



Escuela
Politécnica
Superior

Aplicaciones para el entrenamiento auditivo psicoacústico



Grado en Ingeniería en Sonido e Imagen
en Telecomunicación

Trabajo Fin de Grado

Autor:

Fernando Ayelo Sánchez

Tutor/es:

Pedro José Ponce de León Amador

Junio 2017



Universitat d'Alacant
Universidad de Alicante

Dedicado a mi familia por el esfuerzo empleado para llegar hasta aquí.

Agradecimientos

El presente proyecto no hubiera sido posible sin la inestimable colaboración y apoyo de una serie de personas a las que me gustaría mencionar y agradecer.

En primer lugar agradecer a mi tutor, Pedro José Ponce de León Amador, el cual creyó en mi propuesta desde el primer momento, animándome a su desarrollo y aconsejándome en cada problema o limitación que me encontraba.

En segundo lugar, y como no, a mi padre, mi madre y mi hermano, por su apoyo incondicional a lo largo de estos cuatro años de grado. Gracias por todo el esfuerzo empleado a pesar de los problemas para que hoy pueda escribir éstas líneas.

Por último, agradecer a mis amigos de clase, pilar fundamental para haber llegado hasta aquí. Gracias Manuel. Gracias Yolanda. Gracias Mónica. Han sido cuatro años maravillosos para mí, por lo que os agradezco todo lo que me habéis aportado y os deseo una vida llena de oportunidades y alegrías.

Muchas gracias.

Resumen

En el presente proyecto se va a desarrollar una *suite* de aplicaciones con interfaz gráfica para el entrenamiento auditivo. Las aplicaciones pretenden estar destinadas principalmente a profesionales del sonido (productores, diseñadores sonoros, técnicos e ingenieros de sonido), si bien podrán ser utilizadas por cualquier usuario medio con un nivel de informática básico.

El entrenamiento auditivo psicoacústico busca educar y afinar el oído para que éste sea capaz de identificar y caracterizar las diferentes cualidades y parámetros del material sonoro, lo cual se considera requisito indispensable para los profesionales del audio.

Nuestro grupo de aplicaciones buscará tratar el sonido desde sus diferentes parámetros, como son la localización espacial, detección frecuencial, el ruido y la distorsión y por último la ganancia y la dinámica. Para ello, se hará uso de *Csound* como lenguaje de programación, apoyándonos en *CsoundQt*. Cada una de las aplicaciones contará además con una interfaz gráfica desarrollada a partir de *CsoundQt*, una herramienta *front-end* de desarrollo.

Cada una de las aplicaciones contará con una sección de prueba, en la que el usuario podrá experimentar con todos los parámetros de dicha sección, buscando así la comprensión de los mismos. Por otro lado, cada aplicación también contará con una sección de entrenamiento, donde realmente se pondrá a prueba al usuario en busca de una mejora auditiva psicoacústica.

Palabras clave: Csound, CsoundQt, Cabbage, Síntesis, Entrenamiento, Acústico, Psicoacústica

Abstract

In this project, a suite of applications with graphic interface for the auditive training will be developed. Those applications are ment to be used by sound proffessionals mainly (producers, sound designers and sound engineers), but it could be used by any user with a basic computing level.

The psychoacoustic ear training seeks for the education and tuning of the ear, in order to make it capable of identifying and characterizing the different qualities and parameters of the sound material which is an essential requirement for the sound proffessionals.

Our group of applications looks for proccessing the sound from its different parameters, as the spatial location, frequency detection, the noise and the distortion and lastly the gain and the dinamics are. To achieve this, *Csound* will be the programming language we will use, supported by *CsoundQt*. Every single application will also have a graphic interface designed on *CsoundQt* which is a front-end design tool.

Each application will include a testing section, where the user may be able to experience every parameter of the section. This feature improves the comprehension of those parameters. Furthermore, every application will have a training part, where the user will try out its skills, improving the psychoacoustic auditive.

