Un instrumento virtual basado en síntesis granular para composición musical

A virtual instrument based on granular synthesis for music composition

Por:
Nicolás Pastore Burgos

UNIVERSIDAD COMPLUTENSE
MADRID

Trabajo de Fin de Grado
Grado en Desarrollo de Videojuegos
Facultad de Informática

Dirigido por

Jaime Sánchez Hernández

Madrid, 2021-2022

Agradecimientos

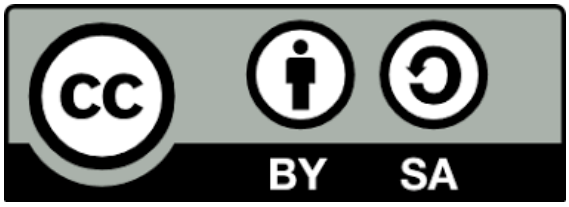
Quiero agradecer a Jaime por la paciencia infinita que ha tenido conmigo, a Ricky, Stiven, Carlos y Ana por echarme una mano con la memoria y haber estado ahí cuando lo necesitaba, a Nacho por enseñarme que es una bibliografía, a Diego y a mis productores por echarle un ojo y darle feedback, a la carrera por haberme dado los conocimientos necesarios para hacer el proyecto una realidad, a mi familia por haberme dado los recursos, haber leído mil veces la memoria y el impulso de haber podido trabajar y seguir adelante.

Autorización y difusión de uso

Se autoriza a la UCM (Universidad Complutense de Madrid) a difundir y usar el trabajo realizado con fines académicos y mencionado a los autores, tanto la memoria como el código.

Nicolás Pastore Burgos

Madrid, 2021-2022



Índice general

Resumen

El objetivo principal de este trabajo es desarrollar un instrumento virtual que, mediante un algoritmo de granulación sobre un archivo de sonido, cree nuevos sonidos para la composición musical de cualquier tipo de proyecto, como la composición de música *techno*, ambiental o para la composición de bandas sonoras para películas o videojuegos. Este módelo también se utiliza para modificar el timbre o duración de los sonidos.

Para ello, se utiliza la síntesis granular, un modelo de producción de sonidos basado en la división de uno o varios sonidos existentes en muchos fragmentos o granos acústicos, que al reorganizarse, secuencial o en paralelo, generan un nuevo sonido. En este proyecto se utiliza un *sampler* para cargar los archivos, siendo este un programa digital que permite cargar un sonido y reproducirlo.

Este instrumento también cuenta con un ecualizador, que actúa como una herramienta para controlar la ganancia de las frecuencias del sonido, al igual que un analizador de estas, tanto para el canal de sonido izquierdo como derecho.

El instrumento funciona tanto como una aplicación independiente, *standalone*, como en formato **VST**, siendo el que utilizan los **DAW**, uno de los tipos de software más utilizados en la creación musical.

Palabras clave: JUCE, C++, música digital, *sampler*, síntesis granular, instrumento virtual, VST, *standalone*, ecualizador, analizador de frecuencias

Abstract

The main objective of this work is to develop a virtual instrument that, by means of a granulation algorithm from a sound file, creates new sounds for the musical composition of any type of project, such as the composition of *techno* music, ambient or for the composition of soundtracks for movies or video games. Granular synthesis can be used to modify the pitch or duration, of sound.

To do this, **granular synthesis** is used, a sound production model based on the division of one or several existing sounds into many fragments or acoustic grains, which, when reorganized, sequentially or in parallel, generate a new sound. In this project a *sampler* is used to load the files. A *sampler* being a digital program that allows you to load a sound and play it.

This instrument also features an equalizer, which acts as a tool to control the gain of sound frequencies, as well as a frequency analyzer for both the left and right sound channels.

The instrument works both as a standalone application and in **VST** format, which is used by **DAWs**, one of the most widely used types of software in music creation.

Key Words: JUCE, C++, digital music, *sampler*, granular synthesis, virtual instrument, VST, *standalone*, equalizer, frequency analyzer

Capítulo 1

Introducción

La **síntesis granular** es un modelo de producción de sonidos que se basa en la composición de sonidos nuevos a partir de la división de un sonido ya existente en secciones llamadas granos acústicos y en su reproducción en un orden diferente al original. El sonido original puede tener muchos orígenes, pero en este trabajo se utilizará un sampler para importar archivos de audio como fuente, y es por ello que muchas veces se utilizará el término de *sampler granular* para referirse al instrumento. Esta también puede ser por síntesis **FM**, **RM**, por tabla de ondas o aditiva entre otras. Este modelo tiene otros usos como extender la longitud de un sonido sin alterar su frecuencia, o alterar su frecuencia sin alterar su duración, útil para programas como *Autotune*, que permiten modificar el timbre de un sonido a otro para ayudar con la afinación de algunas interpretaciones.

Este proyecto además cuenta con un ecualizador, utilizado para poder controlar las frecuencias del sonido resultante. Este permite cortar las frecuencias bajas y las altas con un filtro paso alto y paso bajo respectivamente al igual que un filtro de banda, que permite controlar la ganancia o volumen de una zona de frecuencias.

El instrumento además funcionará en un **DAW**. Un **DAW**, o *Digital Audio Workstation* [3], es un programa informático utilizado para la creación, composición, mezcla y masterización de música digitalmente. Estos programas suelen funcionar como un entorno de trabajo donde se lanzan programas **VST**, o *Virtual Studio Technology*, que cumplen la función de instrumentos o efectos sonoros. Este proyecto, está pensado para poder utilizarse tanto individualmente (*standalone*) o como **VST**.

1.1. Motivaciones

La síntesis granular es uno de los modelos más poderosos para la creación musical. Este es capaz de utilizarse como herramienta para modificar sonidos existentes o crear nuevos como he explicado anteriormente. Aún así es uno de los algoritmos de composición musical más olvidados. Es por ello que he querido crear una herramienta lo suficientemente cómoda para que gente que desconozca la tecnología sea capaz de descubrir este modelo, y lo suficientemente potente para crear sonidos que puedan utilizarse en proyectos. También he querido implementar un control de la envolvente, es decir la manera en la que los granos comienzan y paran de reproducirse, al observar que el resto de *software* con este modelo implementado no suele experimentar con su control.

Además, como una persona que ha estado involucrada con la música desde hace más de ocho años, y como actual compositor para la empresa de videojuego *Mad Cream Games* [5] y baterista de la banda de *El Corral de los Quietos*, trabaje en este proyecto con la intención e ilusión de poder utilizar este *software* para nuestro próximo videojuego *Painting Werther* [6] y para futuras canciones de la banda.

1.2. Objetivos

El objetivo de este trabajo es crear un *Sampler granular* funcional, con una interfaz gráfica que responda en tiempo real y permita modificar los parámetros del modelo. Estos parámetros deben ser la densidad de granos reproduciéndose al mismo tiempo, la ganancia de estos, el rango de donde se puede extraer los granos del archivo, el rango de duración de estos y la envolvente que se puede utilizar de estos. Además este debe contener un ecualizador con un filtro de paso bajo, alto y un filtro de banda para controlar las distintas frecuencias del sonido.

Este debe poder compilarse tanto como *standalone*, es decir, como un programa individual como en formato **VST** con el objetivo de poder integrarlo en un **DAW**, en específico, **FL Studio**, ya que es uno de los *Digital Audio Workstation* más utilizados.

1.3. Estructura de la memoria

Esta memoria está dividida en siete capítulos y la bibliografía. Esta contiene los enlaces y referencias a todas las fuentes de la investigación, trabajo y tecnologías utilizadas en el proyecto. En las secciones se hablará en mayor detalle de los conceptos resumidos a continuación:

- **Capítulo 1 - Introducción**. En este, se hace una primera explicación de los conceptos básicos del proyecto y se desarrollan las motivaciones que llevaron a la elaboración de este trabajo al igual que el objetivo del mismo.
- **Capítulo 2 - Estado del arte**. Aquí se explican los usos de esta técnica en la actualidad y su proyección a futuro.
- **Capítulo 3 - Metodología y tecnología**. Se explica la metodología utilizada para llevar a cabo el proyecto y se profundiza en las tecnologías utilizadas.
- **Capítulo 4 - Diseño de la aplicación**. Análisis del diseño gráfico de la interfaz, desde las necesidades que satisface, inspiración y un ejemplo de uso.
- **Capítulo 5 - Arquitectura e implementación del código**. En este capítulo se profundiza en el código informático del proyecto. En él se verá que arquitectura se ha seguido y sus distintas partes y funcionalidades.
- **Capítulo 6 - Conclusiones y trabajo futuro**. En este, se hace una retrospectiva del trabajo y se analizan los posibles cambios y mejoras que se podrían hacer en versiones futuras.
- **Anexo**. En este apartado se encuentra un manual de uso del instrumento.
- **Bibliografía**. Una colección de referencias e hipervínculos de las fuentes de la investigación y tecnologías utilizadas para crear el proyecto.

Los capítulos uno y siete se pueden encontrar también traducidos al inglés, siguiendo la normativa de los Trabajos de Fin de Grado de la Universidad Complutense de Madrid, al igual que el resumen. El resto de la memoria estará escrita en español.

Capítulo 1

Introduction

Granular synthesis is a sound production model based on the composition of new sounds by dividing an existing sound into sections called acoustic grains and reproducing them in a different order than the original. The original sound can be from many sources, but in this work a sampler will be used to import audio files as the origin, and for this reason the term *granular sampler* will often be used to refer to the instrument. This can also be by synthesis **FM**, **RM**, by wavetable or additive, among many others. This model has other uses such as extending the length of a sound without altering its frequency, or altering its frequency without altering its duration, useful for programs like *Autotune*, which allow you to modify the pitch of one sound to help tune some interpretations.

This project also has an equalizer, used to control the frequencies of the resulting sound. This allows to cut low and high frequencies with a high-pass and low-pass filter, respectively, as well as a band filter, which allows you to control the gain or volume of a zone of frequencies.

The instrument will also work in a **DAW**. A **DAW**, or *Digital Audio Workstation* [3], is a computer program used for digitally creating, composing, mixing, and mastering music. These programs usually work as a work environment where **VST**, or *Virtual Studio Technology* programs are launched, which act as instruments or sound effects. This project is designed to be used both individually (*standalone*) or as a **VST**.

1.1. Motivation

Granular synthesis is one of the most powerful models for music creation. This is capable of being used as a tool to modify existing sounds or create new ones as I have explained before. Still it is one of the most neglected music composition algorithms. That's why I wanted to create a tool comfortable enough for people who aren't knowledgeable with the technology to be able to discover this model, and powerful enough to create sounds that can be used in real music projects. I also wanted to implement a control of the envelope, that is, the way in which the grains start and stop playing, as I observed that the rest of the *software* with this model implemented does not usually experiment with its control.

Furthermore, as a person who has been involved with music for over eight years, and as a current composer for the video game company *Mad Cream Games* [5] and drummer for the band *El Corral de los Quietos*, I've worked on this project with the intention and hope of being able to use this *software* for our next video game *Painting Werther* [6] and for future songs by the band.

1.2. Objectives

The objective of this work is to create a functional *Granular Sampler*, with a graphical interface that responds in real time and allows the modification of the model parameters. These parameters must be the density of grains playing at the same time, their gain, the range from which the grains can be extracted from the file, the range of their duration and the envelope that they use. In addition, it must contain an equalizer with a low and high pass filter and a band filter to control the different frequencies of the resulting sound.

It must be able to be compiled both as *standalone*, that is, as an individual program and in **VST** format with the aim of being able to integrate it into a **DAW**, specifically, **FL Studio** , since it is one of the most used *Digital Audio Workstation*.

1.3. Document structure

This report is divided into seven chapters, an annex and the bibliography. This contains links and references to all sources of research, work and technologies used in the project. The following sections will discuss in greater detail the concepts summarized below:

- **Chapter 1 - Introduction**. A first explanation of the basic concepts of the project is made and the motivations that led to the elaboration of this work are developed as well as its objective.
- **Chapter 2 - State of the art**. Here it is explained the uses of this technique, it's present and it's future projection.
- **Chapter 3 - Methodology and technology**. The methodology used to carry out the project is explained and the technologies used are deepened.
- **Chapter 4 - Application Design**. Analysis of the graphic design of the interface, from the needs it satisfies, inspiration and an example of use.
- **Chapter 5 - Code architecture and implementation**. This chapter delves into the computer code of the project. In it you will see what architecture has been followed and its different parts and functionalities.
- **Chapter 6 - Conclusions and future work**. In this, a retrospective of the work that was done and the possible changes and improvements that could be made in future versions are analyzed.
- **Annex**. Here you can find a user manual that explains how to use the instrument.
- **Bibliography**. A collection of references and hyperlinks to the sources of the research and technologies used to create the project.

