HARMONYPAL: Un Plugin para la práctica con acordes de piano

HARMONYPAL: A Plugin for practicing piano chords

AUTOR: JAIME CALZADA FERRERAS

DIRECTOR: MIGUEL GÓMEZ-ZAMALLOA GIL

TRABAJO DE FIN DE GRADO EN EL GRADO EN INGENIERÍA DEL SOFTWARE

CURSO 2021 - 2022

FACULTAD DE INFORMÁTICA

UNIVERSIDAD COMPLUTENSE MADRID

RESUMEN

HarmonyPal es un programa que recibe las notas tocadas en un piano digital para analizar acordes, generar acordes nuevos que encajen con el actual y mostrarle al usuario toda esta información en un formato fácil de comprender con velocidad. Su objetivo es asistir en la práctica musical a dos perfiles generales de usuarios: por un lado, estudiantes de piano de niveles iniciales e intermedios, a quienes entender cómo llevar el hilo musical con autoconsciencia y agilidad suele resultar complicado sin la presencia de un profesor. Por otro, compositores aficionados o con destreza en aspectos musicales no relacionados con la armonía, de sobra conocedores de la no siempre productiva experiencia compositiva de sentarse largos ratos al piano e improvisar hasta toparse con unos acordes que suenen bien.

En mi experiencia personal, me he visto a mí mismo en estas dos situaciones. incluso, en la primera, en el puesto del alumno y del profesor. Como conocedor de las dificultades que atañe este aspecto de la práctica pianística y compositiva, he decidido dedicar mi Trabajo de Fin de Grado a desarrollar una herramienta que, de una forma u otra, pudiera servir a mis alumnos para mejorar sus habilidades por su cuenta y aumentar su enfoque alejándoles de esfuerzos innecesarios.

La herramienta toma la forma de un plugin MIDI, un programa que puede ser ejecutado de manera independiente o como extensión de un DAW, software de producción musical. Su utilización, para el músico mínimamente afín a la informática, es trivial, y no requiere más que de un teclado de piano con conexión al computador para funcionar. Una vez esté listo y funcionando, su respuesta es inmediata, y tan sólo necesita que el joven músico tenga conocimientos básicos de armonía para entender sus funcionalidades.

Espero que con el desarrollo de esta herramienta pueda llegar a servir de ayuda a alguna persona que comparta mis pasiones y, en el mejor de los casos, llegar a facilitarle las tareas de aprendizaje de la manera en que me hubiera gustado a mí en su momento.

PALABRAS CLAVE: PIANO, ACORDE, ARMONÍA, MIDI, PLUGIN

ABSTRACT

HarmonyPal is a music program wich receives an input form a digital piano keyboard and retrieves from it the information of a played chord, algorithmically generates new chord suggestions to play next and shows all the information to the user in a simple interface. It aims to assist musical practice of two kinds of users in particular: on the one hand, entry-level students, who may at that point struggle with some aspects of the harmony as understood from the piano without their teacher, such as keep on playing chords without interruption and that much thinking. On the other, aspiring composers, or more advanced and creative piano students, especially those not well versed with the harmony aspect of music theory, who may need a little boost to their imagination when they are trying chord sequences for their pieces, or simply improvising.

I have found myself in all of these situations, both as the teacher and the student. With all the knowledge of the ups and downs of these processes, I decided to dedicate this work to the development of a tool that can help those that might share these difficulties at one point or another, thinking of what would my students want to have at their disposal while I´m not there to help them. This tool might help them focus on the more practical aspects and cut time with the musical theory, wich might at the beginning be an entry barrier more than a tool for them.

HarmonyPal takes the shape of a MIDI plugin, a program that can be run either on its own or as an extension of a Digital Audio Workstation. Its usage is simple, needing only the most basic knowledge of music computing, and its requisits go no further than a digital piano, or MIDI controller, and a PC. Once up and running, its results are immediately apparent, and only need from the user basic piano knowledge once again to make use of it.

I implemented this application with the hopes that it can help those people such as my students and my musician pals, and be at least that little push they may need to reach further heights in their musical learning journey, such as I would have liked to have back then.

KEYWORDS: PIANO, CHORD, HARMONY, MIDI, PLUGIN

ÍNDICE

CAPÍTULO 1

Introducción

1.1 Motivación

Este proyecto nace como la consecuencia inevitable de mis circunstancias académicas, compaginando mi tiempo de estudios entre dos grandes disciplinas. Por un lado, la informática, pues mi titulación en Ingeniería del Software ve su conclusión con este Trabajo de Fin de Grado. Por otro, la música, y específicamente las especialidades de mis estudios en el conservatorio (Piano y Composición), que han sido desde el principio la materia en torno a la que he querido desarrollar el trabajo.

El trabajo comienza a desarrollarse junto a Javier Navalón. En la fase inicial pretendíamos desarrollar una aplicación con múltiples funcionalidades, un alcance muy grande y un enfoque poco concentrado. Durante este período él trabajó con la producción de sonidos musicales, y yo con la representación de la información de las notas. Tras la escisión, cada uno persiguió su enfoque por su lado. Su proyecto, RythmPal, terminó siendo una herramienta para generar acompañamientos musicales a partir de una melodía en tiempo real, mientras que el mío parte de los acordes del usuario para generar propuestas de nuevos acordes que encajen musicalmente uno detrás de otro.

A la hora de reenfocar mi proyecto, partí de mi experiencia como profesor de piano y escogí elaborar un programa que me ayudara a hacer comprender a mis alumnos ciertos fundamentos de la práctica básica del instrumento. En el piano, esencialmente, se separan las tareas de mano derecha y mano izquierda en melodía y acordes respectivamente, y es con esta segunda, siendo frecuentemente la mano no dominante y con una función relegada a “secundaria” por no ejecutar las melodías que cualquiera puede tararear, con la que constantemente encuentro los problemas de aprendizaje más severos. El germen de la idea de HarmonyPal está en las dudas y problemas que puede encontrarse un neófito de la práctica pianística cuando trabaja acordes en la mano izquierda por su cuenta, pero creo firmemente en sus posibilidades para asistir a músicos de mayor calibre a la hora de buscar sonoridades menos evidentes y ganar fluidez con la teoría de la armonía.

HarmoyPal es, en definitiva, una herramienta que analiza musicalmente el acorde que toque el usuario, genera tres acordes que suenen bien junto al primero y muestra cómo debería tocarlos, para que descubra nuevas sonoridades, practique la digitación y no se estanque en fórmulas repetidas a la hora de improvisar o componer.

1.2 Contexto de la informática musical

Para entender exactamente qué es HarmonyPal a nivel informático, debemos hablar de un par de ideas. En primer lugar, los **DAW** (Digital Audio Workstation), que son entornos de trabajo para productores musicales en los que grabar, editar, aplicar efectos y dar formato a las creaciones musicales. Para establecer una equivalencia, son herramientas que suponen para un músico lo que Photoshop para un fotógrafo, o el propio Visual Studio para un programador. Hay un buen mercado de ellos, aunque lo importante es saber que la gran mayoría de ellos (Exceptuando, principalmente, ProTools) son compatibles con plugins del mismo formato.

HarmonyPal es un **plugin musical**, que no es diferente a un plugin de cualquier otro tipo de programa (1). Se puede instalar en casi cualquier DAW tan solo arrastrando un archivo, y se puede arrancar desde dentro del DAW, o incluso como aplicación independiente, aunque si se ejecuta de esta manera (llamada standalone) no tendrá todas sus funcionalidades.

Los plugins musicales, en base a su funcionalidad, se dividen en tres tipos, aunque siempre puede haber hibridaciones entre ellos:

-Instrumentos virtuales: estos son los que producen audio en base a muestras pregrabadas y determinados inputs, ya sean del controlador del usuario o del DAW. Son, por ejemplo, los que permiten que, tocando un piano digital, podamos escuchar o grabar sonido de cuerdas, trompetas o un coro entero.

-Procesadores de audio: no producen sonidos, sino que reciben sonidos para manipularlos de determinadas maneras. Por ejemplo, los plugins que introducen efectos al audio de entrada para replicarlo de otra manera: ecualización, reverberación, corrección de afinación…

-Procesadores de MIDI: no tratan con audio, sino con información musical en forma de mensajes del estándar MIDI. Reciben mensajes y los procesan o generan nuevos en base a la entrada. HarmonyPal encajaría en esta categoría.

1.2.1 El estándar MIDI

MIDI son las siglas de **Musical Instrument Digital Interface**. Es un estándar creado para la representación digital de información musical y su transmisión adecuada a través de máquinas diferentes.

Su funcionamiento se sustenta en **mensajes MIDI** de distintos tipos. Se pueden intercambiar entre dispositivos en tiempo real como, por ejemplo, lo que sucedería cuando tocas un piano, y este va enviando las notas que tocas al computador, o se pueden organizar en una línea de tiempo y ser lanzados de manera secuencial, como

pasaría si preprogramas los sonidos de una canción, para que cada nota suene en su preciso momento y en un instrumento concreto, y reproduces la secuencia. Los hay de múltiples tipos, aunque los más habituales y que a nosotros nos interesa son los de **Note On** y **Note Off**, que se corresponden a la señal de tocar una nota o apagarla, respectivamente.