Keywords: Csound, CsoundQt, Cabbage, Synthesis, Acoustic, Training, Psychoacoustics

Índice general

Índice general	VII
Índice de figuras	XI
1 Introducción	1
1.1 Motivación	1
1.2 Objetivos	1
1.3 Organización	2
2 Marco teórico	3
2.1 Psicofísica y psicoacústica	3
2.1.1 Fisiología humana	3
2.1.1.1 El sistema auditivo periférico	4
2.1.1.2 El sistema auditivo central	6
2.1.2 Umbrales de audición	6
2.1.2.1 El umbral absoluto	7
2.1.2.2 El umbral diferencial	7
2.1.3 Métodos de medida	7
2.1.4 Bandas críticas	9
2.1.5 Sonoridad	9
2.1.6 Altura	10
2.1.7 Duración	11
2.1.8 Timbre	11
2.1.9 Localización espacial	12
2.2 Fundamentos del audio digital	13
2.2.1 Conversión AD/DA	13
2.2.1.1 Muestreo	13
2.2.1.2 Cuantificación	14
2.2.1.3 Codificación	14
2.3 Procesado y efectos de audio	14
2.3.1 Frecuenciales	15
2.3.1.1 Ecualizadores	15
2.3.1.2 Distorsión (<i>Overdrive</i>)	15
2.3.2 Dinámicos	15
2.3.2.1 Compresor	15
2.3.2.2 Puerta de ruido	15
2.3.3 Temporales	15
2.3.3.1 Eco	16
2.3.3.2 <i>Reverb</i>	16
2.3.3.3 <i>Delay</i>	16
2.4 Obras y apps de entrenamiento auditivo disponibles	16
2.4.1 <i>Golden Ears</i>	16
2.4.2 <i>SoundGym</i>	17
2.4.3 <i>Train Your Ears</i>	18

2.4.4	<i>iZotope Pro Audio Essentials</i>	19
2.4.5	<i>Quiztones</i>	20
2.4.6	<i>EarMaster</i>	21
2.5	Propuesta de este trabajo	21
2.6	Csound - Herramienta	22
2.6.1	<i>Csound</i>	22
2.6.1.1	Funcionamiento de <i>Csound</i>	22
2.6.2	<i>CsoundQt</i>	23
2.6.3	Otros <i>front-end</i>	25
3	Metodología	26
3.1	Visión general	26
3.2	Apartados didácticos de la aplicación	26
3.2.1	Reconocimiento espacial	26
3.2.2	Reconocimiento frecuencial	28
3.2.3	Ruido y distorsión	28
3.2.4	Volumen y dinámica	30
4	Diseño del sistema	31
4.1	Modos de trabajo	31
4.1.1	Modo de prueba	31
4.1.2	Modo de entrenamiento	32
4.1.3	Visión general de diseño	33
4.2	Interfaz gráfica	34
4.2.1	<i>Widgets</i>	34
4.2.1.1	<i>Presets</i>	35
4.2.2	Nuestro diseño	36
4.2.2.1	Estructura general	36
4.2.2.2	Gama de colores	37
4.3	Evaluación de los ejercicios	37
4.3.1	Auto evaluación	38
5	Implementación	40
5.1	Visión general	40
5.2	Reconocimiento espacial	42
5.2.1	Zona de pruebas	42
5.2.1.1	Main	43
5.2.1.2	Reverb	43
5.2.1.3	Situación espacial	44
5.2.2	Zona de prácticas	44
5.2.2.1	Main	44
5.2.2.2	Situación espacial	45
5.2.2.3	Reverb	45
5.2.2.4	Visualización	45
5.2.2.5	Resultados	46
5.3	Reconocimiento frecuencial	46
5.3.1	Zona de pruebas	46
5.3.1.1	Muestra 1	47
5.3.1.2	Muestra 2	48
5.3.2	Zona de prácticas	48
5.3.2.1	Generar	48
5.3.2.2	Muestra 1 y Muestra 2	48
5.3.2.3	Respuestas - Muestra 1 y Muestra 2	49
5.3.2.4	Resultados	49