This document will be written entirely in Spanish, with the exception of Chapters 1, 7 and the Abstract which will be translated to English, following the regulations of the Final Degree Projects of the Complutense University of Madrid.

Capítulo 2

Estado del arte

La síntesis granular es un **modelo de producción de sonidos** desarrollado por Iannis Xenakis [7] y Curtis Roads [8, 9], y se basa en la creación de sonidos nuevos a partir de la división de otros sonidos en secciones denominadas **granos** acústicos o *sonic grains*. Estos granos suelen durar entre 10 y 50 ms (aunque pueden llegar a ser más largos en función del resultado que se busque) y son comúnmente colocados siguiendo un orden aleatorio. En la figura 2.1 se puede observar una representación gráfica de este proceso. Además, muchos granos suelen reproducirse al mismo tiempo, generando una mayor profundidad en el sonido. Los resultados de esta técnica suelen producir sonidos, manteniendo la textura y carácter del sonido original.

Muchos de los usos modernos de este tipo de técnicas de composición se destinan a música ambiental, para composiciones de bandas sonoras de videojuegos o películas y música electrónica moderna, donde ha encontrado muy buena recepción. También destacan su uso en las distintas herramientas para la composición musical, como en la modificación del timbre de un sonido o su duración. Estas técnicas se pueden encontrar por ejemplo en aplicaciones que ajustan la afinación de sonidos sin alterar su duración como es el famoso programa *Auto-Tune* de Antares.

El sonido puede originar por un sintetizador, como uno FM, *wavetable* o sustractivo, o puede provenir de un *sampler*, que permitiría cargar uno o varios archivos al mismo tiempo. Un *sampler* es un programa informático que es capaz de cargar y reproducir archivos de sonido en momentos de tiempo concretos. Estos se suelen utilizar desde la reproducción de un sonido, a mesas de **DJ**, a un control mucho más preciso como se ve reflejado en los géneros como **Hip-Hop**, **RnB** y otros estilos musicales modernos que utilizan técnicas digitales de composición.

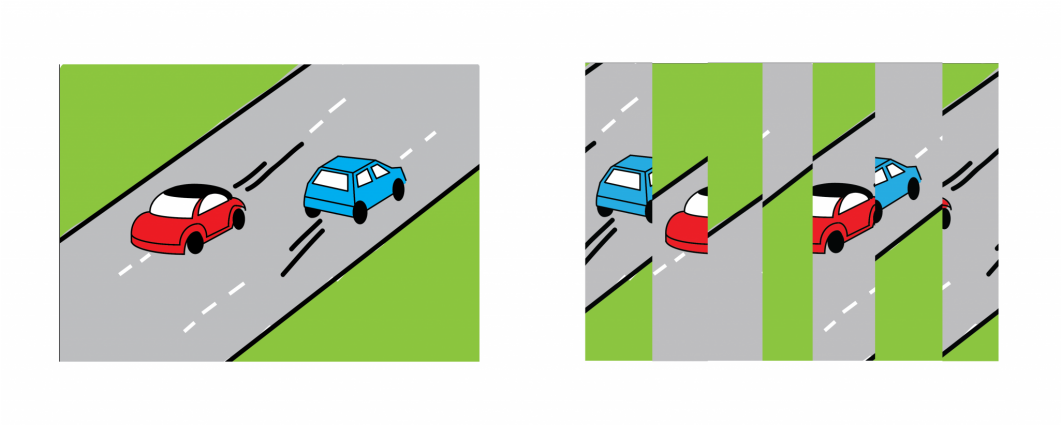


Figura 2.1: Representación visual de la división de un archivo en granos [15]

En la figura 2.2, podemos ver las distintas etapas de la síntesis granular. Primero se selecciona del origen los granos que se quieren reproducir. A continuación se le aplica una envolvente, que gestiona la entrada y salida

del sonido añadiendo un *fadein* y un *fadeout*, que significan una entrada gradual de volumen al comienzo y al final de un sonido respectivamente. Por último se suman los granos con su envolvente a la salida en un orden aleatorio.

La **parametrización** en mi proyecto, ha decidido centrarse en la densidad de granos que suenan al mismo tiempo, en controlar la variación de la longitud de estos, la envolvente que se les aplica y poder limitar una sección determinada de donde se pueden extraer los granos del archivo. Al igual que un control general de frecuencias con un ecualizador, para aportar más control del resultado final del sonido.

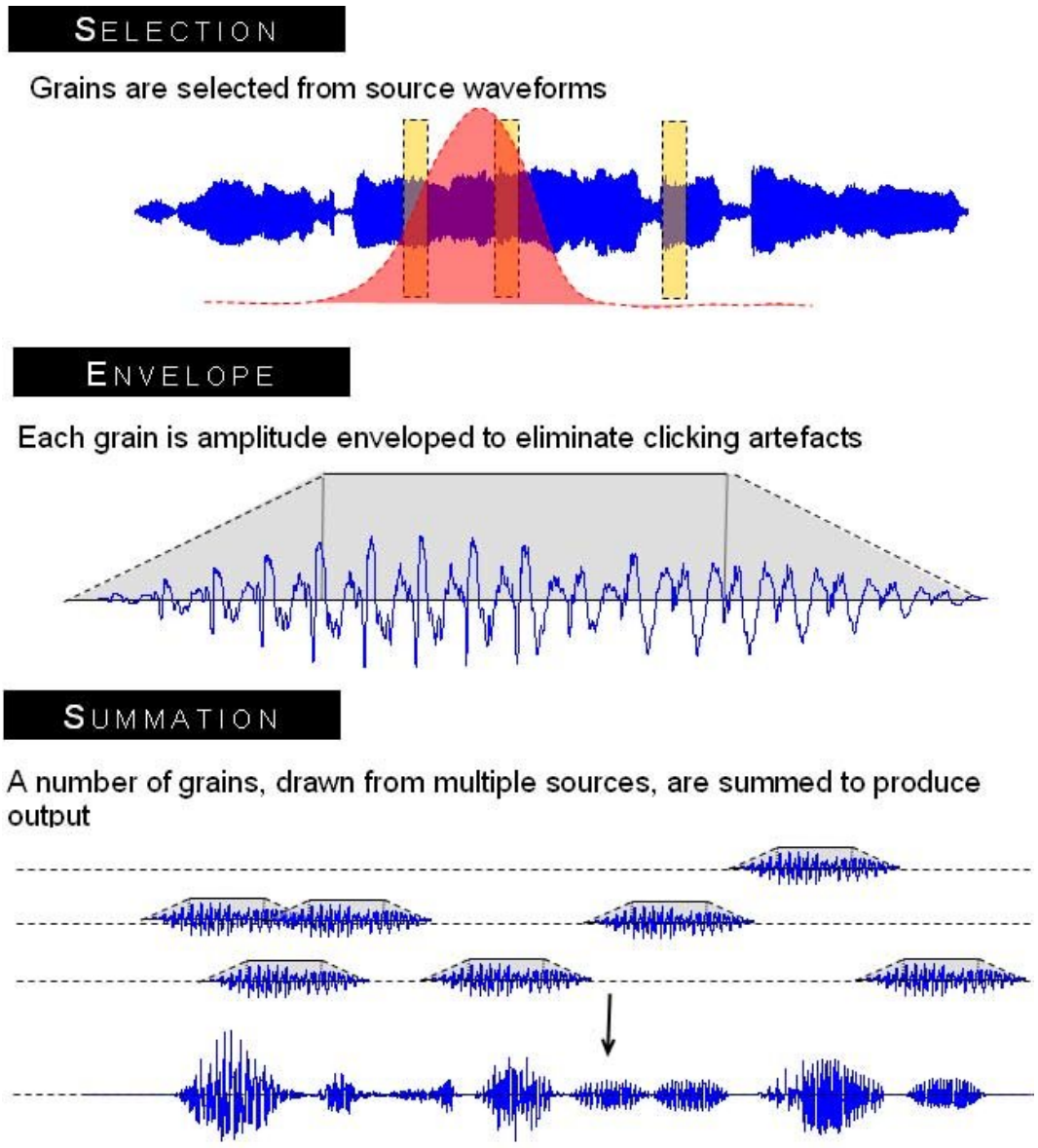


Figura 2.2: Resumen del proceso de la síntesis granular [16]

En internet, se pueden encontrar muchos programas de alto nivel que utilizan este algoritmo para la composición de nuevos sonidos, similar al uso que se da en este proyecto. A continuación se encuentran los programas que se han utilizado como referencia: Cecilia [10], Ableton Granulator II [11], GranuRise [12] y AD046 Quanta [13]. **Cecilia** es del que más ha servido de inspiración y punto de referencia, por ser *Open Source*, por su facilidad de uso y por ser capaz de crear muy buenos resultados. En la figura 2.3 podemos

observar un ejemplo de **Cecilia** en funcionamiento, en el que se puede observar su excelente control de la automatización en la parte superior, y de los controladores en la parte inferior que muchos de ellos decidí implementar en mi instrumento también, como es la densidad de granos sonando en cada momento.

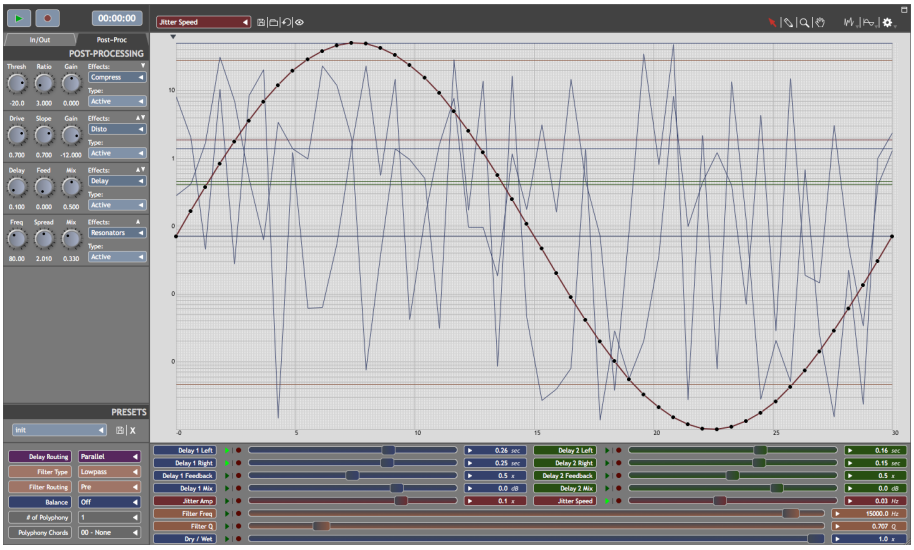


Figura 2.3: Una captura de Cecilia [10]

Muchos músicos ven la síntesis granular como el futuro de la música electrónica y de la música ambiental. Esta tecnología cada vez está siendo más utilizada por todo tipo de artistas y cada vez empieza a ganarse más su espacio entre las grandes formas de síntesis de sonido, no solo por sus usos técnicos explicados anteriormente, sino por su capacidad tan potente y rápida de crear sonidos muy ricos en texturas y profundidad.

Capítulo 3

Metodología y tecnología

3.1. Metodología de Trabajo

Scrum es una metodología ágil iterativa de trabajo diseñada para poder trabajar en un equipo de manera óptima y eficiente. Esta ha sido la metodología que he utilizado a lo largo de toda la carrera desde el primer curso y es la que he utilizado como base. **Scrum** se basa en respetar un ciclo de trabajo, llamado *sprint* de una cantidad de tiempo determinado por el equipo (siendo una, dos o más semanas) en las que se fija unas tareas concretas que se planifican terminar en ese tiempo. Para ello suele haber reuniones frecuentes (diarias o cada unos pocos días) donde se pone en común el trabajo realizado por el equipo y qué problemas han habido para poder actualizar al resto.

