UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

APLICACIÓN MÓVIL DE CREACIÓN MUSICAL Y APRENDIZAJE TEÓRICO

José Vicente Camarasa Sánchez

Tutor: José Manuel Mossi García

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2018-19

Valencia, 3 de Julio de 2019

Resumen

El presente trabajo de fin de grado consiste en la creación y desarrollo de una aplicación musical para Android e iOS a través de la plataforma Unity3D utilizando el lenguaje de programación C#. Algunos de los sonidos utilizados han sido proporcionados por el programa Ableton Live 10 y otros han sido grabados manualmente a través de instrumentos musicales en un estudio de grabación.

La finalidad de esta aplicación es dar la capacidad a cualquier persona que disponga de un smartphone, de poder tanto crear música con distintos instrumentos y escalas como aprender la teoría musical que esto conlleva, ya que el proceso de creación es totalmente visual.

Consiste en una herramienta versátil y práctica que permite al usuario plasmar sus ideas en cualquier lugar y momento, a partir de una interfaz intuitiva en la que se pueden escoger seis instrumentos diferentes y sus principales siete modos musicales mayores y menores, además de la escala correspondiente, lo cual genera miles de combinaciones posibles con las que la persona puede desarrollar su creatividad musical y da la posibilidad de llevar estas ideas a la práctica con instrumentos reales.

Resum

El present treball de fi de grau consisteix en la creació i desenvolupament d'una aplicació musical per a Android e iOS a través de la plataforma Unity3D utilitzant el llenguatge de programació C#. Alguns dels sons utilitzats han sigut proporcionats pel programa Ableton Live 10 i altres han sigut gravats manualment a través d'instruments musicals en un estudi de gravació.

La finalitat d'aquesta aplicació és donar la capacitat a qualsevol persona que dispose d'un smartphone, de poder tant crear música amb diversos instruments i escales com aprendre la teoria musical que açò comporta, ja que el procés de creació és totalment visual.

Consisteix en una ferramenta versàtil i pràctica que permeta a l'usuari plasmar les seues idees en qualsevol lloc i moment, a partir d'una interfície intuïtiva en la qual es poden escollir sis instruments diferents i els seus principals modes musicals majors i menors, a més de l'escala corresponent, el qual genera milers de combinacions possibles amb les quals la persona pot desenrotllar la seua creativitat musical i dóna la possibilitat de portar aquestes idees a la pràctica amb instruments reals.

Abstract

The present final project consists in the creation and development of a musical application for Android and iOS through the Unity3D platform using the C # programming language. Some of the sounds have been provided by the Ableton Live 10 program and others have been recorded manually through musical instruments in a recording studio.

The purpose of this application is to give the ability to anyone who has a smartphone to create music with instruments and scales and learning the musical theory that this implies since the process is totally visual.

It consists of a versatile and practical tool that allows the user to translate their ideas at any place and time from an intuitive interface where you can choose six different instruments and their seven major and minor musical modes, in addition to the corresponding scale, which generates thousands of possible combinations with which the person can develop their musical creativity and offers the possibility of bringing these ideas into practice with real instruments.

Índice

Capítulo 1. Introducción

1.1 Cita

"La música se oye y se siente, ya que si algo provoca la música es una explosión de emociones y de recuerdos, ¿a qué se debe todo esto? Durante mucho tiempo esta cuestión fue un misterio para la ciencia, pero, hoy tenemos la respuesta y es bien sencilla, tiene que ver con nuestras neuronas. La música nos gusta y nos produce un inmenso placer porque activa el sistema límbico y baña nuestro cerebro en endorfinas, las mismas hormonas que segregamos cuando damos un beso apasionado o nos zampamos un trozo de tarta de chocolate.

De las endorfinas se ha descubierto que nos dan motivación ante la vida, que producen alegría y optimismo, que disminuyen el dolor o que por lo menos, ayudan a olvidarlo, que contribuyen a la sensación de bienestar y que estimulan los sentimientos de gratitud y de satisfacción existencial. Además, las respuestas placenteras que nos evoca la música se relacionan con la actividad de las regiones del cerebro implicadas en los mecanismos de recompensa y en los de emociones. Observaciones como esta sugieren que, aunque la música no es estrictamente necesaria para la supervivencia de la especie humana, si constituye un beneficio significativo para nuestro bienestar físico y mental, es más, si cantamos o bailamos en público las endorfinas que segregamos nos ayudan a establecer lazos sociales, a sentirnos parte de la manada y nos hace más atractivos a ojos de los demás.

En definitiva puede constituir una estrategia de supervivencia, y esto es algo que remonta desde bien antiguo. Hoy sabemos que la música surgió antes que el lenguaje, y sabemos que fue utilizada por nuestros antepasados homínidos cuando todavía no sabían vocalizar.

Sin duda la música desata un sinfín de ventajas para nuestro bienestar y para nuestras relaciones sociales, ¿quieres sacarle el máximo partido?, cierra los ojos." (Eduard Punset, 2012)

1.2 Motivación

La principal motivación que me ha llevado a desarrollar este proyecto ha sido mi gran afición por la música, que desde muy pequeño siempre ha estado acompañándome y por la tecnología, razón por la cuál elegí este grado. Ambas disciplinas me parecen muy interesantes y con un gran potencial para ser unidas. Hoy en día, con un dispositivo electrónico que llevamos siempre encima como es un smartphone, con potencia y características suficientes para mover aplicaciones musicales, tenía la oportunidad y necesidad de crear algo que uniera los dos mundos mencionados y que me ayudara a desarrollar todos los conceptos dados a lo largo del grado.

Por este motivo, he querido llevar a cabo el desarrollo de esta aplicación para que cualquier persona en cualquier parte del mundo, se la pueda descargar a su móvil para crear y desarrollar todo tipo de música de forma fácil.

1.3 Objetivos

El objetivo de este proyecto es la creación de una aplicación en el software Unity, desarrollada para plataformas móviles, mediante la cual se da la capacidad para crear música con diferentes instrumentos utilizando una pantalla táctil. Además de dar la posibilidad de elegir la escala musical que se prefiera entre los principales modos mayores y menores, además de poder añadir diferentes efectos creando de esta forma miles de posibilidades en notas y sonidos. La aplicación será capaz de grabar, pista por pista, lo que se está tocando en directo a modo de capas, que se podrán reproducir simultáneamente para crear una composición con los distintos instrumentos musicales.

A parte del objetivo de la creación de música en cualquier lugar, esta aplicación pretende ayudar a comprender la teoría musical de las diferentes escalas, ya que todas las notas saldrán con su nombre en la interfaz visual.

También, se tiene que tener en cuenta el objetivo de la optimización en las funciones que reproducen el sonido cuando se pulsa la pantalla, ya que el tiempo de retraso aquí hará que la experiencia del usuario no sea del todo placentera.

1.4 Plataformas utilizadas

A lo largo del desarrollo del proyecto, se han utilizado los programas Ableton Live 10, un secuenciador de audio (también llamado DAW) y el Unity 2019, una plataforma de desarrollo de videojuegos.

1.4.1 Historia de los DAW

Una estación de trabajo de audio digital o comúnmente llamado DAW, por sus siglas en inglés de “Digital Audio Workstation”, consiste en un sistema electrónico o una aplicación de software dedicada a la grabación, edición y producción de audio digital mediante el uso de un software de edición de audio. Los software DAW (también llamados secuenciadores) se utilizan en todo tipo de producciones o grabaciones de música, podcast, radio, televisión, producción de cine y todo tipo de diseño o efectos de sonido. Esto se debe a que los precios de los principales DAW son asequibles y tanto un profesional como un aficionado pueden adquirir el mismo producto ya que incluso los ordenadores estándar de hoy en día tienen suficiente potencia como para hacerlos funcionar de manera básica sin problemas.

Antes de la aparición de los primeros DAW, los músicos, estudios de grabación e ingenieros, grababan el audio en un magnetófono de bobina abierta o simplemente magnetofón, que permite registrar en un soporte magnético adherido a una cinta plástica, sonidos mediante el procesamiento de señales eléctricas provenientes de micrófonos.

Fue indispensable para toda la industria musical, sobre todo para el rock and roll, ya que era un gran fenómeno que comenzó a crecer durante los años 50 en adelante y este medio fue el estándar hasta finales de la década de los 70, cuando llegó la era digital.

La empresa Soundstream, fundada en 1975 por Thomas Stockham, pionero en el desarrollo del audio digital, hizo los primeros intentos en desarrollar un DAW, pero no dio resultados duraderos, ya que el hardware del ordenador era muy limitado; velocidades de disco insuficientes, capacidad de procesado extremadamente lenta y precio del almacenamiento muy alto. Aunque en 1979, una compañía llamada "Fairlight" fundada en 1975 por Peter Vogel y Kim Ryrie desarrolló el Fairlight CMI (Computer Musical Instrument), considerado el primer DAW de la historia. A pesar de su elevado precio, se utilizó por muchos músicos de los años ochenta, como por ejemplo David Bowie, en España fue utilizado por primera vez por Tino Casal.

A comienzos de los años 80, las compañías comenzaron realmente a impulsar la idea del software DAW, debido a la aparición de computadoras como Apple II, Atari ST y Commodore Amiga, ya que tenían mejor capacidad de procesamiento y podían llevar a cabo tareas de edición de audio digital.

En 1985 se fundó la empresa Opcode por Dave Oppenheim, la cual un año más tarde sacaría el secuenciador y la interfaz MIDIMAC, un producto de gran éxito dentro del mundo del sonido digital considerado de los primeros secuenciadores MIDI[1] comercializados públicamente. Finalmente el producto se conocería más tarde como "Vision", siendo muy revolucionario y utilizado por muchos artistas conocidos del momento.

Otro de los primeros desarrolladores fue Mark Of The Unicorn (MOTU) con su software llamado "Professional Composer" lanzado en 1984. En 1985 la compañía lanzó un secuenciador llamado Performer, basado en la plataforma Macintosh aprovechando al máximo muchas de las características del protocolo MIDI, como dirigir muchos instrumentos simultaneos, elegir qué notas tocar, su volumen y durante cuánto tiempo. Más adelante, en 1990 añadió la capacidad de poder sincronizar el audio digital y relanzó el producto modificándole el nombre a "Digital Performer", todavía muy usado a día de hoy.

La siguiente empresa en darse a conocer y que puso las bases del audio digital hasta el día de hoy fue Steinberg, fundada en 1984 en Alemania por Karl Steinberg y Manfred Rürup. Ambos desarrollaron "Pro 16", un secuenciador MIDI de 16 pistas con una interfaz clara de una sola página y grabación en tiempo real para el Commodore 64. Más tarde desarrollaron el "Pro 24" para la plataforma Atari ST. Esta plataforma fue revolucionaria porque resultaba mucho más barata que la plataforma Macintosh, teniendo unas características elevadas para la época, además incluía interfaz MIDI de serie, lo que hizo que esta plataforma fuera el núcleo de la mayoría de estudios audiovisuales. Finalmente la empresa desarrolló "Cubase" para Atari en 1989, software que aún a día de hoy se mantiene como uno de los grandes líderes en la industria musical.

Junto a "Cubase", otro programa que se popularizó fue de la empresa Emagic, una compañía de software con sede en Alemania fundada en 1992 por Sven Junge, Chris Adam y Gerhard

1 Estándar tecnológico que describe un protocolo, una interfaz digital y conectores que permiten que varios instrumentos musicales electrónicos, ordenadores y otros dispositivos relacionados se conecten y comuniquen entre si.

Lengeling. Adam y Lengeling habían desarrollado anteriormente los programas “Scoretrack” y “Supertrack” para Commodore 64 en la empresa C-LAB y luego “Creator” y “Notator SL” para Atari ST. Bajo la nueva compañía formada, lanzaron el “Notator Logic” en 1993 para Atari ST y Mac, más tarde también para Microsoft Windows.
El nombre del producto fue finalmente acortado a “Logic”. Más tarde, en 2002, la empresa Emagic fue comprada por Apple y cambió el nombre del producto a “Logic Pro”, el standard de Apple que existe a día de hoy.

La existencia y evolución de los DAW ha permitido a los músicos no solo crear música de manera más eficiente, sino que también ha ayudado a desencadenar una revolución creativa en el desarrollo de los sonidos, fomentando la aparición de nuevos géneros musicales que años atrás no existían.