5.4	Ruido y Distorsión	50
5.4.1	Zona de pruebas	50
5.4.1.1	Main	50
5.4.1.2	Ruido	51
5.4.1.3	Cuantización	51
5.4.1.4	Muestreo	52
5.4.2	Zona de prácticas	53
5.4.2.1	Main	53
5.4.2.2	Dificultad	53
5.4.2.3	¿Qué degradación hay?	54
5.4.2.4	Respuestas - Ruido, Cuantización y Muestreo	54
5.4.2.5	Resultados	55
5.5	Ganancia y Dinámica	55
5.5.1	Zona de pruebas	56
5.5.1.1	Main	56
5.5.1.2	Ganancia	56
5.5.1.3	Dinámica	57
5.5.2	Zona de prácticas	58
5.5.2.1	Main	58
5.5.2.2	Ganancia	58
5.5.2.3	Dinámica	59
5.5.2.4	Resultados	60
6	Conclusiones y trabajos futuros	61
6.1	Conclusiones	61
6.2	Propuestas de trabajo futuras	61
	Bibliografía	63
	Apéndices	
A	Código fuente - Reconocimiento espacial	65
B	Código fuente - Reconocimiento frecuencial	73
C	Código fuente - Ruido y distorsión	84
D	Código fuente - Ganancia y dinámica	101

Índice de figuras

2.1	Representación gráfica del sistema auditivo periférico.[3]	4
2.2	Esquema del sistema auditivo periférico con la cóclea desenrollada.[3]	5
2.3	Corte de la cóclea.[3]	6
2.4	Distribución de las funciones a lo largo de los 2 hemisferios cerebrales para individuos con lengua materna occidental.[4]	7
2.5	Gráfico con los umbrales de audición humanos.[7]	8
2.6	Diagrama de explicación para las bandas críticas.[12]	9
2.7	Curvas isofónicas o de sonoridad.[13]	10
2.8	Sonido agudo (izquierda); Sonido grave (derecha)	11
2.9	Sonido largo y sin extinción (izquierda); Sonido corto y con un decaimiento rápido (derecha)	11
2.10	Distintos timbres para una misma altura y duración. Se observa la composición armónica de cada uno en su espectro.[16]	12
2.11	Análisis espectral de un sonido instrumental percusivo de "madera"(izquierda); Otro análisis espectral de un sonido simple (senoidal) estacionario(derecha).[17]	12
2.12	Diagrama para explicar la escucha binaural.[15]	12
2.13	Logo <i>SoundGym</i>	17
2.14	Logo <i>Train Your Ears</i>	18
2.15	Interfaz gráfica <i>Train Your Ears EQ Edition</i>	18
2.16	Logo <i>iZotope</i>	19
2.17	Logo <i>Pro Audio Essentials</i>	20
2.18	Logo <i>Quiztones</i>	20
2.19	Logo <i>EarMaster</i>	21
2.20	Interfas gráfica <i>EarMaster</i>	21
2.21	Logo <i>CsoundQt</i>	23
2.22	Interfaz gráfica <i>CsoundQt</i>	24
3.1	Ejemplo de colocación espacial de una banda estándar.[19]	27
3.2	Instrumentos con su colocación frecuencial a lo largo del espectro audible.	29
3.3	Ejemplo de una señal sin ruido blanco agregado, con ruido leve (10 dB SNR) y con ruido intenso (20dB SNR).	29
4.1	Ventana del "Modo de prueba" de la aplicación de "Reconocimiento frecuencial".	32
4.2	Ventana del "Modo de entrenamiento" de la aplicación de "Reconocimiento frecuencial".	33
4.3	Vista general de la aplicación de "Reconocimiento frecuencial".	34
4.4	Esquema de la estructura general asociada a las 4 aplicaciones.	36
4.5	Ejemplo de diseño final, con la composición de colores y tipografías definitivas.	37
4.6	Esquema del flujo de trabajo mediante auto evaluación.	38
5.1	Esquema del flujo de funcionamiento dentro de la "Zona de prácticas".	42
5.2	Vista general de la aplicación "Reconocimiento espacial".	43