Aún así, al ser una metodología centrada en el trabajo en equipo, la he seguido como guía para dirigir, organizar y priorizar el trabajo. He hecho esto por ser un proceso orientado a la eficiencia y por su enfoque iterativo sobre el proyecto, que le da mucha mayor flexibilidad a haber seguido uno secuencial como sería un modelo en cascada. Además, estoy muy familiarizado con este sistema de trabajo por haberlo utilizado extensamente durante mis cuatro años cursando el grado, por lo que me resultó natural seguirlo. De esta metodología he utilizado los siguientes elementos, ajustándose a mis necesidades:

- **Product Backlog**: También conocido como *Icebox*, es donde se agrupan todas las tareas que quedan por hacer. El proceso suele partir de un documento de diseño en el que se explica y desarrolla el proyecto, y se convierten en una serie de tareas enfocadas desde la perspectiva del usuario, conocidas como **historias de usuario**. Estas tareas quedarán planificadas en función de su dificultad y estarán diseñadas para ser lo más concisas y precisas posibles. Por ejemplo, en este proyecto habían historias como: implementar el *power-cross fade* para sumar los granos entre sí, añadir un botón para cargar un archivo en el *sampler*, añadir un botón para activar o desactivar el ecualizador de tres bandas. Cabe destacar, que al ser un proceso iterativo, es flexible a errores de planificación y/o de estimación, permitiendo dividir la tarea en varias en caso de ser más complicada de lo previsto.
- **Sprint Planning**: Seguí el patrón de *sprints* para obligarme a tener fechas de entrega y no dejar a un lado el trabajo. En esta planificación elegía que tareas eran las más prioritarias para cumplir en el tiempo que me pusiera.
- **Sprint Backlog**: Este es una lista de historias de usuario que se están realizando durante el *Sprint*, tanto las que ya están terminadas, las que se están trabajando en ese momento, o las que aún quedan por empezar. El objetivo que me propuse en el *Sprint Planning* es terminar este *backlog* por completo al terminar.
- **Sprint Review**: Convencionalmente, en esta fase se evalúa si se necesita añadir o retirar historias del *backlog* por problemas que hayan surgido de planificación, para mejorar la estimación de próximos *sprints* y de optimizar en el que están. En mi caso, al estar solo, solía hacer una revisión diaria del *sprint* para evaluar la cantidad adecuada de trabajo que sería capaz de realizar.

- **Sprint Retrospective**: En esta fase se suele evaluar cómo ha sido el *sprint* actual y si se ha planificado y estimado correctamente. Al haber evaluado diariamente el *sprint*, aprovechaba esta sesión para analizar posibles errores de estimación que hubiera podido realizar, al igual que problemas que hubiera podido encontrar en mi proceso de trabajo.

En la figura 3.1 podemos observar un diagrama en el que se representan las distintas fases de **Scrum** explicados anteriormente, al igual que se refleja la naturaleza iterativa del proceso, tanto en los *Daily Scrum* como en los *sprints*.

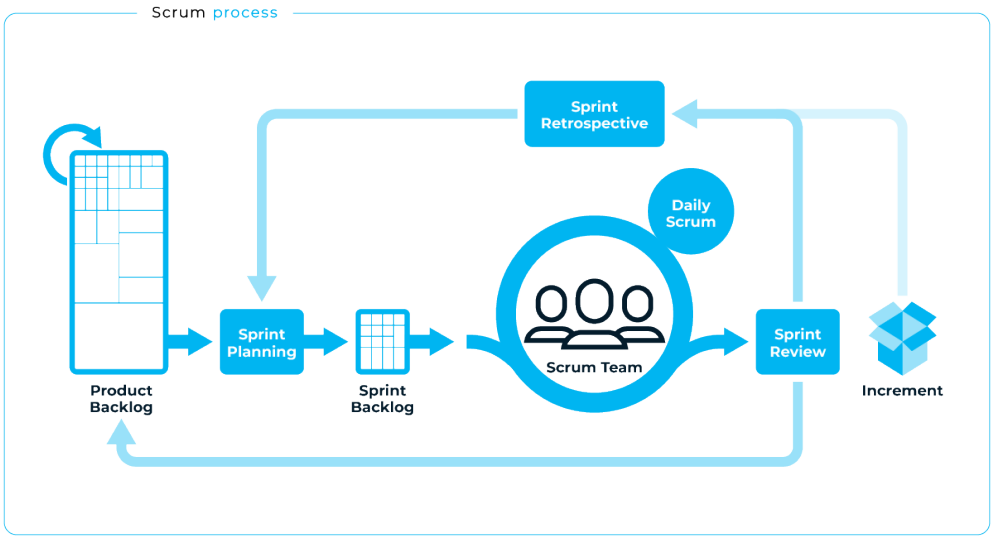


Figura 3.1: Esquema de la metodología Scrum [17]

3.2. Planificación del proyecto

Antes de comenzar a trabajar en el proyecto directamente decidí organizar el trabajo en etapas. Comencé con etapas de investigación y de aprendizaje para poder enfrentarme al proyecto mucho más preparado y con confianza:

- Investigación
- Primer contacto con las tecnologías
- Desarrollo y pruebas
- Documentación de la aplicación

3.2.1. Investigación

La investigación inicial de este proyecto tuvo dos focos muy importantes.

- **Síntesis granular**: En esta investigación decidí centrarme primero en comprender el algoritmo teórico. Me centré primero en entender el concepto de grano, como crearlos a partir de un archivo, como ordenarlos, gestionar los distintos granos que se pudieran generar y cómo conseguir varios granos sonando al mismo tiempo. Además busqué otras aplicaciones con acceso al código, que utilizaban este algoritmo como es *Cecilia* [10].
- **Búsqueda de tecnologías**: La primera decisión que tuve que tomar fue la plataforma en la que iba a crear el programa. Me planteé *Python*, con librerías como *Pyo* [19] o *PyAudio* [20]. Pero acabé optando por utilizar **JUCE** [1] en **C++** por la potencia de poder crear aplicaciones *standalone* optimizadas tanto en memoria como en tiempo de cálculo. Otra ventaja que encontré con esta librería, es que me permitía con el mismo código compilarlo como un **VST** [4].

3.2.2. Primer contacto con las tecnologías

Mi primera toma de contacto la hice siguiendo los tutoriales de la propia librería de **JUCE** [21], para aprender a importar archivos, reproducirlos y entender a rasgos generales las dos hebras que te facilita la base del proyecto para crear aplicaciones de audio. También seguí dos tutoriales muy completos en inglés de Free Code Camp [22]: el primero en el que enseñaba a realizar un ecualizador de tres bandas [23] y en el segundo donde enseñaba a crear un compresor de tres bandas también [24]. El primer vídeo tiene especial relevancia ya que acabé implementando el resultado en mi proyecto final, por lo importante que vi poder filtrar bandas que podrían quedar saturadas al sumar varios granos de una misma textura entre sí. En esta fase también comencé a familiarizarme con la API de la librería [25] y con las aplicaciones, y su código, de ejemplo que vienen con *Projucer*, al igual que a compilar aplicaciones tanto en *Standalone*, como **VST**, y en este segundo, a vincular el texto a la aplicación de *AudioPluginHost* a mi **IDE** de trabajo, en mi caso *Visual Studio 2019*.

3.2.3. Desarrollo y Pruebas

Una vez terminadas las dos fases anteriores, comencé a desarrollar una serie de historias de usuario que podrían completar mi proyecto. Una vez tuve un primer *Icebox*, o *Backlog*, del que estuviera satisfecho, planifique una serie de fechas de Hitos personales con un número de *sprints* en cada uno, y realice una primera estimación demasiado optimista sobre la cantidad de trabajo que podría realizar en el tiempo que tenía.

Teniendo una primera organización de mi trabajo, comencé a trabajar, priorizando mayoritariamente la funcionalidad sobre la interfaz. Desde ya el principio preferí depender de las herramientas de *debug* que trae el propio **JUCE** para hacer muchas de las pruebas antes de crear una interfaz gráfica capaz de modificar esos valores a tiempo real.

Durante el desarrollo, también decidí añadir un ecualizador sencillo al instrumento. Tomé esta decisión primero porque ya había seguido un muy completo tutorial anteriormente en el que mostraban cómo implementar este de procesado de audio, y segundo porque en algunas de mis pruebas, algunas frecuencias solían saturarse al reproducir varios sonidos con una misma textura. Para darle más control al usuario del sonido que se reproducía, decidí añadir estas funcionalidades.

Como observación, las pruebas que realice sobre el código y la aplicación fueron mayoritariamente mías, aunque también pedí *feedback* y opiniones a conocidos del mundo de la música sobre la interfaz y el sonido que producía la aplicación. Estos comentarios fueron mayoritariamente positivos.

3.2.4. Documentación de la aplicación

La documentación del proyecto ha sido lo último que he realizado, mientras terminaba de corregir algunos de los últimos *bugs* y errores que iba encontrando en mi instrumento. La he escrito en LaTeX, utilizando Overleaf y se puede encontrar como Anexo de este documento, o en el repositorio de *Github* de este trabajo [28]. En ella se puede encontrar la funcionalidad de cada elemento de la interfaz y un ejemplo visual de uso.

3.3. Tecnologías utilizadas

A continuación hablaré de las tecnologías que he utilizado para poder hacer posible este proyecto:

3.3.1. C++, JUCE, Projucer y Audio Plugin Host

JUCE [1] [1] es una librería de creación de *software* musical para el lenguaje de programación **C++** [26]. Esta librería ofrece recursos para la síntesis en tiempo real de audio, al igual que la capacidad de crear una interfaz gráfica enlazada al cómputo de audio, permitiendo editar parámetros a tiempo real y que afecten al procesado de audio.

Además, **JUCE** ofrece gratuitamente **Projucer**, una aplicación que permite vincular otras librerías ya incluidas con **JUCE**, como *OpenGL*, y crear proyectos base a partir de modelos eliminando gran parte de la carga de trabajo. Otra ventaja es la automatización de la creación de los proyectos y vinculaciones de las librerías en muchos de los IDEs modernos más utilizados, en mi caso *Visual Studio 2019*. Este al crear el proyecto, permite compilar el proyecto como *Standalone*, es decir, como aplicación por separada, o como **VST** [4], que permite enlazarlo en una **DAW** [3] y usarlo en composición músical. En la figura 3.2 se puede observar el aspecto de esta aplicación conteniendo mi proyecto.

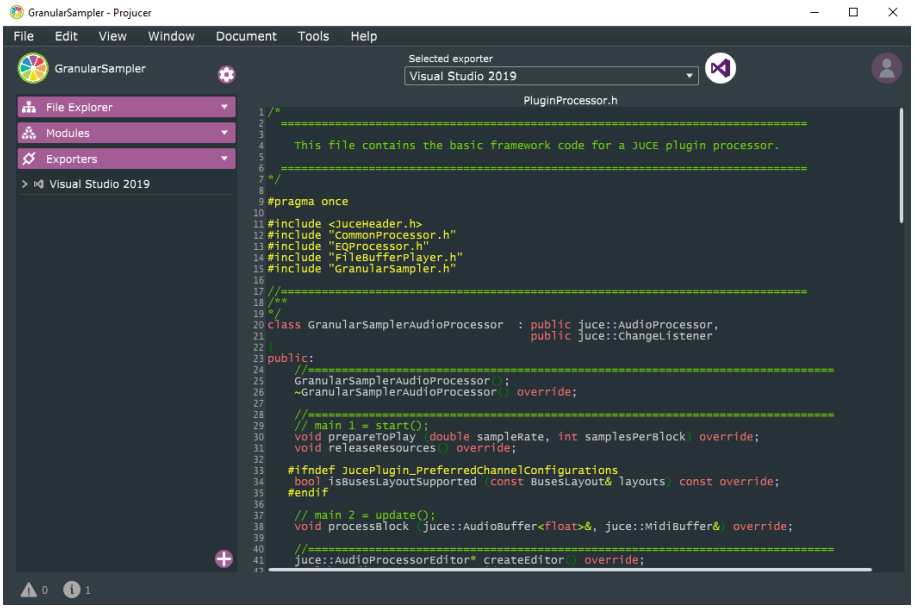


Figura 3.2: La ventana de *Projucer* del proyecto

Otra aplicación muy útil que trae, es **AudioPluginHost**, que permite probar el funcionamiento del software en formato **VST**, permitiendo ver su funcionamiento con otros programas. En la figura 3.3 podemos observar el programa funcionando con el gráfico de entrada y salida tanto **MIDI** (*Musical Instrument Digital Interface*) como de audio que utilicé para probar el funcionamiento del programa en formato **VST**. Para hacerlo tuve que vincularlo como programa de arranque en la ejecución desde *Visual Studio 2019*.

