Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Visualización de partituras en realidad virtual sincronizadas con su interpretación

Performance-synchronized Score Visualization in VR

Carlota Marrero Morales

La Laguna, 13 de marzo de 2023

D. **José Gil Marichal Hernández**, con N.I.F. 78.677.406-H, profesor contratado doctor adscrito al área Teoría de la Señal y Comunicaciones, del Departamento de Ingeniería Industrial de la Universidad de La Laguna, como tutor

D. **Óscar Gómez Cárdenes**, con N.I.F. 78.859.209-Y, doctor en Informática por la Universidad de La Laguna, como cotutor

C E R T I F I C A N

Que la presente memoria titulada:

"*Visualización de partituras en realidad virtual sincronizadas con su interpretación*"

ha sido realizada bajo nuestra dirección por D. **Carlota Marrero Morales**, con N.I.F. 51.169.041-K.

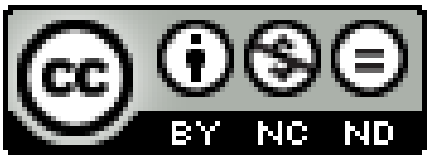
Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 13 de marzo de 2023.

Agradecimientos

A mis padres y a mi hermana por apoyarme siempre, incondicionalmente.

A mis tutores, José y Óscar, por guiarme y acompañarme durante este proyecto.

Licencia



Resumen

El objetivo de este Trabajo de Fin de Grado ha sido representar de manera estéticamente agradable una partitura musical en realidad virtual (RV) en tiempo real y conseguir que esta esté sincronizada con la reproducción de audio correspondiente a la interpretación de la obra. Esto se ha conseguido desarrollando una aplicación de RV utilizando el motor gráfico de videojuegos, Unity, que a partir de la lectura de un fichero MIDI nos permitirá representar en un entorno 3D la diferente información que nos proporciona el MIDI.

El resultado de este trabajo es una herramienta que se pueda utilizar en ámbitos educativos para fomentar el aprendizaje y el gusto por la música de manera visual y entretenida.

Palabras clave: Partitura, Música, MIDI, Realidad Virtual, Unity, Neuroestimulación

Abstract

The purpose of this project has been to represent in an aesthetically pleasing way a musical score in virtual reality (VR) in real-time and get it to be synchronized with the audio playback corresponding to the real life interpretation of the musical piece. This has been achieved by developing a VR application using the video game graphics engine, Unity, that taking a MIDI file as input will allow to represent the different information provided by the MIDI in a 3D environment.

The result of this work is a tool that can be used in an educational field to promote learning and enjoyment of music.

Keywords: Music Score, Music, MIDI, Virtual Reality, Unity, Neuro-stimulation

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Introducción

La música es una forma de expresión artística que ha estado presente en la humanidad desde tiempos prehistóricos y que ha evolucionado con la tecnología y la cultura. Además de ser una forma de entretenimiento y diversión, también se utiliza como medio de comunicación universal y de expresión de emociones y sentimientos que trasciende culturas, idiomas y países. Sin la música presente, no se entendería una sociedad tal y como la conocemos hoy en día.

En este Trabajo de Fin de Grado (TFG) se ha desarrollado una aplicación, que tomando un fichero MIDI (Interfaz Digital de Instrumentos Musicales) como entrada, busca representar de manera estéticamente atractiva una partitura musical en Realidad Rirtual (RV) en tiempo real, de forma que se pueda experimentar una nueva dimensión sensorial, la vista, durante el cotidiano acto de escuchar música.

Además, esto nos va a permitir adentrarnos en el campo de la realidad virtual, para abstraer al usuario del mundo real y colocarlo en un entorno favorable para el disfrute de la experiencia de visualización en su totalidad.

1.2. Objetivos y motivaciones

El objetivo principal de este trabajo, aparte del propósito funcional de facilitar la interpretación de las partituras en pentagrama y de resaltar la belleza simétrica de las partituras, es servir como una herramienta pedagógica en ámbitos educativos.

Según estudios en neuroeducación, nuestro cerebro precisa de la integración multimodal de la información para crear una representación de nuestro entorno, y por lo tanto puede enriquecerse con la realización de actividades que implican la activación de distintas zonas del cerebro que mejorarían el aprendizaje y la conectividad cerebral [1]. Este podría ser el caso de la aplicación que se desarrolla en el presente TFG, al tratarse de la conexión de áreas cerebrales audiovisuales prioritarias en la comunicación social.

Por otro lado, MIDI ha sido empleado para examinar el procesamiento de la interacción

audio-visual de la música con imágenes visuales. Usualmente, el objetivo de estas investigaciones es la mejora de la competencia musical y una mayor efectividad del desempeño de los músicos [2]. En este TFG, la visualización de la música que hemos implementado puede resultar en un mayor enriquecimiento de la experiencia sensorial de la música, lo que redundaría en un mayor disfrute de su audición. Por ello, la aplicación desarrollada podría servir como una herramienta para estimular el gusto por la música en el ámbito educativo.

Las denominadas aulas de estimulación sensorial buscan la mejora de los escolares con necesidades específicas de apoyo educativo mediante la integración de la experiencia auditiva, visual, táctil u olfativa [3]. El presente TFG podría contribuir a la estimulación sensorial de este grupo de escolares. En relación a la discapacidad, en concreto la discapacidad auditiva, se ha demostrado que la integración de estímulos simples multimodales, visual y vibro-táctiles, son capaces de activar procesos emocionales similares a los que se activan cuando se oye una pieza musical; asimismo, refuerza los procesos atencionales [4]. Este proyecto puede contribuir a esta línea de trabajo, al aportar una aplicación donde las piezas musicales se pueden transformar en estímulos visuales en un contexto de realidad virtual.

1.3. Estado del arte

En la actualidad existen diversas formas de representar MIDI de manera gráfica. La forma más común en los programas de edición de audio digital (*Digital Audio Workstation*, DAW) es representar la información MIDI de manera similar a como se hacía en un rollo de pianola, *piano roll*, con el tono de las notas en el eje vertical y su duración en el horizontal (Figura 1.1).