5.3	Vista general de la aplicación “Reconocimiento frecuencial”.	47
5.4	<i>Displays</i> encargados de mostrar la onda elegida por el usuario.	47
5.5	Sección de control de ambas muestras para la “Zona de prácticas”.	49
5.6	Sección de respuestas de ambas muestras para la “Zona de prácticas”.	49
5.7	Vista general de la aplicación “Ruido y Distorsión”.	50
5.8	Vista general de la sección “Ruido”.	51
5.9	Vista general de la sección “Cuantización”.	52
5.10	Vista general de la sección “Cuantización”.	53
5.11	Vista general de la sección “¿Qué degradación hay?”.	54
5.12	Vista general de la sección de respuestas.	55
5.13	Vista general de la aplicación “Ganancia y Dinámica”.	56
5.14	Vista general de la sección “Ganancia”.	57
5.15	Vista general de la sección “Dinámica”.	57
5.16	Vista general de la sección “Ganancia”.	59
5.17	Vista general de la sección “Dinámica”.	60

CAPÍTULO 1

Introducción

1.1 Motivación

Desde el comienzo de la música, uno de los grandes requisitos para el correcto desarrollo de la misma ha sido, entre otras muchas facultades, la capacidad del intérprete en la escucha, necesitando por tanto de un sentido del oído afinado y bien desarrollado.

Paralelamente al desarrollo de la música, y a comienzos del siglo XIX, se hizo cada vez más importante tener la capacidad de poder grabarla y reproducirla de forma que pudiese perdurar en el tiempo y ser escuchada por una gran masa de gente. En éste punto nació la figura del técnico de sonido o ingeniero sonoro. Es fácil imaginar que dicho técnico debía compartir con los músicos su capacidad auditiva para el correcto desarrollo en la captación y mezcla musical.

El siguiente paso en nuestra historia se sitúa en 1965 en el Shea Stadium de Nueva York. En esa fecha y lugar se llevó a cabo a mano de los todopoderosos Beatles el primer concierto multitudinario, y con ello nació la industria de la música en directo. Es fácil pensar que dicha industria aumentó la demanda de técnicos e ingenieros de sonido de manera exponencial, aumentando así el reclamo de cada vez más profesionales capacitados con las cualidades anteriormente mencionadas.

Llegados a este punto se puede deducir de manera sencilla y lógica el potencial que posee ésta industria, con un fuerte reclamo en profesionales cualificados para hacer frente a la gran demanda tanto de ingenieros y técnicos de estudio, como de música en directo, teatros, radios, televisiones, etc.

1.2 Objetivos

El objetivo del proyecto es combinar la experiencia profesional desarrollada como técnico de sonido durante los últimos 3 años con los conocimientos adquiridos estos últimos años en la universidad, para desarrollar un software de entrenamiento auditivo mediante síntesis digital de sonido enfocado a técnicos e ingenieros de sonido principalmente, pero con la intención de que pueda llegar a ser útil para cualquier persona con ganas de afinar y entrenar su audición.

Nuestro objetivo principal, y por tanto del proyecto, consiste en acabar con una serie de aplicaciones basadas en lenguaje *Csound* y con interfaz gráfica para el entrenamiento auditivo. Como objetivos específicos para completar el desarrollo del proyecto se establecen los siguientes puntos:

- Exploración de las características de *Csound* y sus interfaces gráficas que resulten idóneas para la implementación de la aplicación.
- Diseño del sistema de ejercicios de entrenamiento auditivo.

- Desarrollo de una *suite* de aplicaciones con interfaz gráfica para el entrenamiento auditivo psicoacústico basada en *Csound*.

1.3 Organización

La memoria de este proyecto se estructura en 6 secciones de contenido a parte de la parte introductoria y una de documentación anexa.

En primer lugar se hace un repaso sobre los fundamentos básicos de audio y la psicoacústica con la idea de relacionar los conceptos teóricos con los posteriores resultados prácticos, además de mostrar en base teórica las herramientas que se van a utilizar para el desarrollo de nuestra aplicación.