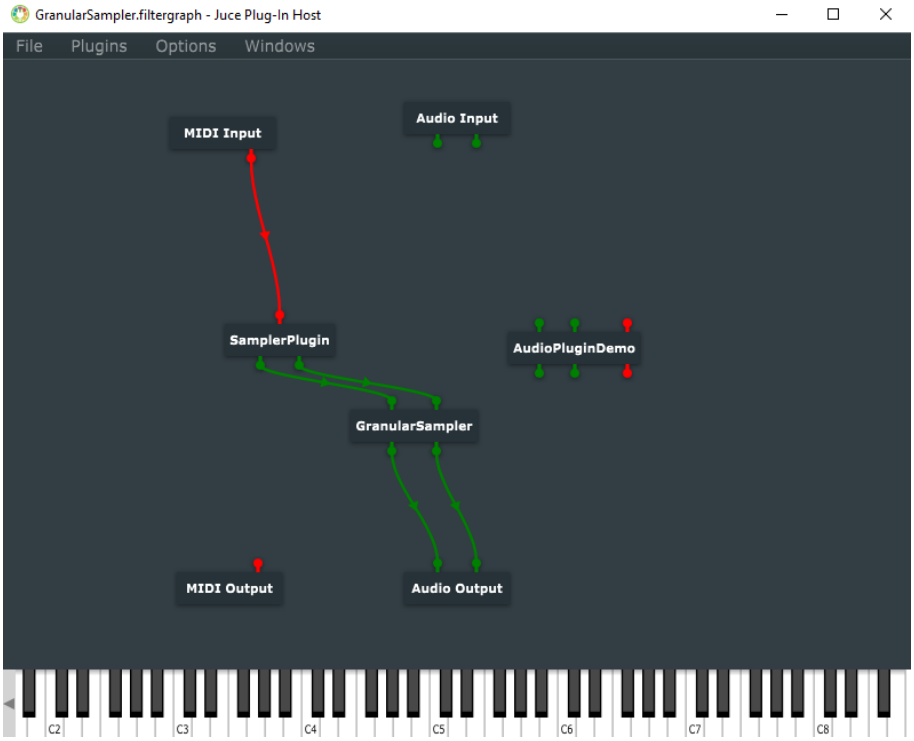


Figura 3.3: La ventana de *AudioPluginHost* utilizada durante el desarrollo.

JUCE ofrece una gran cantidad de tutoriales, tanto oficiales como de la comunidad, un foro activo y de uso gratuito, al igual que muchos ejemplos, dejando disponible el código y el resultado final para los desarrolladores gratuitamente. También ofrece una **API** extensa, que aún con alguna que otra carencia, es muy cómoda de utilizar.

La versión de **JUCE** que he utilizado es la v6.1.2 y para hacer funcionar el proyecto se requiere C++17.

3.3.2. Visual Studio 2019

Visual Studio 2019 es uno de los IDEs (*Integrated Development Environment*) más utilizados del mercado. Desarrollado por *Microsoft* [27], ofrece un entorno bastante cómodo para trabajar en lenguajes como **C++** o **C#**. La versión que he utilizado ha sido la v16.9.4.

3.3.3. Git

Git es un software de control de versiones *open source* que he utilizado para guardar un registro del trabajo realizado. Además he utilizado *GitHub* como servicio para publicar el código del proyecto y para guardar un registro de mi trabajo en la nube. [28]

3.3.4. Trello

Trello es la aplicación de organización que he utilizado para gestionar el *Product* y *Sprint Backlog* que utilizaba para trabajar. En la figura 3.4 se puede ver una captura de las historias de usuario utilizadas durante el desarrollo [31].

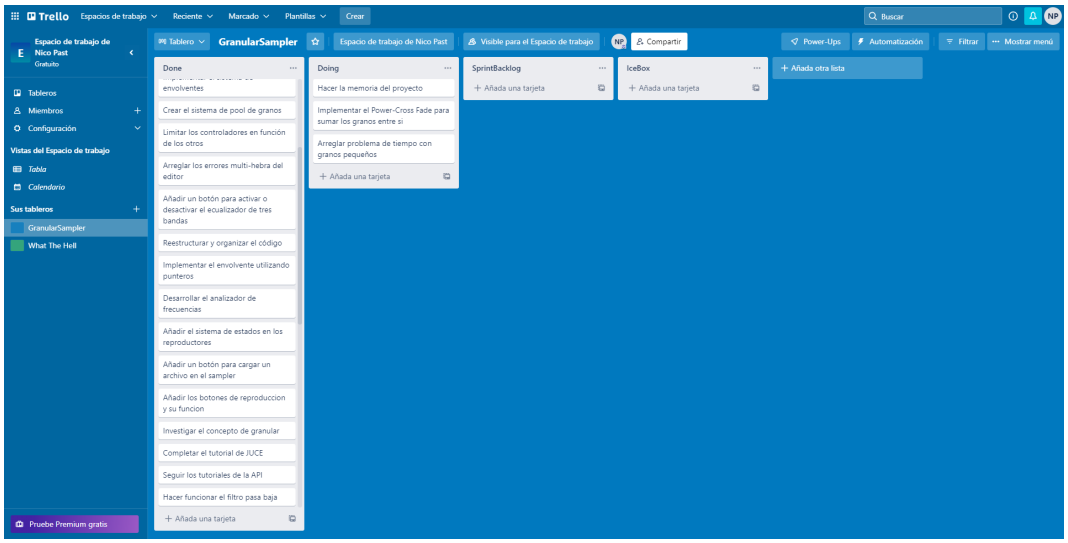


Figura 3.4: Una captura del **Trello** de mi proyecto

3.3.5. Overleaf

Overleaf es un editor en línea de textos LaTeX que he utilizado para escribir la memoria de este Trabajo de Fin de Grado y la documentación para mi proyecto. Además también he utilizado la plantilla que ofrece la Facultad de Informática de la Universidad Complutense de Madrid al igual que la ayuda de unos compañeros de mi curso.

3.3.6. Adobe Photoshop

Photoshop es un software de edición de imágenes que he utilizado para unos primeros diseños de interfaz gráfica y para editar y mejorar algunas de las imágenes de este documento.

Capítulo 4

Diseño de la aplicación

4.1. Principios y requisitos

Para poder diseñar la interfaz del proyecto, primero me centré en cuales serían los principios que priorizaría y cuales iban a ser los requisitos indispensables que debían seguir.

Sobre los principios, tuve claro que quería centrarme en funcionalidad y comodidad por encima del aspecto visual estético, ya que aún habiendo tenido alguna formación en diseño gráfico a lo largo de la carrera, nunca he estudiado en profundidad este tipo de ramas artísticas, así que preferí centrarme en crear una interfaz que a mí me resultaría cómoda de utilizar.

Para explicar los distintos requisitos que debía cumplir el instrumento, voy a dividirlos en función de su uso:

4.1.1. Sampler

Para el *sampler* quise poder tener la funcionalidad de cargar un archivo, reproducirlo sin alterarlo por la síntesis granular y reproducirlo con la granulación. Además también quise añadir la funcionalidad de detener la reproducción de cualquiera de las dos reproducciones en cualquier momento. Como último detalle, estas no podrían sonar al mismo tiempo.

4.1.2. Síntesis granular

Para la síntesis, decidí que los dos controles más importantes eran la densidad de granos reproduciéndose en cada momento y la ganancia de estos. Esto es así ya que son los controles que más modifican el sonido resultante. Después quise modificar el rango de duración de los granos, permitiendo llegar a duraciones mucho superiores a lo común, ya que quería experimentar con granos mucho más grandes y lo que afectaba el sonido final. También vi necesario poder controlar el rango de donde pueden originarse los granos del archivo original. Por último decidí poder controlar el tipo de envolvente que utilizan los granos ya que al investigar los programas que utilizaban este modelo, no muchos exploraban controlar este factor.

4.1.3. Ecualizador

Para el ecualizador de tres bandas quise poder representar la modificación a las frecuencias que se producían al utilizarlo, al igual que controles individuales para los tres filtros que posee. Para el *high-cut* y el *low-cut* quise darle un control a la frecuencia donde comenzaba el corte y la intensidad de la pendiente del corte. Quise añadir estos dos filtros por la posibilidad de *Aliasing* [32] al sumar sonidos que contienen frecuencias superiores o inferiores al rango auditivo, que pueden deteriorar la calidad, y al sumar el error en cada grano, puede multiplicar el problema. También quise aportar el control en los extremos de las frecuencias para dar más control al sonido resultante. Para el tercer filtro, al ser de banda, tuve que añadir además de la frecuencia y de la intensidad, un control de cuán abierto es el filtro, es decir, de a cuántas frecuencias afecta vecinas de la que se quiere modificar. Además, quise añadir botones que activarán y desactivarán los filtros, el análisis

de frecuencias, y el efecto al completo. Este fue añadido, además de para aportar más control del sonido final, para poder mitigar la saturación de frecuencias que se podían producir, es decir, que por la naturaleza aditiva de la síntesis, muchos granos se sumen en la misma frecuencia generando sonidos no deseados en esa región, empastando el sonido.

4.2. Inspiración

Como fuente de inspiración, quise centrarme en sintetizadores modulares analógicos. De estos quise centrarme en el aspecto de colocar los parámetros de distintas funcionalidades juntas, en secciones divididas entre sí y en el control de estos parámetros con controladores circulares, simulando un potenciómetro analógico. En la figura 4.2, se puede ver un ejemplo de un sintetizador de este tipo, y la clara división en secciones de sus funcionalidades.

Otro motivo para seguir este tipo de diseño, es que muchos otros programas ya siguen este tipo de diseño, ayudando a familiarizarse más rápidamente con el programa a la gente que ya tiene experiencia con este tipo de *software*.

Para la paleta de color, he decidido llevarlo a un estilo más futurista, utilizando colores vibrantes emulando el efecto neón. Esto lo he hecho así ya que la síntesis granular suele estar concebida con un estilo más futurista.

En la figura 4.1 se ven dos sintetizadores en el que se ven los colores vibrantes neón y el uso de diales circulares para controlar los parámetros de la composición. Ambos son *plugins* predeterminados del **DAW** *FL Studio*.

(a) Una captura de **Harmor** por ImageLine

(b) Una captura de **SimSynth** por ImageLine

Figura 4.1: Sintetizadores digitales de los que me inspire

Figura 4.2: Un ejemplo de un sintetizador modular analógico [18]

4.3. Resultado

En el resultado final, visible en la figura 4.3, se pueden ver gran parte de las influencias de las que se habla en el punto anterior. Se ve la importancia de los diales circulares y los colores vibrantes y de neón al igual que distintas formas de representar el audio, tanto con un analizador de frecuencias como con una gráfica que representa el archivo que tiene cargado el programa y los distintos picos de volumen a lo largo del tiempo.

También se ve la división por regiones de las distintas funcionalidades, siguiendo las leyes de la *Gestalt* [30] y el diseño de un sintetizador modular analógico, al igual que los programas de ejemplo que hice referencia en el anterior punto.

