se␣puede␣cambiar␣
63
64      fun␣int␣changeNoteSet(string␣newSet){␣
65          <<<"ChangeAttempt",␣newSet␣>>>;␣
66          for(0=>int␣i;i<currentKnownNotes;i++){␣
67              if(knownNotes[i]==newSet){␣
68                  newSet␣=>␣currentSet;␣
69                  <<<␣"Set␣Changed",␣newSet␣>>>;␣
70                  return␣1;␣
71              }␣
72          }␣
73          return␣-1;␣
74      }␣
75
76
77  //␣Funcion␣para␣cambiar␣el␣conjunto␣de␣notas␣y␣la␣transposicion␣
78  //␣a␣la␣vez.␣
79  //␣
80  //␣Entradas:␣un␣string␣con␣la␣nota␣raiz,␣y␣otro␣con␣el␣nuevo␣nombre␣
81  //␣del␣conjunto␣(por␣ejemplo␣"C"␣"mayor")␣
82  //␣Salidas:␣1␣en␣caso␣de␣exito,␣-1␣en␣caso␣de␣error␣
83
84      fun␣int␣changeNoteSet(string␣root,␣string␣name){␣
85          getTranspositionFromNoteName(root)␣=>␣int␣newTransp;␣
86          if(newTransp␣<␣-12)␣
87              return␣-1;␣
88          newTransp␣=>␣currentTranspose;␣
89          return␣changeNoteSet(name);␣
90      }␣
91
92
93  //␣Funcion␣que␣comprueba␣si␣no␣hay␣ninguna␣escala␣elegida.␣
94  //␣
95  //␣Salidas:␣1␣en␣caso␣afirmativo,␣0␣en␣caso␣negativo␣
96
97      fun␣int␣inSilence(){␣
98          return␣currentSet␣==␣"Silence";␣
99      }␣
```

Código A.6: Parte I del fichero *ScaleHandler.ck*, que se encarga de las escalas pero solo en su sentido abstracto de conjuntos de notas. Destacamos esta función llamada specialRound, que es la que hace la interpolación no lineal, partiendo de la interpolación lineal que se hace en el cuerpo de getRoundedMidi. getRoundedMidi es la función que recibe una nota midi, calculada como el número de notas que se ha avanzado desde la posición central de la mano sumado a 60. De manera que la mano frente a la cabeza tiene el valor 60, y si se mueve a la derecha esta nota irá subiendo. En la notación anterior esta función recibe m_2 y devuelve m_3.

```
34
35  fun float getRoundedMidi(float midiNote){ 
36
37      midiNote => float original; 
38      Math.max(0, midiNote) => midiNote; 
39      Math.min(maxNoteTableSize-2, midiNote) => midiNote; 
40      Std.ftoi(Math.round(midiNote)) => int midiNoteIndex; 
41      noteTables[currentSet][midiNoteIndex]+currentTranspose => midiNote; 
42
43      1.0 => float distance; 
44      midiNote => float startingPoint; 
45      original-Math.floor(original) => float decimalPart; 
46
47      if(original > Math.round(original)){ 
48          noteTables[currentSet][midiNoteIndex+1]+currentTranspose-midiNote => 
49          distance; 
50          midiNote + decimalPart*distance => startingPoint; 
51      } 
52      else { 
53          midiNote-noteTables[currentSet][midiNoteIndex-1]-currentTranspose => 
54          distance; 
55          midiNote -(1-decimalPart)*distance => startingPoint; 
56
57      } 
58      return specialRound(startingPoint, midiNote, distance); 
59  }
```

Código A.7: Parte II del fichero *ScaleHandler.ck*. genScaleTable() es la función que a partir de una lista de intervalos calcula el resto de la escala, lon que esta función devuelve es lo que se guarda en la tabla de escalas.

```
93  fun float[] genScaleTable(float scale[]) {
94      float scaleTable[maxNoteTableSize];
95      int startingMidiNote;
96      float currentMidiNote;
97      int i;
98      60 => startingMidiNote;
99      60 => i;
100     startingMidiNote => currentMidiNote;
101     currentMidiNote => scaleTable[i];
102     // Rellenamos la tabla hacia arriba
103     while(i<maxNoteTableSize-1){
104         i++;
105         scale[positiveModulo((i-startingMidiNote-1),scale.cap())] +=>
106         currentMidiNote;
107         currentMidiNote => scaleTable[i];
108
109         if(currentMidiNote>126)
110             break;
111
112     }
113     // Terminamos de rellenar hacia arriba
114     while(i<maxNoteTableSize-1){
115         i++;
116         126 => scaleTable[i];
117     }
118     // Rellenamos la tabla hacia abajo
119     startingMidiNote => i;
120     startingMidiNote => currentMidiNote;
121     while(i>0){
122         i--;
123         scale[positiveModulo((i-startingMidiNote),scale.cap())] =>
124         currentMidiNote;
125
126         currentMidiNote => scaleTable[i];
127
128         if(currentMidiNote<0)
129             break;
130
131     }
132     // Terminamos de rellenar hacia abajo
133     while(i>0){
134         i--;
135         0 => scaleTable[i];
136     }
137     return scaleTable;
138
139
140     }
```

Código A.8: Parte I del fichero *ChordOscHandler.ck*, que se encarga de hacer sonar los acordes. Maneja en distintos osciladores las distintas voces simultáneas, y con funciones se pueden añadir y cambiar los acordes. Lo que vemos aquí es la declaración e inicialización del objeto donde observamos el array de osciladores.

```
8
9   public class ChordOscHandler{␣
10      10␣=>␣int maxNotes;␣//␣Numero␣maximo␣de␣notas␣
11      1␣=>␣int currentlySoundingNotes;␣//␣Numero␣de␣notas␣sonando␣actualmente␣
12      SawOsc␣oscillators[maxNotes];␣//␣Array␣de␣osciladores␣
13      LPF␣filter;␣//␣Filtro␣de␣paso␣bajo␣
14      float gain;␣//␣Ganancia␣
15
16      //␣Conectamos␣los␣osciladores␣al␣filtro␣
17      for(0␣=>␣int i;␣i<maxNotes;i++){␣
18          oscillators[i]␣=>␣filter;␣
19      }␣
20
21      //␣Creamos␣un␣manejador␣de␣acordes␣
22      ChordHandler␣chordHandler;␣
23      //␣Silenciamos␣los␣osciladores␣
```

Código A.9: Parte II del fichero *ChordOscHandler.ck*, que se encarga de hacer sonar los acordes. Maneja en distintos osciladores las distintas voces simultáneas, y con funciones se pueden añadir y cambiar los acordes.

```
43  //␣osciladores.␣
44  //␣
45  //␣Entrada:␣el␣numero␣maximo␣de␣notas.␣
46  //␣Salida:␣el␣numero␣maximo␣de␣notas␣que␣pueden␣sonar␣en␣un␣acorde.␣
47
48      fun␣int setMaxNotes(int n){␣
49
50          SawOsc␣@␣newOscillators[n];␣
51          int i;␣
52          for(0=>i;i<maxNotes;i++){␣
53                  oscillators[i]␣@=>␣newOscillators[i];␣
54          }␣
55          for(;i<n;i++){␣
56                  createNewOsc()␣@=>␣newOscillators[i];␣
57                   newOscillators[i]␣=>␣filter;␣
58          }␣
59          newOscillators␣@=>␣oscillators;␣
60          Std.ftoi(Math.max(1.0,n))␣=>␣maxNotes;␣
61          return maxNotes;␣
62      }␣
63
64
65  //␣Funcion␣para␣conectar␣la␣salida␣de␣este␣modulo␣a␣otro␣de␣ChucK.␣
66  //␣Entrada:␣el␣modulo␣al␣que␣conectar␣
67
68      fun␣void connect(UGen␣output){␣
69          filter␣=>␣output;␣
70      }␣
71
72
73  //␣Funcion␣para␣cambiar␣la␣ganancia␣general␣de␣los␣osciladores.␣
74  //␣Reparte␣la␣ganancia␣entre␣los␣osciladores.␣
75  //␣Entrada:␣nueva␣ganancia.␣
76
77      fun␣void setGain(float g){␣
78          0.7␣*g␣=>␣g;␣
79          if(inSilence())␣
80              0.0=>g;␣
81          int i;␣
82          for(0␣=>i;i<currentlySoundingNotes;i++){␣
83             g/currentlySoundingNotes␣=>␣oscillators[i].gain;␣
84          }␣
```

Código A.10: Parte II del fichero *ChordOscHandler.ck*. En esta figura tenemos la función de cambio de acorde y como se asignan a los distintos osciladores las distintas notas, para al final silenciar los demás.

```
117  fun int changeChord(string root, string quality){
118
119      chordHandler.changeNoteSet(root, quality);
120      chordHandler.getCurrentChordNotes()@=> float newChord[];
121      if(newChord==NULL)
122          return 0;
123      int i;
124      for(0=>i;i<Math.min(newChord.cap(),maxNotes);i++){
125
126          Std.mtof(newChord[i])=>oscillators[i].freq;
127      }
128
129      setCurrentlySoundingNotes(Std.ftoi(Math.min(newChord.cap(),maxNotes)));
130      return 1;
131  }
```

Código A.11: Fichero *ChordHandler.ck*, que se encarga de gestionar los acordes en su forma más abastracta como conjuntos de notas. Es un fichero muy corto por su herencia de NoteSetHandler, su única función es la que aparece aquí que devuelve las notas midi concretas que forman parte del acorde seleccionado.

```
11  1=>int currentlySoundingNotes;//Numero de notas sonando actualmente
12  SawOsc oscillators[maxNotes];//Array de osciladores
13  LPF filter;//Filtro de paso bajo
14  float gain;//Ganancia
15
16  //Conectamos los osciladores al filtro
17  for(0=>int i;i<maxNotes;i++){
18      oscillators[i]=>filter;
19  }
20
21  //Creamos un manejador de acordes
22  ChordHandler chordHandler;
23  //Silenciamos los osciladores
24  setGain(0.0);
25
26
27 //Funcion para cuantos osciladores tienen que estar sonando.