La sección 3 explica la metodología que se va a utilizar para el desarrollo de cada una de las aplicaciones, explicando los primeros pasos que se llevaron a cabo al comienzo del proyecto y una explicación teórica de cada una de las aplicaciones y el porqué se ha decidido llevarla a cabo.

En el capítulo 4 se hablará del diseño del sistema, empezando por los métodos de trabajo que se van a implementar en cada aplicación, así como de la interfaz gráfica, herramientas disponibles para su desarrollo y demás detalles relacionados. En el capítulo 5 se hablará de la implementación propia de cada una de las aplicaciones, enfocada desde un punto de vista técnico y comentando decisiones tomadas para la escritura de cada una de ellas, así como una explicación sobre el código y su funcionamiento.

En el capítulo 6 se pretende evaluar cada una de las aplicaciones, bien desde un punto de vista propio, como a través de profesionales del mundo del audio, recibiendo opiniones para posibles mejoras futuras y, en general, una evaluación externa de las mismas.

Para terminar, tenemos el capítulo 7, en el que se hablará de las conclusiones sobre el presente proyecto así como propuestas futuras relacionadas y que se nos ocurren ampliarán y mejorarán la funcionalidad de nuestras aplicaciones.

CAPÍTULO 2

Marco teórico

2.1 Psicofísica y psicoacústica

La psicofísica se define como una parte de la rama de la psicología experimental dedicada a la investigación de la relación que existe entre la intensidad de un estímulo, sea cual fuere, y la calidad del mismo. Dicho de otra manera, su fin es estudiar la percepción y juicios que recibe y realiza el cerebro frente a un estímulo externo. Otra manera de enfocarlo es verlo como la relación del ser humano con su medio físico y los juicios que forma acerca de las percepciones que de él le vienen.

Por todo lo mencionado, y tal y como señalan reconocidos autores como *Day* (1969), los métodos aplicados por la psicofísica representan y son una forma de investigación precisa para poder cuantificar cuatro tipos básicos de comportamientos perceptivos: la detección, la discriminación, el reconocimiento y la estimación. Actualmente, la psicofísica tiene como objetivo estudiar los procesos que se encuentran entre lo físico y lo subjetivo, es decir, de lo físico a lo psíquico. [1]

La psicoacústica es una rama perteneciente a la psicofísica que estudia la relación existentes entre un estímulo acústico de carácter físico y su respuesta a nivel psicológico. Es decir, estudia la relación entre las propiedades físicas del sonido y la interpretación que hace de ellas el cerebro.[8]

Los objetivos generales de la psicoacústica pueden resumirse en saber determinar:

- La característica de respuesta de nuestro sistema auditivo, es decir, cómo se relaciona la magnitud de la sensación producida por el estímulo con la magnitud real del estímulo
- El umbral (absoluto) de la sensación
- El umbral diferencial de determinado parámetro del estímulo (mínima variación y mínima diferencia perceptible)
- La resolución o capacidad de resolución del sistema auditivo para separar estímulos simultáneos o la forma en la que dichos estímulos provocan una sensación compuesta
- La variación en el tiempo de la sensación del estímulo

2.1.1 Fisiología humana

La fisiología es la rama de la ciencia encargada del estudio de las funciones de los seres pluricelulares, es decir, de los seres vivos. La fisiología humana es, por ende, la ciencia que estudia las funciones vitales de los seres humanos. [2]

2.1.1.1 El sistema auditivo periférico

El sistema auditivo periférico (el oído), en la fig. 2.1, está compuesto por el oído externo, el oído medio y el oído interno.

El sistema auditivo periférico cumple las funciones de percepción del sonido, esencialmente de las variaciones de presión sonora que llegan al tímpano en impulsos eléctricos, además de desempeñar también una función importante en el sentido del equilibrio.