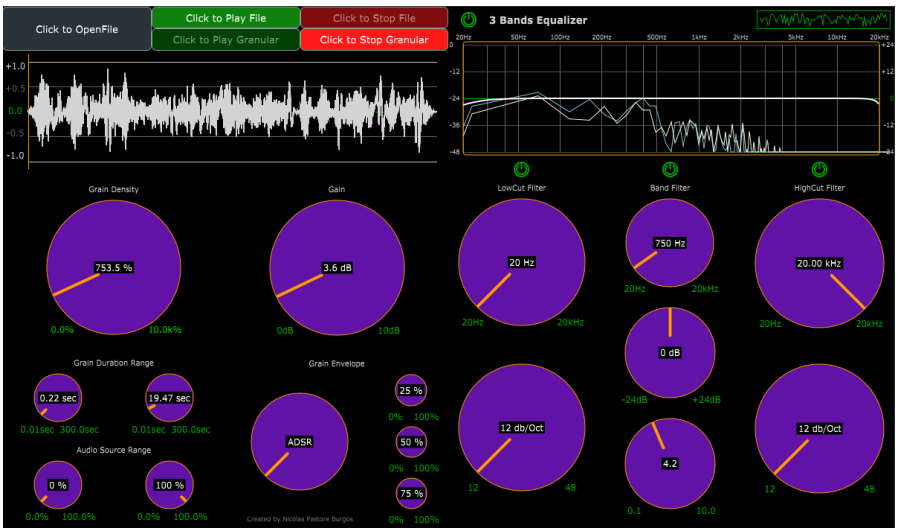
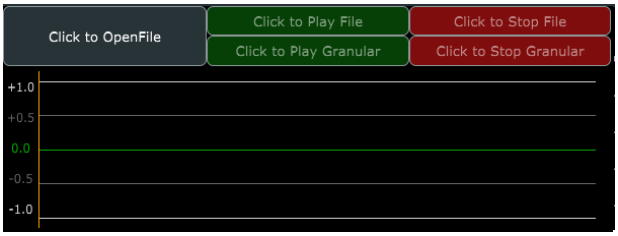


Figura 4.3: La interfaz gráfica completa del instrumento

4.3.1. Sampler

En esta sección del programa es donde se encuentra la capacidad de cargar un archivo de audio, reproducirlo sin ser modificado por la síntesis granular, que es el botón verde superior, y reproducirlo con el tratamiento del algoritmo, que es el inferior. Al pulsar uno de estos botones verdes, el botón rojo de su derecha se iluminará, permitiendo detener su ejecución. Si se pulsa el otro botón verde, también se detendrá. El botón gris de la izquierda es el que se utiliza para cargar los archivos de audio, y lanzará una ventana del sistema operativo para elegir el archivo.

En la figura 4.4, se puede observar como cambia la interfaz, de la imagen (a) a la (b), al cargar exitosamente un archivo, tanto representado en la parte inferior como activando los dos botones verdes para los distintos tipos de reproducciones.



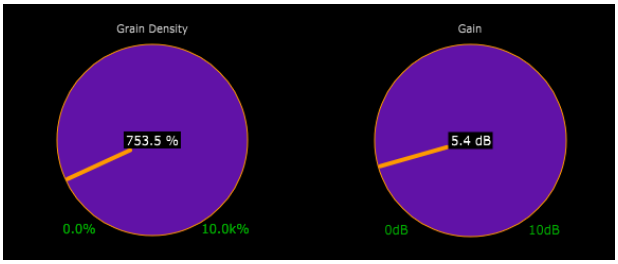
(a) El *sampler* sin un archivo de audio al iniciar

(b) El *sampler* con un archivo de audio

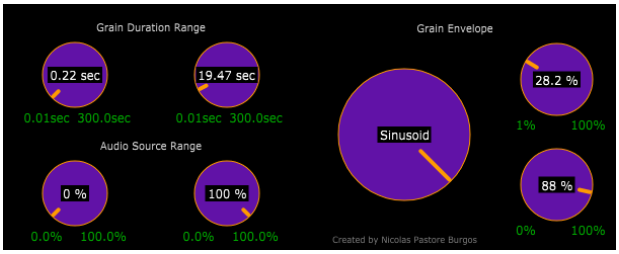
Figura 4.4: El *sampler*

4.3.2. Síntesis granular

Los controladores de la síntesis granular se dividen en dos regiones. La superior representada en la imagen (a) de la figura 4.5, donde se controla la densidad de los granos que se reproducen al mismo tiempo en porcentaje, es decir, que al 100 % hay un grano sonando y la ganancia o volumen de estos. En la parte inferior, representada en la imagen (b) de la misma figura, se encuentran los controladores de los rangos, tanto de la duración de los granos, al igual que del rango donde se pueden extraer del archivo. También se encuentran los controladores de la envolvente de los granos, es decir, el algoritmo y los parámetros que utilizan las distintas maneras en las que los granos comienzan y terminan de reproducirse.



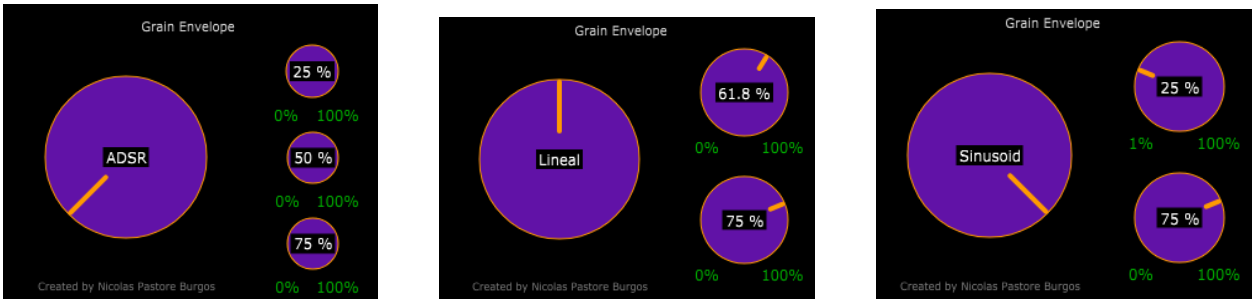
(a) Sección superior del módulo



(b) Sección inferior del módulo

Figura 4.5: Secciones de los controladores de la síntesis modular

En la figura 4.6 se pueden ver las distintas configuraciones que se pueden asignar a la envolvente de los granos. Al modificar el dial de mayor tamaño, los diales más pequeños de la derecha se modifican, ajustándose a los parámetros de cada envolvente. En el caso del **ADSR**, el primer dial desde arriba marca la distancia desde el comienzo hasta el valor del ataque, desde ese valor hasta el siguiente será el *decay*, de la misma manera con el siguiente dial es el *sustain* y el resto es reservado para el *release*. Tanto en el envolvente lineal y sinusoidal el dial superior marca la duración del *fade in*, y el inferior el del *fade out*.



(a) Envolvente **ADSR** (b) Envolvente lineal (c) Envolvente sinusoidal

Figura 4.6: Distintos tipos de envolventes de los granos

4.3.3. Ecualizador

El ecualizador se divide en dos secciones. En la parte superior en la que se representan las distintas frecuencias de los dos canales, izquierdo en azul y derecho en naranja. También se encuentra una línea blanca que representa el efecto del ecualizador sobre las frecuencias y dos botones verdes en la parte superior: el de la izquierda para apagar todo el ecualizador, y el derecho para desactivar el analizador. En la parte inferior se encuentran los controladores del ecualizador. En orden de izquierda a derecha se encuentra el filtro *low-cut*, el filtro de banda y el filtro *hi-cut*. Los dos filtros de corte tienen dos diales, uno que controla la frecuencia, y un segundo que controla la intensidad de la pendiente del filtro. En el caso del de banda, hay tres controladores, uno para la frecuencia del filtro, otro para la ganancia en estas frecuencias y un tercero que marca cuan ancho es el filtro, es decir, cuanto afecta a las frecuencias vecinas.

En la figura 4.7 se puede ver el ecualizador funcionando, con el *low-cut* apagado. También se distinguen la representación de ambos canales de sonido, de la línea blanca que muestra el efecto de los filtros en funcionamiento y de los distintos efectos que puede tener modificar los filtros en distintas maneras.

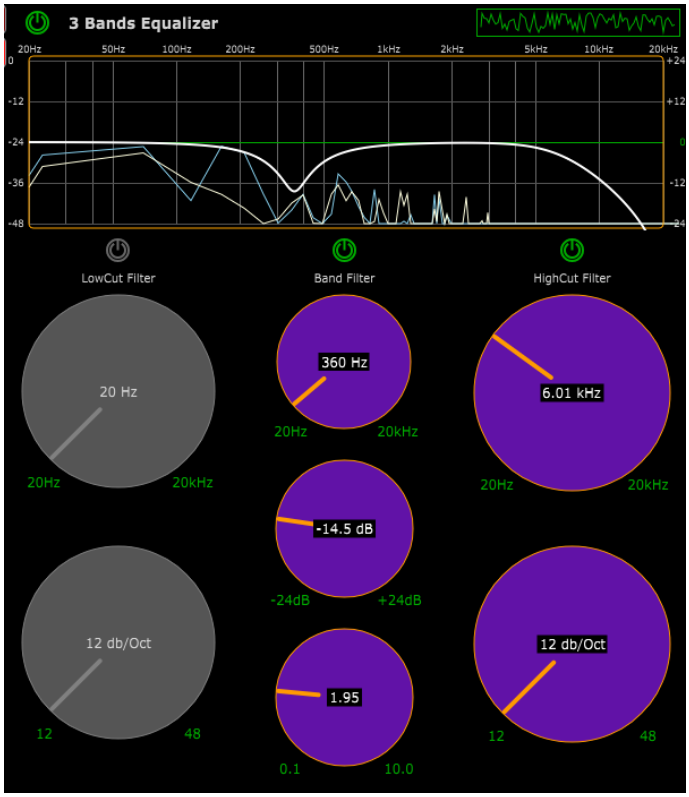


Figura 4.7: El ecualizador de tres bandas

4.4. Ejemplo de uso

A continuación se encuentra el enlace a un vídeo en el que se muestra un ejemplo de uso con anotaciones de cada sección del instrumento: https://www.youtube.com/watch?v=W7c8ZrscYrM

Capítulo 5

Arquitectura e implementación del código

5.1. Arquitectura

La arquitectura base que ha seguido mi proyecto ha sido la del modelo que viene por defecto al crear un proyecto de aplicación de sonido con *Projucer*. Este modelo tiene dos hebras en funcionamiento, la primera para el procesado de audio, que es la hebra más importante y la que lanza la segunda, que es el editor visual del programa. Esta primera es capaz de funcionar sin la segunda, pero el editor es incapaz de funcionar sin la de procesado.

Estas funcionan desde los archivos *PluginProcessor* y *PluginEditor* respectivamente. Ambos archivos tienen el ciclo principal de cada hebra, al igual que los componentes que utiliza cada una.

La comunicación entre las dos hebras sólo se genera por los parámetros de entrada y salida del instrumento. Al ser estos modificados desde la hebra del editor, se necesita enviar los nuevos valores modificados mediante un sistema de *listeners* de vuelta a la hebra de procesado. Como observación, algunos de los controles del proyecto están limitados por otros del instrumento, ya que suelen ser el límite superior que no puede superar. Esto se comunica desde la hebra del procesado al editor, y requiere de una serie de *Mutex* solo al modificar uno de los dos parámetros para evitar errores durante la ejecución.

5.2. Plugin Processor

En este archivo, encontramos toda la funcionalidad y elementos necesarios para poder procesar el audio, desde su lectura de un archivo y reproducción, a la división en granos, la reproducción de estos y su posible ecualización.