1.4.2 Ableton

Ableton fue fundada en 1999 por Gerhard Behles y Robert Henke, quienes lanzaron su primer producto en 2001 llamado “Live”, una estación de trabajo de audio digital y secuenciador de audio que permitía a los músicos almacenar y ejecutar fácilmente muestras de sonido durante sus actuaciones, permitiendo que pudieran trabajar de manera más improvisada según lo que el público demandara. Esto fue revolucionario y permitió que muchas personas no profesionales pudieran acceder a hacer música con sus ordenadores en directo.

El programa fue ampliamente adoptado por muchos profesionales cambiando radicalmente la forma en la que la música electrónica se interpretaba en vivo, dando pie a que se generara un movimiento que impulsó la cultura de los festivales de música electrónica en todo el mundo.

La idea principal que hizo surgir este programa, no fue su comercialización, sino la propia necesidad que tenían los dos músicos para actuar en sus propios directos de la formación “Monolake” a mediados de los años 90. A finales de los años 90, se unió Bernd Roggendorf alentándoles a convertir el código de su programa escrito en “Max”, en un programa más generalizado para la venta al por menor. Así es como en 2001 nació Ableton Live.

Popularizar un nuevo software no era tarea fácil, pero Henke y Behles tenían muchos contactos de artistas en la naciente escena de Berlín ya que formaban parte de ella. Tenían relación con artistas cuya popularidad estaba creciendo rápidamente y con ello más demanda de espectáculos en vivo. Por lo tanto, la necesidad de una herramienta amigable con el usuario para actuaciones en vivo y los contactos de ambos miembros fundadores en la escena electrónica fue crucial para la popularización de este software, que en poco tiempo se convirtió en el estándar de la industria de la música electrónica.

Este programa ha seguido evolucionando año tras año hasta convertirse en un referente no solo de las actuaciones en directo de música electrónica sino de la producción musical. Es utilizado en todo tipo de producciones donde se requiera editar o grabar un sonido y ofrece mucha estabilidad consumiendo muy pocos recursos. Posee dos modos diferentes de creación; el “arrangement view”, común a todos los secuenciadores con el eje de tiempo horizontal y las diferentes pistas dispuestas verticalmente y el “session view”, el original de sus raíces, donde la música creada se organiza en clips que se pueden pausar o reproducir de forma más improvisada, un modo pensado más para el directo.

Figura 1: Logo Ableton

Figura 2: Logo Ableton Live

1.4.3 Unity

Unity es un motor de videojuegos multiplataforma que permite la renderización con gráficos 2D o 3D, ofrece un motor de físicas, detección de colisiones, diseño de sonido, scripting en C#, animaciones, inteligencia artificial y más implementaciones que hacen que el proceso de creación de un videojuego resulte más sencillo que partir desde cero.

El software fue creado por Unity Technologies en 2005 en Dinamarca, después de que un grupo de desarrolladores (David Helgason, Nicholas Francis y Joachim Ante) crearan su primer juego, GooBall, que no tuvo ningún éxito, pero sus creadores pensaron que el motor y las herramientas desarrolladas para crearlo podrían tener valor para un público que buscara un producto asequible, naciendo de esta manera el motor de videojuegos Unity.

El objetivo de Unity era hacer el desarrollo de videojuegos más accesible a todo el público.

Inicialmente, se lanzó para la plataforma Mac OS X y al siguiente año de ser lanzado, fue nombrado subcampeón en la categoría de "Mejor Uso de Gráficos" de Mac OS X en el Apple Desing Awards. Después del éxito, se agregó el soporte para Windows.

Unity siempre se ha mantenido en constante evolución, con una gran actualización cada cierto tiempo, pero constantemente añadiendo nuevas herramientas para facilitar o dar más posibilidades a su uso.
La versión 2.0 llegó en 2007 con multitud de nuevas funciones, como un motor de terreno optimizado para entornos 3D detallados, sombras dinámicas en tiempo real, luces y focos direccionales, reproducción de vídeo y muchas más funciones, como una red online para que los desarrollares crearan juegos multijugador.

En 2008 se agregó soporte para iPhone y más tarde, con la actualización 3.0 en 2010, para varias consolas de videojuegos y Android. Además de añadir nuevas funciones como un editor de árbol, renderizado de fuentes nativas o filtros de audio.

En 2012 ya era mundialmente reconocido y más de 1.3 millones de desarrolladores lo estaban utilizando. Las encuestas de medios especializados lo situaban como el mejor motor de juegos para plataformas móviles. Ya en 2014, Unity ganó el premio al "Mejor Motor" en los premios anuales especializados "Develop Industry Excellence Awards" del Reino Unido. En ese mismo año, se lanzó la versión 4.0, agregando "DirectX 11", soporte para "Adobe Flash" y nuevas herramientas de animación llamadas "Mecanim".

Para el año 2015 se lanzó la versión 5.0, se lanzaron multitud de herramientas como iluminación global en tiempo real, vistas previas del mapeo de luz, Unity Cloud, un nuevo sistema de audio, el motor de física "Nvidia PhysX3.3". También se presentó Cinematic Image Effects, que ayudaba a que los juegos se vieran menos genéricos. La versión 5.6 añadió nuevos efectos de iluminación y de partículas, actualizó el rendimiento general del motor y agregó soporte nativo para Nintendo Switch, Facebook Gameroom, Google Daydream VR y la API de gráficos Vulkan. También se presentó un reproductor de vídeo 4K capaz de ejecutar vídeos en 360 grados para realidad virtual.

En diciembre de 2016, Unity Technologies anunció que cambiarían el sistema de numeración de versiones por uno basado en el año de lanzamiento para alinear el control de versiones dado que sus actualizaciones eran constantes.
Así nació Unity 2017, incluyendo un motor de procesamiento de gráficos en tiempo real, gradación de colores, construcción de mundos, análisis de operaciones en vivo e informes de rendimiento. La versión 2017.2 añadió funciones más allá del mundo de videojuegos, como la herramienta Timeline, que permitía a los desarrolladores arrastrar y soltar animaciones en los

juegos y Cinemachine, un sistema de cámara inteligente dentro de los juegos, también se incluyeron dentro del motor de Unity herramientas de 3DS Max y Maya de Autodesk.

Con Unity 2018 llegaron las actualizaciones para crear gráficos de alta calidad en todo tipo de plataformas, incluyendo realidad virtual, realidad aumentada y realidad mixta. Se incluyeron herramientas de aprendizaje automático como aprendizaje por imitación, donde los juegos aprenden de los hábitos de los jugadores reales a la vez que añadieron compatibilidad con Magic Leap.

En la versión 2019, que es la que se ha utilizado en este proyecto, se han integrado mejoras en los gráficos y la iluminación, pero lo más importante, sobre todo para el presente trabajo es que ha habido mejoras en la optimización del audio y estabilidad en las plataformas móviles. Al ser una aplicación donde la música reacciona con las pulsaciones, con Unity 2019 se ha podido hacer sin problemas este proyecto donde el sonido responde sin apenas milisegundos de retraso al tocar la pantalla.

Figura 3: Logo Unity

Capítulo 2. Metodología

Un proyecto con este tipo de envergadura, necesita una planificación detallada de cada una de las tareas que se van a llevar a cabo para que los diferentes objetivos propuestos se alcancen con éxito.

2.1 Distribución de tareas

A continuación se procede a detallar las distintas tareas en las que se ha dividido el proyecto y la distribución temporal de las mismas.

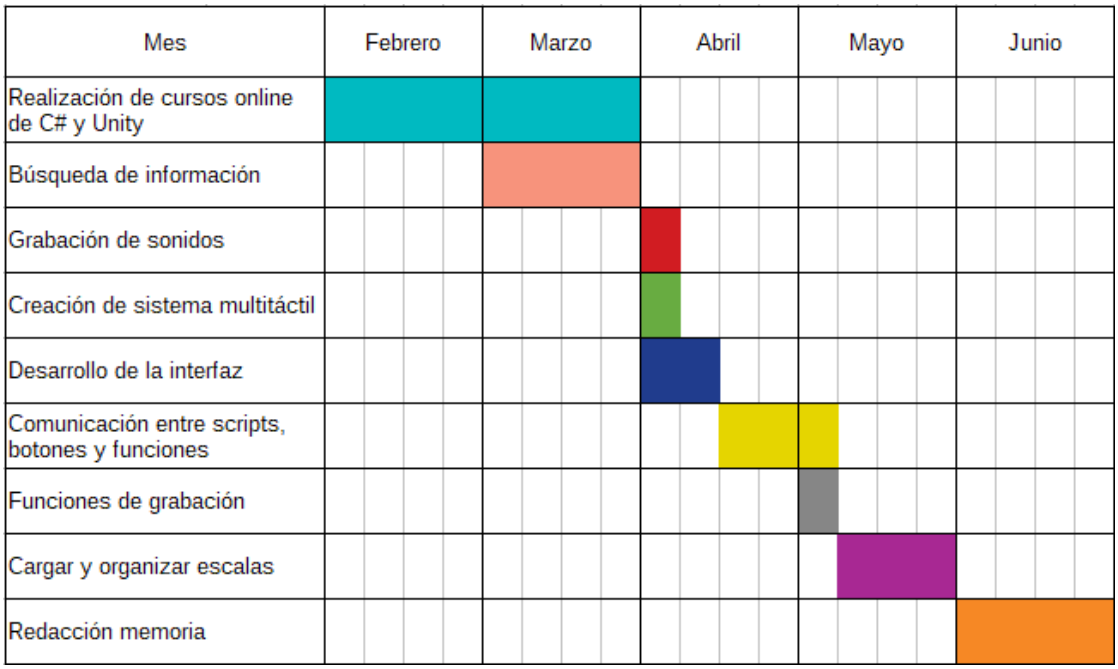

Tabla 1: Diagrama de Gantt del planteamiento inicial

Este planteamiento ha ayudado mucho a llevar un orden y que el proyecto se pueda realizar con éxito, pero aún así la distribución temporal de algunos apartados ha variado un poco. A continuación se detalla la planificación y la variación de la misma en el caso de que la haya habido:

Realización de cursos online:
Antes de la iniciación del proyecto se han realizado diferentes cursos online de Unity. Primero se ha llevado a cabo el de "Introducción al desarrollo de videojuegos con Unity" de UPValenciaX en la plataforma online de edX con un contenido de 18 horas.
Después se han seguido tres tutoriales oficiales de nivel de iniciación e intermedio en la página oficial de Unity. Además de continuar con más cursos de Unity en la plataforma de YouTube. Conjunto con esto, se ha llevado a cabo un curso en la plataforma YouTube de C# de 16 horas, ya que era un lenguaje de programación que nunca había utilizado a pesar de tener similitudes con los aprendidos en el grado.
Este apartado ha tenido una correcta planificación, y se ha desarrollado a tiempo.

Búsqueda de información:
Apartado fundamental durante todo el trabajo, tanto antes de empezar a desarrollar el proyecto para saber cómo llevarlo a cabo, como durante el proyecto para ir solucionando todos los

problemas que constantemente van surgiendo pero que al fin y al cabo son lo que te hacen aprender realmente.
La búsqueda de información ha sido necesaria constantemente ya que, a pesar de haber hecho los múltiples cursos y tutoriales, esto no es suficiente para llevar a cabo aplicaciones propias. Cada apartado ha requerido constantemente de búsqueda de información nueva.

Grabación de sonidos:
Los sonidos que se reproducen en la aplicación al usar los controles táctiles se han obtenido en su mayoría del software Ableton Live, excepto la guitarra eléctrica que se ha grabado personalmente en un estudio propio mediante el emulador de amplificadores y efectos Kemper.
En este apartado no ha habido ningún contratiempo y se ha podido realizar sin problemas.

Creación de sistema multitáctil:
En la aplicación hay 8 cubos representando las 7 notas de la escala y la primera octava, en algunas ocasiones se tendrá que pulsar con varios dedos a la vez para crear acordes, por lo tanto se necesita un sistema multitáctil.
La creación de este sistema se ha alargado una semana más debido a que las funciones táctiles de serie en Unity no permiten que haya reacción pulsando diferentes zonas simultáneamente, por lo tanto se han tenido que desarrollar mediante la búsqueda de información.

Desarrollo de la interfaz:
La interfaz tiene un estilo propio y es sencilla e intuitiva para que todo el mundo pueda utilizarla sin problemas y sin previas instrucciones.
A pesar de que en un principio se había desarrollado una interfaz válida, el software Unity se tuvo que actualizar durante el desarrollo de la aplicación ya que la nueva actualización permitía un mejor funcionamiento de las herramientas de sonido. Esto hizo que se desajustara toda la interfaz y se tuviera que hacer de nuevo.