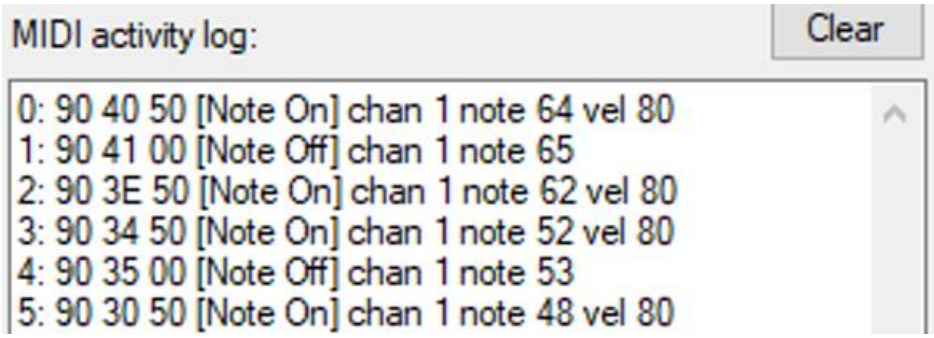

Figura 1.1: Ejemplo de log de mensajes MIDI en el DAW Reaper.

Podemos crear, recibir y manipular mensajes MIDI como con cualquier otro tipo de información digitalizada, de manera que un programa puede, sin ningún problema, recibir la entrada de un piano digital (o, mejor dicho, **controlador MIDI**), convertirla en un formato más procesable, interpretarla y generar nuevos mensajes a partir de ella, para mostrarlos por pantalla en el formato deseado. Así es como funciona HarmonyPal.

CHAPTER 1

Introduction

1.1 Motives

This project is born as a combination of my two academic fields of study. On the one hand, of course, computer science, as in fact this is the last work before I graduate on Software Engineering. On the other, music, with my specialties in piano and composition, that have been the driving forces behind this whole project.

In the beginning, this work was shared with my companion Javier Navalón. In the early stages of the Project, we envisioned a much bigger and ambitious application, with a not so very sharp focus on its features. During this time, he worked more with the audio aspect of the program, and I focused on the mathematical representation of the musical information. After our split, each decided to pursue its own vision. His work culminated in RythmPal, a tool to generate musical accompaniments while playing, and has been published this same year. My work has reached its own conclusion being a tool that creates chord suggestions from the chords the user plays, so that they can play musically rich sequences.

When deciding how to face my project, I had my experience working as a piano teacher to understand how my students might need help understanding certain aspects of the basics of harmony theory before they can get playing with ease. When we talk about the piano, we need to understand the different roles of the two hands, with the right being usually the one that plays melodies and the left playing chords. It's the left one that gives the most headaches to most students, and not only because of their own dominant hand being overwhelmingly the other one, but because of this role of "chord player" that it has in modern western music in particular. It not playing memorable and catchy melodies, has a sometimes-overlooked responsibility that I try to address in my classes, but that always needs a little more attention than I can give with limited time. The idea of HarmonyPal is precisely to aid me and my students with these issues when I´m not there, in their own practicing time at home. Nevertheless, I would not underestimate the potential of the application to aid more mature musicians with their improvising or composing sessions, particularly those with general knowledge of music, but not so much of harmony theory.

At the end of the day, HarmonyPal is a tool that analyzes the chords the user plays, generates candidate chords that musically follow, and shows the way to play them with the correct keys, so that they can play them with good technique, hear them and

discover new musical possibilities, and not get used to the same four chords over and over.

1.2 Music computing basics

To understand what the working elements of HarmonyPal are, we must first address a few concepts. Firstly, the **Digital Audio Workstations**, or DAW, are work environments that can be installed in most PCs and with which one can record, edit, and produce musical works with professional tools. They are to a musician what Photoshop is to a photographer, or Visual Studio to a programmer. There are plenty of them to choose from, free and paid for, but the vast majority of them (most notably excluding ProTools) are compatible with the same formats of plugin.

HarmonyPal is a **music plugin**, which is no different from any other plugin for any other program (1). It can be installed in any DAW just by dragging one file, and it can be run from the DAW or even independently as a standalone application.

Music plugins are sorted, according to the data they handle and the tasks they accomplish, in three basic groups which can be hybridized:

-Virtual instruments: these are the ones that actually produce audio, taken from pre-recorded samples or sound waves. They need some input from the user or another plugin within the DAW, and are the reason why we can hear strings, brass or even drums by playing a digital keyboard.

-Audio processors: these do not produce audio, but rather make alterations to existing audio signals. Such are the reverbs, equalizers, vocoders and many more.

-MIDI processors: they deal with musical information rather than audio. This information takes the shape of MIDI standard messages. These messages are created and processed so that they can be sent to other plugins or programs. HarmonyPal falls in this category.

1.2.1 MIDI Standard

MIDI, or **Musical Instrument Digital Interface**, is an informatic standard established to accurately represent musical information and share it across different kinds of computers.

It works with **MIDI messages**, which can be of many different kinds. They can be transmitted across different machines or programs in real time, and so be the basis for the digital musical practice, like playing a digital keyboard and be able to hear the audio you generate from it. They can also be pre-programmed in a sequence and played at a

certain speed, thus offering the possibility of playing a music piece from carefully placed MIDI signals.

MIDI messages can be of 9 kinds, but the most common ones are **Note On** and **Note Off** messages. These represent the act of playing and releasing a note, respectively, be it on a keyboard or on any other music instrument you can imagine.

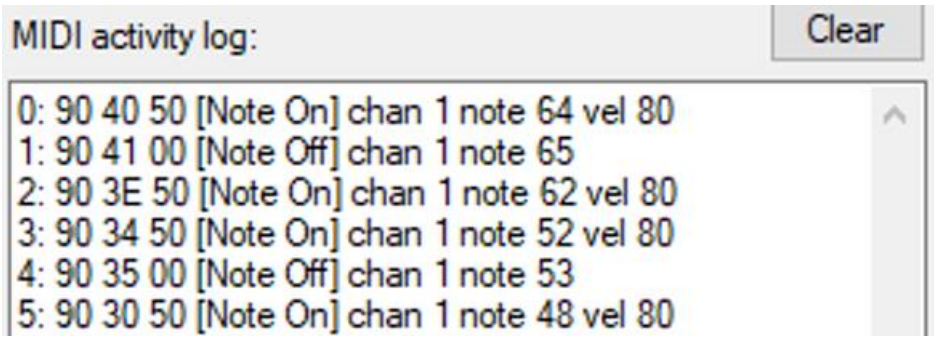

Figure 1.1: MIDI message log in the DAW REAPER.

We can create, recieve and alter MIDI messages just like we can with any other type of digital information, so a program can easily get the input from a MIDI controller, make it into a more manageable format, process it and generate new messages with it, to show the final user whatever may be the results. Such is the way HarmonyPal Works.

CAPÍTULO 2

Contenido teórico

Para comprender los aspectos más técnicos del funcionamiento del programa, debemos establecer una base de teoría musical que explique los conceptos involucrados (2).

2.1 La armonía

HarmonyPal, como su propio nombre indica, se enfoca en la **armonía**, el llamado aspecto "vertical" de la música, y complementario al horizontal, que es la melodía. La melodía es, esencialmente, el desarrollo de los sonidos musicales en el tiempo, y trata con las diferentes notas de manera secuencial, una después de la otra. La armonía, por el contrario, se encarga de las relaciones entre notas cuando estas suenan **de manera simultánea**, conformando **acordes** (de tres a más notas sonando al mismo tiempo de manera estructurada). Cuando se puntean las cuerdas de la guitarra una tras otra, se está haciendo una melodía y, cuando se rasguean las seis a la vez, suena un acorde. A pesar de estas diferencias, estos dos aspectos de la música están estrechamente ligados y los conjuntos de reglas que los gobiernan tienen multitud de puntos de encuentro. Tanto es así que existen sistemas de análisis que comprenden las dos dimensiones, vertical y horizontal, como un todo indistinto, como puede ser el sistema acorde-escala. Una melodía, aunque suene en vacío, siempre implica una armonía que encajaría con ella, y aunque analicemos los acordes como un bloque aislado del tiempo, su enlace con el que suene a continuación depende también de aspectos melódicos.

En cualquier caso, para ambos aspectos, hay combinaciones de notas que suenan mejor que otras, y del mismo modo que la teoría del color establece bajo qué criterios unos colores funcionan mejor juntos (siempre asumiendo una cierta visión culturalmente determinada), la armonía hace lo propio con las notas musicales. Evidentemente el estudio de estas combinaciones da para más de una vida, pero para entender nuestro programa será suficiente con revisar los siguientes aspectos.

2.1.1 Las notas musicales

El sistema musical occidental separa la continuidad del espectro auditivo en doce sonidos fundamentales, representados en las teclas de un piano: las notas. La realidad

es que un piano tiene muchas más teclas, pero son todas repeticiones, más agudas o graves, de estos doce nombres: Do, Do# (leído “Do sostenido”), Re, Re#, Mi, Fa, Fa#, Sol, Sol#, La, La# y Si.

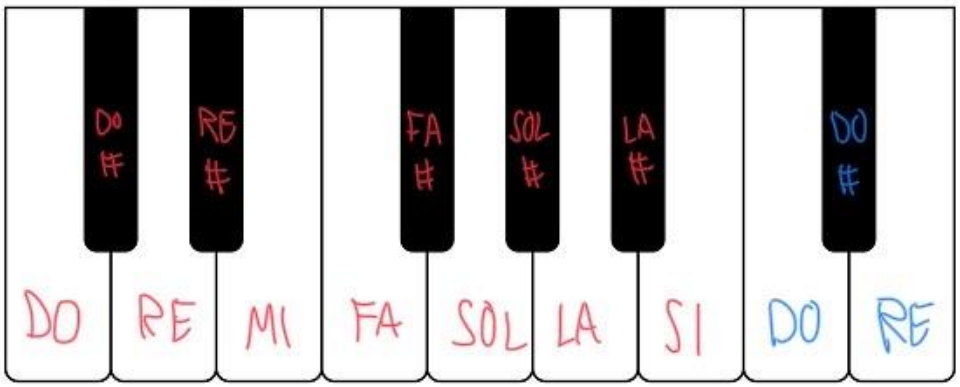

Figura 2.1: Las notas musicales en el piano.