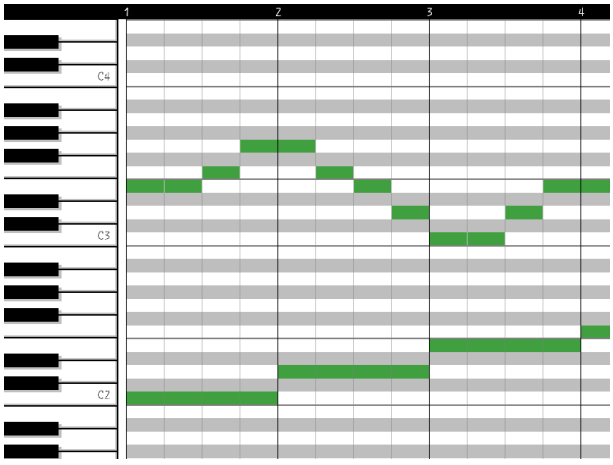


Figura 1.1: Representación de *piano roll* horizontal.

Fuera del entorno profesional existen numerosas representaciones en tiempo real, siendo la más popular una pequeña variación de la anterior, en donde, ahora, las notas musicales caen verticalmente hacia un teclado situado en la parte inferior de la pantalla. Existen varias aplicaciones que realizan esta función como PianoVFX [5] y Synthesia [6] que además de la visualización, tiene una componente pedagógica, en tanto que facilitan la interpretación de piezas para teclado, sin recurrir a las partituras convencionales, que

son sustituidas por una representación 2D más directa de las mismas y con el tempo adecuado, como se muestra en la figura 1.2.

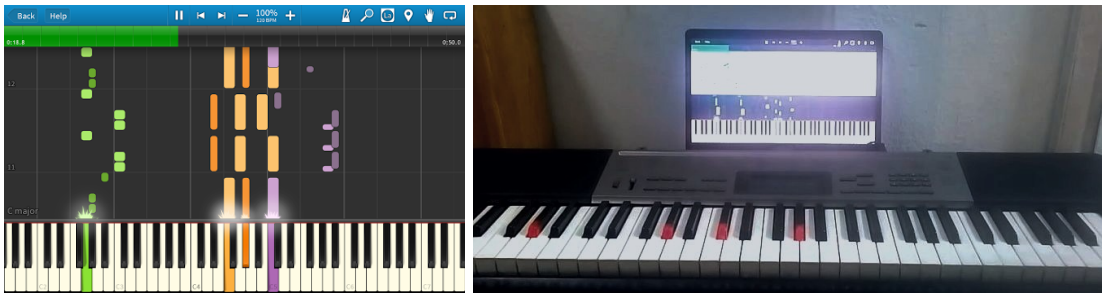


Figura 1.2: Aplicación para Android de Synthesia. A la derecha, conectada a un teclado cuyas teclas se iluminan.

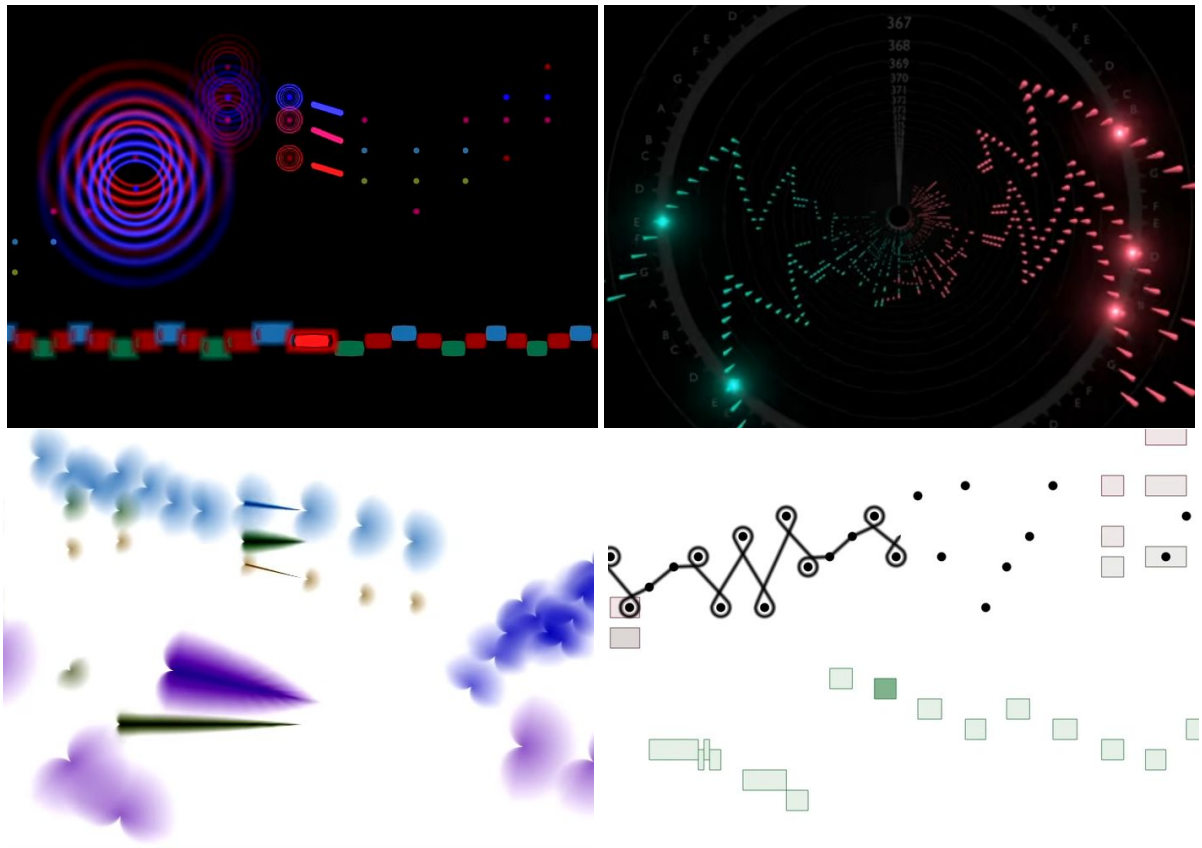


Figura 1.3: De izda. a dcha. y arriba a abajo: representación estéticamente atractiva de las partituras “El vuelo del moscardón” de Rimsky-Korsakov, la “Rapsodia Húngara, no. 2”, de Liszt, el “1^{er} Arabesco” de Debussy, y el “Concierto para violín en La menor” de Bach, disponibles en YouTube [7–10].

Estas formas de representación tienen un propósito funcional: facilitar la interpretación de las partituras en pentagrama. Alternativamente se puede considerar además el plano estético, para resaltar la belleza simétrica de las partituras musicales (Figura 1.3), como añadido al placer del oyente. En este plano, más cercano a los propósitos de este proyecto, destaca el software *Music Animation Machine MIDI Player* (MAMM Player [11]) que mediante algoritmos genera animaciones visuales en tiempo real. En estas animaciones cada nota musical se representa con diferentes formas geométricas que se conectan

temporalmente mediante líneas. Salvo la subfigura superior derecha, las imágenes, y vídeos correspondientes, de la figura 1.3, fueron realizados con esta herramienta.