Comunicación entre scripts, botones y funciones:
Cada botón tiene que tener su función correctamente asignada para el buen funcionamiento de la aplicación.
Este apartado se ha tenido que ir modificando constantemente más de lo esperado ya que cada pequeño detalle tiene que asignarse para el correcto funcionamiento.

Funciones de grabación:
En este apartado se tiene que dar la capacidad de poder grabar lo que el usuario crea y poder reproducirlo luego junto con más instrumentos.
Implementando las funciones de grabación es donde más problemas ha habido a la hora de la realización, ya que no tenía conocimiento de todo el proceso que tiene que hacer un script para coger los datos de audio y guardarlos, debido a esto, este apartado se alargó más de lo esperado.

Cargar y organizar escalas:
Hay 168 escalas que tienen que escribirse y cargarse una a una sin ningún error, por lo tanto es una tarea que requiere de tiempo.

En este apartado no ha habido problemas, únicamente se ha desarrollado un poco más tarde de lo deseado, pero en el tiempo establecido.

Redacción de la memoria:
Este apartado es indispensable para la evaluación, por lo tanto hay que incluirlo en la planificación del desarrollo.
Se ha tomado nota sobre todo el proceso seguido a lo largo del desarrollo de la aplicación, de este modo es más fácil la redacción final, ya que se trata de un texto extenso y con muchos detalles que si no se tienen en cuenta antes, pueden perderse a la hora de la redacción.

Este proceso se ha desarrollado con éxito en el tiempo establecido.

2.1.1 Distribución temporal final

A lo largo del desarrollo de la aplicación ha habido varios percances que han hecho que el planteamiento inicial varíe un poco. A pesar de esto ha sido un planteamiento bastante exitoso y que ha ayudado a llevar a cabo correctamente todas las tareas en el tiempo que más o menos se había definido.
A continuación se incluye un diagrama de Gantt con la duración que ha tenido cada tarea finalmente.

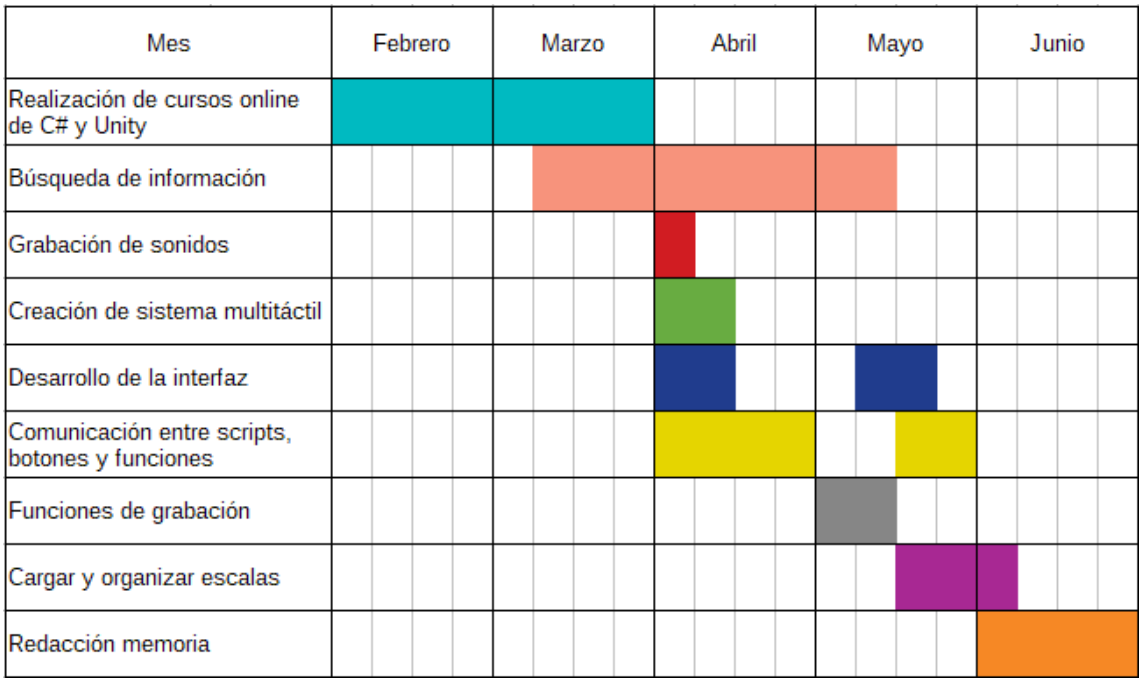

Tabla 2: Diagrama de Gantt final

Capítulo 3. Desarrollo

3.1 Obtención de los sonidos.

Lo primero que se ha llevado a cabo para el desarrollo del proyecto ha sido la obtención de los sonidos utilizados en la aplicación. Estos han sido casi en su totalidad grabados en Ableton Live 10.

Se han grabado piano, violín, sintetizador, batería y bajo. El proceso de grabación ha sido el mismo para todos pero cambiando el preset de sonido al instrumento correspondiente. Para grabar el piano se ha utilizado el preset Grand Piano utilizando el piano roll, que es el piano digital que puedes tocar dentro del programa Ableton con un simple clic en la pantalla, eligiendo la nota y la duración de la misma. Aquí se presenta un ejemplo con la nota F# en su segunda octava, o la equivalente en la nomenclatura que se utiliza en España Fa#.

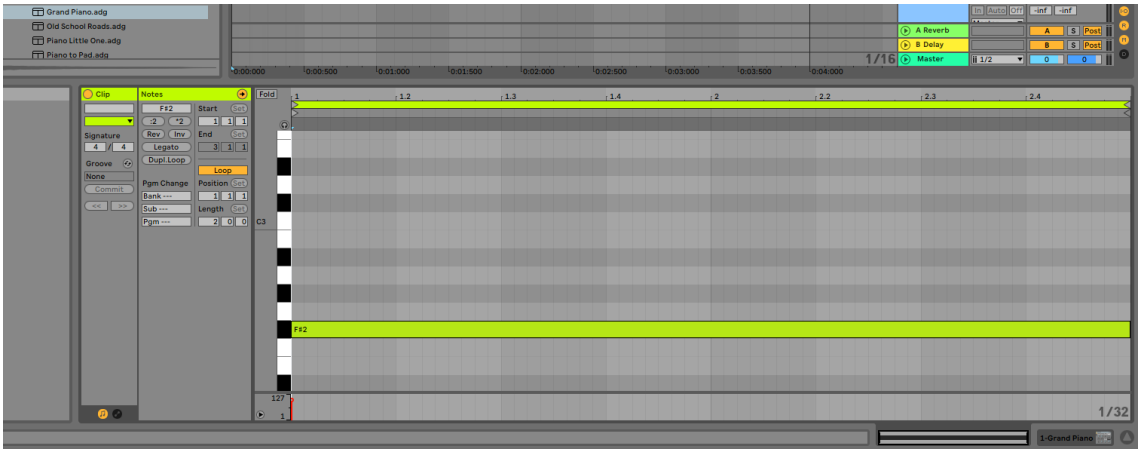

Figura 4: Piano roll en el programa Ableton Live 10 con la nota F#2

Se ha utilizado esta nomenclatura ya que es la utilizada universalmente y el propósito de este trabajo es que pueda ser una aplicación utilizada en cualquier país. Para la correcta comprensión de todos, el equivalente de las notas es:

C	D	E	F	G	A	B
Do	Re	Mi	Fa	Sol	La	Si

Figura 5: Equivalente de nomenclatura de notas musicales universal

Las notas grabadas han sido desde F#2 hasta F4, pasando por todas las siguientes que se pueden observar en este esquema:

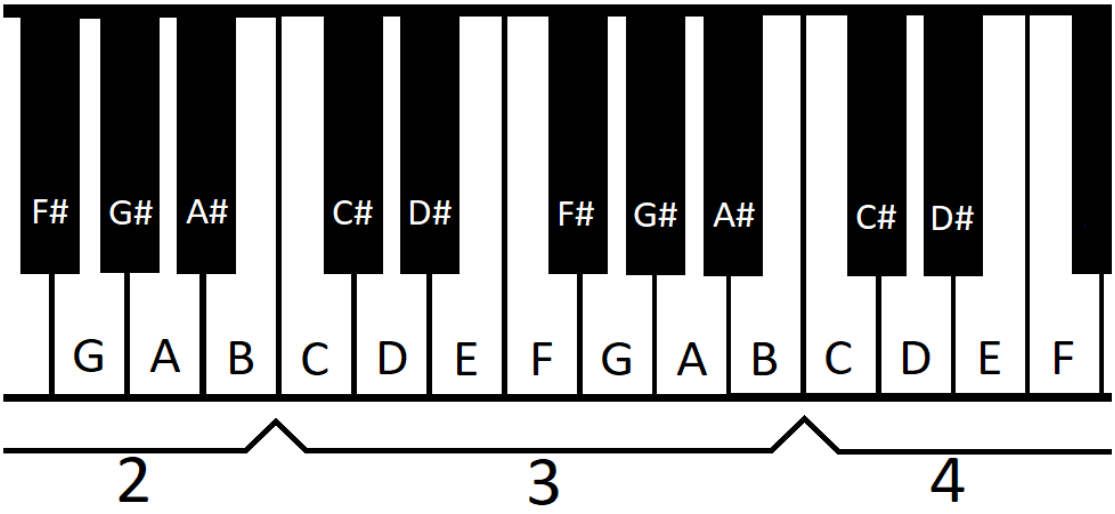

Figura 6: Rango de notas musicales utilizado en el desarrollo de la aplicación

De esta manera se obtiene un rango de notas con las que se pueden tener todas las escalas que más adelante se utilizarán. Se ha optado por empezar la escala en una octava más abajo cuando la nota elegida es superior a F, ya que así el sonido final de éstas se queda más centrado y no se van a la cuarta octava quedando demasiado agudas para ser las notas bases.

En el proceso de grabación, se ha utilizado una automatización para crear un fade out y que el sonido no acabe de forma abrupta resultando molesto o artificial. De este modo se obtienen 4 segundos de sonido en los que creamos una sensación orgánica y natural del mismo audio.

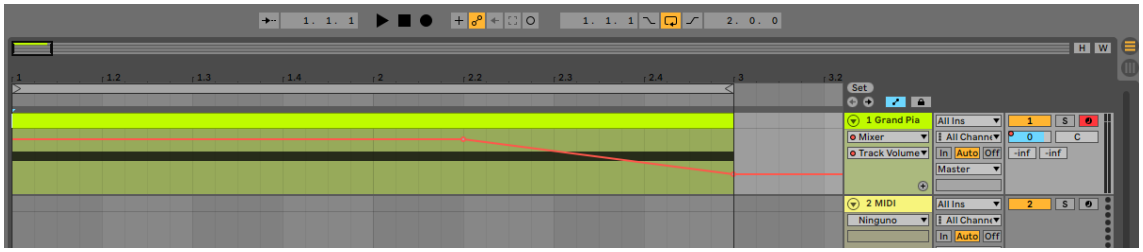

Figura 7: Automatización de fade out utilizada en la grabación de cada nota en Ableton Live

Una vez tenemos el sonido deseado se exporta en formato wav, ya que Unity lee de forma más eficiente el archivo si no tiene ningún tipo de compresión, y en esta ocasión, la velocidad de lectura del audio será de vital importancia, ya que la aplicación tendrá que responder a tiempo real con el sonido cuando se pulsan ciertas zonas de la pantalla.

Este proceso se ha de repetir para todas las notas del rango anteriormente nombrado, lo que son 24 notas exportadas por instrumento. En este caso, se han exportado utilizando Ableton Live cinco instrumentos: piano con el preset Grand Piano, violín con el preset Violin Section, sintetizador con el preset Jarble Perator, bajo con el preset Electric Bass y batería con el Kit StudioBasic. En todos los instrumentos el proceso ha sido el mismo, exceptuando la batería ya que aquí no tenemos notas, y lo que se ha hecho es cargar 8 sonidos diferentes de percusión.

Para la grabación de la guitarra, aprovechando conocimientos personales sobre ella, se ha grabado en un pequeño estudio de música personal, utilizando el sistema Kemper, que se trata de un emulador digital de todo tipo de amplificadores y de efectos diferentes. El sonido recogido ha sido grabado en el mismo software de grabación utilizado para todos los demás sonidos.