Los métodos más importantes y que más he modificado de esta clase son:

- **prepareToPlay()** En este se inicializan y cargan todos los recursos o configuraciones necesarias antes de comenzar la reproducción. En la figura 5.1, se puede observar este método y cómo se preparan los distintos elementos como el ecualizador y las cadenas que utiliza, un oscilador que se utilizó para pruebas, el sistema de carga de archivos y el tamaño del bloque máximo que deben devolver los granos.

```cpp
//==============================================================================
void GranularSamplerAudioProcessor::prepareToPlay (double sampleRate, int samplesPerBlock)
{
    // Use this method as the place to do any pre-playback
    // initialisation that you need.

    transportSource.prepareToPlay(samplesPerBlock, sampleRate);

    juce::dsp::ProcessSpec spec;

    spec.maximumBlockSize = samplesPerBlock;

    spec.sampleRate = sampleRate;

    spec.numChannels = 1;

    leftChain.prepare(spec);
    rightChain.prepare(spec);

    updateFilters();

    leftChannelFifo.prepare(samplesPerBlock);
    rightChannelFifo.prepare(samplesPerBlock);

    osc.initialise([](float x) {return std::sin(x); });

    spec.numChannels = getTotalNumOutputChannels();
    osc.prepare(spec);
    osc.setFrequency(100);

    granularSampler.prepareGrains(2, samplesPerBlock);
}
```

Figura 5.1: Captura del método **prepareToPlay()**

- **processBlock()** Este es el método crítico del programa ya que pide el audio procesado por bloques que se planean reproducir en cada *frame* del programa. Aquí es donde se deben realizar el procesado del audio para conseguir el resultado objetivo. En la figura 5.2, podemos observar una sección de este método en la que en función de cuál de los dos reproductores está funcionando, copia el bloque correspondiente, al igual que la preparación y procesado del ecualizador en ambos canales y la actualización de la estructura **FIFO** para el analizador de frecuencias.

```cpp
if(buffPlay.getState() != Stopped)
    buffPlay.copyNextBlockFromBufferFileTo(buffer);

if (granularSampler.getState() != Stopped)
{
    granularSampler.getNextAudioBlock(buffer);
    buffer.applyGain(apvts.getRawParameterValue("GranularSampler Gain")->load());
}

//buffer.copyFrom(0, 0, *bufferToFill.buffer, 0, bufferToFill.startSample, buffer.getNumSamples());

//transportSource
//readerSource->read(&audioBuffer, 0, reader->lengthInSamples, 0, true, true);

auto leftBlock = block.getSingleChannelBlock(0);
auto rightBlock = block.getSingleChannelBlock(1);

juce::dsp::ProcessContextReplacing<float> leftContext(leftBlock);
juce::dsp::ProcessContextReplacing<float> rightContext(rightBlock);

leftChain.process(leftContext);
rightChain.process(rightContext);

// ================================================

leftChannelFifo.update(buffer);
rightChannelFifo.update(buffer);
```

Figura 5.2: Captura del método **processBlock()**

- **createParameterLayout()** Es el método donde se declaran y se definen los rangos y tipos de los parámetros con los que se comunicará con el editor bidireccionalmente. En la figura 5.3 se puede observar como se crean estos parámetros.

```
juce::AudioProcessorValueTreeState::ParameterLayout
    GranularSamplerAudioProcessor::createParameterLayout()
{
    juce::AudioProcessorValueTreeState::ParameterLayout layout;

    // name, parameter name, range of parameter, step, skew value, default Val

    layout.add(std::make_unique<juce::AudioParameterFloat>(
        "GranularSampler Gain",
        "GranularSampler Gain",
        juce::NormalisableRange<float>(0, 10.f, .01f, 1.f),
        1.f));

    layout.add(std::make_unique<juce::AudioParameterFloat>(
        "Grain Density",
        "Grain Density",
        juce::NormalisableRange<float>(0.f, 10000.f, .1f, 0.25f),
        2000.0f));

    layout.add(std::make_unique<juce::AudioParameterFloat>(
        "Grain Min Length",
        "Grain Min Length",
        juce::NormalisableRange<float>(0.01f, 300.f, 0.01f, 1.f),
        0.01f));

    layout.add(std::make_unique<juce::AudioParameterFloat>(
        "Grain Max Length",
        "Grain Max Length",
        juce::NormalisableRange<float>(0.01f, 300.f, 0.01f, 1.f),
        20.5f));

    layout.add(std::make_unique<juce::AudioParameterFloat>(
        "Grain Min StartPos",
        "Grain Min StartPos",
        juce::NormalisableRange<float>(0.f, 100.f, 0.01f, 1.f),
        0.f));

    layout.add(std::make_unique<juce::AudioParameterFloat>(
        "Grain Max StartPos",
        "Grain Max StartPos",
```

Figura 5.3: Captura del método **createParameterLayout()**

Entre el resto de elementos, encontramos muchos de los métodos necesarios para hacer funcionar la clase padre *juce::AudioProcessor*, además de los parámetros del programa que serán los que modifique los controles de la interfaz gráfica. Después de estos la clase está dividida en secciones los distintos elementos del procesado por su funcionalidad, y a su vez, la funcionalidad de estas se encuentran en una clase propia.

Como observación, dentro de **CommonProcessor**, se encuentran los elementos básicos del procesador, como definiciones de *structs*, *enums* y otro código que usan todas las clases que procesan el audio.

5.2.1. File Buffer Player

Esta clase realiza la función de poder leer un archivo de audio, cargarlo y poder reproducirlo en bloques de audio después. Esta no modifica la entrada de ninguna manera.

Este tiene un *juce::AudioBuffer<float>* donde guarda toda la información, y tiene un sistema de estados del tipo **TransportState** en el que puede tener un estado de parado, empezando, reproduciéndose y parándose.

Los métodos más importantes de esta clase son **setState()**, que cambia el estado del reproductor, **copyNextBlockFromBufferFileTo()**, que realmente copia el siguiente bloque de audio en el buffer de entrada, y **setBuffer()**, que cambia el buffer de origen, cambiando el archivo que reproduce.

5.2.2. Granular Sampler

Esta clase realiza la función de un sampler granular, utilizando el *File_ Buffer_ Player* para cargar el archivo y leerlo, y usa el mismo sistema de estados que la clase. Además este se encarga de crear, gestionar y sumar distintos granos para poder reproducirlos a tiempo real.

Los métodos más significativos son **getNextAudioBlock()**, en el que se crean, reproducen y liberan los granos que se van a reproducir, y **changeState()**, donde se puede cambiar el estado del *sampler*.

Los granos se implementan a través de la clase **Garin**, donde se guardan los valores de la reproducción de cada grano individual, como los rangos de duración, la sección de sonido del archivo y métodos para aplicar la envolvente correspondiente.

Además, la gestión de los granos se hace a través de un diseño de *pool*, es decir, que los granos solo se construyen al comienzo del *sampler*, y después se van reciclando a medida que se vean utilizando.

5.2.3. EQ Processor

En este archivo encontramos gran parte de la implementación necesaria para realizar los cálculos del ecualizador del instrumento. Estas se componen principalmente de definiciones de nombres, structs y *enums* y de algunos métodos *template* que se utilizan durante el procesado de audio para componer el filtro.

Para hacer funcionar el ecualizador, se añade el módulo *juce_ dsp* (*Digital Signal Processing*) al proyecto, que aporta gran parte del cálculo de los filtros que se aplican, como las **FFT** (*Fast Fourier Transforms*), utilizadas en el analizador de frecuencias, o **IIR** (*Infinite Impulse Response filters*), aplicado en este archivo [29].

El ecualizador funciona utilizando una serie de filtros importados del módulo *juce_ dsp*, explicado arriba. Al llegar al método del procesado de audio **processBlock()**, primero se actualizan los filtros según los parametros actuales del instrumento. Después se procesan ambos canales al aplicarse por la cadena de filtros (llamada en el código **MonoChain**). Al ser ecualizador lo último que se procesa, se acaba enviando el resultado a una estructura **FIFO** (*First In, First Out)*, donde se prepara para que el analizador de frecuencias pueda convertir mediante transformaciones de Fourier (**FFT**), en una línea para su renderizado.

5.3. Plugin Editor

En este archivo encontramos todos los componentes gráficos de la interfaz, al igual que referencias al procesador de audio y los métodos necesarios para crear y renderizar una interfaz gráfica con **JUCE**. Todas las implementaciones de los métodos se encuentran en el archivo *PluginEditor.cpp*.

En el constructor de esta clase, podemos ver cómo se configuran los componentes al igual de cómo se asocian a los parámetros de la otra hebra. También se ve cómo se configuran funciones para los botones y se añade un *LookAndFeel*, que se utilizará para ajustar el diseño de la aplicación. También se ve cómo se crea la ventana con un tamaño determinado de píxeles. En el destructor, se destruye la referencia al *LookAndFeel* de los componentes.

La función de renderizado que se utilizará para crear la aplicación será la de **resized()**, dejando **paint()** como una función de borrado de pantalla. Esto mejora el rendimiento del programa ya que solo volverá a renderizar ciertos elementos que necesiten ser renderizados, en vez de continuamente. En **resized()**, es donde se asignará a cada elemento su correspondiente área de renderizado, permitiendo que se puedan pintar en pantalla. En la figura 5.4 se puede observar una sección del método **resized()**, donde se puede observar como se asignan las secciones de interfaz gráfica a cada elemento.

```
void GranularSamplerAudioProcessorEditor::resized()
{
    // This is generally where you'll want to lay out the positions of any
    // subcomponents in your editor..

    auto bounds = getLocalBounds();

    auto boundsEQ = bounds.removeFromRight(bounds.getWidth() * 0.5f);

    eqResized(boundsEQ);
    //eq.resized(boundsEQ);

    auto fileButtonArea = bounds.removeFromTop(bounds.getHeight() * 0.333f);

    auto fileRendererArea = fileButtonArea.removeFromBottom(fileButtonArea.getHeight() * 0.75f);

    fileRenderer.setBounds(fileRendererArea);

    auto buttonsArea = fileButtonArea.removeFromRight(fileButtonArea.getWidth() * 0.666f);

    openFileButton.setBounds(fileButtonArea);

    auto playerButtonsArea = buttonsArea.removeFromTop(buttonsArea.getHeight() * 0.5f);

    auto stopPlayerButtonArea = playerButtonsArea.removeFromRight(playerButtonsArea.getWidth() * 0.5f);

    playPlayerButton.setBounds(playerButtonsArea);
    stopPlayerButton.setBounds(stopPlayerButtonArea);

    auto stopSamplerButtonArea = buttonsArea.removeFromRight(buttonsArea.getWidth() * 0.5f);

    playSamplerButton.setBounds(buttonsArea);
    stopSamplerButton.setBounds(stopSamplerButtonArea);

    //auto infoBar = bounds.removeFromTop(25);
    auto rightArea = bounds.removeFromRight(bounds.getWidth() * 0.5f);

    // left side
    auto densityBoundsSlider = bounds.removeFromTop(bounds.getHeight() * 0.5f);
    grainDensityLabel.setBounds(densityBoundsSlider.removeFromTop(25));
    grainDensitySlider.setBounds(densityBoundsSlider);

    // lower-upper
    auto lengthSliders = bounds.removeFromTop(bounds.getHeight() * 0.5f);
    auto titleDurationArea = lengthSliders.removeFromTop(25);
    grainDurationLabel.setBounds(titleDurationArea);

    auto minLengthSlider = lengthSliders.removeFromLeft(lengthSliders.getWidth() * 0.5f);
    grainMinLengthSlider.setBounds(minLengthSlider);
    grainMaxLengthSlider.setBounds(lengthSliders);
```

Figura 5.4: Captura del método **resized()**

Por último, el resto de métodos de la clase se diseñan para organizar mejor el código y para añadir funcionalidades como la activación y desactivación del ecualizador, la carga de archivos a través de un botón, la reproducción de un botón u otro, o el cambio de parámetros modificables al cambiar el tipo de envolvente utilizada.

Como observación, se encontraron muchos problemas durante la creación de nuevas clases para ayudar con la organización en el editor. Esto, sospecho, se debe a la necesidad de tener acceso a todos los componentes gráficos del editor juntos. En el caso de separarlos intentando agrupar todos los elementos de ecualizador en una única clase, generaba errores.

El editor depende de las siguientes clases:

- **PluginProcessor.h** Que como ya se ha explicado antes, es donde se encuentra el procesador de audio.
- **CommonEditor.h** Donde se encuentran elementos básicos de la hebra del editor, como el *LookAndFeel* utilizado, definiciones de abreviaturas utilizadas en el código y *enums* y *structs* utilizados.
- **Analyzer.h** Encargado de renderizar el analizador de frecuencias del ecualizador. Este funciona gracias a un algoritmo de Tranformaciones de Fourier (*Fast Fourier Transform*) que permite reconvertir las frecuencias en una única línea. Uno de los inconvenientes del método que he utilizado es que suele perder calidad en frecuencias altas o bajas. En mi caso, he preferido reducir la calidad en las frecuencias graves.

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones y valoración del proyecto

Como conclusión principal de este proyecto, el trabajo ha resultado en la creación de un instrumento virtual, completamente funcional, y se han completado todos los objetivos que se propusieron en un inicio. Este es funcional como aplicación independiente, *standalone* y como formato **VST**, que utilizan otros programas de composición musical como los **DAW**.

Además de la síntesis granular, el instrumento cuenta con un ecualizador de tres bandas en el proyecto, con un filtro de paso bajo y alto, al igual que un filtro de banda. Estas funcionalidades eliminan cualquier saturación que pueda ocasionar la suma de los granos al reproducir más de uno al mismo tiempo.

El proyecto cuenta también con sistemas visuales en tiempo real que permiten analizar tanto el archivo de audio de entrada, como las frecuencias que se reproducen como resultado, mejorando notablemente el control y entendimiento del resultado final.