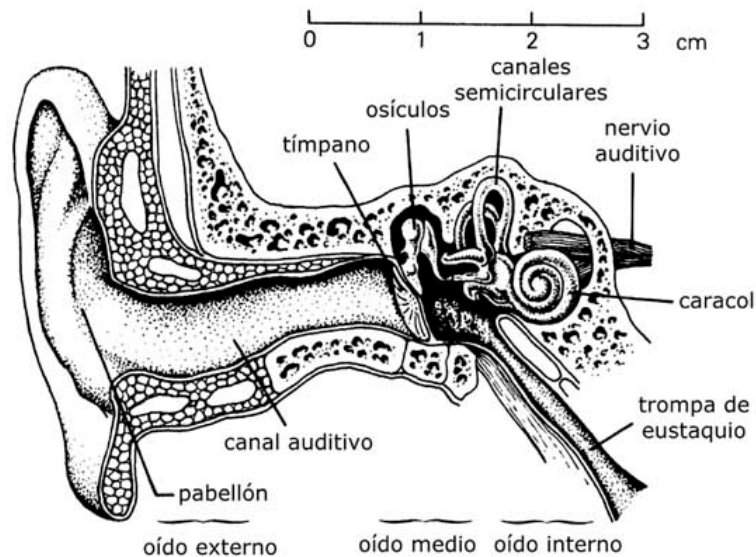


Figura 2.1: Representación gráfica del sistema auditivo periférico.[3]

A continuación se expone una descripción de cada una de las tres secciones en las que se divide el sistema auditivo periférico:

■ Oído externo

El oído externo se compone por el pabellón auditivo (la oreja), la cual se encarga de concentrar las ondas sonoras incidentes en el canal auditivo, el cual conduce las ondas hasta el tímpano.

La ubicación lateral de los pabellones derecho e izquierdo en el ser humano ha hecho prácticamente innecesaria la capacidad de movimiento de los mismos, a diferencia de lo que sucede en muchos otros seres vivos que tienen una amplia capacidad de movimiento en sus pabellones, pudiendo enfocarlos en la dirección de proveniencia del sonido. De esta manera se contribuye a la función del pabellón, que es la de concentrar las ondas sonoras en el conducto auditivo externo.

La no linealidad en las funciones de transferencia del oído comienzan ya en el pabellón, ya que por sus características físicas, éste tiene una frecuencia de resonancia entre los 4.500 Hz y los 5.000 Hz.

El canal auditivo tiene unos 2,7 cm de media en su longitud, y un diámetro promedio de 0,7 cm. Al comportarse como un tubo cerrado en el que oscila una columna de aire, la frecuencia de resonancia del canal es de alrededor de los 3.200 Hz, lo que provoca una ganancia subjetiva en la señal en este rango de frecuencias, siendo la causa de la alta sensibilidad del oído en esta banda.[3]

■ Oído medio

El oído medio, en la fig. 2.2, está lleno de aire y está compuesto por el tímpano (es el encargado de separar el oído externo del oído medio), los osículos (martillo, yunque

y estribo) que no es ni más ni menos que una cadena ósea de huesos minúsculos conectados entre sí y la trompa de Eustaquio.

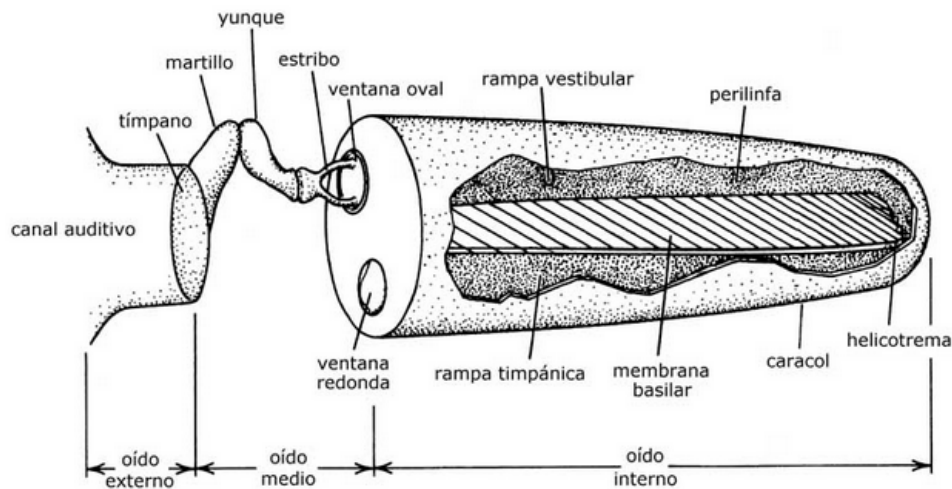


Figura 2.2: Esquema del sistema auditivo periférico con la cóclea desenrollada.[3]