Una vez obtenidos todos los archivos wav de cada nota de los diferentes instrumentos, se han introducido organizados en diferentes carpetas para cada instrumento dentro de la carpeta Assets/Resources en Unity para posteriormente usarse en los scripts.

3.2 Descripción

Elementos utilizados de Unity:
Antes de describir las partes principales de Unity, se va a proceder a desarrollar una breve descripción sobre las herramientas básicas que se utilizan dentro del software.

GameObjects:
Los "GameObjects" son objetos fundamentales en Unity que representan personajes, props (modelos 3D) y el escenario. Estos no logran nada por sí mismos pero funcionan como contendedores para "Components", que implementan la verdadera funcionalidad.

Un GameObject siempre tiene el componente "Transform" adjunto (representa la posición y orientación) y no es posible eliminarlo. Los otros componentes que le dan al objeto su funcionalidad pueden ser agregados del menú "Component" del editor o desde un script.

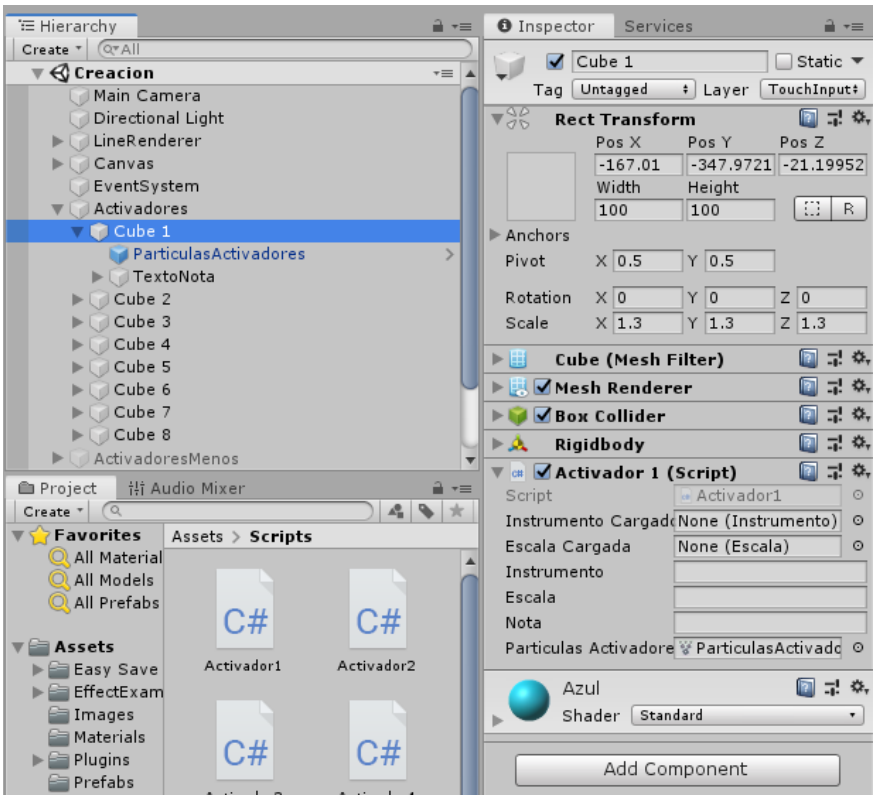

Figura 8: GameObject de un cubo con sus diferentes componentes a la derecha en la interfaz de Unity

Prefabs:
El sistema de prefabs te permite crear, configurar y guardar GameObjects de forma completa con sus components, sus valores concretos y con sus GameObjects hijos que le pertenecen como un Asset más para reutilizarlo de forma sencilla.

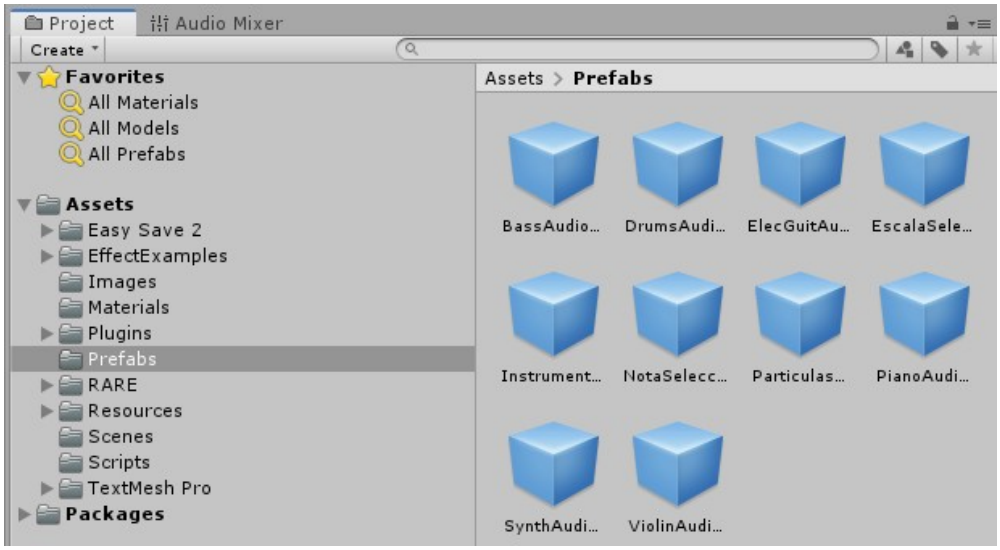

Figura 9: Vista de los prefabs utilizados en el proyecto en la interfaz de Unity

Scripts:
Los scripts pueden utilizarse para crear efectos gráficos, controlar el comportamiento físico de objetos o implementar sistemas de inteligencia artificial para los personajes del juego. El lenguaje utilizado es C# en la plataforma Microsoft Visual Studio.

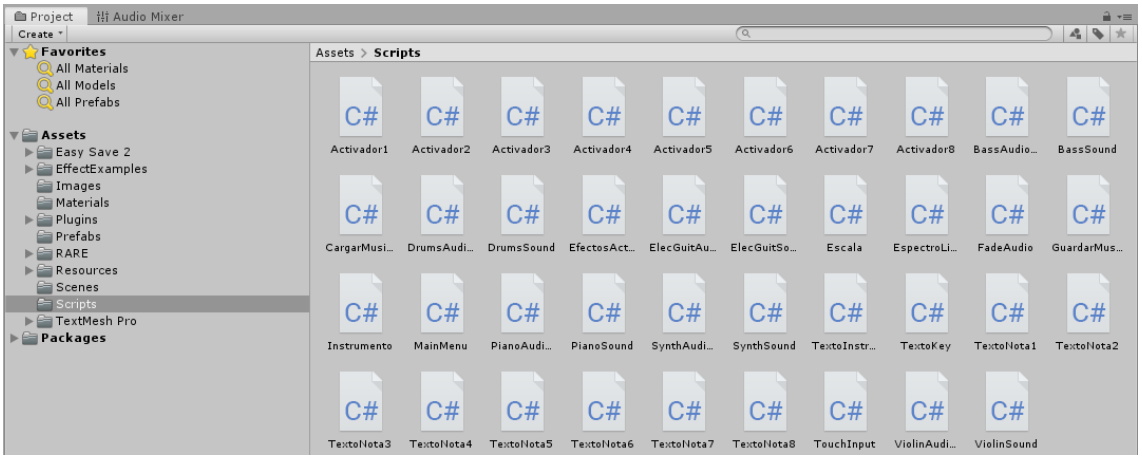

Figura 10: Vista de todos los scripts utilizados en el desarrollo del proyecto en Unity

Assets:
Un asset puede venir de un archivo creado fuera de Unity, como un modelo 3D, un script, un archivo de audio, una imagen o cualquiera de los tipos de archivos que Unity admite. También hay tipos de assets que se pueden crear dentro de Unity, como controladores de animaciones, mezcladores de audio o texturas de renderizado.

Unity tiene disponible una asset store donde toda la comunidad puede publicar sus propios assets para que otras personas puedan usarlos en sus proyectos, los hay de pago y de forma gratuita.

Unity Editor:
El editor de Unity esta compuesto por diferentes subventanas, las cuales tienen esta forma:

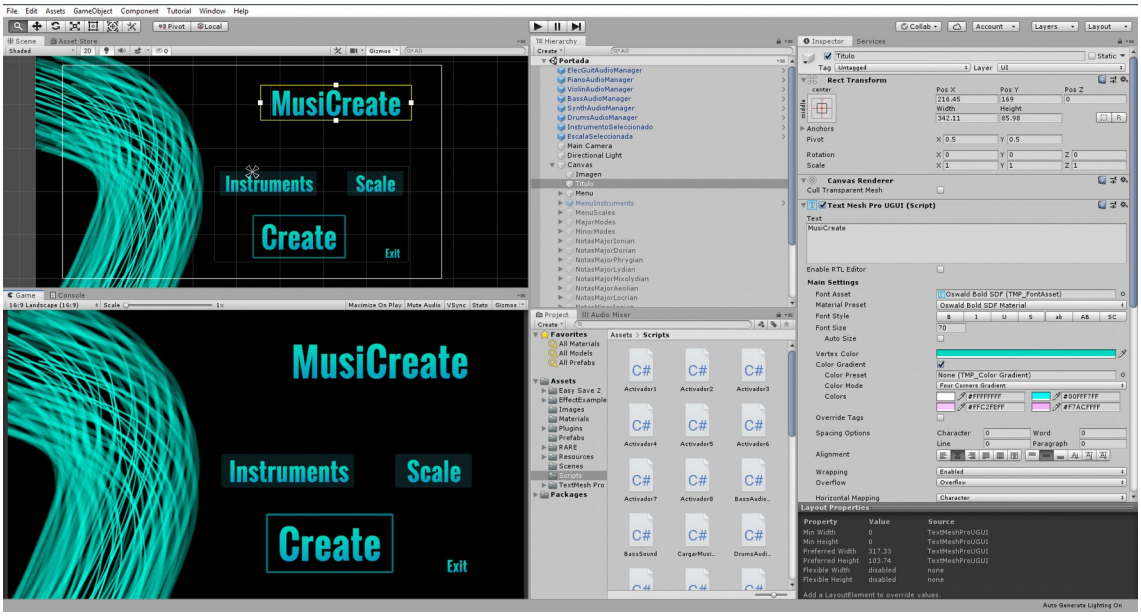

Figura 11: Vista completa de la interfaz del editor de Unity

A continuación se va describir brevemente la utilidad de las principales ventanas.

La Ventana del Proyecto (Proyect Window):

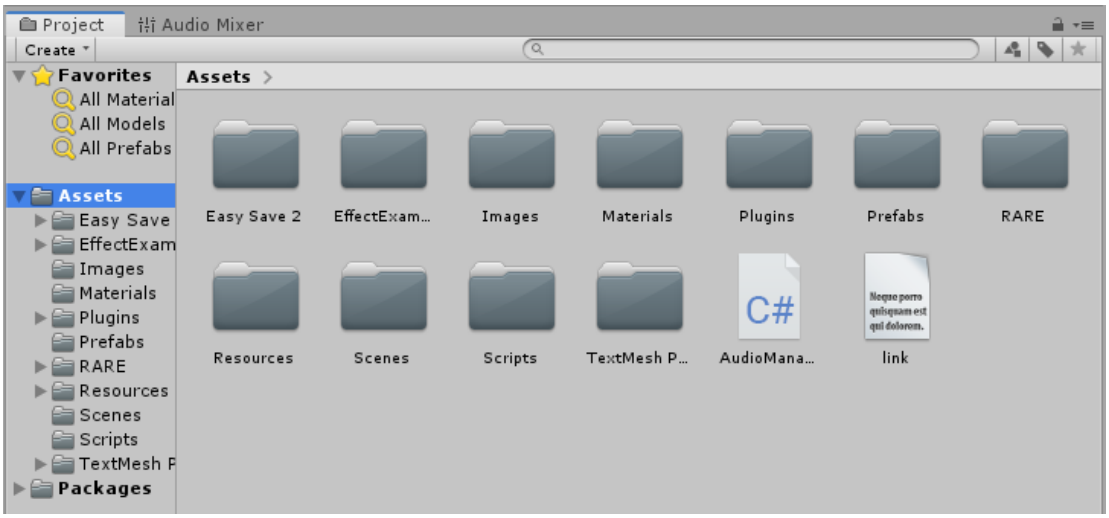

Figura 12: Vista de la ventana del proyecto

En esta vista, se puede acceder y gestionar los assets que pertenezcan a su proyecto. El panel izquierdo del navegador muestra la estructura de carpetas del proyecto en jerarquía. Cuando una

carpeta es seleccionada de una lista haciendo clic, su contenido va a ser mostrado en el panel a la derecha.