Por otro lado, el mundo de la realidad virtual ha tenido un gran auge en los últimos años [12, 13]. Sin embargo, al contrario que lo visto con las representaciones en 2D y 3D, no abundan las aplicaciones que permitan visualizar un MIDI en realidad virtual. Entre las pocas existentes, podemos destacar:

- PianoVision [14], aunque del ámbito de la realidad aumentada, es una aplicación para aprender a tocar el piano que permite conectar un teclado MIDI a las gafas de RV que muestra en tiempo real las notas que deben ejecutarse;
- Modulia Studio [15], un controlador MIDI con diversos módulos para crear tu propio estudio de música personalizado, por lo que está más centrado en la creación y edición de música que en la representación gráfica de un MIDI (Figura 1.5).

Como se puede observar, ninguna de las aplicaciones mencionadas está destinada a la visualización de partituras en tiempo real y que además esté sincronizada con la reproducción de audio correspondiente, que es otro de los objetivos de este proyecto.

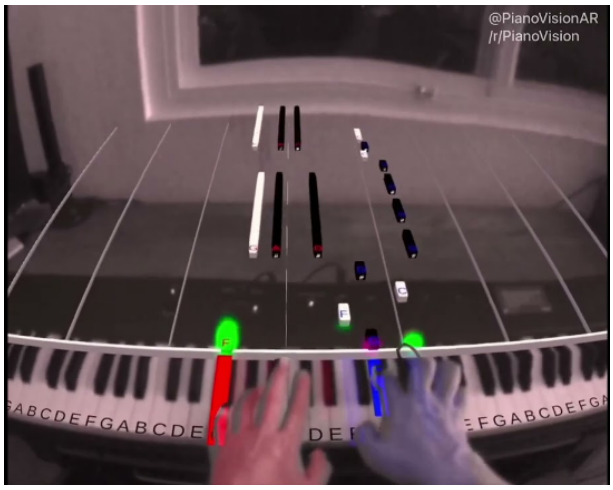


Figura 1.4: Ejemplo de la aplicación PianoVision.

Figura 1.5: Captura de uno de los modos disponibles en Modulia Studio.

Capítulo 2

Metodología de trabajo

2.1. Metodología ágil

La metodología de trabajo que más se identifica con la que se ha utilizado durante el desarrollo de este proyecto ha sido la herramienta de Tableros **Kanban** [16], que son parte de la metodología ágil. Esta técnica se caracteriza por dividir el proceso en tareas mínimas que se organizan en un tablero de trabajo separadas en varias columnas, donde se puede visualizar rápidamente las tareas pendientes, en proceso y finalizadas.

Esto proporciona una excelente visión del estado del proyecto en todo momento así como la capacidad de pautar el desarrollo del trabajo de manera óptima ya que se puede centrar el trabajo en finalizar las tareas prioritarias que todavía no han sido completadas.

En los siguientes apartados se exponen las tecnologías utilizadas para llevar a cabo la metodología nombrada.

2.1.1. Trello

Trello [17] es una herramienta en línea para la gestión de proyectos y del flujo de trabajo que utiliza tableros virtuales y tarjetas para organizar y visualizar la información. En este proyecto se ha hecho uso de un tablero a modo de Kanban con cuatro columnas para separar las tareas que estaban pendientes de realizar, en proceso, en revisión, y las completadas, de modo que siempre se tenía una línea de trabajo clara para el alumno y los tutores, en la que cada vez que se realizaba una tarea esta se sometía a revisión antes de darla por finalizada. Esto permitió tener siempre presentes los objetivos que se querían alcanzar tanto a corto como a largo plazo. En la Figura 2.1 se muestra el tablero, creado con esta herramienta, utilizado durante el trabajo.

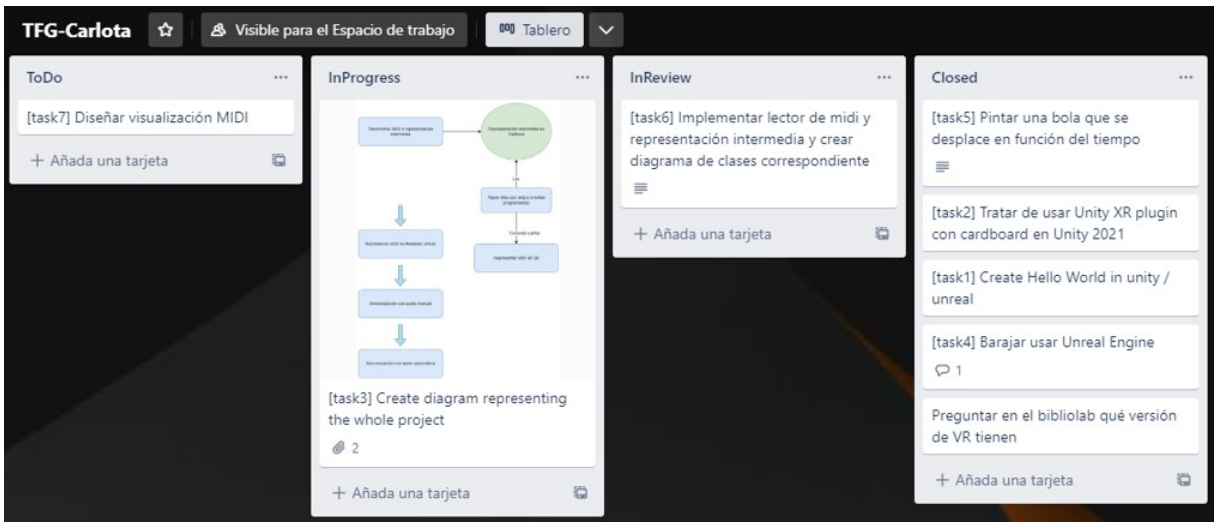


Figura 2.1: Captura del tablero de Trello en un momento dado del proyecto.