```

B

Blueprints Unreal Engine

Se presentan a continuación fragmentos de los blueprints del módulo controlador. Se recuerda al lector que esto es un equivalente a código en Unreal Engine, donde también se puede programar en C++, pero el método de blueprints es más sencillo para un proyecto con un único desarrollador, sobre todo si este es primerizo en esta herramienta como ha sido el caso.

En los sistemas de blueprints, el *Event Graph* (gráficos de eventos) es el código raíz, que se podría decir que se parece al código escrito directamente dentro del objeto, y que llama a las funciones correspondientes. En este apéndice veremos los gráficos de eventos de los tres submódulos del controlador, y algunas funciones internas interesantes.

Todo lo necesario para reconstruir el proyecto, incluyendo los blueprints con las clases de las que hemos hablado a lo largo de este documento, se puede hallar en la subcarpeta *LoosyController* del repositorio:

https://github.com/juan260/Loosy

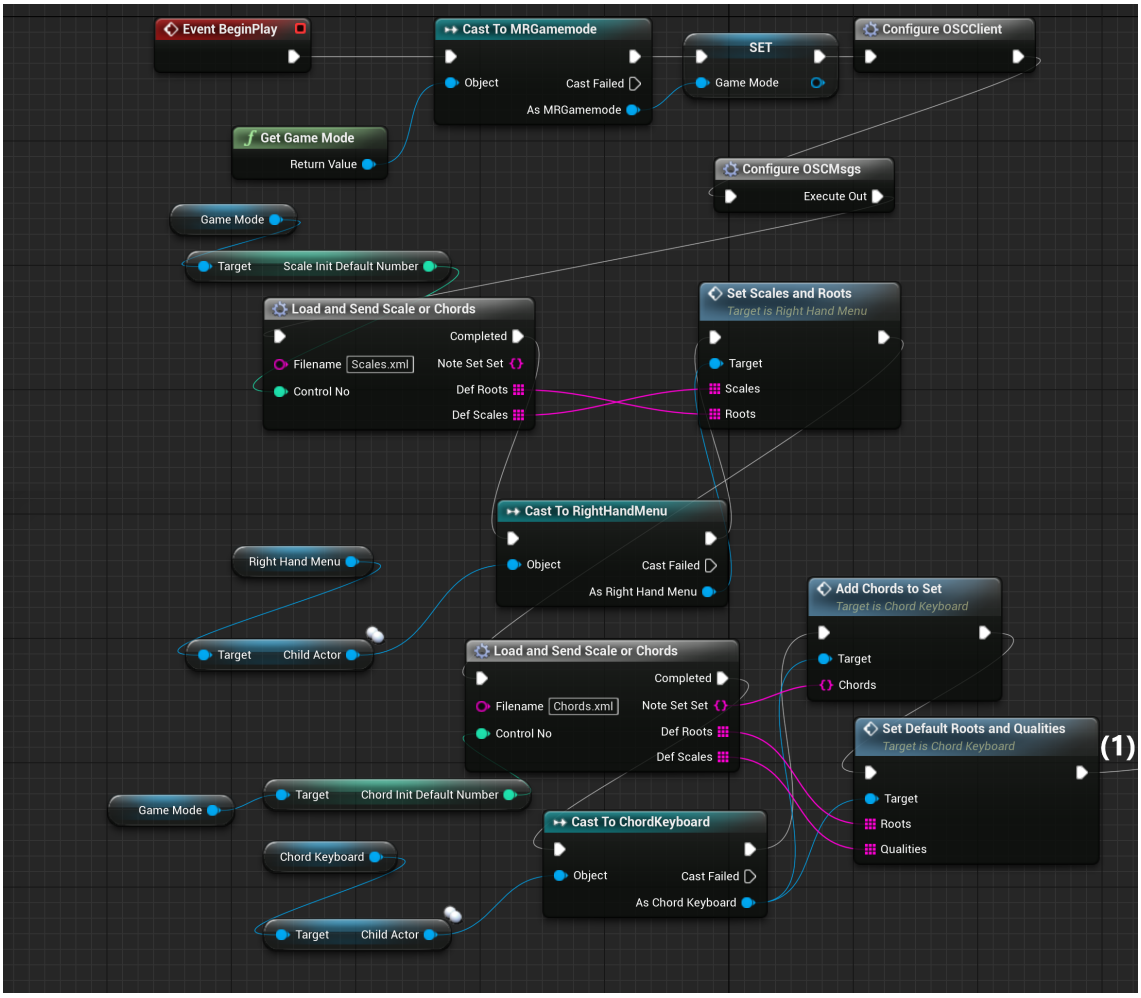


Figura B.1: SynthController: EventGraph pt. 1. Este es el gráfico de eventos del SynthController, el objeto padre. Como vemos configura el cliente OSC y los mensajes, carga Scales.xml y Chords.xml y le comunica esa informacion al menú de la mano derecha y al tablero de acordes.

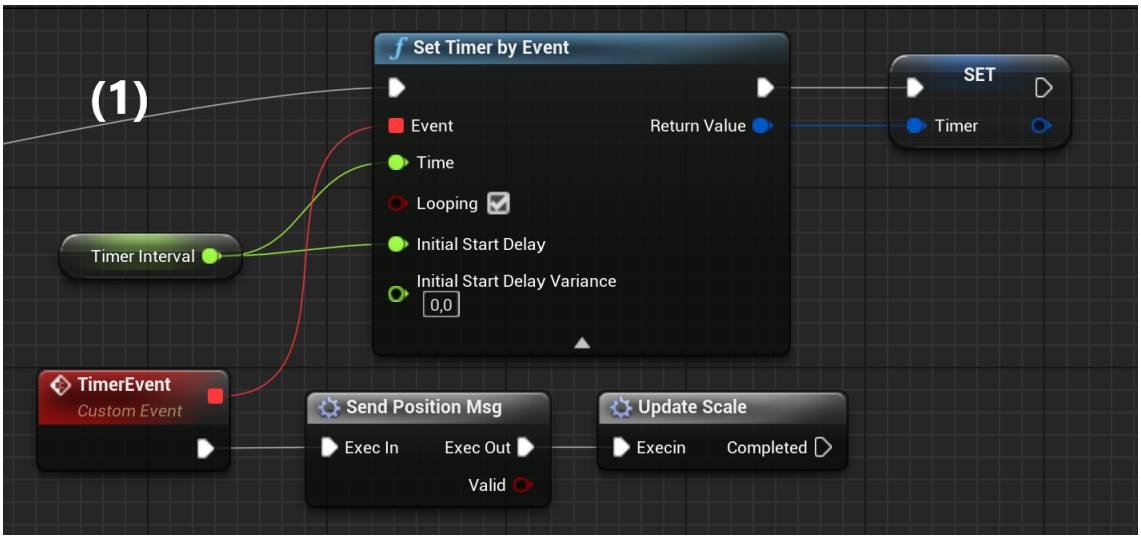


Figura B.2: SynthController: EventGraph pt. 2. En esta segunda mitad crea un evento que se ejecuta cada cierto tiempo y que llama a SendPositionMsg y UpdateScale, que analizamos a continuación.

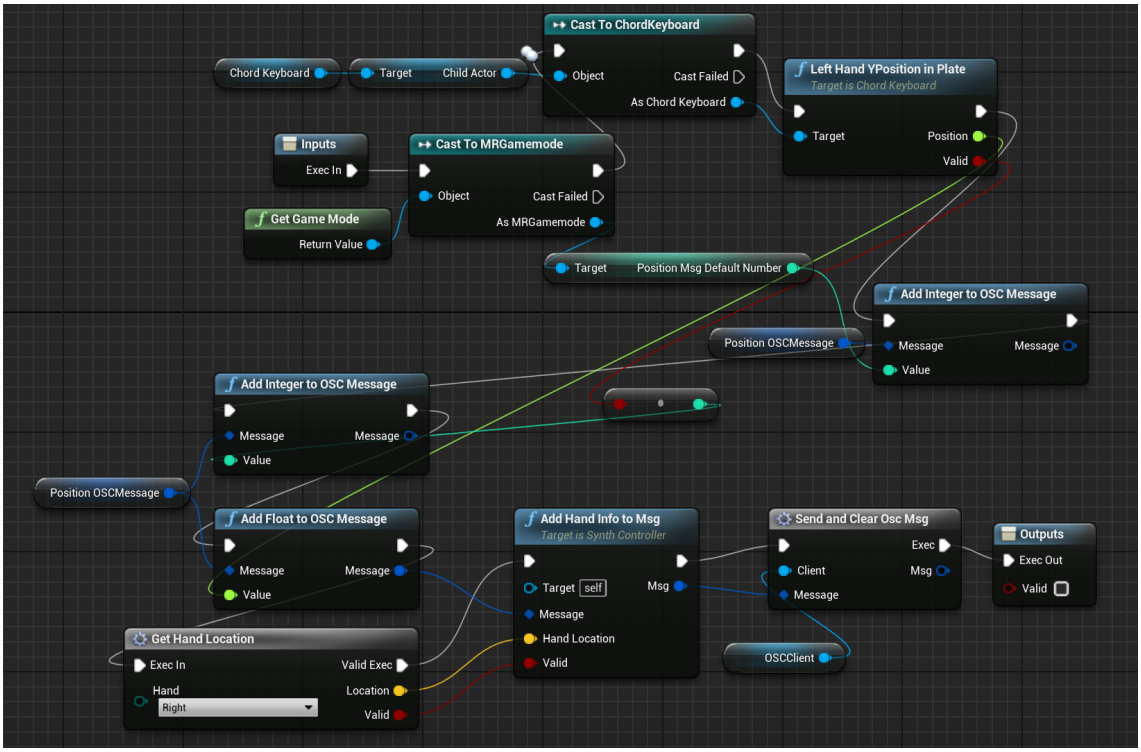


Figura B.3: SynthController: SendPositionMsg. Esta función analiza las posiciones de las manos y las envía al sintetizador, además de preparar el mensaje de posición descrito en la sección 5.2.3.