La ventana de escena (Scene View):

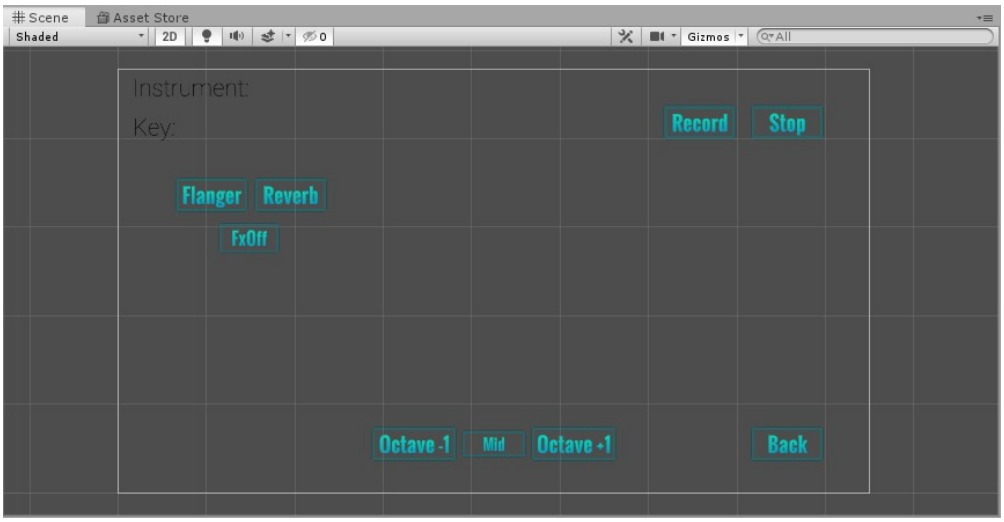

Figura 13: Vista de la ventana de escena

Es la ventana interactiva donde se ve todo lo que se está creando. Se utiliza para seleccionar o posicionar los escenarios, personajes, cámaras, luces y todo tipo de GameObjects.

La Ventana de Jerarquía (Hierachy):
Contiene una lista de cada GameObject en la escena, pueden ser instancias directas de assets, de prefabs o cualquier objeto que se haya creado para el juego.
Unity utiliza un concepto llamado “Parenting” donde se puede meter un objeto o un grupo a uno en concreto, entonces se convertirán en hijos de este objeto, de esta manera si por ejemplo cambias la posición del objeto padre, los hijos cambiarán con él.

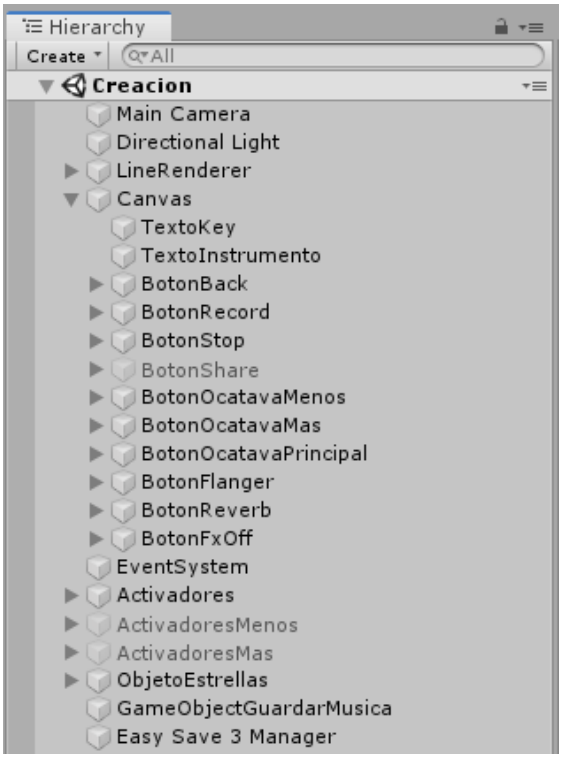

Figura 14: Vista de la ventana de jerarquía

La ventana del Inspector (Inspector):
Muestra todo la información detallada del GameObject actualmente seleccionado, incluyendo todos los components adjuntos y sus propiedades. También nos permite modificar la funcionalidad de los GameObjects de la escena.

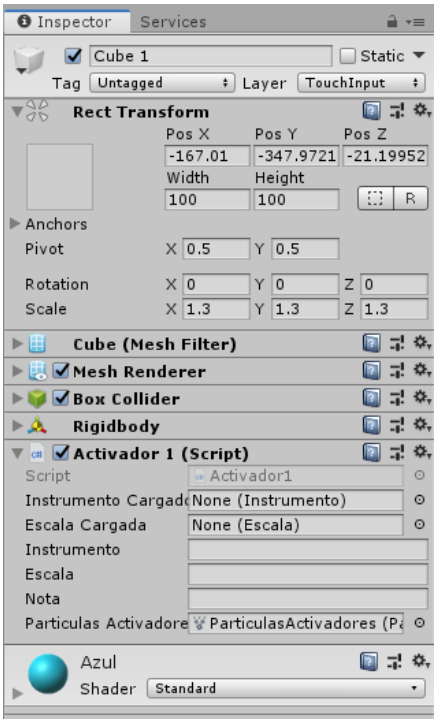

Figura 15: Vista de la ventana del inspector

La Ventana de Juego (Game View):
Se representa la vista de cómo se vería el juego desde la posición de la cámara.

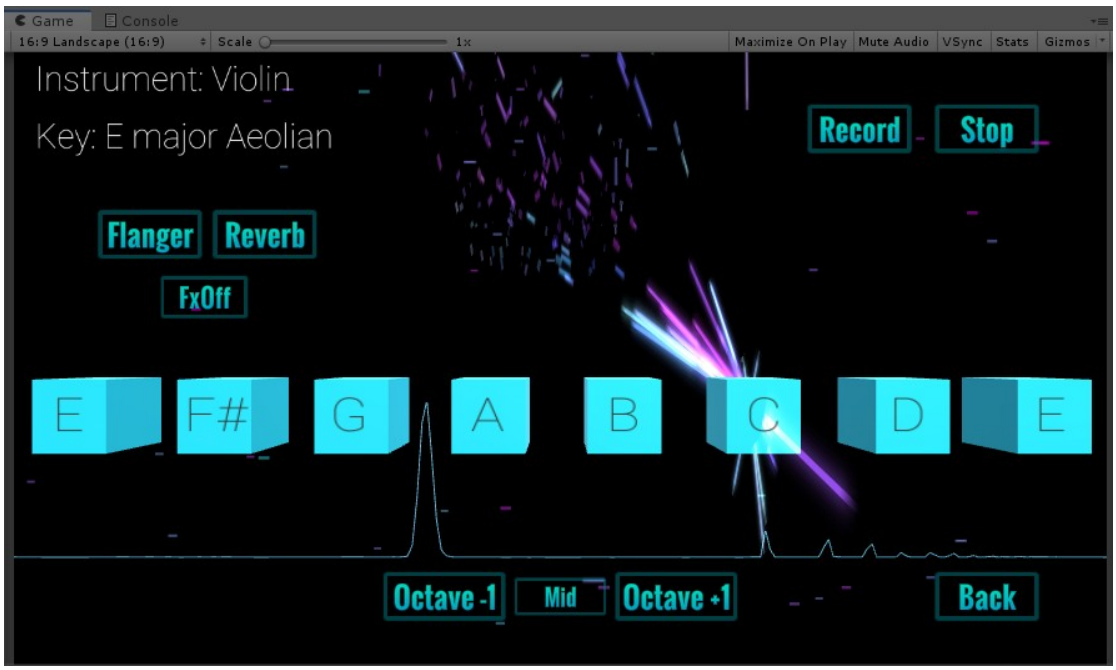

Figura 16: Vista de la ventana de juego

La barra de herramientas (ToolBar):
Se encuentran los controles básicos:

Figura 17: Vista de la barra de herramientas principal en la interfaz de Unity

El primer apartado son las herramientas del componente "Transform", para lo relacionado con mover objetos en la escena.

En el segundo apartado están los componentes "Transform" de los "Gizmos", que son gráficos asociados a los GameObjects en la escena.

Los siguientes son el botón "Play"/"Puase"/"Step", usados con la ventana de Juego.

Después se encuentran las opciones que tiene Unity para trabajar en la nube con sus propios servicios y con la cuenta personal.

En "Layers" se encuentran las capas, las cuales tienen objetos asociados y de esta forma elegimos la capa que queremos ver en la ventana de escena.

Con el último botón se selecciona la distribución de la interfaz visual que se prefiera, ya que las ventanas se pueden mover al gusto del usuario y guardar la distribución.

3.3 Instrucciones

Al iniciar la aplicación, aparece la escena de portada, formada por distintos botones. Para el correcto funcionamiento, se debe elegir el instrumento y la escala concreta que se desea tocar.

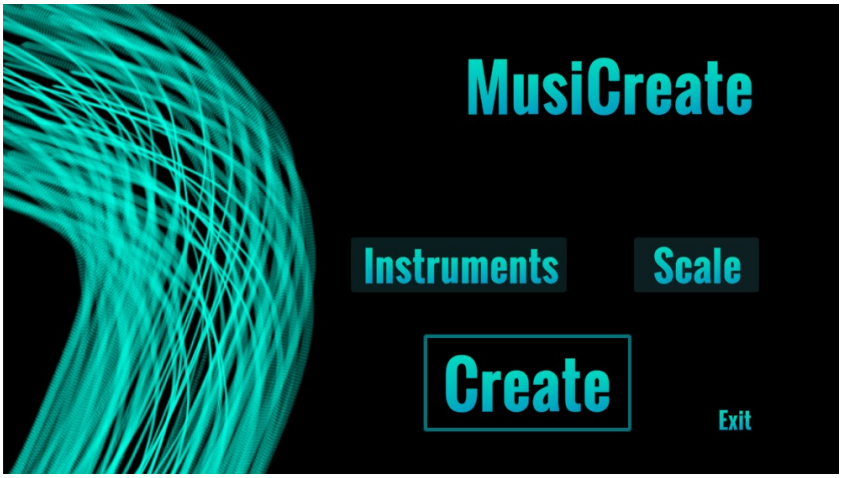

Figura 18: Escena de portada de la aplicación

El botón “Instruments” nos lleva al menú donde se encuentran todos los instrumentos disponibles para seleccionar. Si se pulsa en “Exit” la aplicación se cerrará y a la vez eliminará todos los archivos de audio generados para así no acumular datos en los móviles donde se ejecute.

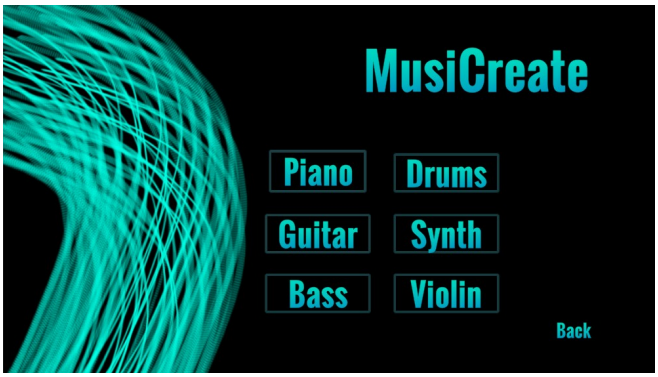

Figura 19: Menú Instruments de la aplicación

En el menú que aparece al presionar “Instruments” se seleccionará el botón del instrumento que se quiera tocar y después, dándole al botón “Back” se volverá a la escena inicial de portada.

Ahora en la escena de portada hay que elegir la escala que se prefiera. Este botón nos lleva a un menú donde elegiremos si queremos tocar modos mayores o modos menores. Comúnmente se asocian los modos mayores a música más alegre y los menores a la más triste. En esta aplicación cada usuario podrá investigarlos y entenderlos practicando con ellos por su propia cuenta.