La distancia de una a otra se mide en **semitonos**, ascendentes si van hacia la derecha y descendentes en caso contrario, y se calculan en el orden anteriormente descrito. La distancia de una nota a otra es un **intervalo**. En el código de una aplicación todo habrá de interpretarse numéricamente, pero aquí hablaremos de las notas con nombres y de los intervalos en valores numéricos.

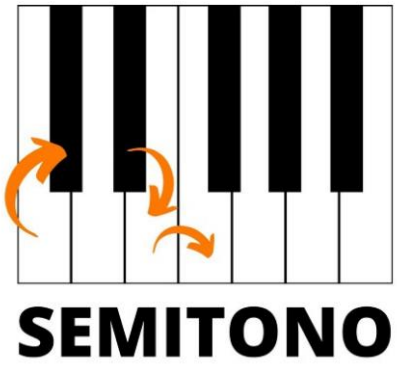

Figura 2.2: Los semitonos en el piano.

Por ejemplo, de la nota Do a la nota Mi hay un intervalo de 4 semitonos. Como las notas se repetirán siempre en el mismo orden, también podemos hablar del intervalo **inverso**, desde el Mi hasta el siguiente Do, cuyo valor es de 8 semitonos. Lógicamente, el valor del intervalo inverso es el total de notas, 12, menos el intervalo sin invertir.

2.1.2 El acorde

Ahora debemos tratar la unidad mínima de la armonía, el **acorde**. Un acorde es un conjunto de notas de diferente nombre (generalmente de tres o más) que suenan simultáneamente. Los acordes se definen por las notas que los conforman, de entre las 12 disponibles. Por ejemplo, un acorde puede estar conformado por las notas Do, Mi y Sol.

Figura 2.3: El acorde de Do Mayor en el piano.

Este es el acorde de **Do Mayor**. Do, porque es su **nota fundamental** (la primera), y Mayor, porque es el nombre que se le da por convenio a esta distribución de intervalos, la llamada **especie** del acorde. Es equivalente, por ejemplo, al acorde Sol, Si, Re, porque aunque no tengan las mismas notas, si comparten las distancias entre ellas, su **interválica**. De Do a Mi, igual que de Sol a Si, hay 4 semitonos, y de Mi a Sol, como de Si a Re, hay 3. Es conveniente tratar con las especies abstrayéndonos de la nota fundamental, es decir, que trataremos con acordes Mayores en general, no con Do Mayor, Mi Mayor y Si Mayor por separado, por ejemplo.

Especies de acordes existen tantas como combinaciones de intervalos se puedan realizar, aunque las más importantes históricamente, y los más sencillos de entender para un estudiante de piano, son los acordes Mayor (Que ya hemos visto) y Menor (Equivalente a Do, Re#, Sol).

Hay notaciones numéricas que tratan con las especies de acordes representando exclusivamente sus intervalos de esa manera: da igual si es Do, Re o Sol, que si se trata de un acorde Mayor, se lo representará por los intervalos desde su fundamental hasta sus notas (es decir, 4 y 7). En mi caso he encontrado más conveniente una pequeña variación de esta notación, que comprende no la distancia en semitonos desde la fundamental al resto, sino de una nota a la siguiente, incluyendo el intervalo de vuelta de la última nota a la primera. Así, el acorde Mayor se representaría como 4, 3, 5, y el Menor como 3, 4, 5. El segundo intervalo, que en la anterior notación hubiera sido más grande, ahora se deduce por la suma de los dos primeros. Aparece también el intervalo

final, que no es más que la cantidad necesaria para dar la vuelta y llegar de nuevo a la primera nota, sumando 12.

La decisión de adoptar esta notación en el programa se debe a la **facilidad de inversión** del acorde. Del mismo modo que el intervalo Do –> Mi = 4 tiene un inverso, Mi –> Do = 8, los acordes tienen también sus inversiones, de igual funcionamiento y que detallaremos más adelante. Si el acorde tuviera las mismas notas, pero en distinto orden (en vez de Do Mi Sol, Mi Sol Do), la interválica medida desde la fundamental resulta compleja: la nueva representación partiendo desde el Mi sería 3, 8, en lugar de 4, 7, y nos hallamos ante dos representaciones irreconocibles de, esencialmente, las mismas notas. Con nuestra representación nota a nota, vemos que el acorde que antes era 4, 3, 5 se convierte ahora en 3, 5, 4, y que hacer algo tan sencillo como la rotación de la interválica un valor a la derecha nos aporta la misma información, pero con una representación numéricamente muy similar. Las ventajas de este segundo formato para un programador son evidentes si nuestro objetivo pasa por cambiar el orden de las notas indistintamente y con frecuencia.

2.1.3 Las inversiones

Este cambio de orden de las mismas notas de un acorde es la inversión. Un mismo acorde de *n* notas tiene hasta *n* estados posibles: el llamado **fundamental**, que tiene como primera nota la fundamental del acorde, y *n-1* **inversiones** nombradas por ordinales. Sol, Do, Mi, por ejemplo, es la **segunda inversión** del proverbial acorde de Do Mayor. Nótese que, aunque en las inversiones la primera nota pasa a ser otra distinta, la fundamental del acorde seguirá siendo Do, y viene dada por la que sería la primera nota con su interválica original.

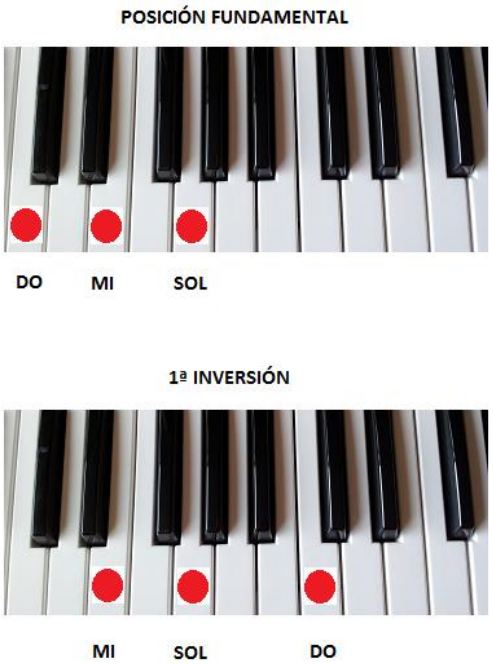

Figura 2.4: Las inversiones de acordes.

Las inversiones son importantes para nosotros en tanto que lo son para la ejecución pianística: el programa ha de interpretar cada input de tres notas como un determinado tipo de acorde, y no tenemos la garantía de que el usuario las introducirá siempre en un mismo orden, así que es necesario operar con los conjuntos de notas tal y como lleguen, y partir de la base de que puede tratarse de un acorde en inversión. Además, a la hora de mostrar las opciones de acordes para tocar a continuación, puede ser conveniente para la técnica musical que estos acordes aparezcan también invertidos, para que, por ejemplo, la mano del pianista no tenga que moverse más de lo necesario a lo largo del teclado entre un acorde y el siguiente.

2.1.4 La jerarquía

Del mismo modo que un retrato pone en el foco a la persona representada, y relega a papeles secundarios el fondo, los objetos o el paisaje, la armonía de un determinado fragmento musical se enmarca en una **tonalidad** que determina a qué nota se le otorga un mayor peso, cuáles están supeditadas a ella y de qué manera. El concepto mismo de tonalidad es complejo y altamente cambiante según el estilo y época, de manera que veremos una versión simplificada, basada en cómo la aplicación la entiende.

Esencialmente, una tonalidad es el conjunto de notas dentro de las doce totales que *podemos* usar con un mayor grado de coherencia y, dentro de ellas, qué emoción evocaran en el oyente por su relación con las demás. Por lo que a nosotros respecta, la tonalidad está formada por **siete** notas o **grados**. Si, por ejemplo, una canción tiene por

tónica (nota de inicio y fin, y de mayor cantidad de veces usada) la nota Do, diremos que esa nota es el **primer grado**, escrito en números romanos. Cuáles son los demás grados es un asunto muy estudiado y concreto, pero lo simplificaremos en el siguiente esquema que ayuda a visualizar cómo se procesa en el algoritmo:

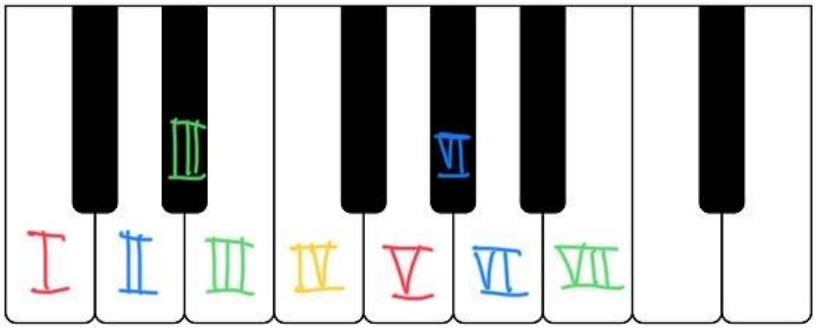

Figura 2.5: Los grados según HarmonyPal.