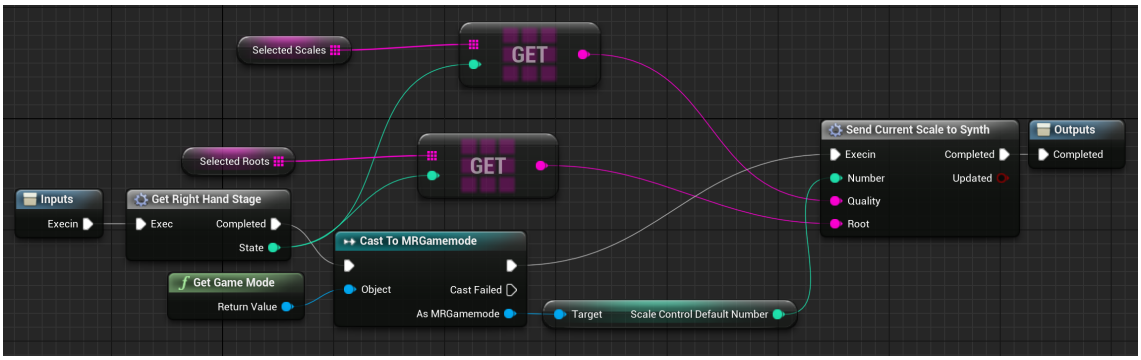


Figura B.4: SynthController: UpdateScale. Esta función obtiene la fase (*stage*) de la manod derecha y averigua cuál es su escala asignada (root y quality). Esta información se la da a SendCurrentScaleToSynth, que detecta si las entradas son distintas a la de su llamada anterior y si es así comunica al sintetizador el cambio de escala.

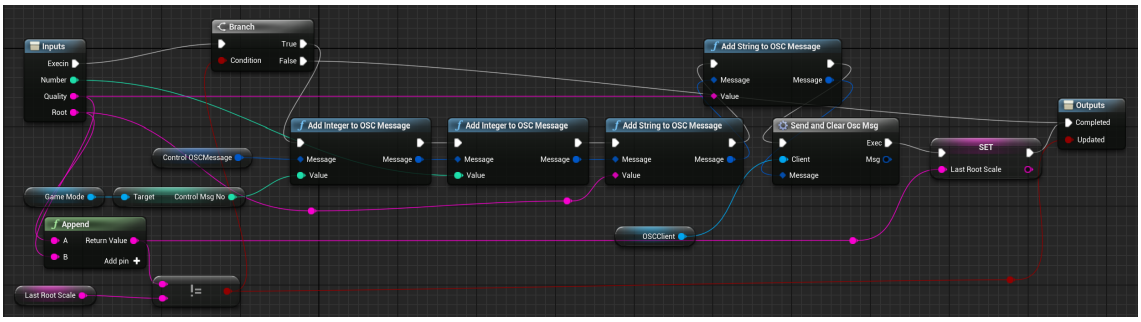


Figura B.5: SynthController: SendCurrentScaleToSynth. Esta función comprueba si está siendo llamada con unas entradas distintas a las anteriores y si es así prepara el mensaje de control correspondiente descrito en la sección 5.2.3 y lo envía.

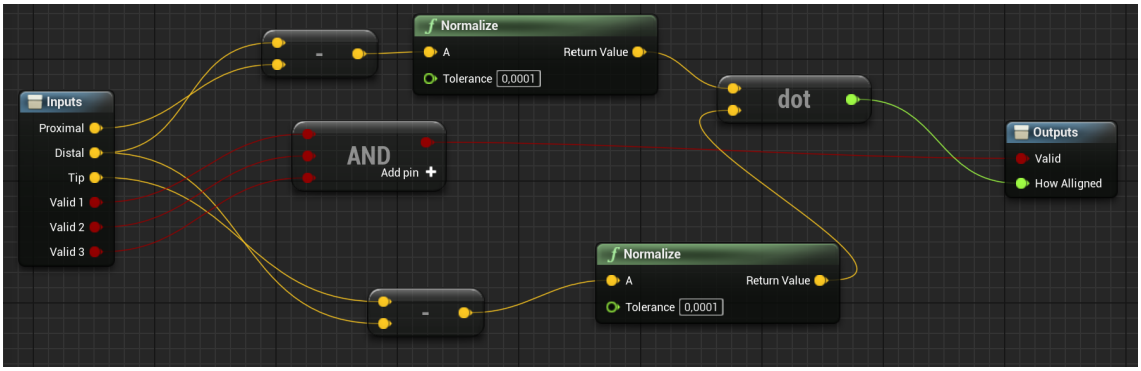


Figura B.6: SynthController: IsIndividualFingerPointing. De entre las funciones de gestión de vectores descrita en la sección 4.1.1 se ha seleccionado esta para mostrar por ser la más elegante y sencilla. Se reciben la punta (tip), falange intermedia (distal) y puño (proximal) de un dedo cualquiera, y tres booleanos que expresan si estos datos son válidos o si están fuera del ángulo de detección de las gafas. Se calculan los vectores normalizados falange intermedia $\rightarrow$ punta y puño $\rightarrow$ falange intermedia, y se hace el producto escalar para ver cómo de alineados están.

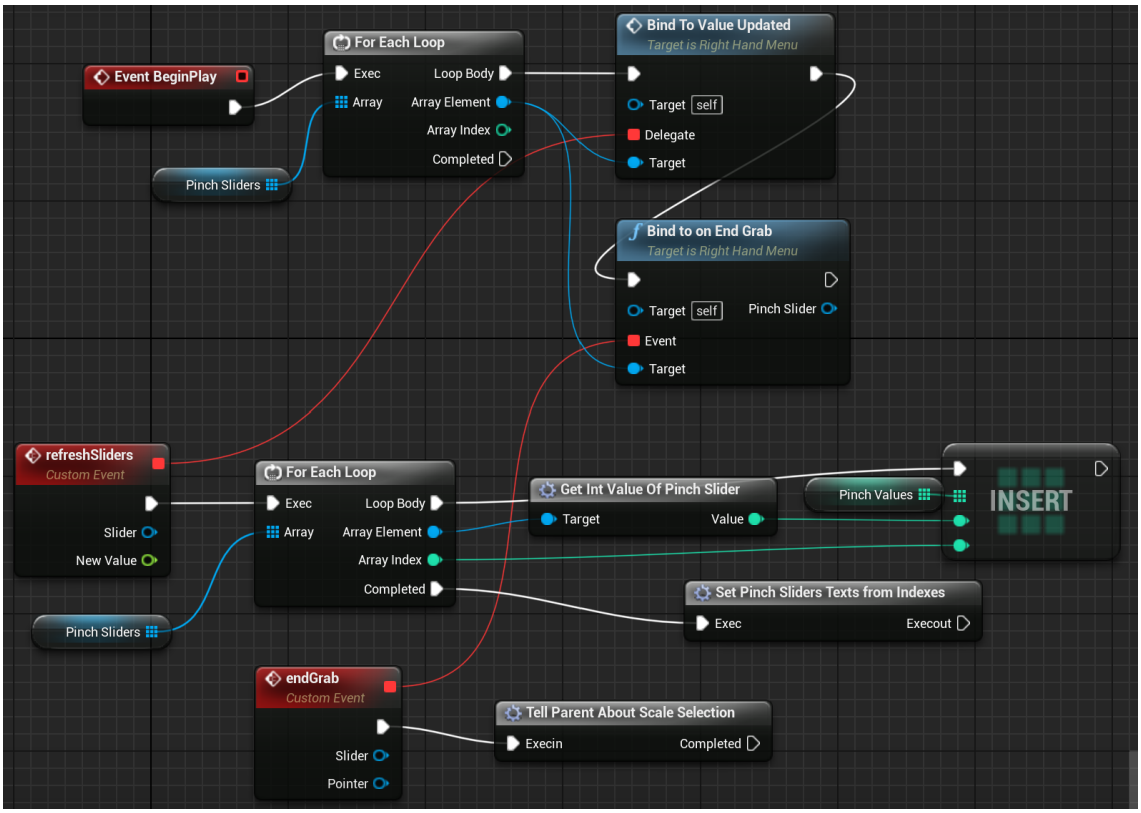


Figura B.7: RightHandMenu: EventGraph pt. 1. El gráfico de eventos del módulo *RightHandMenu*, en el que creamos un evento que se activa cuando se mueve un *slider* para actualizar el texto, y otro que se activa cuando se suelta un *slider*, para actualizar el sonido.

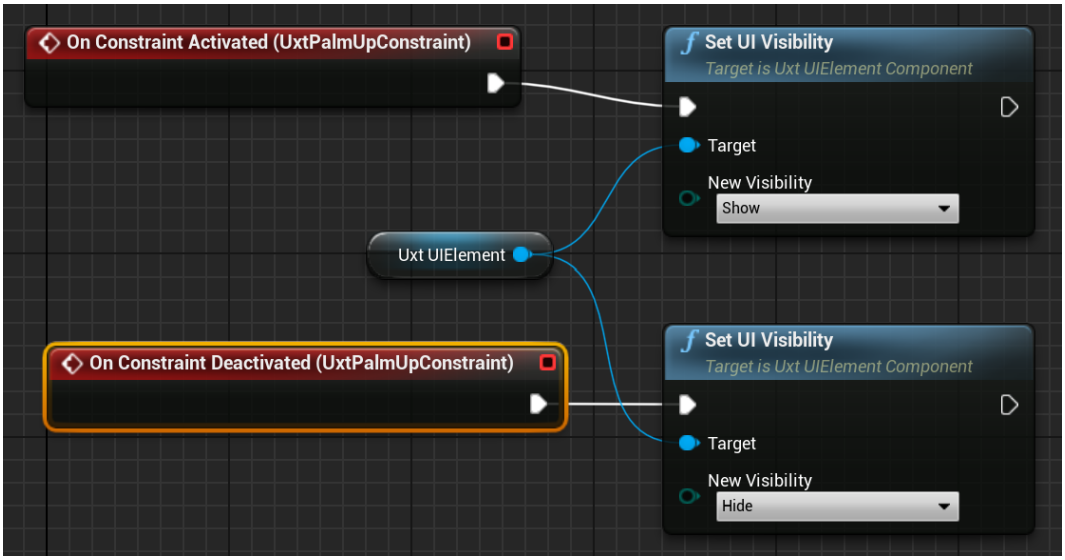


Figura B.8: RightHandMenu: EventGraph pt. 2. En esta párte del gráfico de eventos nos valemos de MRTK de Microsoft, la librería antes mencionada de desarrollo en Unreal para Hololens 2, y la utilizamos para detectar cuándo la palma de la mano mira al usuario y cuándo no, para ocultar o mostrar el menú.