2.1.2. Control de versiones

Bitbucket [18] es la plataforma de alojamiento de repositorios de Git y Mercurial en la nube que hemos elegido para este proyecto. En todo desarrollo de software es sumamente importante hacer uso de un sistema de control de versiones como Git [19], ya que permite realizar un seguimiento de todos los cambios en el código fuente a lo largo del proyecto. Además, permite que los desarrolladores no interrumpan su flujo de trabajo dado que pueden trabajar simultáneamente en ramas o versiones de código diferentes. El esquema de desarrollo que se siguió en este proyecto constaba de una rama principal *develop* y, por cada nueva funcionalidad a añadir, se creaba una rama *feature*, de modo que el proceso de trabajo fue el siguiente:

1. Crear una rama *feature*.
2. Implementar funcionalidad (con sus *commits* correspondientes).
3. Revisar funcionalidad por parte del tutor.
4. Fusionar rama con rama *develop*.
5. Eliminar rama creada.

Este esquema de trabajo con ramas es una adaptación de GitFlow [20], un modelo que ha sido muy utilizado en la última década.

Capítulo 3

Desarrollo del proyecto

En este capítulo se describe el proceso de desarrollo del proyecto, las decisiones tomadas, las herramientas utilizadas y las tareas realizadas.

3.1. Dispositivo de realidad virtual

El dispositivo de realidad virtual disponible para el desarrollo de este proyecto ha sido las gafas *Google Daydream VR* [21]. Estas gafas lanzadas en 2016 permiten disfrutar de la realidad virtual en 3D y 360 grados, pero solo actúan a modo de soporte y no disponen de pantalla integrada (Figura 3.1), por lo que para que el dispositivo funcione se debe introducir en él un *smartphone* compatible con la tecnología bautizada como "*Daydream Ready*". Son compatibles entre otros los modelos de móvil Google Pixel y Pixel XL, y los Motorola Moto Z y Moto Z Force, pero no muchos más. Desafortunadamente, el producto ha sido descontinuado por lo que ha dejado de estar a la venta y su *plugin* dedicado ya no recibe soporte [22]. Sin embargo, siguen siendo compatibles con el *plugin* para las gafas *Google Cardboard* [23], de su misma línea de producto, por lo que se ha continuado el desarrollo con este segundo *plugin*.

Figura 3.1: Gafas Google Daydream VR. A la derecha, las Google Cardboard.

3.2. Unity

El dispositivo de RV disponible tuvo una influencia absoluta en la elección del motor gráfico de videojuegos utilizado para desarrollar el proyecto. Al inicio del proyecto, luego de haber especificado el alcance y los objetivos, se valoraron las ventajas y desventajas de los tres motores más populares del mercado, Unity [24], Unreal Engine [25] y Godot Engine [26], para determinar el más adecuado a utilizar en este caso. En primer lugar, se descartó Godot ya que no disponía de soporte oficial para Cardboard ni para Daydream. Después, se realizó un proyecto de prueba en cada motor gráfico restante para terminar de determinar qué motor era más favorable para los requisitos del proyecto. Sin embargo, pronto se tuvo que descartar Unreal dado que la obsolescencia del *plugin* de Cardboard de la plataforma dificultaba enormemente lograr un entorno estable y mantenible para desarrollar. Por lo tanto, fue el proyecto funcional de Unity al que se le dio continuidad.

Unity [24] es un motor gráfico de desarrollo de videojuegos multiplataforma 2D y 3D, que permite una gran versatilidad para desarrollar un proyecto. Posee una gran cantidad de herramientas, documentación, y recursos gratuitos que facilitan la implementación de las tareas propuestas. Para la realización de los scripts, Unity utiliza el lenguaje de programación C#, un lenguaje con el que la autora no estaba habituada pero que es bastante parecido a C++, el cual sí se utiliza a lo largo del grado, por lo que no hubo grandes problemas en la adaptación. Además, la licencia de Unity es completamente gratuita para estudiantes y también para particulares hasta cierto umbral comercial.

La versión de Unity utilizada en este proyecto ha sido la 2021.3.15f1, incluyendo los módulos de Android SDK, NDK Tools y OpenJDK con la plantilla básica de proyecto 3D. Además, se han añadido los *plugins* necesarios para Google Cardboard y realidad extendida (XR) compatibles con el dispositivo de RV que tenemos para el desarrollo. Con todo lo mencionado, la escena de prueba inicial proporcionada por el *plugin* de Cardboard funcionaba correctamente en Android.

3.3. MIDI

Una vez elegida la plataforma de desarrollo, se investigó sobre qué era y qué estructura sigue un fichero MIDI [27], así como la información que proporciona, para conocer con exactitud los datos de los que disponíamos para trabajar y modelar después la visualización.

MIDI es un estándar tecnológico creado a principios de los años 80 con el fin de estandarizar la comunicación entre los diferentes tipos de hardware musical (computadores, sintetizadores, secuenciadores, controladores, etc.). Describe un protocolo, una interfaz digital y conectores para que los dispositivos se conecten y se comuniquen entre sí. Una simple conexión MIDI puede transmitir hasta 16 canales de información que pueden ser conectados a diferentes dispositivos cada uno.

Un aspecto importante de MIDI es que no transmite el audio en forma de onda –*waveform*– sino que transporta información sobre los eventos de la pieza musical, que es-

pecifican tono, velocidad (intensidad), duración, entre otras características de las notas y contiene las señales de reloj necesarias para sincronizar el tempo entre varios dispositivos. Por este motivo, el tamaño del archivo es mucho menor que si contuviera una grabación musical. Así un fichero con formato .wav, .ogg o .mp3, que sí contienen sonido en forma de onda reproducible (una vez descomprimido) por un altavoz, ocupa megabytes, donde su equivalente .mid ocupará kilobytes. A cambio el .mid no es reproducible directamente por un altavoz, sino que esas instrucciones deben ser procesadas antes por un dispositivo con capacidad de sintetizar sonidos para recrear la forma de onda.

Se ha elegido un fichero en formato MIDI como entrada de la aplicación porque estos archivos tienen un tamaño compacto y permiten una fácil manipulación. Además hay disponibilidad pública y gratuita de todo tipo de piezas musicales representadas en este formato. Están formados por una cabecera que contiene la información general de la obra (número de pistas y tempo) y por segmentos de pista, con una cabecera similar a la del fichero, seguida de una serie de eventos MIDI.

3.4. Librería DryWetMIDI

Se determinó que la manera más eficaz y eficiente de hacer la lectura del fichero MIDI sería a través de la librería de .NET, **DryWetMIDI** [28]. Esta librería, muy completa, permite leer, crear y modificar ficheros MIDI con suma facilidad, así como su reproducción mediante un sintetizador, aunque esta característica solo está disponible en Windows y MacOs. Además, es altamente personalizable y tiene una extensa y organizada documentación muy útil para la correcta importación e implementación de sus funcionalidades.