Este es un ejemplo a partir de la tónica Do. Lo importante, como con los acordes, es la interválica y no las notas concretas. Por ejemplo, para la tonalidad de Re, el segundo grado sería Mi, y el tercero correspondería a las notas Fa y Fa#. Hablar de **grados**, en lugar de notas concretas, nos permite nuevamente ponernos en un nivel de abstracción conveniente para la programación.

2.1.5 Funciones tonales

Cada uno de estos grados tiene unas características y una relación sonora con el resto de la tonalidad que es necesario conocer en su nivel más elemental. Hugo Riemann y Diether de la Motte hablan de la **armonía funcional** en un contexto contemporáneo y comprenden todos los grados de la escala en tres funciones:

-Tónica: los grados primero, tercero y sexto. Suenan estables, reposados, y es de donde la música parte y a donde llega. La misma nota tónica es la más importante de la tonalidad por su función de anclaje.

-Subdominante: los grados cuarto y segundo. Una función de transición, que sugiere progresión y movimiento, pero sin llegar a generar choques ni inestabilidades.

-Dominante: Los grados quinto y séptimo. La función de tensión máxima, necesidad de resolución hacia la tónica, expectación y exaltación. El clímax de las frases musicales antes del cierre en tónica. Contiene la nota sensible, el onceno de los semitonos, que es la de mayor energía resolutiva y segunda más importante de la tonalidad.

El funcionamiento básico de la armonía occidental necesita de la alternancia de estas funciones, generalmente en este orden y cíclicamente, para ser reconocible y satisfactorio.

2.1.6 Los enlaces en el piano

A la hora de trasladar todo esto al teclado de un piano, debemos entender que un joven músico no trabajará día a día con todos estos valores numéricos y conceptos teóricos, sino que ha de desenvolverse con la vista del teclado y su memoria muscular. Cuando se sienta a improvisar, elige arbitrariamente una tonalidad y lanza su acorde de tónica, necesita dos cosas: saber qué acorde tocará a continuación, y tener claro dónde están las teclas correctas.

Para elegir un acorde no basta con seguir este principio básico de Tónica – Subdominante – Dominante y repetirlo sin más meditación, pues si bien se obtendrá un resultado correcto, nunca será demasiado interesante ni servirá al intérprete para descubrir posibilidades técnicas o sonoras. Debemos tener en cuenta todos los aspectos brevemente explicados en este resumen teórico y otros muchos más para poder garantizar unos acordes sucesivos interesantes al oído como a la mano del pianista, y de eso se encargará nuestro algoritmo.

Y, sobre todo, una vez elegido un acorde, ha de quedar claro cómo enlazarlo de manera cómoda y natural, sin realizar desplazamientos innecesarios y manteniendo en su lugar las notas comunes en caso de haberlas. Aquí es donde es necesaria la representación gráfica del acorde en cuestión sobre un teclado en el que quede meridianamente claro cómo han de moverse cada uno de los dedos del intérprete. Los acordes para enlazar han de tomar en cuenta sus potenciales inversiones para presentarse de la manera más pianística posible.

Esencialmente la regla es: cuando haya notas comunes entre los dos acordes, éstas se mantienen en su posición, y para las que sean diferentes, han de moverse la menor cantidad de semitonos posible. El acorde resultante no debe superar los doce semitonos entre su nota más grave y la más aguda, manteniendo la que llamamos **posición cerrada**.

2.2 MIDI

El funcionamiento de este y cualquier plugin o aplicación de informática musical gira en torno al estándar **MIDI**. Creado en los años 80, y motivado por las fuertes necesidades de los incipientes videojuegos domésticos e instrumentos electrónicos (como los sintetizadores), MIDI establece cómo se transmiten los mensajes musicalmente significativos entre programas o dispositivos, y es el que permite que hoy en día un teclado digital pueda conectarse a cualquier computador con un puerto USB cualquiera,

ser detectado y lanzar sonidos pregrabados, modificar sus parámetros y, en esencia, producir música digitalmente.

A nivel básico, MIDI son sus **mensajes**. Cada mensaje MIDI consta de, como máximo, tres bytes, siendo el primero de ellos uno de control, llamado byte de estado. Este byte se divide a su vez en tres fragmentos: el primero, un solo bit siempre por valor 1. El segundo indica el tipo de mensaje MIDI que se va a describir a continuación, de entre los ocho que aceptan tanto el sistema como los tres bits que se le destinan. Por ejemplo, un mensaje puede ser de Note On, si especifica que se está tocando una nota, o de Pitch Bend, si se está haciendo un bending (técnica de modificación del tono típica en guitarra y sintetizadores) sobre una nota concreta. El tercer fragmento indica el canal MIDI, de entre los 16 que dejan disponibles los últimos cuatro bits, al que está destinado el mensaje. MIDI soporta distintos canales para que, por ejemplo, puedan coexistir múltiples instrumentos virtuales con sus diferentes notas y modificaciones en el mismo fragmento de música. Los demás bytes, llamados de estado, determinan valores relevantes para cada tipo de mensaje. Para el mensaje Note On, por ejemplo, el primer byte especifica qué nota se está tocando, y el segundo su "velocity" o intensidad de pulsación.

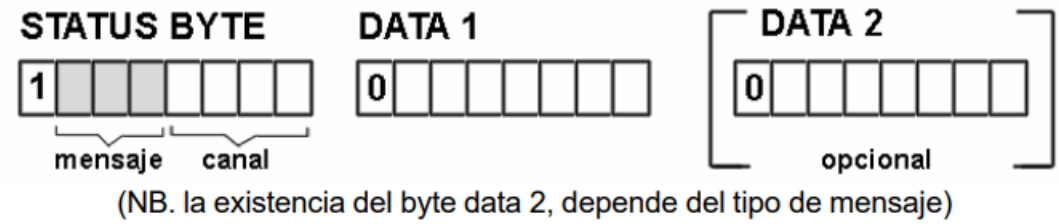

Figura 2.6: Estructura de un mensaje MIDI.

Con programas capaces de generar mensajes MIDI a partir de una entrada (como las pulsaciones sobre un controlador MIDI) y otros capaces de interpretar estos mensajes y generar señales sonoras (los llamados instrumentos virtuales), tenemos los elementos necesarios para crear música con equipo exclusivamente digital. Inmediatamente, el límite de las posibilidades salta tan lejos que, ni se ha alcanzado más de cuarenta años después de la creación del estándar, ni parece que vaya a alcanzarse en el futuro cercano.

2.3 JUCE

Para poder sacar provecho de las capacidades de los mensajes MIDI, se puede simplemente programar en un IDE de preferencia y generar los mensajes MIDI de manera artesanal, pero es conveniente disponer de librerías y funcionalidades específicas para la informática musical. Para este propósito existe **JUCE**, un framework

que utiliza el lenguaje C++ y librerías propias para facilitar el desarrollo de software musical, incluyendo gestión de audio y MIDI, e independizar el desarrollo de código de la compilación específica del entorno.

Las aplicaciones desarrolladas con JUCE se dividen esencialmente en aplicaciones y plugins. Cada una tiene su estructura de clases propia, pero esencialmente se diferencian en su propósito: aunque un plugin suele poder ejecutarse de manera independiente (standalone), está pensado para funcionar en conjunto con otras herramientas, instrumentos virtuales o efectos, es decir, dentro de un DAW. En cambio, las aplicaciones están pensadas desde un inicio para funcionar completamente por sí mismas, como podría suceder, por ejemplo, con una App para dispositivos móviles que ofrezca un teclado con sonidos. En el caso de HarmonyPal, tratamos con uno de esos plugins que puede funcionar de manera independiente si se está dispuesto a renunciar a funcionalidades como el propio audio, que no es estrictamente necesario.

2.3.1 Projucer

Projucer es el nombre que recibe la aplicación destinada a la implementación de aplicaciones con las librerías de JUCE. Se puede descargar y usar en minutos, y su funcionamiento es tan sencillo como esenciales son sus implicaciones (3).

Al desarrollar un plugin con JUCE han de tenerse muy en cuenta las distribuciones de las clases y determinados ajustes del proyecto, como sus funciones (si va a recibir o transmitir audio y/o MIDI) o las clasificaciones de la aplicación a la hora de buscarla en una lista de plugins dentro de un DAW, por ejemplo. Con Projucer podemos gestionar todos estos aspectos del proyecto y asegurarnos de que podemos compilar una solución en formato VST3 (aceptado por la mayor parte de DAWs y Sistemas Operativos), o el deseado para nuestro plugin, con apenas unos clicks al crear el proyecto.

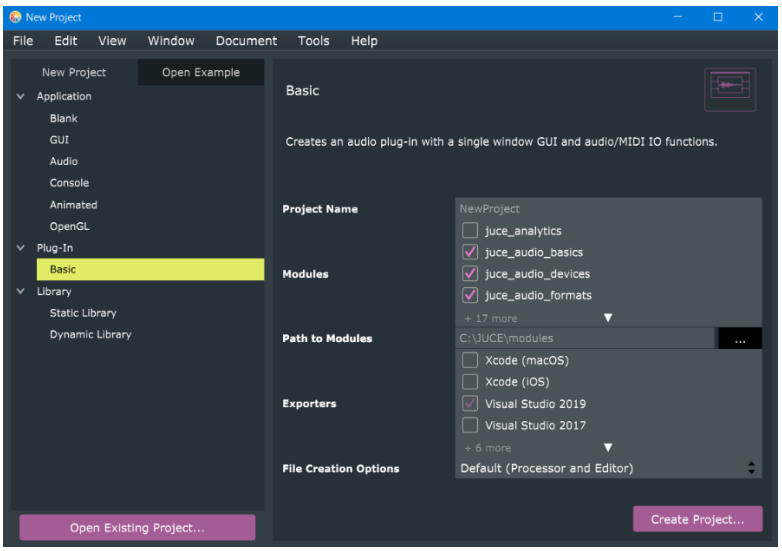

Figura 2.7: Pantalla de creación de un nuevo proyecto en Projucer.