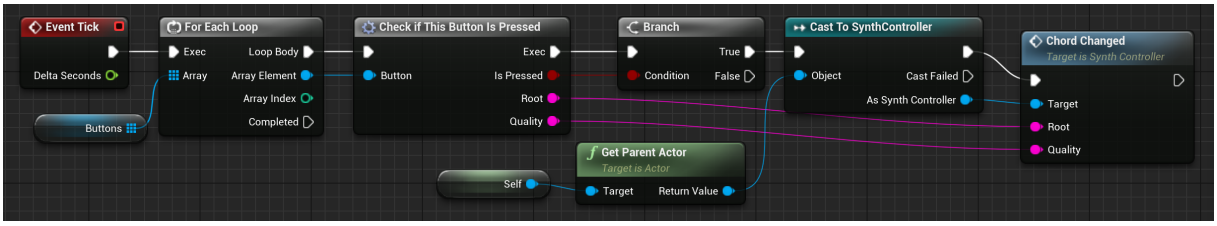


Figura B.9: ChordKeyboard: EventGraph. Gráfico de eventos de ChordKeyboard, en el que en cada tick, que es la unidad de tiempo más pequeña accesible, se comprueba si la mano izquierda está en contacto con algún botón, y en tal caso se comunica a SynthController.

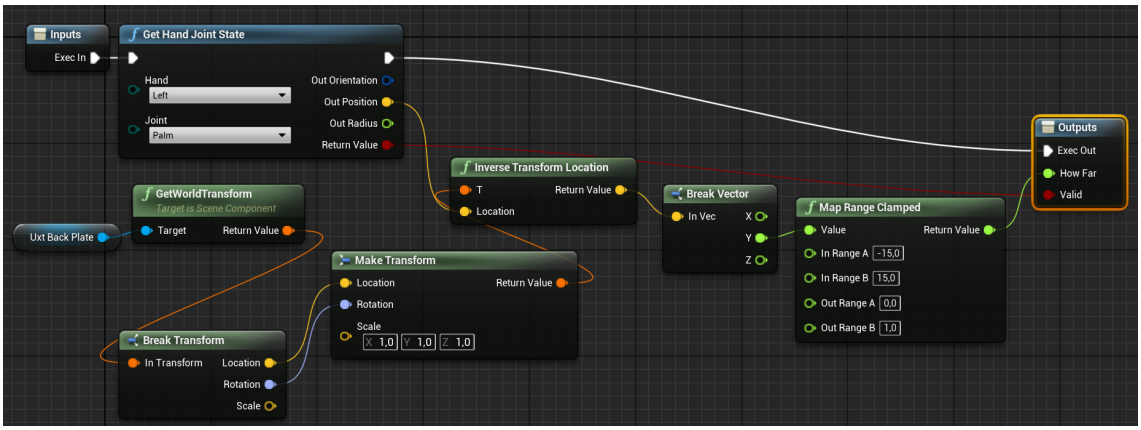


Figura B.10: ChordKeyboard: HowFarIsLeftHandInThePlate. Función que mide cuánto se ha adentrado la mano izquierda sobre el plano extendido de acordes. Se proyecta sobre el plano y se interpola para que nos devuelva un valor entre 0 y 1.

C

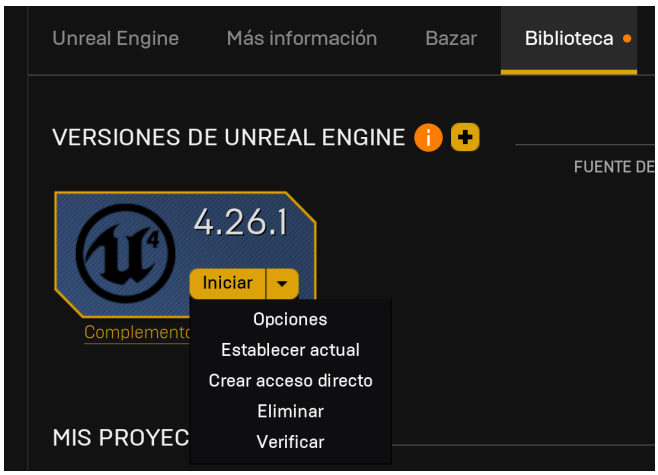
Tutorial de configuración de proyecto Hololens 2 para Unreal Engine

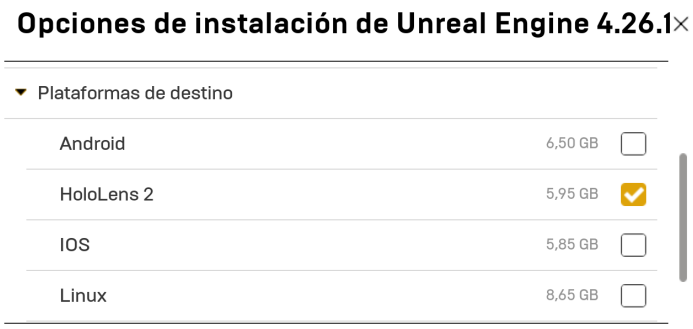
A modo de agradecimiento al equipo de investigación que le ofreció al alumno a principio de curso unas gafas Hololens para poder desarrollar este proyecto, el alumno redactó las siguientes páginas que constituyen un tutorial de instalación y configuración de un proyecto en Unreal Engine para Hololens 2. Está pensado para que el equipo de investigación se valga de él cuando las gafas estén disponibles.

Instalación de herramientas

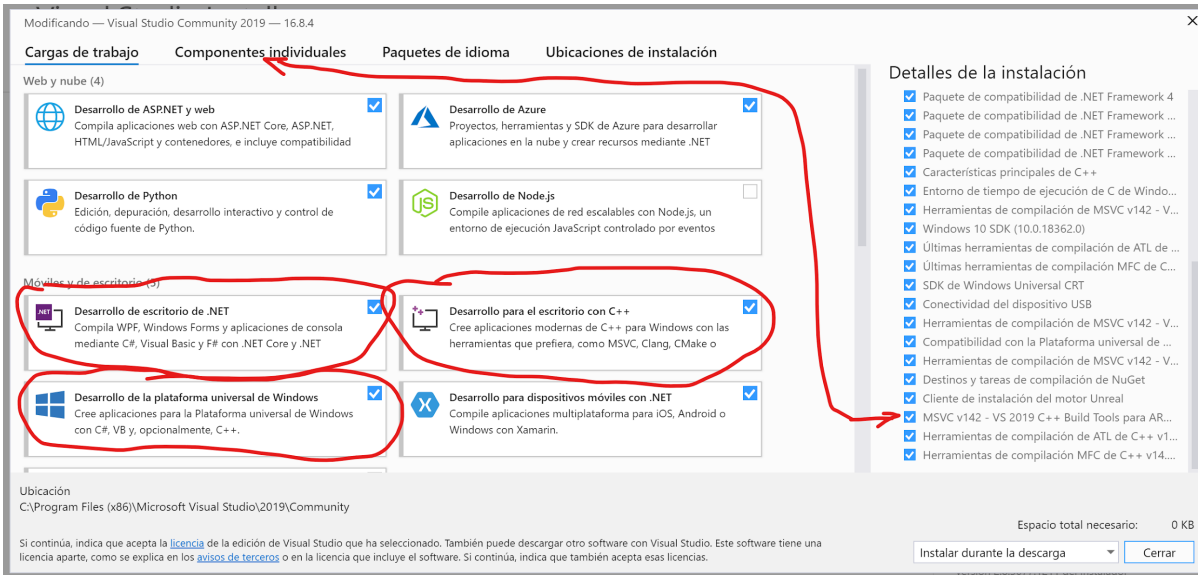
Herramientas necesarias:

- Unreal Engine (en este tutorial se ha utilizado la versión 4.26)
  https://www.unrealengine.com/en-US/download
  Con al menos las el paquete de Hololens. En epic games launcher > Biblioteca clickando a la derecha de “Iniciar” y después en “Opciones”:





- Mixed Reality Toolkit (MRTK) for Unreal Engine
  https://github.com/microsoft/MixedReality-UXTools-Unreal/releases
- Visual Studio con al menos los siguientes paquetes:
  - Desarrollo para el escritorio C++
  - Desarrollo para la plataforma universal de Windows
  - Desarrollo de escritorio .NET
  - En “componentes individuales”: La última versión de MSVC para ARM



También pueden ser útiles las siguientes herramientas:

- Open-XR, librerías adicionales de microsoft para Mixed Reality (muy recomendable): https://github.com/microsoft/Microsoft-OpenXR-Unreal#installing-the-packaged-plugin
- Hololens 2 Emulator, para testear sin tener que ponerse las gafas (opcional, se puede testear sin las gafas en el propio editor de todas formas): https://docs.microsoft.com/en-us/windows/mixed-reality/develop/platform-capabilities-and-apis/using-the-hololens-emulator
- Pix, debugger (opcional)

He creado un proyecto “template” para que no haya que hacer los pasos que vienen a continuación. Para instalar el template, seguir las instrucciones del siguiente enlace:

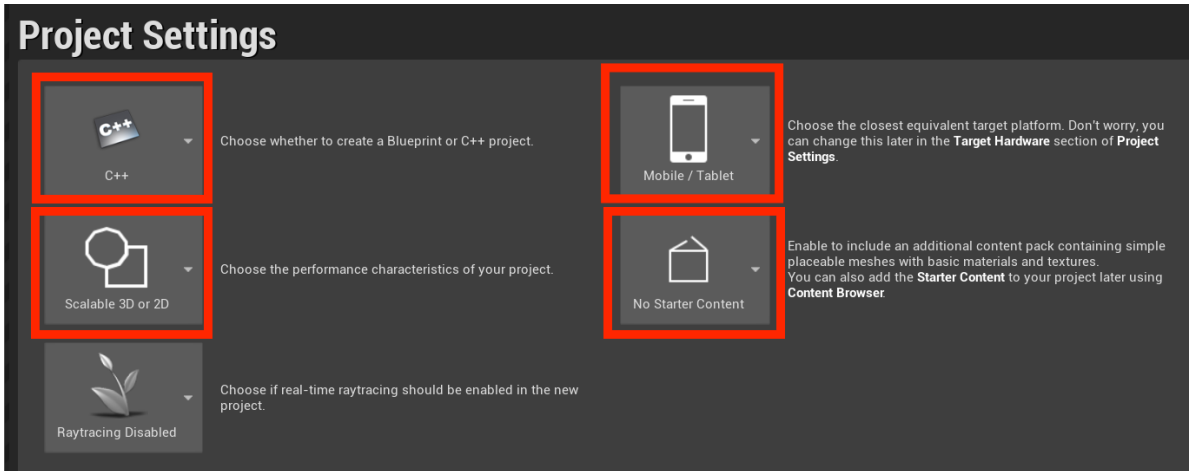
https://drive.google.com/drive/folders/1I0_c_7fIddxJQXiIPsv1iPxYEKIplrwo?usp=sharingc

Con ese template instalado, al intentar crear un nuevo proyecto en Unreal Engine, aparecerá la opción “Hololens 2”, que tiene todo lo que aparece en la siguiente sección ya hecho. A excepción del último paso, que no estoy seguro de que sea necesario aún.