Para poder obtener el fichero en la aplicación Android sin tener que descargarlo y colocarlo en la ruta adecuada manualmente, fue necesario hacer uso de `UnityWebRequest`, ya que Android utiliza archivos `.apk` comprimidos y no se puede acceder a la carpeta con el fichero directamente. Como su nombre indica, `UnityWebRequest` [29] es un sistema modular para componer peticiones y manejar respuestas `HTTP`. En nuestro programa, primero enviamos una petición `GET` del fichero localizado en la carpeta `StreamingAssets`, si la respuesta del servidor es exitosa entonces descargamos el fichero en el móvil en la ruta apuntada por `Application.persistentDataPath()` ya que en esta ruta los datos se guardan entre las actualizaciones de la aplicación.

Los parámetros MIDI que se extrajeron del fichero fueron los siguientes:

- **Nota ON**. Momento en el que se pulsa la tecla.
- **Nota OFF**. Momento en el que se suelta la tecla.
- **Tono**. Tono de la nota
- **Duración**. Tiempo que dura la nota en microsegundos.
- **Velocidad**. Intensidad con la que se toca la nota
- **Canal**. Indica el canal MIDI en el que se reproduce la nota.

3.5. Diseño de la visualización y el entorno

Tras comprobar que la lectura del fichero funcionaba correctamente el siguiente paso fue diseñar la representación gráfica de los eventos MIDI. Para este paso, se creó un diagrama (Figura 3.2) para poder tener una mejor idea general sobre cómo relacionar los diferentes elementos extraídos del fichero MIDI con una cualidad, efecto u objeto que se pudiera programar dentro del entorno de Unity, es decir, asignar un significado visual a la información musical.

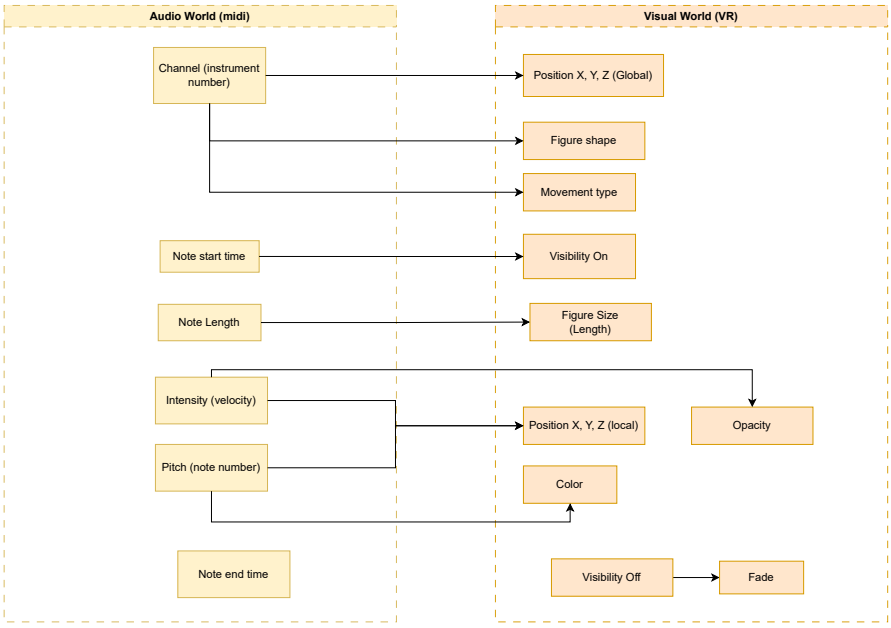


Figura 3.2: Esquema inicial del diseño de visualización.

En otras aplicaciones las notas se disponen sobre un fondo de color sólido, generalmente oscuro, y se mueven acercándose a un plano o línea que supone el momento en que deben comenzar a sonar. La longitud de las notas es tal que se sincroniza su entrada y salida del plano de sonido con su duración sonora mediante la velocidad de avance de las notas acercándose a este “horizonte de eventos”.

Siguiendo esas mismas ideas se generó un modelo, que es el mostrado en la figura 3.3 y supone un compendio de los métodos de la figura 1.3. Tenemos hasta 5 generadores de notas (`spawners`) colocados alrededor del usuario. Se ha elegido un número bajo de generadores para que colocados con una separación adecuada no abarroten la escena y no produzcan una sensación caótica, dado que lo que queremos conseguir es lo contrario, armonía y sosiego.

Al realizar pruebas y mostrarlo a otras personas, se comprobó que este diseño de visualización no era lo suficientemente agradable e interesante.

Lo achacamos a que se crea unas relaciones entre notas pasadas, presentes y futuras que puede ser útil para la interpretación, pero no coincide con la sensación de un oyente

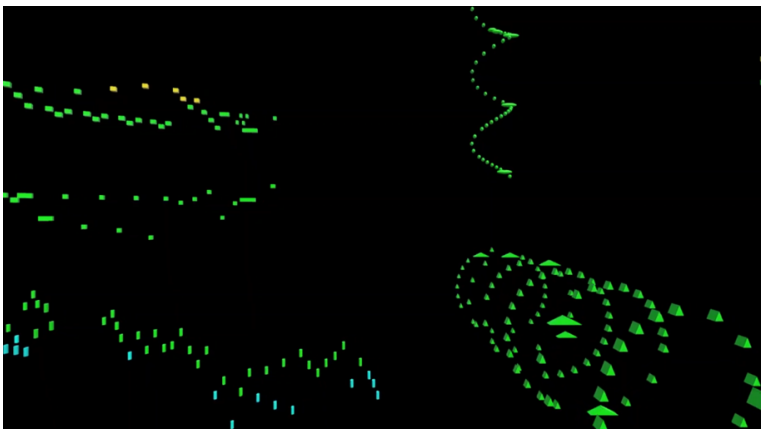


Figura 3.3: Aspecto de la primera versión.

al cual las notas solo se le presentan mientras suenan. Hemos optado por un método de representación visual más similar a la experiencia auditiva, donde las notas no existen hasta que suenan, y solo existen en el recuerdo por un breve momento tras sonar.