El diseño de la implementación está pensado para poder expandirse con facilidad, permitiendo aumentar el tamaño de la interfaz gráfica y añadiendo funciones. Estas posibles mejoras o cambios se encuentran en el punto inferior.

La realización del instrumento ha sido posible gracias al uso de **JUCE**, una librería para el lenguaje de C++ que me ha aportado herramientas para el procesado, renderizado y testeo del proyecto. Además de que ofrece una gran cantidad de documentación, tutoriales y una comunidad activa al ser una tecnología utilizada en el mundo profesional.

Desde el punto de vista del desarrollo del proyecto, el haber realizado todo el trabajo solo ha supuesto una carga de trabajo superior a la que estimé. Esto supuso que para terminar el proyecto tuve que posponer la entrega. Aún así, pude mitigar gran parte del esfuerzo utilizando metodologías ágiles de trabajo que llevo toda la carrera utilizando. Otros problemas que he tenido durante el desarrollo fueron problemas con las implementaciones por defecto de **JUCE** para implementar la lectura de archivos de sonido, que me forzaron a rehacer mi propio reproductor. Surgieron otros problemas con la memoria dinámica que reservaban los granos. Para solucionarlo se eliminó la copia interna del archivo del grano y se tuvo que añadir un sistema que aplicaba la envolvente en función del momento actual en el que estuviera en su reproducción y del tipo que se utilice.

Personalmente, este trabajo ha sido muy interesante y enriquecedor, y me ha permitido tener una primera experiencia con una rama de la informática que desde que la conozco siempre he querido explorar y con un modelo de síntesis del que tampoco tenía mucha experiencia. El resultado final ha sido más que satisfactorio y recomiendo a cualquiera interesado en utilizar mi aplicación.

6.2. Trabajo Futuro

Aunque el trabajo ha quedado completo según mi visión original, al implementarlo seguí un diseño modular pensando en la posibilidad de que pudiera expandirse en el futuro. Por el carácter del proyecto, hay muchas formas en las que se puede seguir desarrollando. Estas son algunas de las posibles expansiones o mejoras que se podrían implementar:

- **Mejorar la interfaz gráfica**: Al haberme centrado en el desarrollo de la aplicación a un nivel técnico, me hubiera gustado haber dedicado más tiempo al diseño de la interfaz gráfica, y haber conseguido gente con conocimiento del tema.
- **Optimizar el rendimiento del código**: Para ser más concreto, en ordenadores más lentos el rendimiento del programa sufre bastante al reproducir una gran cantidad de granos, en especial si son cortos. Para ello se podría mejorar la gestión de granos en funcionamiento.
- **Aumentar el número de parámetros de la granulación**: A mayor número de parámetros, más libertad tendría el usuario de modificar el sonido resultante. Un ejemplo podría ser la modificación del *pitch*, o timbre, de los granos que se reproducen, la modificación del *panning*, es decir, de poder reproducir los granos más por el canal izquierdo o derecho de sonido o aumentar la libertad del origen de los granos.
- **Mejorar la implementación del algoritmo de Power-Cross-Fade:** Este se utiliza para conseguir un volumen cercano al sonido del archivo al realizar la suma de granos. En este proyecto solo se pudo hacer una investigación inicial del problema llegando a un resultado satisfactorio, pero no óptimo y se debería poder investigar la posibilidad de aplicar este algoritmo para un número n de sonidos de manera matemáticamente precisa, y no la aproximación utilizada.
- **Añadir la posibilidad de importar más de un archivo como origen**: Esto permitiría mezclar las diferentes texturas de varios archivos, volviendo más complejo e interesante el sonido resultante.
- **Añadir un compresor**: Este módulo ayudaría a mitigar los archivos de audio con ataques muy marcados, permitiendo igualar el sonido de los granos entre sí y conseguir un efecto más monótono de volumen en el resultado.
- **Mejorar el analizador de frecuencias**: Este funciona con un algoritmo de **FFT** que aún funcional, tiene muy baja resolución, especialmente en las frecuencias graves. Como solución, se podrían buscar algoritmos más sofisticados para su implementación.
- **Añadir entrada MIDI para la modificación del *pitch***: Esta herramienta cambiaría el funcionamiento del instrumento de un sampler a algo más parecido a un sintetizador, permitiendo controlar las frecuencias de los granos, pero respetando sus texturas.

Capítulo 6

Conclusions and future work

6.1. Conclusions and work evaluation

As the main conclusion from this project, the final result from this project is the creation of a fully functional virtual instrument. All the objectives that were initially proposed have been completed. It is functional as a stand-alone application, and as a **VST**, which is used by other music composition programs such as **DAW**.

In addition to granular synthesis, the instrument has a built-in three-band equalizer, with a low and high pass filter, as well as a band filter. These functionalities eliminate any saturation that can be caused by the sum of the grains when playing more than one at the same time.

The project also has real-time visual systems that allow analyzing both the input audio file and the frequencies that are reproduced as a result, notably improving the control and understanding of the final result.

The implementation design is intended to be easily expandable, allowing you to increase the size of the graphical interface and add functions. These possible improvements or changes will be explained in the next point.

The realization of the instrument has been possible thanks to the use of **JUCE**, a library for the C++ programming language that has provided me with tools for the processing, rendering and testing of the project. In addition to offering a large amount of documentation, tutorials and an active community as it is a technology used for professional and casual development.

From the point of view of the development of the project, As I worked alone, the amount of work I had to accomplish was greater than the one I originally estimated. This meant that in order to finish the project I had to postpone the delivery. Even so, I was able to mitigate much of the effort using agile work methodologies that I have been using all my career. Another problem I've had during development were with the default implementations of **JUCE** to implement reading sound files, which forced me to remake my own player. Other problems arose with the dynamic memory reserved by the grains. To solve this, the internal copy of the grain file was eliminated and a system had to be added that applied the envelope depending on the current moment in which it was in its reproduction and the type used.

Personally, this work has been very interesting and enriching, and it has allowed me to have a first experience with a branch of computer science that I have always wanted to explore ever since I learned about it. I also got to learn a lot from a synthesis model that I did not have much experience with. The final result has been more than satisfactory and I recommend anyone interested in my instrument to experiment and play with it.

6.2. Future work

Although the work has been completed according to my original vision, I followed a modular design considering the possibility that it could be expanded in the future. Due to the nature of the project, there are many ways in which it can be further developed. These are some of the possible expansions or improvements that could be implemented:

- **Improving the graphical interface**: Having focused on the development of the application at a technical level, I would have liked to have spent more time designing the graphical interface, and to have gotten people with knowledge of the subject helping with the design.
- **Optimize the performance of the code**: To be more specific, on slower computers the performance of the program suffers quite a bit when playing a large number of grains, especially if they are short. For this, the management of grains playing could be improved.
- **Increase the number of grain parameters**: The greater the number of parameters, the more freedom the user would have in modifying the resulting sound. An example could be the modification of the *pitch* or frequencies of the grains that are reproduced, the modification of the *panning*, that is the control to reproduce the amount of grains playing on the left or right channel or increase the control of the origin of each grain.
- **Improve the implementation of the Power-Cross-Fade algorithm:** This is used to get a volume close to the sound of the file when performing grain addition. In this project, it was only possible to carry out an initial investigation of the problem, reaching a satisfactory, but not optimal, result, and it should be possible to investigate the possibility of applying this algorithm to a number N of sounds in a mathematically precise manner, and not the approximation used.
- **Add the possibility to import more than one file as a source**: This would allow mixing the different textures of several files, making the resulting sound more complex and interesting.
- **Add a compressor**: This module would help mitigate audio files with very strong attacks, allowing the sound of the grains to match each other and achieve an even more monotonous volume effect in the resulting sound.
- **Improve the frequency analyzer**: This works with a **FFT** algorithm that even if working, has very low resolution, especially in the lower frequencies. As a solution, more sophisticated algorithms could be sought for its implementation, or combining multiple **FFT** passes into one.
- **Add MIDI input for *pitch* modification**: This tool would change the behavior of the instrument from a sampler to something more like a synthesizer, allowing you to control the frequencies of the grains, but respecting their textures. It would also need to add a overall envelope to control how a key note fades in and out.

Anexos

A - Manual de usuario - GranularSampler

Controles generales

El uso de este programa se controla principalmente mediante un ratón informático. Este se utiliza para pulsar con el botón izquierdo de este los distintos botones y controladores del programa. Los controladores pueden ser modificados mediante la rueda del ratón, o arrastrando hacia arriba y hacia abajo al pulsar sobre el controlador.

Como observación importante, se puede ajustar con mayor precisión el valor del dial, si al modificarlo se mantiene pulsado el botón control del teclado.

Secciones del instrumento

Las secciones del instrumento son: en la parte superior de la izquierda se encuentra el *sampler*; donde se carga el archivo de audio que utilizará la síntesis granular y donde se controla la reproducción del instrumento, la parte inferior de la izquierda; donde se encuentran los controladores de la síntesis granular, y a la derecha el ecualizador, donde se divide arriba en el analizador de frecuencias; y en la parte inferior los controles de los tres filtros; el de pasa baja, alta y el filtro de banda.

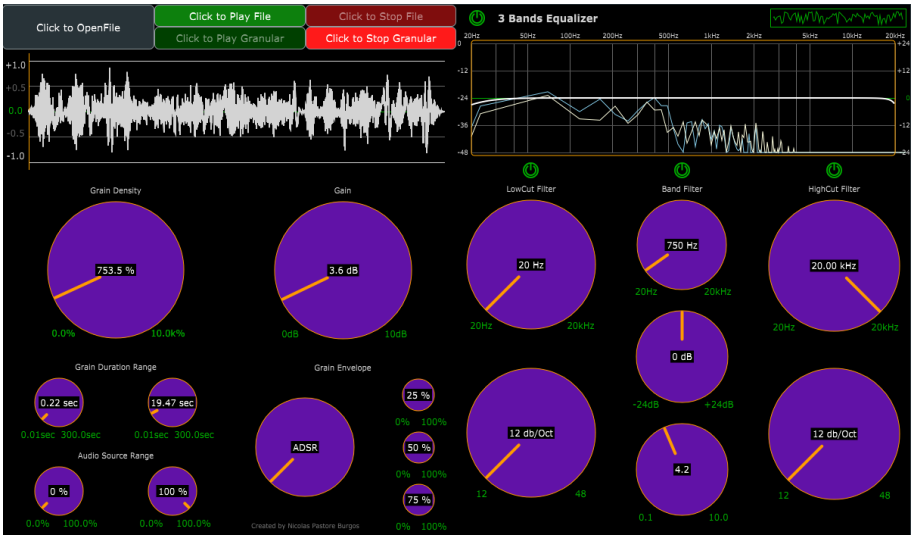


Figura A.1: El instrument funcionando

Carga de un archivo y reproducción

Para cargar un archivo en el instrumento, se debe pulsar el botón superior de la izquierda gris. Este lanzará una ventana del sistema operativo que te permitirá navegar por el dispositivo hasta encontrar el archivo deseado. Una vez seleccionado y añadido, si el instrumento ha cargado correctamente el archivo deberán iluminarse los botones verde de reproducción del archivo y del modelo granular, al igual que se debe representar el archivo de audio debajo de los botones.

Como última comprobación, se puede reproducir el archivo de audio pulsando el botón verde superior. Si se quiere parar en cualquier momento se debe pulsar el botón rojo iluminado de la derecha. Este solo se iluminará si se está reproduciendo.

Si se quiere comenzar a granular el archivo, se deberá pulsar el botón verde inferior. Este funcionará igual que el que se utiliza para reproducir el archivo.