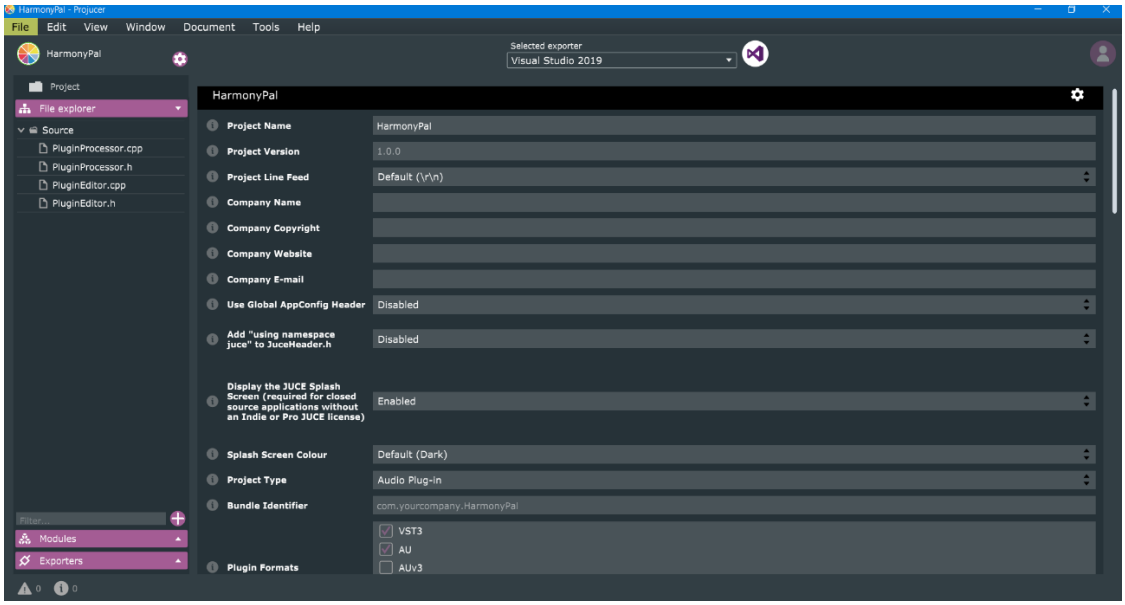

Figura 2.8: Pantalla de ajustes de un proyecto existente en Projucer.

Los programas creados con Projucer tienen la distribución de archivos inicial ya establecida, y contienen un archivo especial de extensión *jucer* que podemos abrir con la aplicación Projucer para cambiar ajustes del proyecto después de su creación.

Es gracias a estas herramientas, y sobre todo a su gran repertorio de documentación, tutoriales y ayudas (4), que ha sido posible el desarrollo del programa que nos ocupa.

CAPÍTULO 3

La aplicación

3.1 Prestaciones

HarmonyPal es una herramienta para los pianistas y teclistas que detecta los acordes introducidos y genera sugerencias de acordes correctamente dispuestos para poder continuar un acompañamiento rico y funcional. Este apartado es un recorrido detallado del funcionamiento de estas tres prestaciones.

Para utilizar HarmonyPal como plugin dentro de un DAW, hay que disponer de uno y configurar un nuevo proyecto de cierta manera. Se ilustra aquí con un ejemplo en REAPER, un DAW de descarga gratuita y fácil utilización.

En primer lugar debemos ubicar los plugins necesarios en las rutas que el DAW pueda escanear. En Options -> Preferences -> VST, tenemos una lista editable de estas rutas. Bastará con copiar allí el .vst de HarmonyPal y de un instrumento virtual.

Ya en el proyecto DAW, creamos una pista con el instrumento virtual, que proporcionará el sonido a nuestro plugin. Puede ser un piano o cualquier otro instrumento, siempre que acepte mensajes MIDI externos. Lo añadimos a la pista con la pestaña *FX*.

Figura 3.1: La pista del instrumento virtual, con la pestaña FX marcada en verde.

Después, añadimos pistas con MIDI routing hacia esta. La pista que creemos enviará su MIDI saliente a la pista del instrumento, y en esta nueva pista colocaremos el plugin HarmonyPal. Puede hacerse, por ejemplo, en *FX -> Options -> Build 16 channels of MIDI routing to this path*, para después dejar sólo una de las pistas creadas.

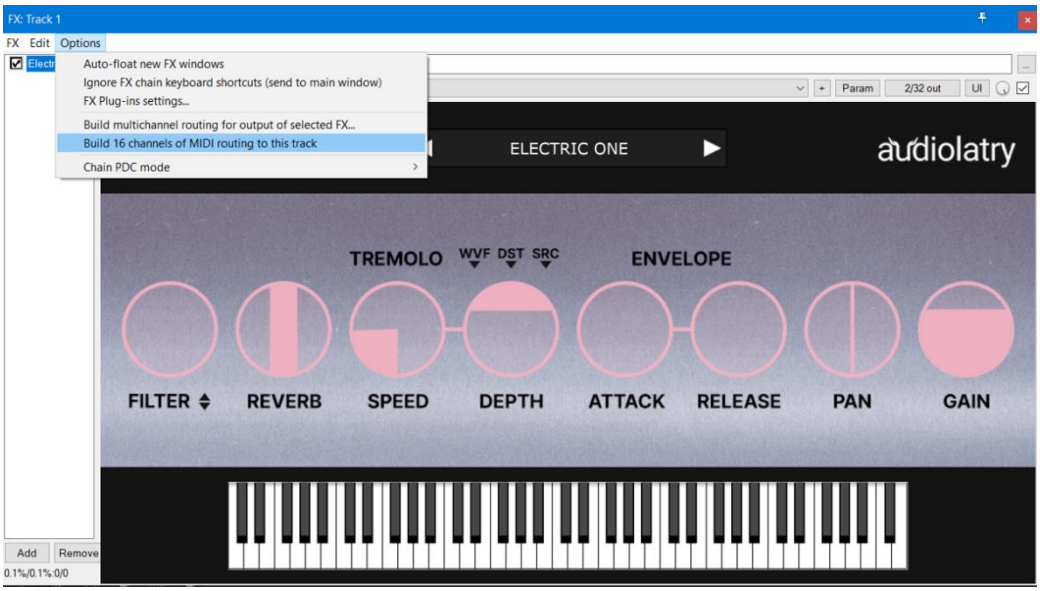

Figura 3.2: Ejemplo del proceso descrito.

Por último, añadimos el plugin HarmonyPal a la pista enrutada mediante la pestaña *FX* como hemos hecho con el instrumento virtual y añadimos nuestro controlador MIDI como entrada de la pista. Ya está listo para funcionar, pulsamos en *FX* para mostrar el plugin y podemos empezar a tocar.

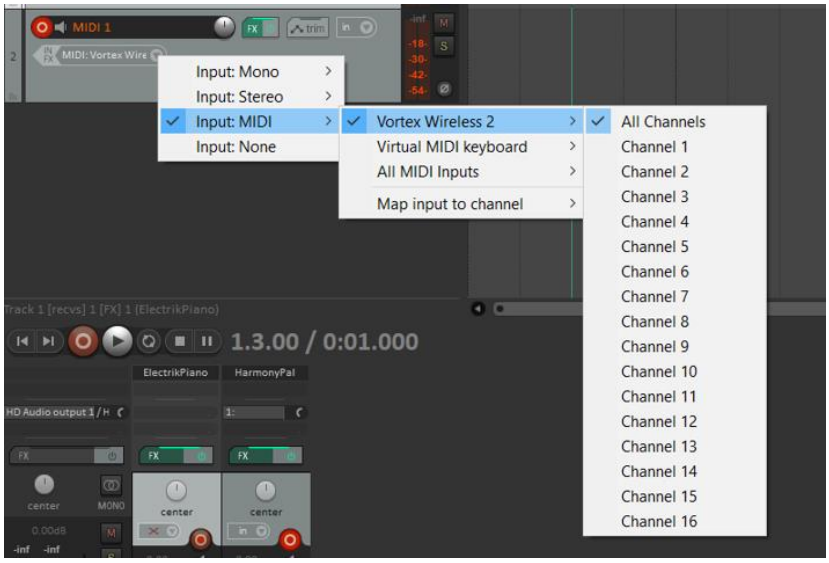

Figura 3.3: Cómo asignar el controlador MIDI como entrada de la pista de HarmonyPal.

Para el usuario, la experiencia consistirá en conectar su controlador MIDI al ordenador, asegurarse de que su entrada se está recibiendo en la pestaña de ajustes del plugin (modo standalone) o en la entrada de la pista (modo DAW), y comenzar a pulsar notas. Si toca una nota a partir del Do 6 en el estándar MIDI (es decir, de la parte más a la derecha en un piano normal), cambiará la tonalidad en la que está trabajando el

algoritmo. Al tocar el resto de las notas del teclado, el programa escuchará hasta que haya tres teclas pulsadas de manera simultánea. Cuando esto suceda, el programa lo procesará y generará las sugerencias de acordes. Tanto estas como los nombres de todos los acordes serán mostrados por pantalla.