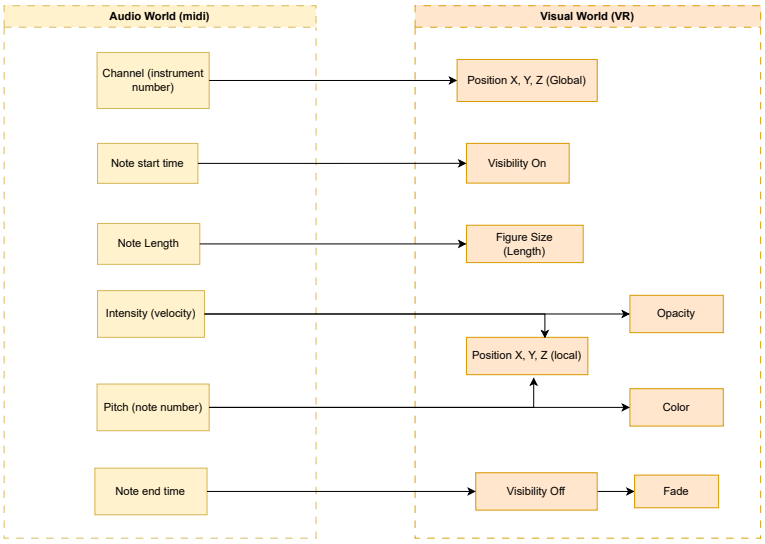


Figura 3.4: Esquema final del diseño de visualización.

En este nuevo diseño además se mejoró el aspecto de las notas al añadir un brillo y una luz alrededor de cada objeto y se creó un nuevo entorno (Figura 3.4).

En el diseño final las notas se “encenderán” y flotarán en el vacío rodeando al oyente. Siempre más o menos en la misma posición para las notas de un mismo instrumento. Aparecerán cuando un evento de comienzo de nota así lo indique en el MIDI y desaparecerán con los eventos de nota terminada. Además lo harán de forma gradual, disminuyendo la opacidad del objeto instanciado hasta llegar a desaparecer completamente. Esto se basa en la idea de que la música solo existe en el presente y lo único importante es la

atmósfera sonora que existe en ese momento. Cualquier evento pasado o futuro no tiene importancia para el disfrute en dicho instante.

El entorno que rodeará al oyente es el vacío, o casi, dado que se debe colocar una serie de superficies rodeando al sujeto para que los efectos de luz que emitan las notas resalten al chocar con las mismas.

Esta última versión descrita (Figura 3.5) ha sido la que ha prevalecido en el desarrollo final de este proyecto aunque las funcionalidades mencionadas en la primera versión son totalmente válidas para implementar, con alguna pequeña modificación de sus características, en el futuro.

Figura 3.5: Aspecto de la versión final.

3.6. Sincronización audio

Por último se realizó la sincronización manual del audio para Android. De tal manera que en la aplicación de realidad virtual solo la visualización se extrae del MIDI, mientras que el audio proviene de una grabación en formato `MP3` de la interpretación con instrumentos reales de la misma obra musical. Esto es necesario ya que la interpretación de la partitura MIDI mediante sintetizadores peca de “robótica”, carente de los matices que incorporan los músicos a la partitura, y muchas veces incompleta o inexacta respecto de la obra musical como la planteó su autor. Para que la concordancia entre tiempos de representación de notas y su ejecución en la obra pregrabada sea total se requiere alinear previamente ajustando los tiempos de inicio y la duración.

Capítulo 4

Conclusiones y líneas futuras

4.1. Conclusiones

Se ha concluido una primera versión completa de la aplicación que cumple con todos los objetivos marcados al principio del proyecto, excepto el más complejo, la alineación automática del audio, que no ha podido realizarse debido a la escasez de tiempo. La búsqueda de una visualización estéticamente atractiva y a la vez educativamente significativa junto al uso de una tecnología nunca utilizada anteriormente (Unity) por la autora han jugado un factor importante para que sólo se haya podido completar la sincronización manual del audio. No obstante, la aplicación sigue siendo completamente funcional y cumple adecuadamente con su enfoque didáctico.

En cuanto al valor estético del proyecto, algo mucho más difícil de evaluar, se ha logrado implementar una visualización que encaja de la forma más universal posible a la gran variedad de gustos personales que existen sin olvidar las connotaciones para el aprendizaje musical.

Para finalizar, hubiera estado bien haber podido utilizar algún dispositivo de RV más moderno de forma que se ofreciera una experiencia más inmersiva y de más calidad en el producto final, ya que las Google Daydream solo sensan la rotación de la cabeza del usuario, mientras que otros dispositivos como las Oculus Rift o las HTC Vive son capaces de sensar también la posición espacial absoluta.

4.2. Líneas futuras

Aunque el proceso de desarrollo de la aplicación para este Trabajo de Fin de Grado haya finalizado, de cara al futuro, a la aplicación todavía le queda un amplio margen de mejora para añadir nuevas funcionalidades y características. Algunas de las ideas más destacadas que se podrían implementar son las siguientes:

- **Soporte para un dispositivo de RV con capacidades hápticas**. Adaptar el proyecto a un dispositivo más moderno y más inmersivo para completar la experiencia

visual y auditiva ya ofrecida con tecnología háptica que simule la intensidad y las vibraciones que genera la música en la realidad.

- **Personalización de la visualización**. Añadir parámetros para que los usuarios puedan modificar las opciones de visualización a su gusto como elegir una paleta de colores, cambiar la forma que adoptan las notas, añadir movimientos, cambiar el entorno en el que se encuentran, etc.
- **Información musical**. Permitir la interacción con los diferentes elementos del entorno de RV para mostrar información sobre las notas y la pieza musical en reproducción (tono, duración, intensidad, . . .).

Además de las tareas que no se pudieron llevar a cabo en la planificación del proyecto inicial:

- **Sincronización automática de audio**. Utilizar algún algoritmo de alineación automática para soportar la reproducción de una mayor cantidad de piezas musicales.
- **Utilización de controladores RV**. Obtener un dispositivo con mandos de RV con los que poder interpretar música, de forma que se genere información MIDI en tiempo real y se muestre visualmente a través de la aplicación.

Capítulo 5

Conclusions and future lines

5.1. Conclusions

A first major version of the application has been completed that meets all the objectives set at the beginning of the project, except for the most complex one, the automatic audio alignment, which could not be completed due to time constraints. The search for an aesthetically appealing and, at the same time, educationally meaningful visualization together with the use of a technology never used before (Unity) by the author has played an important factor so that only the manual audio synchronization could be completed. Nevertheless, the application remains fully functional and adequately fulfills its didactic approach.

As for the aesthetic value of the project, something much more difficult to evaluate, it has been possible to achieve the implementation of a visualization that fits as universally as possible to the wide variety of personal tastes that exist without forgetting the connotations for musical learning.