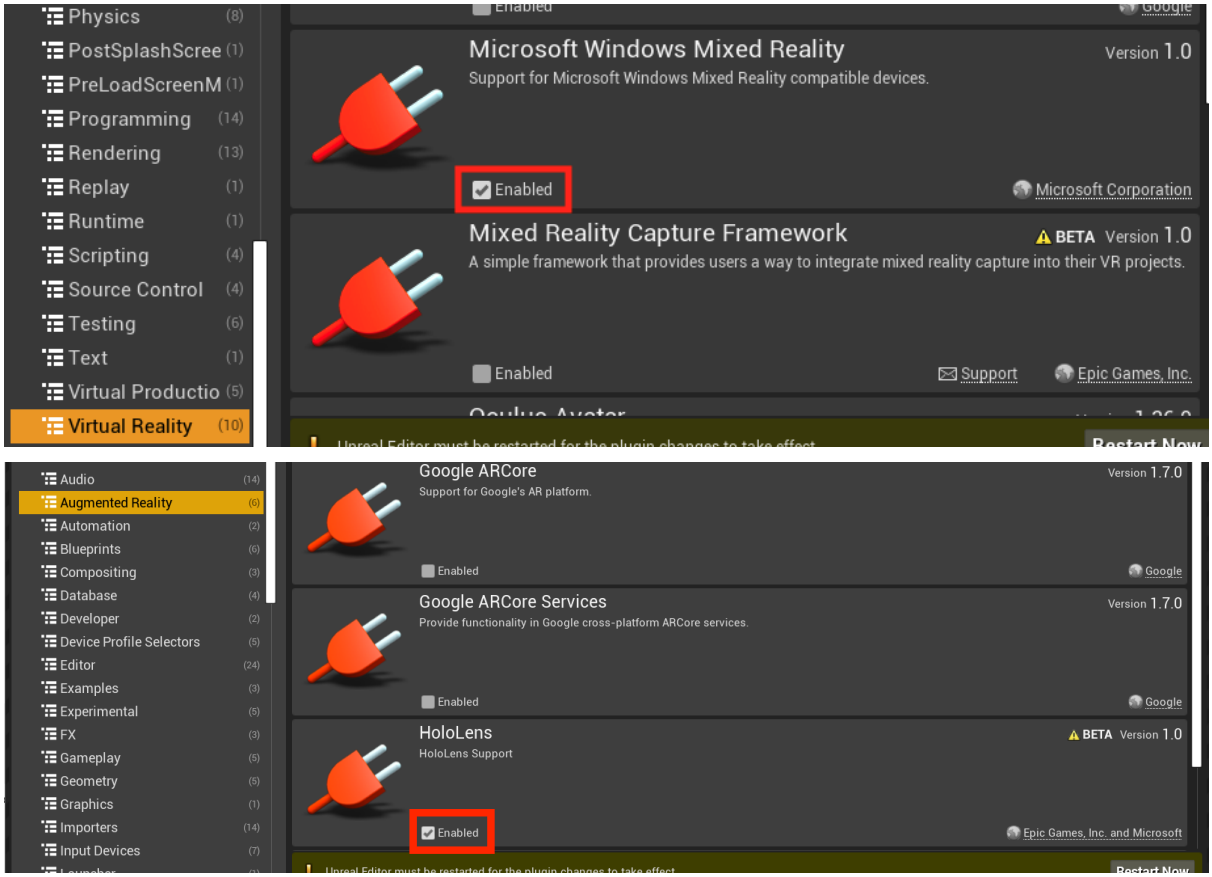
Si por alguna razón alguien quisiera rehacer los pasos necesarios, aquí el tutorial sobre cómo configurar un proyecto de Unreal Engine 4.26 para Hololens 2:

Configuración del proyecto

1) Nuevo proyecto: C++, Scalable 3D or 2D, Mobile/Tablet, and No Starter Content

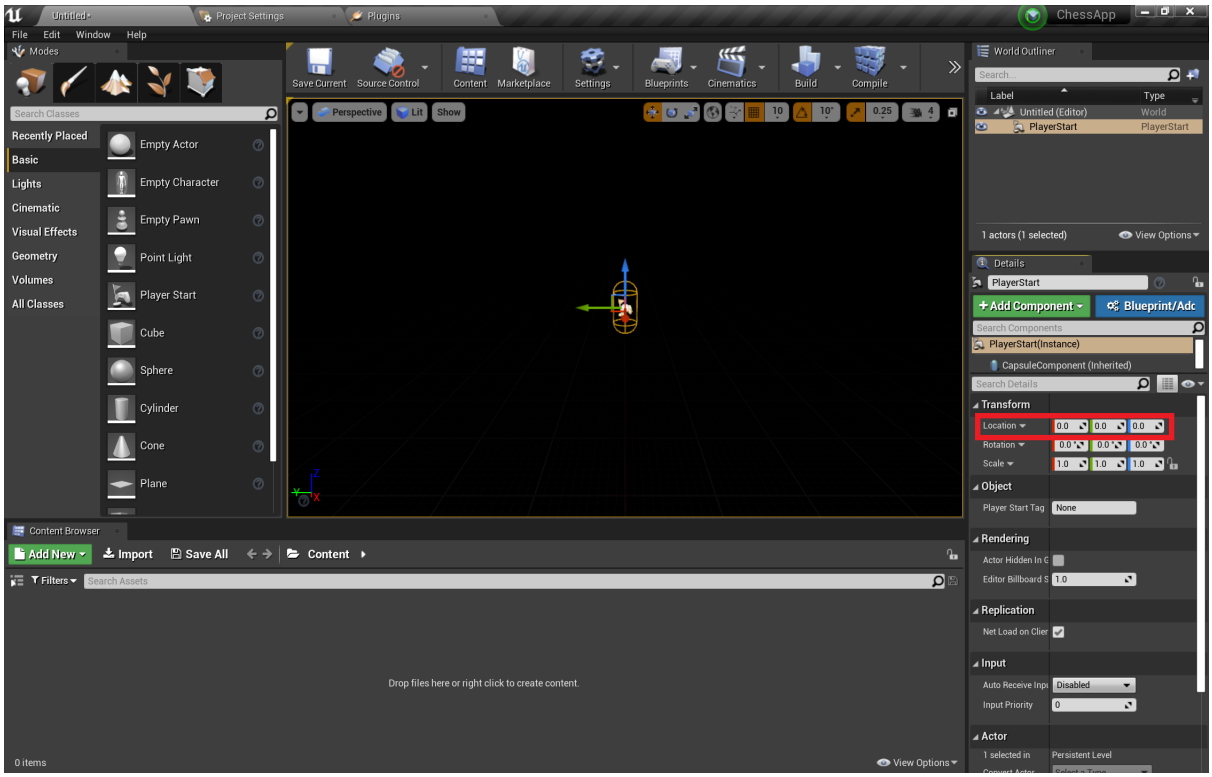


2) Nos vamos a Edit > Plugins y habilitamos los plugins “Hololens”, “Microsoft Windows Mixed Reality”, “Microsoft OpenXR” y “



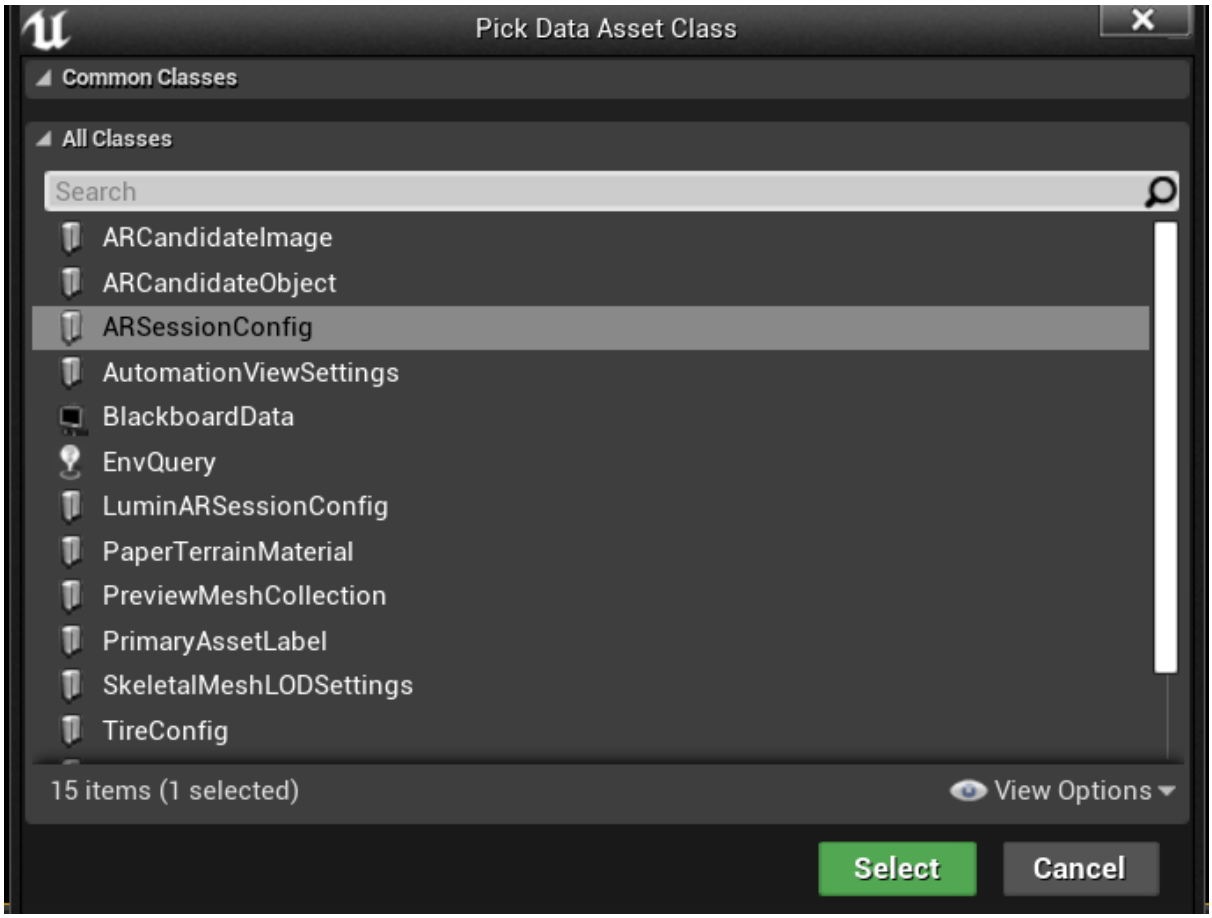
3) Creamos un nivel nuevo vacío en File > New Level, eligiendo “Empty Level”

4) Desde la izquierda creamos un nuevo Basic > Player Start arrastrándolo al nivel. Después lo seleccionamos y lo colocamos en las coordenadas (0, 0, 0)



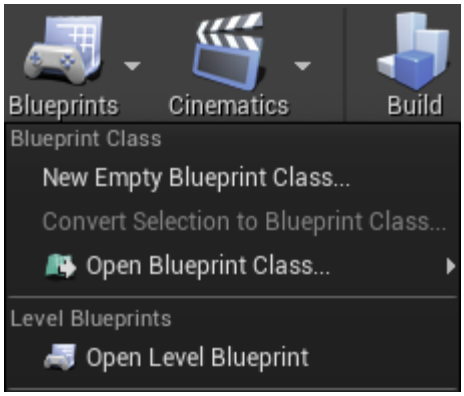
5) Haciendo click derecho abajo o clickando en “Add New/Import” o “Add New” dependiendo de la versión, pulsamos Miscelaneous > Data Asset y elegimos

“ARSessionConfig” de las opciones de la lista.