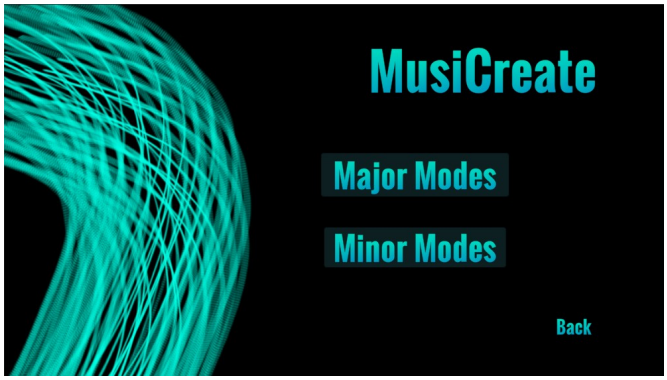

Figura 20: Menú Scales de la aplicación

Una vez seleccionado el modo, nos lleva a las escalas principales de cada uno, se han integrado siete de cada modo.

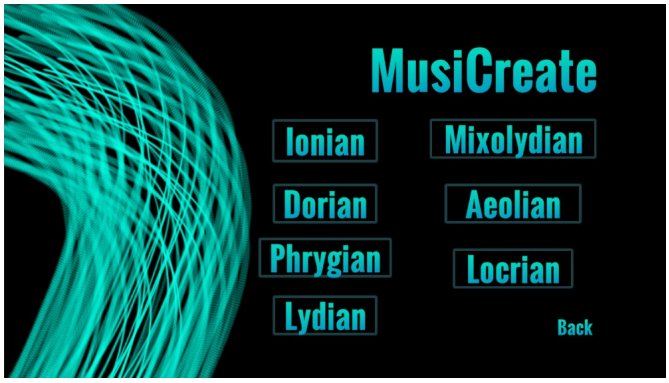

Figura 21: Menú Major Modes

Figura 22: Menú Minor Modes

Ahora, una vez seleccionado el modo que se haya preferido, la aplicación lleva al menú de selección de nota y se ha de seleccionar la nota concreta con la que queremos que se forme la escala.

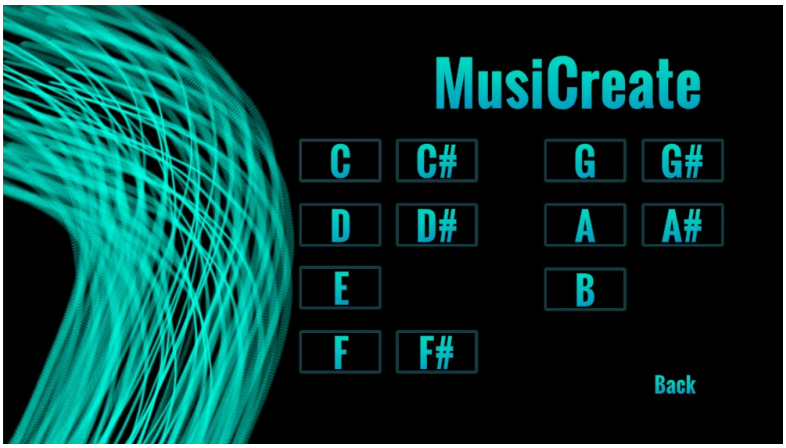

Figura 23: Menú siguiente de elegir el modo para seleccionar la nota en la que se desea tocar la escala

Finalmente hay que dar al botón "Back" para volver a la escena de portada y se pulsará el botón "Create" que nos lleva a la escena de creación.

Figura 24: Escena de creación

En esta escena nos encontramos de nuevo con diferentes botones y con los cubos que serán los que harán sonar la nota del instrumento concreto al ser pulsados. La función de estos botones se explicará detalladamente junto con la descripción de los scripts.

Para mejorar la interfaz visual se han añadido unas líneas de colores simulando estrellas en movimiento. También se ha añadido un efecto visual al pulsar cada activador que deja de funcionar cuando deja de pulsarse para añadir más dinamismo. Además de la interacción en directo con el espectro de audio.

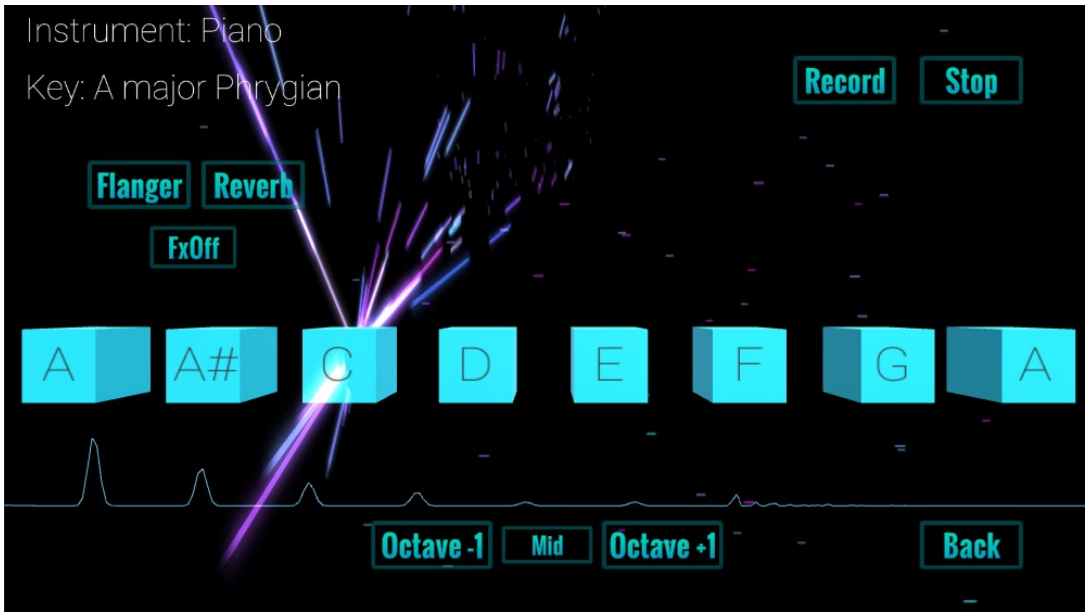

Figura 25: Escena de creación con los efectos visuales de un cubo pulsado

3.4 Desarrollo de Scripts

Activadores:
En este script se organizan las 168 diferentes escalas para cada instrumento y la llamada de la función al archivo de sonido que debe sonar según lo que se haya escogido dentro de la aplicación. También se encuentran las funciones para hacer que pare el sonido cuando ya no se está pulsando el cubo activador. Cada vez que se pulsa un cubo, también se activa un efecto de partículas para dar más dinamismo a la escena.

Para que este script funcione correctamente, ha de recoger de los demás scripts una variable string de instrumento, escala y nota a la vez que también debe poder acceder al script donde se encuentran todos los archivos de sonido para poder activarlos.

```
void OnTouchDown()
{

    particulasActivadores.Play();

    if (instrumento == "Piano")
    {
        PianoAudioManager playable = FindObjectOfType<PianoAudioManager>();

        ////////////////////////////////////////////////////////////////////////////////
        /// MAJOR IONIA /// MAJOR LYDIAN ///MAJOR MIXOLYDIAN
        ////////////////////////////////////////////////////////////////////////////////

        if (escala == "C major Ionian" || escala == "C major Lydian" || escala == "C major Mixolydian")
        {
            playable.Play("PianoE3");
        }
        if (escala == "C# major Ionian" || escala == "C# major Lydian" || escala == "C# major Mixolydian")
        {
            playable.Play("PianoF3");
        }
        if (escala == "D major Ionian" || escala == "D major Lydian" || escala == "D major Mixolydian")
        {
            playable.Play("PianoF#3");
        }
        if (escala == "D# major Ionian" || escala == "D# major Lydian" || escala == "D# major Mixolydian")
        {
            playable.Play("PianoG3");
        }
        if (escala == "E major Ionian" || escala == "E major Lydian" || escala == "E major Mixolydian")
        {
            playable.Play("PianoG#3");
        }
        if (escala == "F major Ionian" || escala == "F major Lydian" || escala == "F major Mixolydian")
        {
            playable.Play("PianoA3");
        }
```

Figura 26: Ejemplo de una parte del código de la organización de las escalas ionia, lydia y mixolydia para piano

Para llevar a cabo este script, como hay 168 funciones play por cada instrumento hay que intentar optimizar lo máximo posible las notas que se repiten o que son distintas en las diferentes escalas, para que no se convierta en un script de miles de líneas. Las escalas de los modos mayores funcionan de la siguiente manera:

Ionian Mode	1	2	3	4	5	6	7
Lydian Mode	1	2	3	#4	5	6	7
Mixolydian Mode	1	2	3	4	5	6	b7
Dorian Mode	1	2	b3	4	5	6	b7
Aeolian Mode	1	2	b3	4	5	b6	b7
Phrygian Mode	1	b2	b3	4	5	b6	b7
Locrian Mode	1	b2	b3	4	b5	b6	b7

Tabla 3: Esquema de cómo se organizan las escalas de los modos mayores

Como hay 8 activadores y el octavo es la misma nota que el primero pero una octava más arriba, también se repite para todas las escalas, por lo tanto, en el primer activador y en el octavo el script se puede optimizar de forma sencilla ya que la nota es la misma para todos los modos, repitiéndose la primera nota en el primer activador y la primera octava en el octavo.

En los segundos activadores, como se aprecia en la imagen, solo deben cambiar el modo Frigio y el Locrio, bajando un semitono. De este modo, se observa que se puede optimizar todo el código agrupando las escalas y las notas mediante grupos donde aún siendo diferente escala, la nota es la misma, debido a esto, se han utilizado bucles "if" con condicionales "OR" para todas las escalas que comparten nota.
De esta manera, como podemos observar en la imagen, tenemos dos grupos para los activadores 2, 3, 4, 5, 6 y 7, donde una "b" significa un semitono menos y un "#" un semitono más. Este es el proceso para los modos de las escalas mayores, en la aplicación también están incluidos los principales modos menores de las escalas. En esta imagen se pueden ver representados de forma esquemática:

Melodic Minor	1	2	b3	4	5	6	7
Dorian b2	1	b2	b3	4	5	6	b7
Lydian Augmented	1	2	3	#4	#5	6	7
Lydian Dominant	1	2	3	#4	5	6	b7
Aeolian Dominant	1	2	3	4	5	b6	b7
Locrian Natural 2	1	2	b3	4	b5	b6	b7
Altered Scale	1	b2	b3	b4	b5	b6	b7

Tabla 4: Esquema de cómo se organizan las escalas de los modos menores

Al igual que con los modos mayores, en los modos menores también se han hecho grupos para optimizar el código ya que de no ser así, habría miles de líneas de código por cada script de los diferentes activadores.

-Sound:
Cada instrumento tiene su propio script de "Sound", en este se crea un array de clips de audio, donde están almacenados todos los diferentes archivos wav de cada instrumento con su respectiva nota, de F#2 hasta F4. También se puede variar independientemente el volumen de cada archivo cargado y el pitch.
Para que se pueda manejar archivos de audio y modificarlos desde el script se ha añadido la librería "UnityEngine.Audio".

-AudioManager:
Al igual que con "Sound", cada instrumento tiene su propio script de "AudioManager". Su finalidad es poder encontrar mediante una cadena de texto, el archivo de sonido contenido en el array que debe activarse cuando se pulsan los cubos y poder reproducirlo o pausarlo. También añade todos los sonidos del array a un "AudioMixer", una herramienta propia de Unity que permite separar los diferentes sonidos que se necesiten en diferentes canales y así tener un sistema de audio más organizado y sencillo de manejar para posibles efectos sonoros que se quieran añadir a instrumentos concretos.

Este script se compone de 3 funciones principales:
La primera para añadir todos los archivos de audio del array particular de cada instrumento al AudioMixer y al propio GameObject al que se le asigna este script para poder acceder a ellos y reproducirlos o pausarlos.
Después está la función de "Play" donde mediante búsqueda por cadena de texto se selecciona el sonido correspondiente al cubo pulsado y se reproduce.
Finalmente está la función "Silenciar" donde de forma similar a la anterior función. Se hace el mismo proceso pero esta vez no se reproduce el sonido sino que se llama a una función del script "FadeAudio" que silencia el sonido de forma progresiva y se explica más adelante.