Finally, it would have been nice to have been able to use a more modern VR device in order to offer a more immersive and higher quality experience in the final product, given that the Google Daydream only sense the roationt of the head of the user while other devices such as the Oculus Rift or the HTC Vive headsets are capable of sensing the absolute spatial position as well.

5.2. Future development

- **Support for a VR device with haptic capabilities**. Adjust the project to a more modern and immersive device to complete the visual and auditory experience already offered with haptic technology that simulates the intensity and vibrations generated by music in real life.
- **Customization of the visualization**. Add parameters so that users can modify the visualization experience by changing options to their liking such as choosing a color palette, changing the shape of the notes, adding different movements, changing the

VR environment, etc.

- **Musical information**. Allow interaction with the different elements of the VR environment to display information about the notes and the piece being played (pitch, duration, intensity, ...).

In addition to the tasks that could not be carried out from the initial project:

- **Automatic audio synchronization**. Use an automatic alignment algorithm to support the playback of a larger number of musical pieces.
- **Use VR controllers**. Acquire a device with VR controllers to be able to play music, so that MIDI information is generated in real time and displayed visually through the application.

Capítulo 6

Presupuesto

6.1. Licencias de software

Nombre del software	Coste	Anotaciones
Unity	0.00 €	Gratuito. Plan educativo/personal
Visual Studio	0.00 €	Plan gratuito
Git	0.00 €	Gratuito. Código libre
Trello	0.00 €	Plan gratuito
Overleaf	0.00 €	Gratuito. Código libre
LaTeX	0.00 €	Gratuito. Código libre
Windows 10	0.00 €	Coste incluido en el precio del equipo

Cuadro 6.1: Coste de las licencias de software utilizadas.

6.2. Materiales

Material	Valor	Factor de amortización	Coste
Portátil	1.000 €	5 %	200 €
Gafas Google Daydream VR / Cardboard	100 €	5 %	20 €
Total:			220 €

Cuadro 6.2: Coste de los materiales utilizados

6.3. Costes de personal

Asumiendo un coste por hora de 20€.

Tarea	Horas	Coste
Investigación mercado	5	100.00 €
Diseño entorno/visualización	100	2000.00 €
Implementación en RV	120	2400.00 €
Sincronización audio	25	50.00 €
Elaboración informe	35	70.00 €
Total	285	4620.00 €

Cuadro 6.3: Costes de personal.

6.4. Presupuesto final del proyecto

Motivo	Coste
Licencias de software	00.00 €
Materiales	220.00
Mano de obra	4620.00 €
Coste	4840.00 €
Beneficio (6 %)	290.40 €
Presupuesto total del proyecto	5130.40 €

Cuadro 6.4: Presupuesto final del proyecto.

Apéndice A

Códigos

A.1. Clase MidiReader (Métodos más destacados)

```
1  public class MidiReader : MonoBehaviour
2  {
3      private const float TimeScale = 1000000f;
4      private const float SizeScale = 10.0f;
5      private const float Height = 0.1f;
6      private const float Width = 0.1f;
7
8      private MidiFile _midiFile;
9
10     public string filename = "Badinerie.mid";
11     public GameObject[] shapes;
12     public GameObject[] spawners;
13     public Dictionary<int, GameObject> Notes = new();
14
15     private Transparency _transparency = new();
16
17
18     private IEnumerator GetFilePath()
19     {
20         var request =
           ↪  UnityWebRequest.Get(Path.Combine(Application.streamingAssetsPath,
           ↪  filename));
21         yield return request.SendWebRequest();
22
23         if (request.result != UnityWebRequest.Result.Success)
24         {
25             Debug.Log(request.error);
26         }
27         else
28         {
29             File.WriteAllBytes(Path.Combine(Application.persistentDataPath,
               ↪  filename), request.downloadHandler.data);
30         }
31     }
32
33     private void SpawnNote(Note note, TempoMap tempoMap, int noteId)
34     {
```

```
35          float pitch = note.NoteNumber / SizeScale;
36          float noteVelocity = note.Velocity / SizeScale;
37          GameObject gameObject = null;
38          Vector3 position;
39          Vector3 spawnPoint;
40
41          switch (note.Channel)
42          {
43              case 0:
44                  spawnPoint = spawners[0].transform.position;
45                  position = new Vector3(Random.Range(-6f, 6f), spawnPoint.y +
                    ↪  pitch, noteVelocity);
46                  gameObject = Instantiate(shapes[0], position,
                    ↪  transform.rotation);
47                  ModifyNoteVisuals(gameObject, note, tempoMap);
48                  break;
49
50              case 1:
51                  spawnPoint = spawners[1].transform.position;
52                  position = new Vector3(-noteVelocity, spawnPoint.y + pitch,
                    ↪  Random.Range(-6f, 6f));
53                  gameObject = Instantiate(shapes[1], position,
                    ↪  Quaternion.AngleAxis(90f, Vector3.up));
54                  ModifyNoteVisuals(gameObject, note, tempoMap);
55                  break;
56
57              case 2:
58                  spawnPoint = spawners[2].transform.position;
59                  position = new Vector3(noteVelocity, spawnPoint.y + pitch,
                    ↪  Random.Range(-6f, 6f));
60                  gameObject = Instantiate(shapes[2], position,
                    ↪  Quaternion.AngleAxis(90f, Vector3.up));
61                  ModifyNoteVisuals(gameObject, note, tempoMap);
62                  break;
63
64              case 3:
65                  spawnPoint = spawners[3].transform.position;
66                  position = new Vector3(Random.Range(-6f, 6f), spawnPoint.y +
                    ↪  pitch, -noteVelocity);
67                  gameObject = Instantiate(shapes[3], position,
                    ↪  transform.rotation);
68                  ModifyNoteVisuals(gameObject, note, tempoMap);
69                  break;
70
71              case 4:
72                  spawnPoint = spawners[4].transform.position;
73                  position = new Vector3(Random.Range(-6f, 6f), spawnPoint.y +
                    ↪  pitch, noteVelocity);
74                  gameObject = Instantiate(shapes[4], position,
                    ↪  transform.rotation);
75                  ModifyNoteVisuals(gameObject, note, tempoMap);
76                  break;
77
78              default:
79                  Debug.Log("The file has more than 5 channels");
80                  break;
81          }
82
```

```
83          if (gameObject != null)
84          {
85              if (Notes.Keys.Contains(noteId))
86              {
87                  throw new Exception("This note ID was already registered");
88              }
89              Notes.Add(noteId, gameObject);
90          }
91      }
92
93      private void ModifyNoteVisuals(GameObject gameObject, Note note, TempoMap
        ↪  tempoMap)
94      {
95          gameObject.transform.localScale =
96              new
                ↪  Vector3(note.LengthAs<MetricTimeSpan>(tempoMap).TotalMicroseconds
                ↪  / TimeScale, Height, Width);
97
98          gameObject.GetComponent<MeshRenderer>().material.color = new
            ↪  Color(Random.Range(0f, 1f), Random.Range(0f, 1f), Random.Range(0f,
            ↪  1f));
99          var glowColor = Random.ColorHSV(0, 1f, 0.5f, 1f, 0.7f, 1f);
100         Material material = gameObject.GetComponent<MeshRenderer>().material;
101         material.SetColor("_EmissionColor", glowColor);
102         gameObject.GetComponent<Light>().color = glowColor;
103
104         var materialTransparency =
            ↪  gameObject.GetComponent<MeshRenderer>().material.color;
105         materialTransparency.a = _transparency.SetTransparency(note.Velocity);
106         gameObject.GetComponent<MeshRenderer>().material.color =
            ↪  materialTransparency;
107     }
108
109
110     private void OnNotesPlaybackStarted(object sender, NotesEventArgs e)
111     {
112         var tempoMap = _midiFile.GetTempoMap();
113         foreach (var note in e.Notes)
114         {
115             var noteId = note.GetHashCode();
116             UnityMainThread.Wkr.AddJob(() =>
117             { SpawnNote(note, tempoMap, noteId); });
118         }
119     }
120
121     private void OnNotesPlaybackFinished(object sender, NotesEventArgs e)
122     {
123         UnityMainThread.Wkr.AddJob(() =>
124         {
125             foreach (var note in e.Notes)
126             {
127                 GameObject gameObject = Notes[note.GetHashCode()];
128                 gameObject.GetComponent<Fade>().StartFade();
129             }
130         });
131     }
132 }
```