6) Doble click sobre la sesión de configuración que acabamos de crear, y pulsamos “Save” para guardarla.

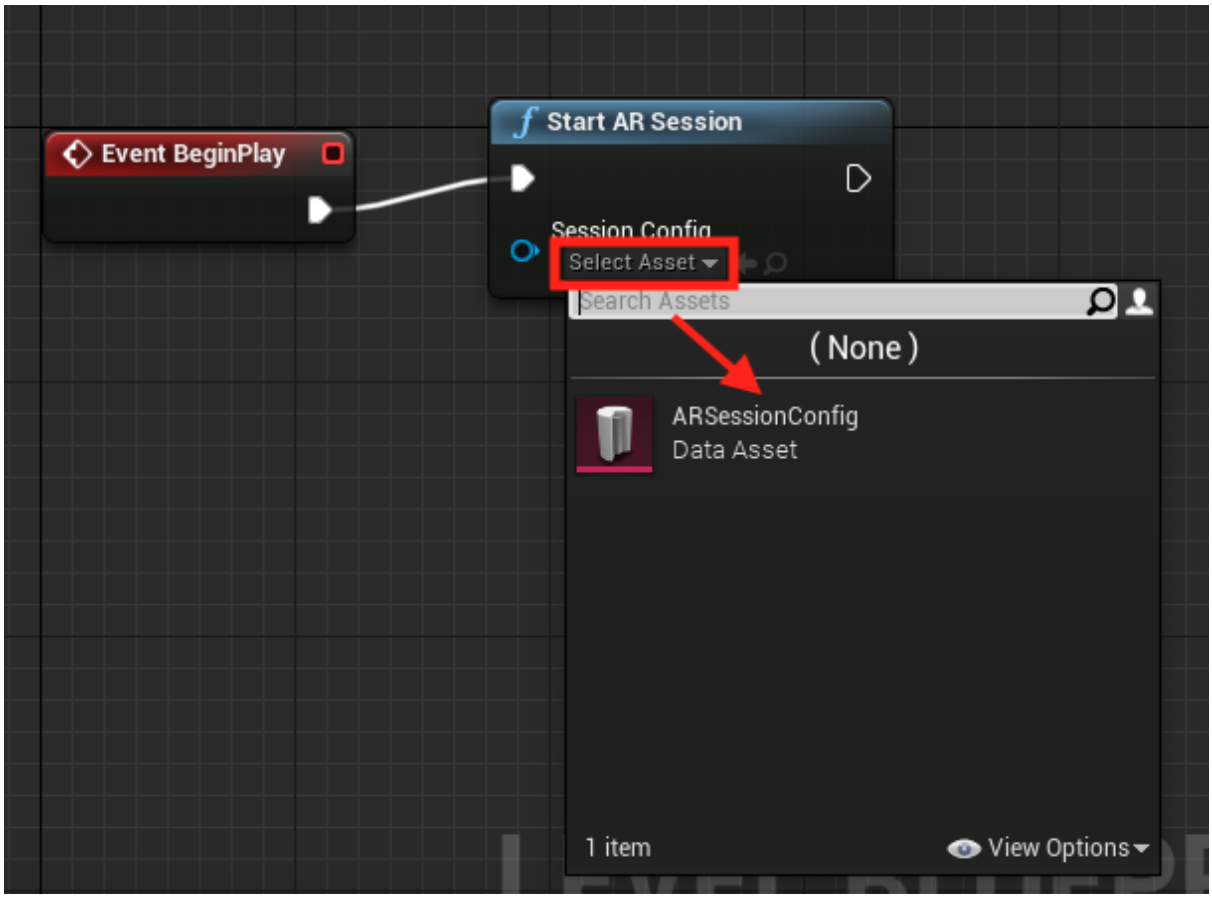
7) Ahora vamos a hacer que la sesión de Realidad Aumentada comience y termine con nuestro nivel. Pulsamos arriba sobre Blueprint > Open Level Blueprint. Este es el lugar en el que se escriben las órdenes para el funcionamiento general del nivel.



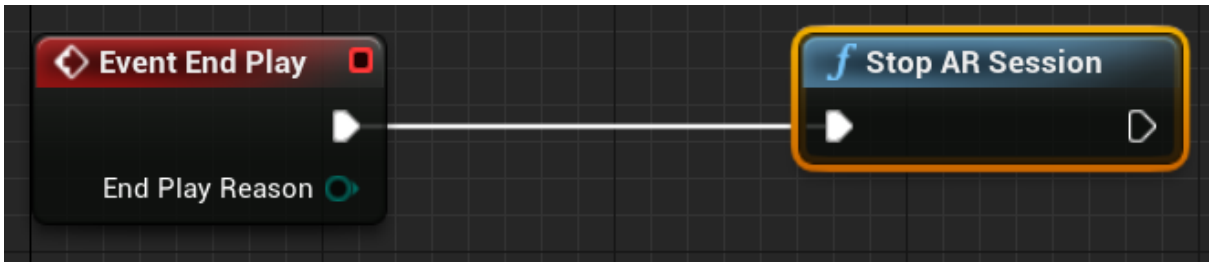
8) Vemos el nodo “Event Begin Play”, arriba tiene un triángulo que apunta hacia la derecha. Esa rama que sale del triángulo controla el orden de ejecución, y este nodo representa el evento “Comienza el juego”. De manera que

clickamos sobre el triangulito y arrastramos hacia la derecha, para soltar la línea en mitad del plano vacío. Nos aparece una lista de nodos posibles que representan la siguiente acción que debe tomar el programa, elegimos (buscamos) “Start AR Session”

9) Hemos creado así un nodo nuevo que comienza la sesión, en el campo “Session Config” elegimos la configuración de sesión que hemos creado antes.