Para que el script pueda encontrar los archivos de audio dentro del array correspondiente se debe añadir al principio del mismo la librería "System" y para poder tratar con las funcionalidades de audio incluidas en Unity la librería "UnityEngine.Audio".

```
using UnityEngine.Audio;
using System; // encontrar un sonido en un array
using UnityEngine;
using System.Collections.Generic; //para poder usar las listas

public class PianoAudioManager : MonoBehaviour{

    public PianoSound[] PianoSounds;
    public static PianoAudioManager instance; // nos aseguramos de que solo haya 1 instancia de audiomanager
    public AudioMixerGroup PianoMixer; // Pongo pública la variable de Mixer de piano.

    void Awake()   //antes que start
    {

        if (instance == null)
            instance = this;
        else
        {
            Destroy(gameObject);
            return;
        }

        DontDestroyOnLoad(gameObject); //para que en las transiciones de escenas no se destruya

        foreach(PianoSound s in PianoSounds)
        {
            s.source = gameObject.AddComponent<AudioSource>(); //el sonido que estamos buscando
            s.source.outputAudioMixerGroup = PianoMixer;  // añado todos los sonidos del array al mixer
            s.source.clip = s.clip;

            s.source.volume = s.volume; //controlar el volumen del sonido buscado
            s.source.pitch = s.pitch; //controlar el pitch del sonido
        }

    }
```

Figura 27: Primera parte del script AudioManager del piano

```
public void Play (string name)
{
    PianoSound s = Array.Find(PianoSounds, sound => sound.name == name);
    // buscamos el sonido en el array de sonidos y lo metemos en la variable s
    if (s == null)
    {
        Debug.LogWarning("Sound: " + name + " not found!"); //por si lo escribimos mal
        return;
    }
    s.source.Play();

}

public void Silenciar(string name)
{
    PianoSound s = Array.Find(PianoSounds, sound => sound.name == name);
    // buscamos el sonido en el array de sonidos y lo metemos en la variable s
    if (s == null)
    {
        Debug.LogWarning("Sound: " + name + " not found!"); //por si lo escribimos mal
        return;
    }
    StartCoroutine(AudioFadeOut.FadeOut(s.source, 0.02f));

}
```

Figura 28: Código para reproducir y silenciar el sonido del piano del script AudioManager

-EfectosActivadores:
En este script se recogen todos los diferentes efectos que podemos aplicarle al sonido final mediante los botones que van asignados; “Flanger”, “Reverb”, “FxOff”, “Octave-1”, “Octave+1” y “Mid”, que se explicarán más detalladamente de manera individual a continuación.

Para que el script funcione correctamente, se ha de añadir el paquete de UnityEngine Audio agregándolo al principio del código utilizando “using” delante, de esta forma, tendremos incluidas las funciones que tienen que ver con el tratamiento del audio.
Por otra parte, antes de eso, se han de añadir todos los instrumentos al AudioMixer, que se llamará “Sonidos”.

Para modificar cualquier parámetro concreto en tiempo real, se tendrá que enviar al script de la siguiente forma: pulsando clic derecho y eligiendo la función “Expose” para luego ponerle el nombre deseado en la ventana de “ExposedParameters” y finalmente modificarlo con los parámetros elegidos en el script. Se han añadido todos los efectos a cada instrumento en lugar de al máster, ya que para algunos instrumentos el sonido con el efecto suena mejor con los parámetros ligeramente diferentes.

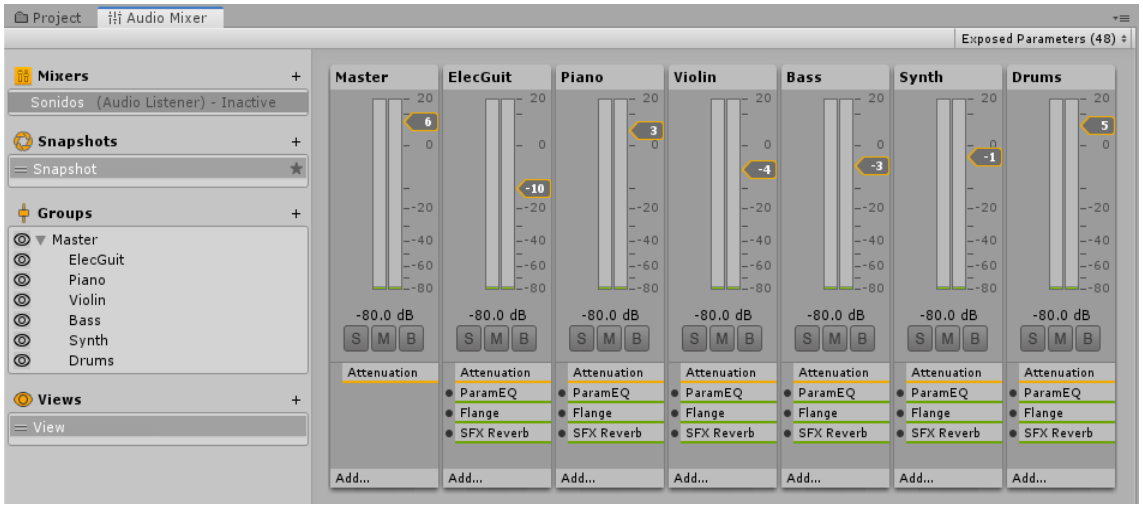

Figura 29: Vista del AudioMixer que recoge todos los instrumentos utilizados y sus efectos

Las funciones que se han hecho para los diferentes efectos mencionados anteriormente son:

-OctavaMenos(), (Octave-1): Este efecto consiste en, de forma digital, bajar una octava toda la escala, para obtener un rango más amplio y tener más variedad de sonidos. Los parámetros que se han seleccionado son, para cada instrumento, el "Pitch", que le asignaremos un valor de "0.5f", lo que hará que la frecuencia se reduzca a la mitad y por lo tanto obtendremos la octava anterior. Además del efecto de pitch, se utilizará un filtro paramétrico que ayudará a retocar el sonido de forma digital para que quede más natural variando en una frecuencia determinada la ganancia. En este caso se han variado los parámetros de frecuencia y de ganancia.

```
public void OctavaMenos()
{
    AudioMixer audioMixer = Resources.Load<AudioMixer>("Escalas/Sonidos");
    audioMixer.SetFloat("PianoPitch", 0.5f);
    audioMixer.SetFloat("PianoCenterFreq", 30f);
    audioMixer.SetFloat("PianoFreqGain", 0.21f);
```

Figura 30: Ejemplo de cómo modificar los parámetros de un efecto para disminuir el pitch del piano del audiomixer desde los scripts

-OctavaMas(), (Octave+1): Es exactamente igual que el anterior pero el valor de "Pitch" es "2f", por lo tanto obtendremos la octava siguiente, dando como resultado unos sonidos más agudos dentro de la misma escala.

-OctavaBase(), (Mid): Esta función sirve para poder volver a la octava principal. El código es el mismo que en las anteriores funciones pero el "Pitch" valdrá "1f".

-FlangeEffect()divideger): El efecto flanger consiste en un duplicado de la onda sonora original, una se mantiene sin procesar y a la otra se le aplica un retraso muy pequeño, normalmente menor a 20 milisegundos, lo que produce un sonido con un efecto metalizado oscilante. Los parámetros que se han asignado en esta función, son "Drymix" (porcentaje de la señal original que va a la salida) con un 90%, "Wetmix" (porcentaje de la señal procesada que va a la salida) también con un 90% y "Rate" (marca la frecuencia del "Low Frequency Oscillator" o LFO) en el que el parámetro es 300Hz.

-Reverb(), (Reverb): El efecto reverb consiste en un aumento de las reflexiones o tiempo de reflexion de las ondas de sonido, dando una sensación de mayor permanencia de este. Los parámetros que se han asignado en este caso son “Room” (para dar la sensación de que el sonido se encuentra en una habitación más grande) con un valor de 0, para dar una sensación de mayor espacio y “Decay Time” (cantidad de tiempo que tardan las reflexiones de la señal en parar) con 7s.

-FXOFF(), (FxOff): Esta función sirve para cambiar todos los valores que han sido alterados en las otras funciones a su valor original y así eliminar cualquier modificación del sonido.

-Instrumento:
La utilidad de este script es poder almacenar en una cadena de texto el instrumento elegido y que se pueda leer tanto en la escena de portada como en la de creación, para así poder seleccionar correctamente el sonido final del instrumento adecuado.

Este script se divide en dos partes, en la primera hay una función “Awake()” (esta función se ejecuta en primer lugar) con la finalidad de hacer que este script se pueda ejecutar en la escena de portada y la de creación, ya que si no tenemos una función que ordene que no se destruya, en el cambio de escena de portada a creación, se destruiría y no podríamos saber qué instrumento se ha seleccionado.

```
private void Awake()
{
    if(instance == null)
    {
        instance = this;
    DontDestroyOnLoad(gameObject);
    }else if(instance != this)
    {
        Destroy(gameObject);
    }

}
```

Figura 31: Función para que los objetos no se destruyan en el cambio de escena

En la segunda parte, hay una función por cada instrumento y que cada una está asignada a su botón de la interfaz visual correspondiente. De esta manera, cuando se pulse el botón de cada instrumento, guardaremos una cadena de texto del instrumento elegido.

```
public void ActivarPiano()
{
    instrumentoCargado = "Piano";
    Debug.Log("Has cargado el Piano");
}

public void ActivarElecGuit()
{
    instrumentoCargado = "Electric Guitar";
    Debug.Log("Has cargado la Guitarra Eléctrica");
}

public void ActivarViolin()
{
    instrumentoCargado = "Violin";
    Debug.Log("Has cargado el Violín");
}

public void ActivarBass()
{
    instrumentoCargado = "Bass";
    Debug.Log("Has cargado el Bass");
}

public void ActivarSynth()
{
    instrumentoCargado = "Synth";
    Debug.Log("Has cargado el Synth");
}

public void ActivarDrums()
{
    instrumentoCargado = "Drums";
    Debug.Log("Has cargado las baterías");
}
```

Figura 32: Código del script Instrumento con las diferentes funciones asociadas a cada instrumento

-Escala:

Este script tiene una finalidad similar al anterior, pero en lugar de almacenar el instrumento seleccionado, almacena la escala de la nota correspondiente, por lo tanto se divide en 2 partes al igual que el anterior, la primera es igual, para que el objeto al que está asignado no se destruya en el cambio de escena a creación y la segunda, dividida en 168 funciones, las cuales se pueden dividir en 7 grupos para las escalas de los modos mayores y otros 7 grupos para las escalas de los modos menores.

Estos subgrupos están divididos cada uno en 12, ya que tenemos 12 notas en total y cada nota tiene su propio orden de escala. Este script, a parte de tener la finalidad de pasar la cadena de texto con la escala seleccionada, también se usará para pasar la cadena de texto de la nota concreta que tiene que ir escrita en cada activador, de esta forma se consigue poder visualizar la nota que estamos tocando o las que conforman la escala que se ha seleccionado y poder aprender teoría musical.

```
public void ActivarCmajorIONIA()
{
    escalaCargada = "C major Ionian";
    a1nota = "C";
    a2nota = "D";
    a3nota = "E";
    a4nota = "F";
    a5nota = "G";
    a6nota = "A";
    a7nota = "B";
    a8nota = "C";
    Debug.Log("Has cargado la escala C major Ionia");
}
```

Figura 33: Ejemplo de una parte del código donde se organiza la escala jónica de Do

-TextoNota:
De este script se necesitan 8 scripts diferentes, uno por cada activador, ya que cada uno se utiliza para pasar la cadena de texto correspondiente a la nota que corresponda con la escala elegida. Este script recoge los valores de las notas definidas en el script "Escala" y asigna a un objeto de texto el texto correspondiente.