A.2. Clase Fade

```
public class Fade : MonoBehaviour
{
    private bool _fading;
    private Color _fadeColor;
    private float _fadeStartTime = -1;
    private const float FadeDuration = 0.20f;

    // Start is called before the first frame update
    private void Start()
    {
         _fadeStartTime = Time.time;
    }

    // Update is called once per frame
    private void Update()
    {
        if (_fading)
        {
            float p = 1.0f - (Time.time - _fadeStartTime) / FadeDuration;
            if (p < 0)
            {
                Destroy(gameObject);
            }
            else
            {
                _fadeColor.a = p;
                GetComponent<MeshRenderer>().material.color = _fadeColor;
            }
        }
        else
        {
            _fadeStartTime = Time.time;
        }
    }

    public void StartFade()
    {
        _fadeColor = gameObject.GetComponent<Renderer>().material.color;
        _fading = true;
    }
}
```

Bibliografía

[1] H. A. Bucheli-López, B. P. Rojas-Arango, S. M. Vergara-Henao, and M. C. Rodríguez-Niño, "Aspectos motivacionales para generar actividades cerebrales óptimas en el proceso de aprendizaje en un Ambiente Virtual de Aprendizaje," *Trilogía Ciencia Tecnología Sociedad*, vol. 13, p. 65–88, ene. 2021.

[2] A. Kapur, G. Wang, P. Davidson, and P. Cook, "Interactive network performance: A dream worth dreaming?," *Organised Sound*, vol. 10, pp. 209 – 219, 12 2005.

[3] L. Agudelo Gómez, L. A. Pulgarín Posada, and C. Tabares Gil, "La estimulación sensorial en el desarrollo cognitivo de la primera infancia / sensory stimulation in cognitive development of early childhood," *Revista Fuentes*, vol. 19, p. 73–83, mar. 2017.

[4] Á. García López, "Nuevos sistemas hápticos para la evocación de emociones en eventos audiovisuales en personas con discapacidad auditiva," 2022.

[5] "Visualize your piano experience!." https://piano-vfx.com/.

[6] "Synthesia, piano for everyone." https://synthesiagame.com/.

[7] S. Malinowski and Nikolai Rimsky Korsakov, "El vuelo del moscardón, el cuento del Zar Saltan." https://youtu.be/cFdY-XQG4uk.

[8] A. Fillebrown and Franz Liszt, "Rapsodia Húngara, no. 2." https://youtu.be/m6xWGVhZl1g.

[9] S. Malinowski and Claude Debussy, "Primer Arabesco." https://youtu.be/Yt1jfX5C1u0.

[10] S. Malinowski and J. S. Bach, "Concierto para violín en La menor, BWV 1043." https://youtu.be/tbWqPnRbq3M.

[11] S. Malinowski, "Music Animation Machine MIDI Player." https://www.musanim.com/Player/.

[12] T. Mazuryk and M. Gervautz, "Virtual reality - history, applications, technology and future," December 1999. http://www.cg.tuwien.ac.at/research/publications/1996/mazuryk-1996-VRH/TR-186-2-96-06Paper.pdf.

[13] Y. M. Akinola, O. C. Agbonifo, and O. A. Sarumi, "Virtual reality as a tool for learning: The past, present and the prospect," *2*, 2020.

[14] "An augmented reality piano platform." https://www.pianovision.app/.

[15] “Infinite midi controller.” https://www.modulia-studio.com/.

[16] “Metodología kanban.” urlhttps://kanbantool.com/es/metodologia-kanban.

[17] “Trello brings all your tasks, teammates, and tools together.” https://trello.com/.

[18] Atlassian, “Git solution for teams using jira.” https://bitbucket.org/.

[19]

[20] “A successful git branching model.” https://nvie.com/posts/a-successful-git-branching-model/.

[21] “Daydream view hardware.” https://support.google.com/daydream/answer/7185269?hl=en&ref_topic=7105096.

[22] “Daydream service update.” https://support.google.com/daydream/answer/7185096?hl=en&ref_topic=7105096.

[23] “Google cardboard.” https://arvr.google.com/intl/es_es/cardboard/.

[24] U. Technologies. https://unity.com/.

[25] “Unreal engine.” https://www.unrealengine.com/es-ES.

[26] G. Engine, “Godot game engine.” https://godotengine.org/.

[27] “Standard MIDI-file format spec. 1.1, updated.” http://www.music.mcgill.ca/~ich/classes/mumt306/StandardMIDIfileformat.html.

[28] “DryWetMIDI Overview.” https://melanchall.github.io/drywetmidi/index.html.

[29] U. Technologies, “Unity Web Request.” https://docs.unity3d.com/ScriptReference/Networking.UnityWebRequest.html.