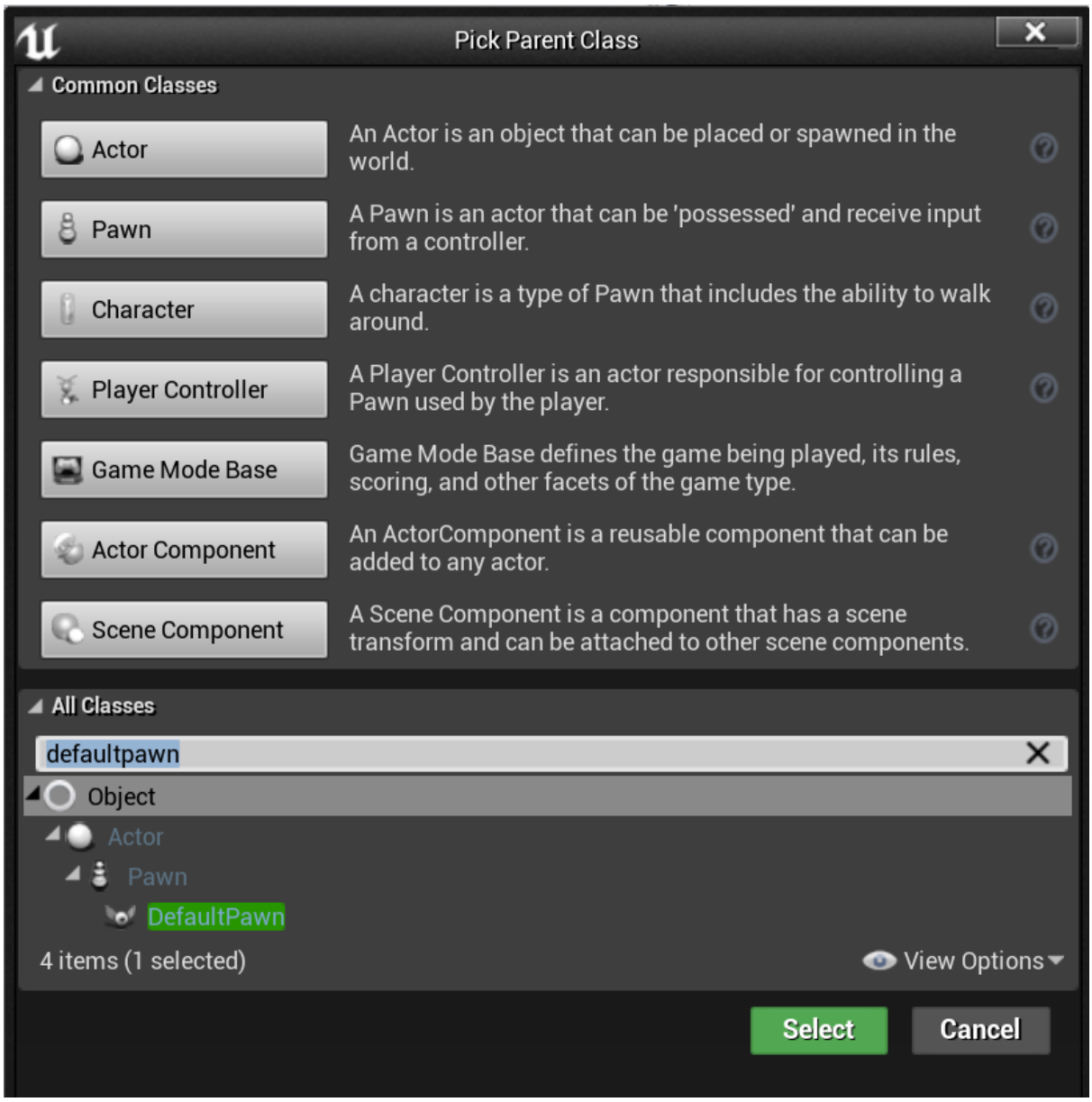


10) En otra zona del plano que quede vacía hacemos doble click y buscamos el nodo “Event End Play” que representa el evento de que termina el juego. Nuevamente arrastramos desde el triangulito y soltamos, para elegir “Stop AR Session”



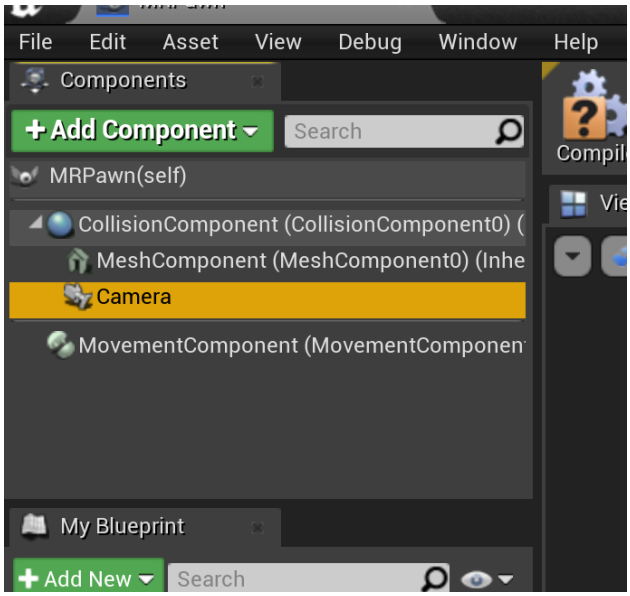
11) Pulsamos en “Compile” y “Save” para compilar y guardar estas nuevas directrices, podemos darle el nombre que queramos como “LevelBlueprint”. Cerramos esta ventana, volviendo a la ventana principal.

12) Vamos a crear al jugador en sí mismo, nuevamente Add New > Blueprint Class y abajo buscamos “DefaultPawn”, le damos el nombre que queramos, lo guardamos y le damos doble click para abrirlo en el editor.

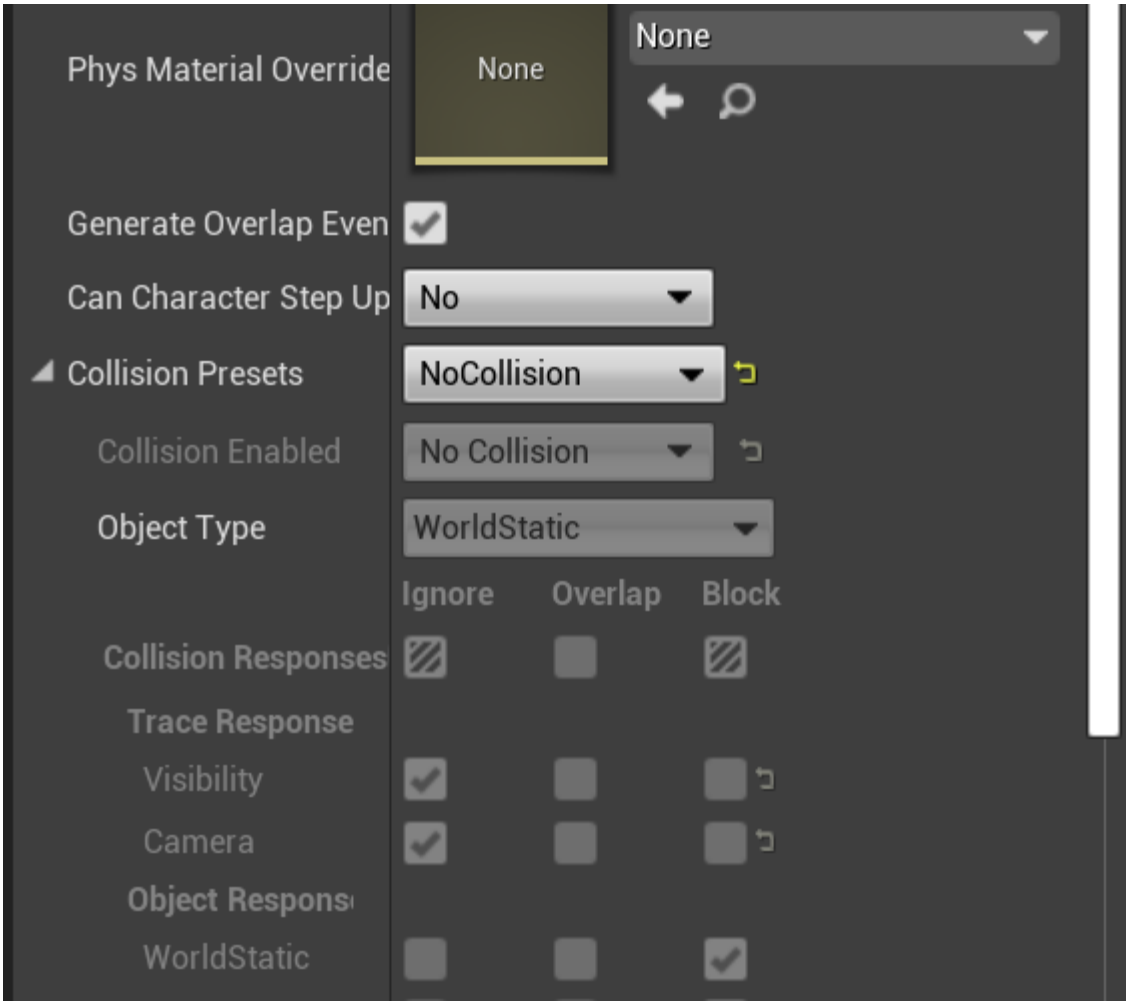


13) Arriba a la izquierda pulsamos en “Add Component” y buscamos “Camera”. Una vez creado nos aseguramos de que sea hijo del CollisionComponent para que la cámara se mueva con las gafas, para eso arrastramos “Camera” sobre el “Collisioncomponent”. Si no nos deja probablemente quiere decir que

ya es hijo, y no hace falta que lo arrastremos. Debería quedar así:

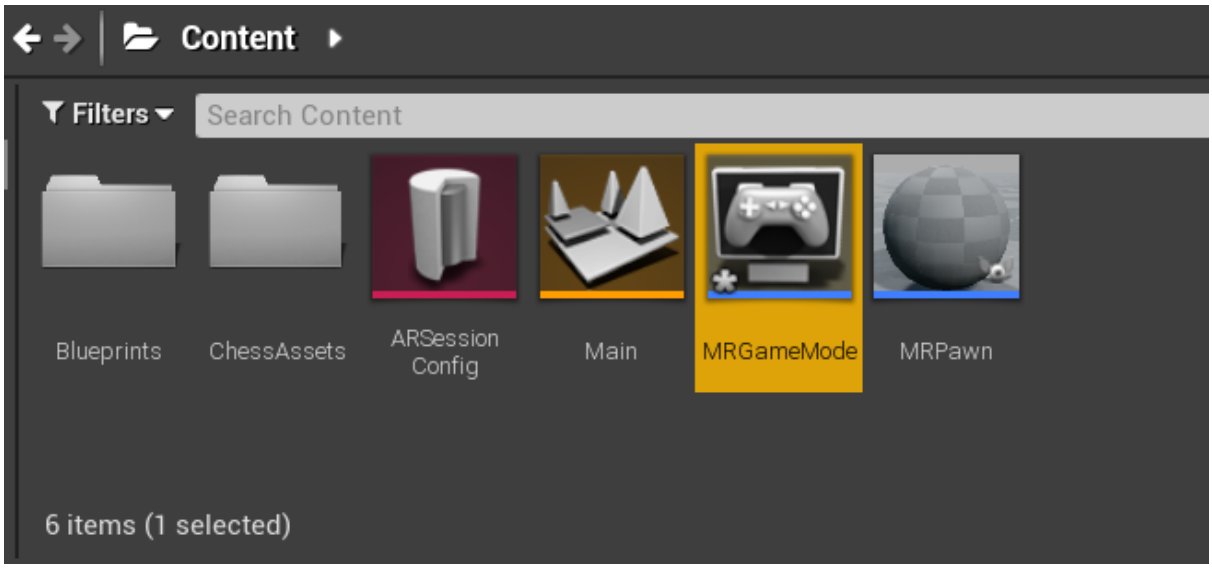


14) Haciendo un click sobre CollisionComponent nos aparecen a la derecha las propiedades de éste, buscamos “CollisionPreset” y elegimos “NoCollision”. Esto hará que la cámara no se pare al chocarse contra algún objeto 3D que haya en el espacio. Hacemos lo mismo con el mesh component.