Figura 3.4: El plugin HarmonyPal.

3.1.1 Analizador de acordes

Cuando el usuario utiliza su teclado MIDI externo y toca un acorde, el programa se dispone a recoger esta entrada, reducirla a información numérica básica y extraer la interválica. Los valores resultantes de restar los números de nota de cada una tocada se comparan con una base de datos de acordes, cargada tras el arranque del programa, disponible en un archivo de texto. La comparativa se hará con cada acorde de la base de datos (que comprende todas las combinaciones de intervalos matemáticamente posibles) y sus inversiones, de manera que está garantizada la coincidencia, aunque no la identificación de la especie del acorde. En un pequeño cuadro de texto se muestra la fundamental, especie e inversión del acorde introducido, con la única salvedad de que la especie del acorde no figure en la base de datos, en cuyo caso se marcará como "Desconocido". Esta información, que ya por sí misma puede ser de cierta utilidad para el teclista neófito, se utilizará como dato de entrada del siguiente componente del programa.

Figura 3.5: El resultado del analizador de acordes, en un campo de texto junto al teclado.

3.1.2 El algoritmo

Con la base teórica establecida en el capítulo anterior, establecemos una serie de directrices para elegir, de entre todos los acordes disponibles en la base de datos, y sus doce posibles tónicas, tres acordes que pudieran ser tocados a continuación con un buen resultado. La idea para su funcionamiento está basado en mi implementación del algoritmo A* durante mis clases de Ingeniería del conocimiento, en tanto que el algoritmo considera todos los acordes posibles y maximiza la puntuación de cada candidato (otorgará a cada acorde una puntuación numérica en base a determinadas características, y se quedará con las más altas).

Es importante tener en cuenta que, si bien los criterios que sigue el algoritmo son razonables y están apoyados sobre un cuerpo teórico abundante, en ningún caso son universales en la práctica musical, ni siquiera en la occidental, y que las valoraciones en puntos que éste realiza han de verse como, hasta cierto punto, subjetivas y sujetas a una mirada más cultural que objetiva. El valor del algoritmo está más en los parámetros de acorde que es capaz de considerar, y menos en el cómo los considera para elegir un candidato.

3.1.3 Enlace de acordes

Una vez obtenidos los tres acordes para mostrar, se realiza con el acorde anterior la operación de enlace pianístico para elegir la disposición de notas o inversión que se le va a mostrar al usuario. La operación es sencilla y su proceso se basa en la práctica pianística y el mantra que personalmente repito sin cansarme a mis alumnos: si de un acorde a otro hay notas comunes, mantenerlas en la misma tecla (no cambiar un Do por otro más a la derecha del teclado), y entre las que no lo sean, realizar los desplazamientos más reducidos posibles. Esto ayuda a la fluidez sonora y a la técnica del estudiante, aunque ignora ciertos aspectos musicales como la conducción de voces o la existencia de la sexta y cuarta cadencial y acordes similares. Considero sin embargo que la forma ideal de mostrar los acordes es con estos criterios de inversión, y que un pianista más experimentado sabrá valerse de la herramienta eligiendo por sí mismo una inversión musicalmente más efectista cuando sea posible.

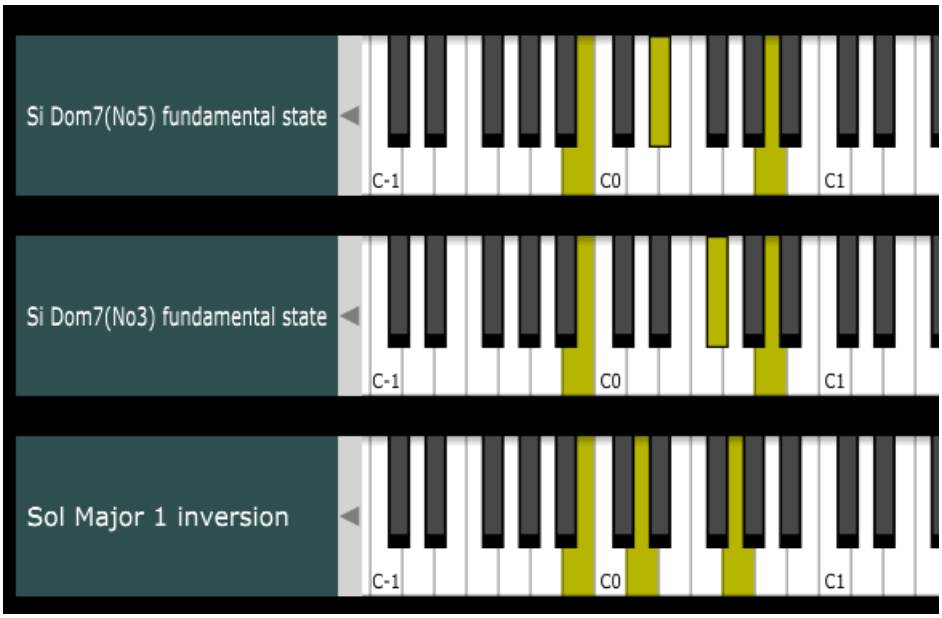

Figura 3.6: Los acordes sugeridos, con su información completa e inversión.

3.1.4 Reproducción de acordes

Junto a los tres teclados de los acordes sugeridos hay un botón con la etiqueta “PLAY” que permite la escucha de los acordes sugeridos por la aplicación. Una vez haya acordes listos para reproducir, podemos pulsar sobre los botones con el ratón, y se enviará el MIDI de los teclados como salida. El instrumento virtual lo reproducirá como hace con la entrada de nuestro controlador MIDI.

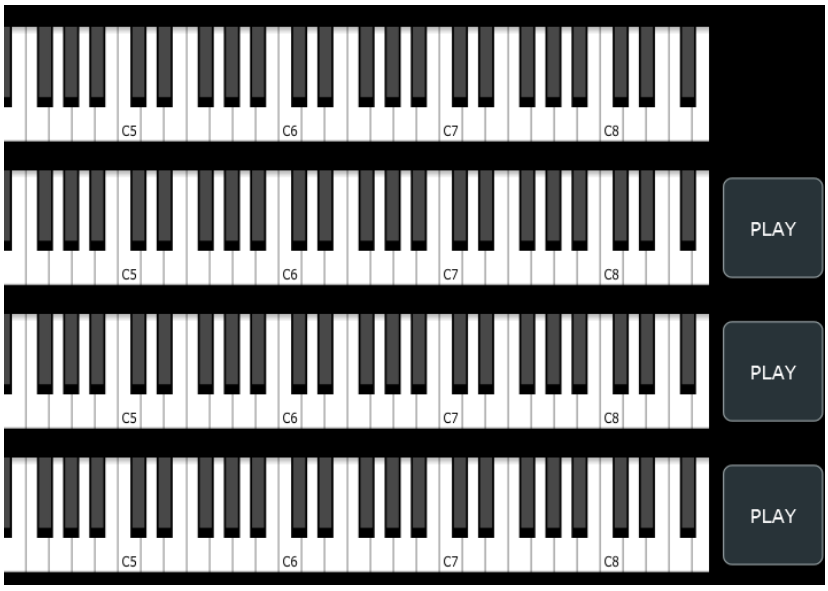

Figura 3.7: Los botones PLAY.

3.1.5 La tonalidad

Con la parte final del teclado MIDI, el usuario puede establecer la nota principal de la tonalidad en la que quiere trabajar tocando la tecla correspondiente del que sería el último conjunto de 12 notas de un piano normal. Si el teclado es más pequeño, se puede aumentar el valor de las teclas disponibles con botones propios del controlador, generalmente señalados como *Octave up / down*.

Al ser una parte tan extrema, se considera menos necesaria, ya que la mayoría de la interpretación en el piano se efectúa en las zonas centrales, y generalmente los acordes, considerados acompañamiento de la melodía, se tocan por la parte izquierda. Si se cambia la tonalidad con éxito, el mensaje *Tonic* de la parte superior izquierda cambiará para mostrar la tónica nueva.

CAPÍTULO 4

Implementación

4.1 El directorio

El programa tiene una estructura muy sencilla. Por un lado, librerías y el archivo .jucer, generados por el framework JUCE y que no he modificado desde su creación. Por otro, los archivos de código propiamente dicho. Incluyen los llamados Processor y Editor, dos nombres estándar para los archivos que JUCE destina a procesamiento de sonido / MIDI e interfaz de usuario, respectivamente, y los módulos auxiliares, ChordData y ChordProcessor.

4.2 PluginEditor

Los archivos PluginEditor.h y PluginEditor.cpp son los destinados a crear la ventana del plugin, dibujar los teclados y mostrar al usuario los resultados de las operaciones realizadas por el algoritmo. En esencia, se limitan a inicializar los MidiKeyboardComponent (el tipo encargado de gestionar la apariencia de un teclado interactivo en pantalla) y los MidiKeyboardState (encargado de operar con las notas y activar las teclas cuando le lleguen las instrucciones oportunas). El constructor asocia cada Component a su State correspondiente, y la función resized, convocada junto a la creación de la ventana del plugin, delimita el espacio adecuado para cada uno de ellos y los sitúa unos sobre otros.

```
addAndMakeVisible(mainKeyboard);
addAndMakeVisible(keyboard1);
addAndMakeVisible(keyboard2);
addAndMakeVisible(keyboard3);

setSize (1240, 600);
```

Figura 4.1: Órdenes para dibujar teclados en PluginEditor.cpp

4.3 PluginProcessor

Como en el caso anterior, se dividen en .h y .cpp, aunque el primero tan solo contiene los encabezados de las funciones, variables globales y la declaración de los MidiKeyboardState que utilizará el Editor.