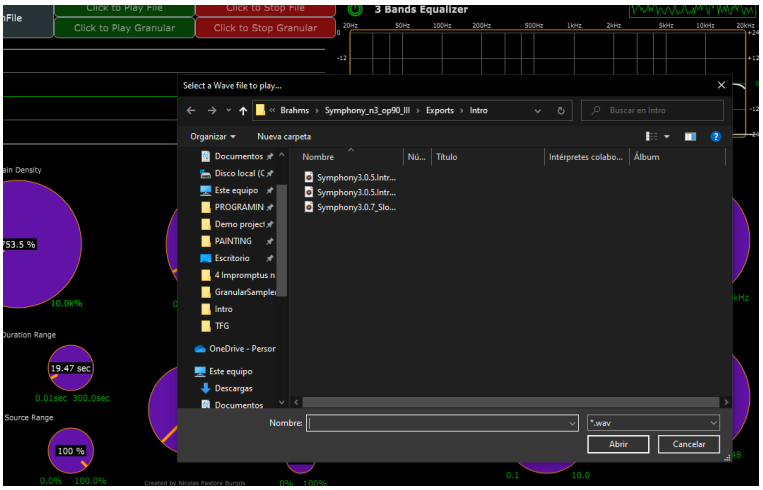
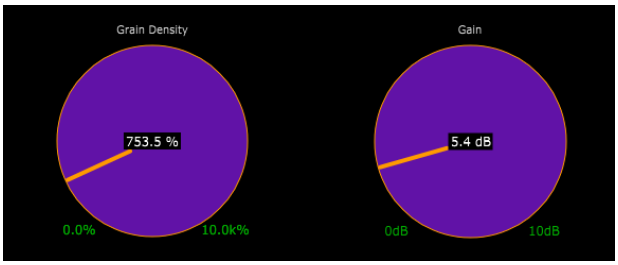


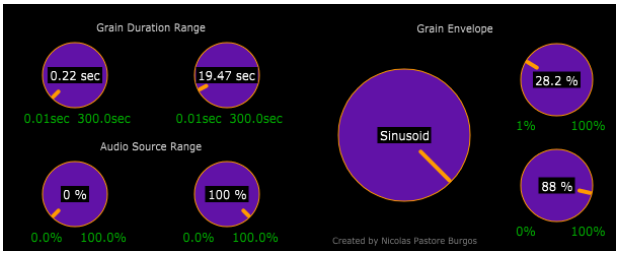
Figura A.2: El selector de archivos de Windows

Control de la granulación

Para controlar los granos que se reproducen, se ha de modificar los diales inferiores al *sampler*. En orden de izquierda a derecha y arriba a abajo, los controladores son: el de la densidad de granos al segundo, en porcentaje (es decir que el 100 % es equivalente a un grano sonando en el momento), el de la ganancia de los granos, los controladores del rango de duración de los granos y rango del archivo de donde se pueden extraer los granos, y un controlador de la envolvente, que permite cambiar y el tipo y controlar los parámetros de cada uno al modificarlo. De las distintas envolventes que hay, el **ADSR** tiene tres controles. Estos se aplican uno detrás del otro, así que empezando desde el cero hasta el valor del primer controlador es el ataque, del primero al segundo el *decay*, del segundo al tercero el *sustain* y del tercero al final el *release*. Las otras dos envolventes tienen dos controladores que definen el *fade in* y *fade out* de la envolvente.

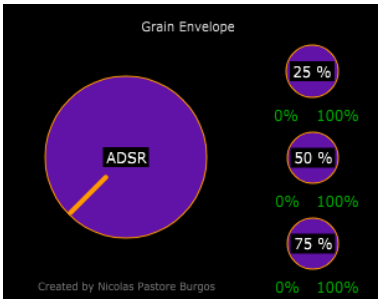


(a) Sección superior del módulo

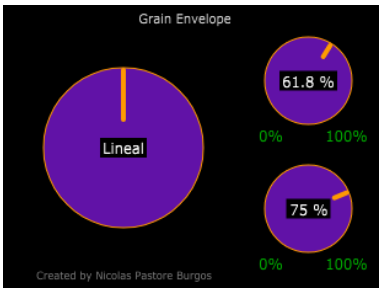


(b) Sección inferior del módulo

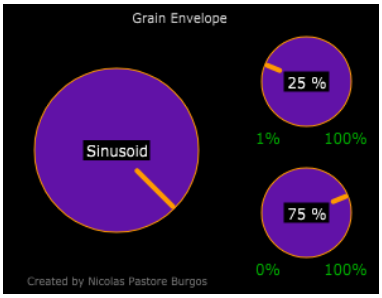
Figura A.3: Secciones de los controladores de la síntesis modular



(a) Envolvente **ADSR**



(b) Envolvente lineal



(c) Envolvente sinusoidal

Figura A.4: Distintos tipos de envolventes de los granos

Control del ecualizador

El ecualizador cuenta con tres filtros diferentes, uno de paso bajo y alto, y un tercero de banda. Estos tres se controlan con los diales de la parte inferior, siendo la primera columna del filtro de paso alto, la segunda el filtro de banda y la tercera de paso bajo. Los dos filtros de corte cuentan con un controlador de la frecuencia y uno para ajustar la fuerza con la que se aplica. El de banda tiene tres filtros, que controlan la frecuencia, la ganancia y un tercer controlador que ajusta cuanto afecta a las frecuencias vecinas. Cada filtro cuenta con un botón para desactivar este filtro, y uno general en la esquina superior izquierda para desactivarlo por completo. Además tiene un analizador de frecuencias en la parte superior, que representa en azul el canal izquierdo y en naranja el canal derecho, una linea blanca que representa el efecto del filtro sobre las frecuencias y un botón en la esquina superior derecha para desactivarlo.

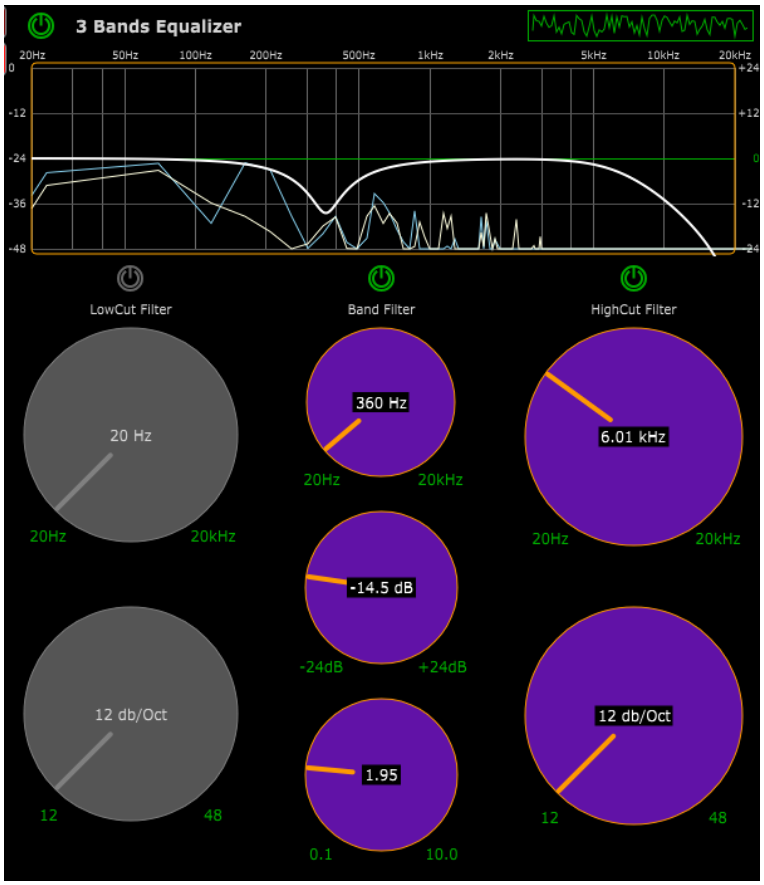


Figura A.5: El ecualizador de tres bandas

Índice de figuras

Bibliografía

[1] Referencia de la librería JUCE: https://juce.com/ [Último acceso: 14/07/2022]

[2] Enlace de Spotify de El Corral de los Quietos: https://open.spotify.com/artist/6VjfONjz3MbMnFh6dBclVo [Último acceso: 14/07/2022]

[3] Entrada sobre los **DAW** de Wikipedia: https://en.wikipedia.org/wiki/Digital_audio_workstation [Último acceso: 14/07/2022]

[4] Entrada sobre los **VST** de Wikipedia: https://en.wikipedia.org/wiki/Virtual_Studio_Technology [Último acceso: 14/07/2022]

[5] Entrada de Steam de la empresa Mad Cream Games: https://store.steampowered.com/developer/MadCreamGames/?utm_source=tfg [Último acceso: 14/07/2022]

[6] Enlace de Steam del videojuego Painting Werther: https://store.steampowered.com/app/1434260/Painting_Werther [Último acceso: 14/07/2022]

[7] Entrada sobre Iannis Xenakis de Wikipedia: https://es.wikipedia.org/wiki/Iannis_Xenakis [Último acceso: 14/07/2022]

[8] Página web de Curtis Roads: https://www.curtisroads.net/ [Último acceso: 14/07/2022]

[9] Roads, Curtis (2001). Microsound. Cambridge: MIT Press. ISBN 0-262-18215-7.

[10] Enlace de Cecilia, el instrumento virtual: http://ajaxsoundstudio.com/software/cecilia/ [Último acceso: 14/07/2022]

[11] Enlace del plugin de ableton de granulación: https://www.ableton.com/en/packs/granulator-ii/ [Último acceso: 14/07/2022]

[12] Enlace del instrumento virtual GranuRise: https://granurise.com/ [Último acceso: 14/07/2022]

[13] Enlace del instrumento virtual Quanta: https://www.audiodamage.com/products/ad046-quanta [Último acceso: 14/07/2022]

[14] Enlace a los distintos proyectos profesionales creados por JUCE: https://juce.com/discover/made-with-juce [Último acceso: 14/07/2022]

[15] Fuente de una de las imágenes explicativas de la granulación: https://ears2.dmu.ac.uk/es/courses/sonidos-generados/lessons/sintesis-granular [Último acceso: 14/07/2022]

[16] Crossan, A. & Williamson, J. & Murray-Smith, Roderick. (2004). Haptic granular synthesis: Targeting, visualisation and texturing. Proceedings of the International Conference on Information Visualization. 8. 527- 532. 10.1109/IV.2004.1320195.

[17] Fuente de la imágen utilizada para explicar el SCRUM: https://www.plainconcepts.com/es/scrum-que-es/ [Último acceso: 14/07/2022]

[18] Fuente de la imágen de ejemplo de un sintetizador granular: https://www.hispasonic.com/anuncios/sintetizador-modular-eurorack-completo-tambien-cambio/1039582 [Último acceso: 14/07/2022]

[19] Enlace al módulo de Python, PYO, para el procesado de señales: http://ajaxsoundstudio.com/software/pyo/ [Último acceso: 14/07/2022]

[20] Enlace al módulo de Python, PyAudio, para la gestión E/S de audio: https://pypi.org/project/PyAudio/ [Último acceso: 14/07/2022]

[21] Enlace a los tutoriales de JUCE: https://juce.com/learn/tutorials [Último acceso: 14/07/2022]

[22] Enlace a la compañía que creo algunos de los tutoriales que seguí para familiarizarme con JUCE: https://www.freecodecamp.org/ [Último acceso: 14/07/2022]

[23] Enlace a YouTube de uno de los videos de Free Code Camp: https://www.youtube.com/watch?v=i_Iq4_Kd7Rc [Último acceso: 14/07/2022]

[24] Enlace a YouTube de uno de los videos de Free Code Camp: https://www.youtube.com/watch?v=Mo0Oco3Vimo [Último acceso: 14/07/2022]

[25] Enlace a la API de JUCE: https://docs.juce.com/master/index.html [Último acceso: 14/07/2022]

[26] Enlace a la documentación del lenguaje de programación de C++: https://cplusplus.com/reference/ [Último acceso: 14/07/2022]

[27] Enlace a la página web del IDE Visual Studio: https://visualstudio.microsoft.com/es/[Último acceso: 14/07/2022]

[28] Enlace al repositorio de GitHub del proyecto: https://github.com/NicoPast/GranularSampler-TFG [Último acceso: 14/07/2022]

[29] Enlace al tutorial de uso del módulo de DSP de JUCE: https://docs.juce.com/master/tutorial_dsp_introduction.html [Último acceso: 14/07/2022]

[30] Entrada de las leyes de la Gestalt de Wikipedia: https://es.wikipedia.org/wiki/Psicología_de_la_Gestalt [Último acceso: 14/07/2022]

[31] Enlace al trello del proyecto: https://trello.com/b/Qz8n9nCY/granularsampler [Último acceso: 14/07/2022]

[32] Entrada sobre el Aliasing de Wikipedia: https://es.wikipedia.org/wiki/Aliasing [Último acceso: 14/07/2022]