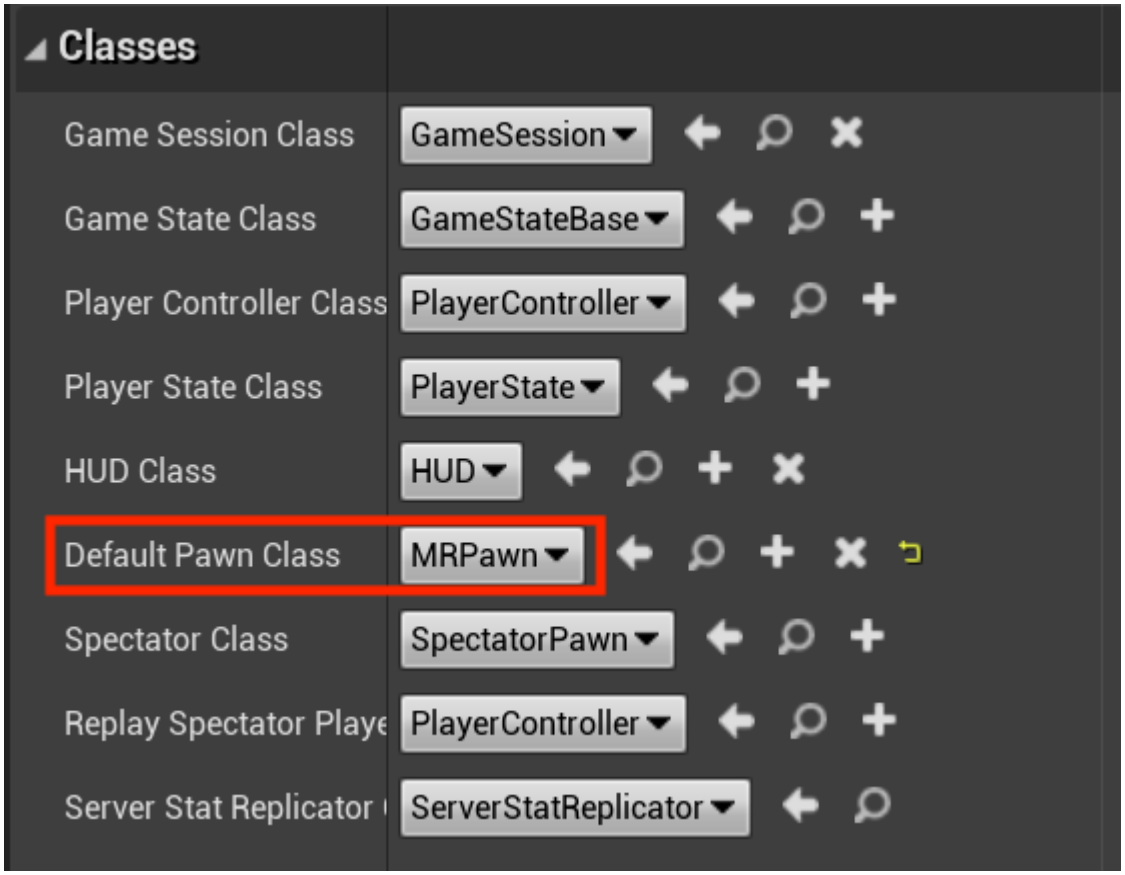


15) Compilamos, guardamos y cerramos la ventana.

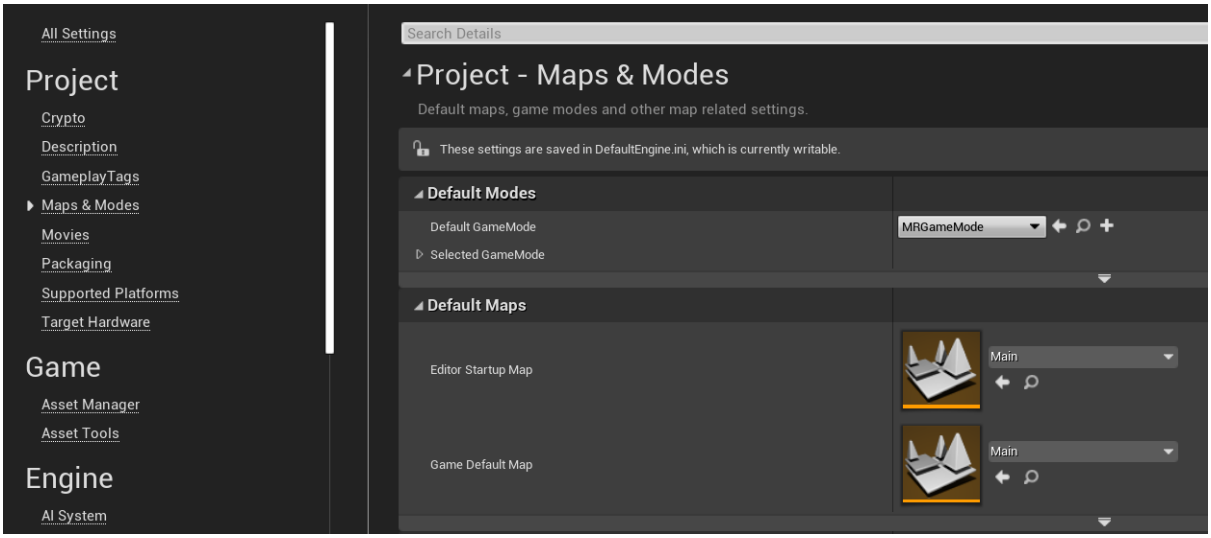
16) Por último vamos a crear un “Modo de Juego”. Add New > Blueprint Class > Game Mode Base.



17) Doble click sobre el modo de juego, y en las propiedades elegimos que el “Default Pawn Class” sea el Pawn que hemos creado antes. Cerramos esta ventana.

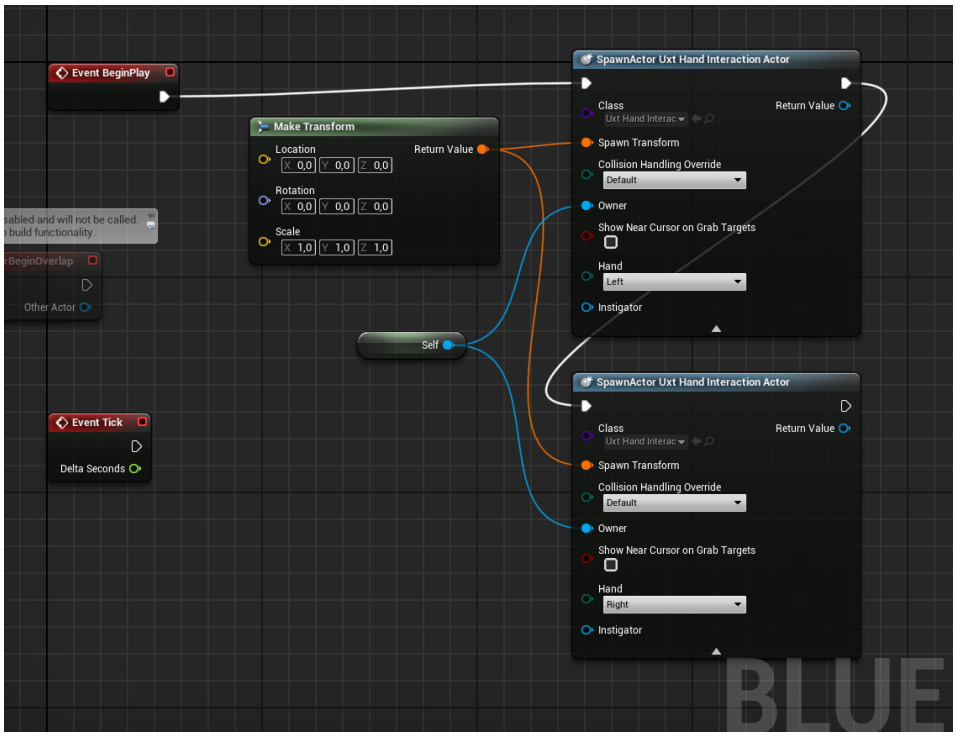


18) Por último elegimos que este modo de juego sea el predeterminado. Para ello nos vamos a Edit > Project Settings > Maps & Modes y elegimos que el “Editor Startup Map” y el “Game Default Map” sean el mapa (level) que hemos creado al principio. Elegimos el “DefaultGameMode” para que sea el modo de juego que hemos creado antes.



19) Guardamos el proyecto y hemos terminado. Notar que no hemos añadido ninguna fuente de luz y los objetos creados no se verán por defecto. Para ello la mejor solución que he encontrado hasta la fecha es que los objetos “emitan” luz. Para ello les creamos un material nuevo y en el campo shading model elegimos “Unlit”. En emissive color elegimos el color del material. De esta manera los materiales quedan autoiluminados, aunque no iluminen otros objetos.

20) ÚLTIMO PASO: no estoy seguro de que este paso sea completamente necesario y no lo he hecho en el template. En el MR Pawn, añadir al blueprint lo siguiente:


Event BeginPlay
Make Transform
Location
Rotation
Scale
Return Value
SpawnActor Uxt Hand Interaction Actor
Class
Spawn Transform
Collision Handling Override
Default
Owner
Show Near Cursor on Grab Targets
Hand
Left
Instigator
Self
Event Tick
Delta Seconds
Right
BLUE