PluginProcessor.cpp contiene un constructor en el que se inicializan la semilla de números aleatorios y algunas variables, además de llamarse a la función de carga de información de acordes. La otra función principal es processBlock, que recibe activaciones constantes, multitud de veces por segundo. En ella comprobamos el número de teclas que el usuario está tocando en su piano, contando con duplicaciones, hasta que se forme un acorde con tres notas nominalmente diferentes. Entonces se llama a la función que procesa la información del teclado, en el módulo ChordProcessor. Por último, se borran las notas pulsadas en los teclados de display (si ya se han tocado acordes antes) y se actualizan para mostrar las nuevas.

```
for (const auto messages : midiMessages) {
    const auto message = messages.getMessage();
    if (message.isNoteOn()) {                       // SI EL USUARIO PULSA UNA NOTA,
        if (message.getNoteNumber() >= 96) {        // SI ES DE LAS MÁS AGUDAS DEL TECLADO,
            tonic = message.getNoteNumber() % 12;   // ES PARA CAMBIAR LA NOTA TÓNICA.
            tonicName = noteNames[tonic];
        }
        else {
            notesDownList.push_back(message.getNoteNumber());   // SI NO ES TAN AGUDA, SE CUENTA COMO TOCADA.
            midiBuffer.addEvent(message, (message.getTimeStamp() * sampleRate));
        }
        mainKeyboardState.noteOn(1, message.getNoteNumber(), 0.5f);
        chordKeyboardState[0].allNotesOff(0);
        chordKeyboardState[1].allNotesOff(0);
        chordKeyboardState[2].allNotesOff(0);
    }
    else if (message.isNoteOff()) {                 // SI EL USUARIO LEVANTA UNA TECLA,
        notesDownList.remove(message.getNoteNumber());  // SE QUITA DE LA LISTA DE NOTAS TOCADAS,
        midiBuffer.addEvent(message, (message.getTimeStamp() * sampleRate));
        mainKeyboardState.noteOff(1, message.getNoteNumber(), 0.0f);
        if (notesDownList.size() < CHORDNOTESNUMBER)    // Y SI DEJA DE TENER PULSADAS LAS NOTAS SUFICIENTES, SE ENTIENDE QUE SE CAMBIA DE ACORDE.
            ready = true;
    }
}
```

Figura 4.2: Captura de notas en la función processBlock.

processBlock es una función que maneja dos buffers, uno de audio y uno de información MIDI. Nuestro plugin no utiliza el primero, pues no genera ni modifica audio como ya hemos tratado. Utilizaremos el buffer MIDI, pero para mayor seguridad, lo sustituimos por uno de creación propia con la orden swapWith. Cada nota que pulsemos o dejemos de pulsar no ha de actualizarse sólo en el keyboardState, sino también en dicho buffer.

processBlock también comprueba a cada llamada el estado de los botones de PLAY: si alguno de ellos está pulsado, se añaden al buffer los mensajes MIDI correspondientes a las notas del acorde asociado al botón en cuestión.

4.4 ChordProcessor

Aquí es donde verdaderamente se encuentra el grueso del código. Su funcionamiento se fragmenta esencialmente en tres funciones, el analizador de acordes tocados por el

usuario, el algoritmo de sugerencias de acordes nuevos, y el algoritmo de enlace que selecciona las notas exactas.

4.4.1 getPlayedChord

La primera función, getPlayedChord, actualiza un tipo estructurado propio pasado por referencia con la información completa del acorde: el Chord. Éste comprende su ChordType (la información propia de cada especie cargada del archivo), su nota fundamental y su inversión.

Para obtener estos datos y construir el Chord, partimos identificando la nota más grave y los nombres de las notas tocadas con la operación módulo 12. Con esa información es fácil calcular la interválica entre cada nota, con simples restas numéricas (y pequeñas operaciones auxiliares) entre las tres notas. Esto nos da un conjunto de tres intervalos, pero no su estado de inversión (el usuario puede tocar un acorde en fundamental o cualquier inversión, y no tenemos modo de saberlo por el momento), de modo que invertimos el acorde y salvamos sus tres posibles estados. Por último, realizamos comparaciones entre cada uno de los tres estados y cada uno de los acordes cargados del fichero. Está garantizado que se dará una coincidencia, cuando el acorde introducido esté en fundamental y se alcance la especie que le corresponda, pues el fichero las contiene todas por defecto. En ese momento tenemos nuestro acorde, que cargamos de la información guardada y al que añadimos su nota fundamental (dada por el usuario) y su inversión (en base al estado fundamental, que da el fichero, y el estado en que el usuario lo ha introducido).

```
////////// OBTENER NOTAS DEL ACORDE //////////

int bass = 300;                                    //
int numNotes = 0;                                  //
bool scaleNotes[12] = { false };                   //
for (auto const& i : notesDownList) {              //
    if (i < bass) bass = i;                        //
    int note = i % 12;                             //
    if (scaleNotes[note] == false) {               //
        scaleNotes[note] = true;                   //
        ++numNotes;                                //
    }
}
bass = bass % 12;                                  //

////////// OBTENER INTERVALOS DEL ACORDE /////////

int index = 1;                                     //
std::list<int> intervals;                          //
for (int note = 1; note < 12; ++note) {
    if (scaleNotes[(note + bass) % 12] == true) {
        intervals.push_back(index);                //
        index = 1;                                 //
    }
    else ++index;                                  //
}
intervals.push_back(index);                        //
```

Figura 4.3: Fragmento de la función getPlayedChord, con el análisis de la interválica.

4.4.2 getProgressionSuggestions

La segunda función es la que contiene el algoritmo que crea las sugerencias de acordes para tocar a continuación, independientes de inversiones.

Su funcionamiento, informáticamente hablando, es sencillo: un array de tres enteros almacena las puntuaciones máximas, y otro almacena los acordes propiamente dichos en posiciones correspondientes. Para cada especie de acorde, excepto aquellos denominados “Unknown” (que no tenemos en cuenta por ser matemáticamente posibles, pero musicalmente irrelevantes), evaluamos cuál es la nota fundamental a partir de la cual se construiría su versión mejor puntuada, y calculamos su puntuación total. Si esta es superior a alguna de las anteriores, el nuevo candidato se selecciona, escogiendo al azar en caso de empate, y se actualizan ambos arrays. Examinadas todas las especies, tendremos en el array los tres acordes elegidos.

```
bool getProgressionSuggestions(Chord chord) {
    bool ok = false;
    int okc = 0;

    int points = 0;
    int highestPoints[] = { 0, 0, 0 };
    int function = getFunction(chord.root);
    int leadingTone = (tonic + 11) % 12;

    for (const ChordType candidateType : ChordLoadedData) {

        if (candidateType.chordType != "Unknown") { ... }
    }

    if (okc > 2)
        ok = true;
    return ok;
}
```

Figura 4.4: Estructura del algoritmo en la función getProgressionSuggestions, con los criterios de puntuación omitidos.

Para calcular las puntuaciones, como se ha dicho previamente, es más importante saber qué parámetros se tienen en cuenta que cómo se valoran exactamente. De entre los conceptos explicados anteriormente, el algoritmo tendrá en cuenta las funciones (favorece el ciclo Tónica-Subdominante-Dominante-Tónica en este orden), las luminosidades (especialmente comparando las del acorde anterior y el posterior), las notas importantes de la tonalidad (la tónica, la fundamental del anterior acorde, la sensible) y las relaciones entre todos estos parámetros. También se incluye una lista FIFO de acordes llamada chordHistory que almacena los últimos acordes tocados y los tiene en cuenta en este punto, principalmente para penalizar los acordes que harían una armonía más estática por repetición excesiva de funciones.

4.4.3 linkChords

Esta tercera función es la encargada de transformar la información en abstracto de los acordes elegidos en una serie de notas concretas, que serán las que finalmente se le muestren al usuario.

Su implementación toma elementos de getPlayedChord, y es la más compacta de las funciones principales. En primer lugar, extrae los números de las notas del acorde sugerido, del cual conocemos su fundamental y su interválica. Después, conociendo la nota más grave del acorde del usuario, elegimos la que será la nota más grave del nuevo acorde, que puede o no ser su fundamental. De entre las notas posibles, escogemos la que sea más cercana al bajo del usuario, sea ascendente o descendentemente, buscándola con un bucle que irá aumentando el valor de dos auxiliares, uno en cada sentido.

```
int noteup = absoluteBass % 12, notedown = absoluteBass % 12, count = 0;
bool ok = false;
while (!ok) {                                          // ESTE BUCLE BU
    if (scaleNotes[noteup] || scaleNotes[notedown]) {  // CUANDO HALLEM
        ok = true;

        if (scaleNotes[notedown]) {
            linkedBass = absoluteBass - count;         // ESCOGEMOS DIC
            if (linkedBass < 0)
                linkedBass += 12;
        }
        else {
            linkedBass = absoluteBass + count;
        }
```

Figura 4.5: Código para elegir el bajo del nuevo acorde en linkChords.