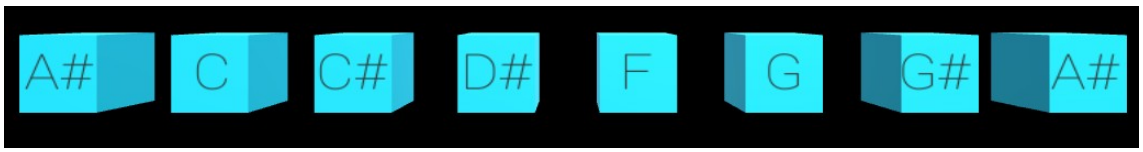

Figura 34: Ejemplo de los cubos activadores de la interfaz visual con la escala A# dórica

-TextoKey:
Su finalidad es sacar por la pantalla en tiempo real en la escena de creación, una cadena de texto en la que aparece la escala y la nota que hemos seleccionado. Este script se comunica con el de "Escala" y coge la cadena de texto de la escala elegida y la asigna al objeto de texto creado con el editor visual para que así pueda salir por pantalla.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI; //necesaria para cosas de UI

public class TextoKey : MonoBehaviour {

    public Escala escala;  //Cogemos el script Escala y lo llamamos con escala

    Text Key;  //creamos la variable de texto Key

    void Start()
    {
        Key = GetComponent<Text>(); //cogemos la variable Key
    }

    void Update()
    {
        Key.text = "Key: " + Escala.escalaCargada;
        //metemos en la variable Key la escalaCargada del script Escala
    }


}
```

Figura 35: Código para representar el texto de la nota por pantalla

-TextoInstrumento:
La finalidad de este script es la misma que la del anterior pero en este caso, se saca por pantalla la cadena de texto correspondiente al instrumento que se haya elegido.

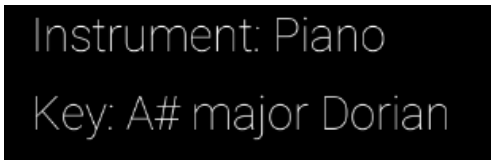

Figura 36: Representación visual de los scripts de TextoKey y TextoInstrumento

-FadeAudio:
Cuando un sonido digital se pausa bruscamente, al estar la onda de sonido desplazada, se produce un clic molesto en el audio. Para solucionar el problema, se ha creado este script, que silencia el sonido de forma progresiva, pero muy rápida, así se consigue un efecto de silencio de forma natural.

Para desarrollar esta solución, se han seguido diferentes pasos. Los scripts de Unity se ejecutan en un mismo frame temporal, por lo tanto si esta función se ejecutara sin utilizar ninguna herramienta diferente, las órdenes escritas no servirían de nada, ya que el sonido sería como pararlo de golpe y los valores intermedios nunca se verían.
Con la finalidad de evitar este tipo de problemas, Unity dispone de corrutinas, que hacen que se puede ejecutar una misma función en diferentes frames según el tiempo establecido para ello. Estas corrutinas se utilizan poniendo antes del nombre de la función "IEnumerator" y en el bucle dentro de la función acabar con "yield", de este modo la función se ejecutará en frames diferentes.

Después de conocer el uso de las corrutinas, el código añadido en esta función recoge el volumen de la fuente de audio que esté sonando y con un bucle lo baja progresivamente hasta que ese valor sea cero. Finalmente asigna el valor del volumen a 0 y vuelve a ponerlo como estaba de inicio para que no haya ningún error en los siguientes audios.

```
using UnityEngine;
using System.Collections;

public class AudioFadeOut : MonoBehaviour
{

    public static IEnumerator FadeOut(AudioSource audioSource, float FadeTime)
    {
        float startVolume = audioSource.volume;

        while (audioSource.volume > 0)
        {
            audioSource.volume -= startVolume * Time.deltaTime / FadeTime;
            //tiempo entre cada frame
            yield return null;
        }

        audioSource.Stop();
        audioSource.volume = startVolume;
    }

}
```

Figura 37: Script para el fade out

-EspectroLinea:
La finalidad de este script es que salga por la pantalla de la aplicación una línea que represente el espectro del audio que está sonando en ese mismo instante.

Para conseguir este objetivo, el script asigna la variable del sonido de salida total con la función "GetSpectrumData()" utilizando una ventana de FFT (Fast Fourier Transform) de "Blackman Harris".
Con este tipo de ventana conseguimos un pico muy amplio a la vez que una buena compresión del lóbulo lateral, lo que nos permite visualizar correctamente en la línea del espectro la frecuencia y sus octavas del sonido que se están reproduciendo, representado de 0 a 22050 Hz (la mitad de 44100 Hz que es a la frecuencia de sonido común que trabajan los dispositivos electrónicos).

Esta línea del espectro, está dividida entre 2048 secciones (número que tiene que ser potencia de 2 para un funcionamiento más óptimo) por lo tanto, en cada sección nos encontramos con una muestra de 10,76 Hz.

Para que los picos se visualicen correctamente hay que añadirlos mediante la frecuencia máxima de cada grupo de valores y luego volverlos a asignar a 0 para que no produzca errores al añadir los siguientes.

Para una correcta visualización, se marcará un máximo de altura a la que pueden llegar los picos de las frecuencias.

-TouchInput:
Con este script se consigue que se puedan pulsar diferentes cubos a la vez, haciendo que la aplicación sea multitáctil para poder hacer acordes con los instrumentos, aumentando la variedad de posibilidades de la aplicación.
Esta utilidad se consigue asignando el script a la cámara principal de la escena de creación, creando una capa de la aplicación diseñada para las funciones táctiles y construyendo un array que contenga todas las pulsaciones registradas en la pantalla para que así el programa pueda leerlas e interactuar con ellas.

```
if (Input.touchCount > 0)
{
    touchesOld = new GameObject[touchList.Count];
    touchList.CopyTo(touchesOld);
    touchList.Clear();

    foreach (Touch touch in Input.touches)
    {
        Ray ray = GetComponent<Camera>().ScreenPointToRay(touch.position);


        if(Physics.Raycast(ray,out hit, touchInputMask))
        {
            GameObject recipient = hit.transform.gameObject;
            touchList.Add(recipient);

            if(touch.phase == TouchPhase.Began)
            {
                recipient.SendMessage("OnTouchDown", hit.point, SendMessageOptions.DontRequireReceiver);
            }
            if (touch.phase == TouchPhase.Ended)
            {
                recipient.SendMessage("OnTouchUp", hit.point, SendMessageOptions.DontRequireReceiver);
            }
            if (touch.phase == TouchPhase.Stationary || touch.phase == TouchPhase.Moved)
            {
                recipient.SendMessage("OnTouchStay", hit.point, SendMessageOptions.DontRequireReceiver);
            }
            if (touch.phase == TouchPhase.Canceled)
            {
                recipient.SendMessage("OnTouchExit", hit.point, SendMessageOptions.DontRequireReceiver);
            }
        }
    }
    foreach(GameObject g in touchesOld)
    {
        if (!touchList.Contains(g))
        {
            g.SendMessage("OnTouchExit", hit.point, SendMessageOptions.DontRequireReceiver);
        }
    }
}
```

Figura 38: Apartado del script TouchInput para que la plicación sea multitáctil

-GuardarMusica:
Este script se utiliza para grabar y guardar el sonido que se haya generado mediante las pulsaciones en los cubos táctiles. Se puede dividir en 3 partes:

La primera es una función "Awake" ya mencionada en los scripts de "Instrumento" y "Escala" que nos permite cambiar entre la escena de creación y portada sin que se destruya este objeto, permitiendo mantener el sonido grabado y a la vez poder cambiar de instrumento o de escala y tocar encima sin que el sonido se pierda.

La segunda función es la que nos permite grabar el audio que esté sonando, asignándola al botón de la interfaz "Record" y guardando cada pista de audio en un archivo .wav almacenado en la carpeta "persistentDataPath" que es la predeterminada para guardar cualquier archivo de las aplicaciones con el nombre de "AudioRecording(i)" siendo "i" el número de pista grabado. En esta misma función, también se activa el archivo de audio con los números anteriores al que se esté grabando, de esta manera podemos escuchar todo lo grabado y a la vez, grabar encima una pista nueva.

Después está la función de "Stop" que se asigna al botón "Stop". Esta función consiste en comprobar el nombre del archivo de audio que está sonando con su número adecuado y hacer que pare de grabar.

Para que el objeto al que está asignado el script pueda cambiar entre escenas, se necesita añadir al principio del código la librería "UnityEngine.SceneManagement" y para permitir el guardado de archivos, "System.IO".

-MainMenu:
Con este script se hacen diferentes funciones para darle la utilidad correcta a los botones de "Create", "Back" y "Exit".

Para conseguir esto, primero se ha de añadir la librería "UnityEngine.SceneManagement" para poder hacer cambios de escenas y "System.IO" para poder acceder a archivos guardados. En la función asignada al botón "Create" se carga la escena siguiente, es decir, la de creación.

La función del botón "Back" se usa múltiples veces tanto para salir del menú de instrumentos como para salir del menú de escalas y notas a la escena principal de "Portada".

Para el botón de "Exit" la función elimina todos los archivos de audio grabados accediendo al "persistentDataPath" para no generar una cantidad de datos grande en los móviles donde se ejecute y finalmente sale de la aplicación.

Capítulo 4. Clonclusiones y trabajo futuro

A lo largo de la realización del trabajo de final de grado desarrollado en la presente memoria, se han aprendido multitud de conceptos sobre la programación en C# enfocada al desarrollo de videojuegos y aplicaciones móviles, además del software Unity.

Este campo era prácticamente nuevo para mí, ya que no conocía el lenguaje C# y nunca había desarrollado ninguna aplicación de estas características siguiendo una idea propia. He tenido que hacer un aprendizaje durante los meses previos en cursos online, aprendiendo según surgían nuevas complicaciones.

La plataforma de desarrollo de Unity me ha sorprendido mucho, ya que tiene un enorme potencial para desarrollar cualquier tipo de videojuego o aplicación, mezclando el editor visual y los scripts en C#. Además, con ella también se pueden crear animaciones enfocadas al sector audiovisual o aplicaciones para arquitectura y construcción. Para el desarrollo se ha utilizado la versión de Unity Personal, ya que es totalmente gratuita si tus ingresos son menores a $100 000 por año.

Por otra parte, para este proyecto también se ha llevado a cabo un estudio sobre teoría musical, indagando en cómo funcionan los diferentes modos y escalas.

Personalmente elegí llevar a cabo esta idea ya que siempre he sido un apasionado de la música y la tecnología y unir los dos mundos (que hoy en día ya no son tan diferentes) me ha hecho sentir una gran superación personal ya que he aprendido mucho sobre la plataforma Unity, el lenguaje C# y teoría musical y me siento más preparado para llevar a cabo proyectos más perfilados y profesionales.

-Dificultades Encontradas:

Durante el proceso de desarrollo de la aplicación, el principal problema que tuve fue en el proceso de grabación, ya que después de mucho tiempo de investigación no conseguí desarrollar un código que almacenara correctamente en el móvil todos archivos de sonido para volver más tarde a reproducirlos y grabar más pistas encima. Así que recurrí a la biblioteca principal de Unity, la asset store y obtuve un asset el cuál tenía muy buen soporte detrás. De esta manera, pude avanzar en el proceso del juego pudiendo grabar el sonido con buena calidad.

-Implementaciones futuras:
Hacer esta aplicación me ha ayudado mucho al aprendizaje sobre programación y desarrollo de aplicaciones, por lo tanto mi intención es seguir llevando a cabo el desarrollo de esta misma, para hacerla aún más profesional y finalmente subirla a la plataforma de la play store de android.

Las mejoras que se investigarán para llevar a cabo serán: implementar un botón de compartir para que la música creada pueda enviarse al instante a otras aplicaciones. Esta mejora se intentó llevar a cabo, pero debido a que para que unas aplicaciones tengan acceso a otras hay que programar permisos de android dentro de la aplicación, no se llevó a cabo por falta de tiempo.

Otra mejora será el control individual del volumen de cada instrumento que se quiera añadir. También se estudiará el implementar otro instrumento a modo de sintetizador digital, que permita la personalización completa del sonido final mediante diferentes parámetros.

Finalmente me gustaría poder añadir un sistema que permitiera crear bucles con el sonido para dar mayor creatividad y ser una aplicación más completa.

Capítulo 5. Bibliografía

[1] Punset, Eduard “Redes - Pregúntale a Punset - ¿Por qué las personas cierran los ojos cuando escuchan música?”
http://www.rtve.es/alacarta/videos/redes/redes-preguntale-punset-porque-personas-cierran-ojos-cuando-escuchan-musica/1412267/# [Online]

[2] Logitunes “History of DAW”
http://logitunes.com/blog/history-of-daw/ [Online]

[3] Thump by Vice “The Untold Story of Ableton Live—the Program That Transformed Electronic Music Performance Forever”
https://www.vice.com/en_us/article/78je3z/ableton-live-history-interview-founders-berhard-behles-robert-henke [Online]

[4] PianoRed http://www.pianored.com/acordes-de-piano.html [Online]

[5] Unity Documentation https://docs.unity3d.com/es/current/Manual/ [Online]

[6] Unity Forum https://forum.unity.com/forums/audio-video.74/ [Online]

[7] StackOverflow https://stackoverflow.com/ [Online]

[8] YouTube, Sebastian Lague, “Unity Touchscreen Input Tutorial”
https://www.youtube.com/watch?v=SrCUO46jcxk [Online]

[9] YouTube, Brackeys, “Introduction to AUDIO in Unity”
https://www.youtube.com/watch?v=6OT43pvUyfY [Online]