Una vez elegido el bajo, solo resta saber qué notas quedarán por encima (el resto del acorde) infiriendo su inversión. El proceso es simple, tan solo hay que ir restándole a la nota del bajo los intervalos del acorde hasta que hallemos la nota fundamental. Si restamos el primer intervalo y hallamos la nota, es que el acorde está en primera inversión, si sucede con los dos primeros, estará en segunda. Véase: si el acorde es Do Mayor (Do, Mi, Sol) y tenemos como nota bajo Sol, vamos restando los intervalos del acorde (4, 3, 5). Si a Sol le restamos 4 semitonos, obtenemos una nota irrelevante (no es la fundamental, Do), pero si seguimos un paso más y le restamos otros 4, llegamos al Do y la inversión equivale al número de intervalos que hemos restado.

```
int auxNote = linkedBass % 12, inversion = 0;    // PARA
    for (const int a : selectedChord.type.intervals) {
        if (auxNote == selectedChord.root) {
            selectedChord.inversion = inversion;
        }
        else {
            ++inversion;
            auxNote -= a;
            if (auxNote < 0)
                auxNote += 12;
        }

    }
```

Figura 4.6: Mecanismo de deducción de inversión a partir de un bajo dado.

Con la inversión actualizada, devolvemos el valor del bajo a la función principal, pues es la nota más cercana, y a partir de la cual construiremos el acorde nuevo en el teclado.

4.5 ChordData

El analizador de acordes funciona extrayendo información de un atributo de texto simplificado ubicado en el módulo “ChordData.h”. Esta información podría ser modificada de manera relativamente sencilla para añadir acordes poco frecuentes, cambiar su denominación o incluso reconfigurar hasta cierto punto las normas de la armonía que analiza el algoritmo. Se detalla aquí el formato de la información reflejada:

Cada línea del archivo comienza con los intervalos del acorde, incluyendo el intervalo de vuelta de la última nota a la primera. Por lo tanto, ya que estamos trabajando con acordes de tres notas, cada línea debe comenzar con tres valores numéricos cuya suma sea 12. Se considera que el orden de intervalos tal y como figura en su representación será interpretado por el programa como el estado fundamental del acorde (sin invertir).

A continuación, aparece la especie del acorde en forma de texto sin espacios. La especie es el “nombre” del tipo de acorde, por ejemplo, mayor, menor, disminuido, etc. La interpretación de un acorde determinado es un aspecto hasta cierto punto cultural y subjetivo, y es relevante solo para mostrarlo al usuario y no para el algoritmo propiamente dicho.

Por último, un solo valor entre 0 y 6, ambos inclusive, que representa la “luminosidad” o consonancia sonora del acorde, un valor tan socialmente determinado como la especie, y que sirve para que el algoritmo empareje acordes con cierto interés. Por ejemplo, un acorde Mayor sería más luminoso que uno Menor.

```
4 6 2   Dom7(No5) 6
5 2 5   Sus4 4
3 1 8   Unknown 0
7 3 2   Dom7(No3) 6
3 3 6   Diminished 0
3 4 5   Minor 2
```

Figura 3.1: Ejemplo de acordes en el formato del módulo "ChordData.h"

ChordData.h también tiene responsabilidades relacionadas con la representación de la información musical en otros formatos, como el atributo noteNames, que convierte un valor numérico a un string con el nombre de la nota correspondiente, y la función chordToString, que hace lo propio para los acordes que se muestran a la izquierda de los teclados.

CAPÍTULO 5

Conclusiones y trabajo futuro

El desarrollo del programa y del documento que nos ocupa ha supuesto mi primera incursión real, independiente y enfocada en el mundo del estándar MIDI y el desarrollo de plugins y aplicaciones de informática musical que conocía casi exclusivamente desde el lado del usuario. Es un mundo que me resultaba intimidante, por su comunidad abundante pero descentralizada, su relación tan solo tangencial con la parte más clásica de la música de donde viene mi formación, y por mis anteriores intentos, siempre infructuosos, de desarrollar alguna aplicación similar, que se acumulaban haciendo que afrontara cada nuevo proyecto con miedos renovados. Si bien este programa es modesto en contenido y mejorable en formato, puedo decir satisfecho que he desarrollado un plugin musical, y puedo afrontar con energías renovadas tanto el desarrollo de herramientas como el propio entendimiento de las ya existentes, algunas de las cuales siempre se me han resistido por su complejidad.

El trabajo con JUCE ha facilitado mucho las cosas, especialmente por los tutoriales con programas prefabricados, sus ejercicios sencillos y su extensa documentación. Como cada framework o herramienta informática, por supuesto que cuenta con foros en los que se tratan los temas más preguntados, pero el simple hecho de disponer de excelentes tutoriales básicos y de muchos algo más avanzados para cada pequeño aspecto del framework es una ventaja mayúscula y que se echa de menos trabajando en otros ámbitos.

Ya con el programa funcionando, puedo ver qué hay más allá y pensar en varias funcionalidades que expandieran su utilidad, siempre dando prioridad a las necesidades de mis alumnos, a quienes más me imagino usando un plugin similar a este. Lo primero que añadiría sería la gestión de acordes de distinta cantidad de notas, pues ni siquiera saliendo de la armonía clásica he echado de menos el verdadero acorde de séptima de dominante, de cuatro notas. Sería de utilidad también añadir variables al algoritmo que evalúen acordes no ya dentro de una tónica, sino para escalas concretas, es decir, distinguiendo entre Do Mayor y Do menor, por ejemplo. A hilo de esto, sería fácilmente extensible con selectores de estilo, para evaluar las armonías como clásico-románticas, de jazz, contemporáneas, barrocas, de pop, etc. Considero particularmente interesante esta función ya que, siendo consciente de que el que programa es alumno de música clásica, no es un estilo por el que la mayoría de los alumnos de piano de los primeros años se ilusionen especialmente, y su análisis para valorar armonías en el algoritmo sería sin duda un trabajo musicalmente enriquecedor.

Con algo más de ambición se podría intentar expandir la aplicación haciendo que la tónica no la proporcione el usuario explícitamente con una sección del teclado, sino que el propio programa la deduzca a partir del historial de acordes ya implementado. Si tres acordes seguidos comparten ciertas características, es asequible que un algoritmo decida cuál es la tonalidad a partir de sus fundamentales, relaciones funcionales y demás. Para hacerlo más aplicable al ámbito escolar o educativo en general, el primer paso ideal sería mejorar la interfaz gráfica, haciendo los teclados más grandes y legibles, o incluso dejando la opción "tutorial" en la que sólo se vea un teclado con uno de los acordes, que el alumno tiene que conseguir tocar exactamente antes de continuar, o con opciones para repetir enlaces en bucle y así pulir la técnica entre ciertos acordes empleados con frecuencia. Sería interesante poder almacenar los acordes de un tema concreto y que el programa los vaya intercalando de ciertas maneras para que el estudiante entrene con el enlace de acordes dentro de una de las canciones que tendrá que tocar con soltura más adelante. Las posibilidades son muchas y confío en que inspiren a alguien a llevar un proyecto similar más allá.

CHAPTER 5

Conclusions and future work

The development of this program and document has been my first real dive into the world of MIDI and musical plugin programming, which I knew exclusively from the user standpoint. It's a vast and overwhelming world, specially when coming from a classical music background, and all of my earlier attempts at making a project of my own have been failures. However, after finishing HarmonyPal, I can much better understand al lot of the technicalities and begin a new potential project with renewed strength. HarmonyPal is no big new development in the digital music industry, but I can proudly say that I finished it and I can face new challenges with this humble tool on my arsenal.

Working with JUCE has been a blessing, because its much detailed and adequate tutorials and documentation are not a given in many other areas of programming. I would love to see this kind of communities with many other computing tools.

With the application up and running, I can already envision a lot of changes and additions that could be made to expand its functionalities and make it more interesting to the end user.

The first thing I would add is a way to work with chords of four or more notes, because of how prevalent they are in western modern (and even classical!) music. It would also be of much help to expand the algorithm such that it can distinguish between different kinds of scales, and not only chords, like working in a different way with C Major and C minor scales. Moreso, the harmony is not limited to western and modern, so it would be nice to consider classic, jazz, or baroque sonorities, all with their own particular rules. The analysis to implement these kinds of music would be very artistically enriching.

Even further beyond, it could be made so that the program itself infers the scale that the user is playing, making that they don´t need to input it themselves and that it flows more naturally with the user´s chords, as it would in practice. It can also be very useful for the educational aspect of it to improve interface design and add new features such as tutorials for early beginners, as those were my main objective across the whole project. There are many possibilities, and I hope someone gets to them now or in the future.

CAPÍTULO 6

Bibliografía

Galería de imágenes

2.2: https://medium.com/musi-co/tonos-semitonos-sostenidos-bemoles-curso-de-piano-online-lecci%C3%B3n-2-a4a12377dbc9

2.3: https://marcolara.net/acordes-piano/acorde-do-mayor-c

2.4: http://elclubdelautodidacta.es/wp/2015/07/inversion-de-acordes/

2.6: https://www.ccapitalia.net/reso/articulos/audiodigital/pdf/08-EspecificacionMIDI.pdf

Teoría e investigación

(1) Plugins MIDI: https://www.marcogalvan.com/2021/02/tipos-de-plugins-para-audio-y-musica.html

(2) Teoría armónica básica: http://www.bustena.com/curso-de-armonia-i/unidad-1/

(3) Manual de Projucer: https://juce.com/discover/stories/projucer-manual#1.1-welcome-to-the-projucer!

(4) Tutoriales de JUCE: https://juce.com/learn/tutorials