La energía acústica recibida del oído externo a través del tímpano es transformada en energía mecánica y se trasmite hasta la entrada del caracol (oído interno). De dicha transformación y transmisión se encargan los osículos, que además amplifican el sonido y son los encargados de acoplar las impedancias de los diferentes medios (aire-líquido). Todo ello se realiza a través de una membrana conocida como la ventana oval. La trompa de Eustaquio sirve para igualar la presión de ambos lados del tímpano, ya que dicha trompa está comunicada con la parte superior de la faringe, y por tanto, con el aire exterior. [3]

■ Oído interno

El oído interno es la parte más importante del sistema auditivo desde el punto de vista psicoacústico. Está formado por la cóclea o caracol, fig. 2.3, en donde se encuentra la membrana basilar, la cual transforma las vibraciones mecánicas en impulsos eléctricos. Además también se encuentran en el oído interno los canales semicirculares y el nervio auditivo.

El fluido de la cóclea es excitado por el pie de estribo al empujar la ventana oval, y estas ondas son las que se propagan hasta llegar a las células ciliares (también conocidas como células capilares) del órgano de Corti. Dichas células, al ser estimuladas, generan los impulsos eléctricos que las fibras nerviosas situadas en el nervio auditivo transmiten al cerebro para su procesamiento. [3]

■ Transmisión ósea

Además, debemos tener en cuenta el factor de que el sonido no nos llega al oído interno únicamente a través del oído medio (tímpano, osículos, etc), ya que las ondas sonoras también son capaces de llegar al oído interno directamente por medio de la oscilación de los huesos que componen el cráneo.

Es algo fácilmente observable y comprobable, ya que si colocamos un diapasón para una frecuencia aleatoria dentro del espectro audible vibrando sobre el hueso parietal o sobre el hueso mastoideo (detrás del pabellón auditivo), somos capaces de escuchar su sonido sin necesidad de que nos llegue a través del oído medio.

Dado que el oído interno se encuentra dentro de una cavidad del hueso temporal, las oscilaciones del cráneo hacen entrar en oscilación directamente al fluido linfático. Dado que se ha demostrado que cualquiera de las dos formas de recepción del

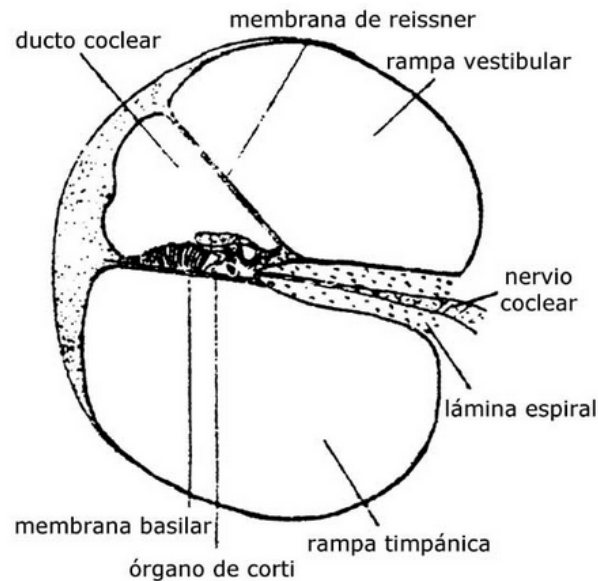


Figura 2.3: Corte de la cóclea.[3]

sonido en el oído interno es igualmente efectiva, éste método es válido para personas que puedan sufrir de enfermedades en el oído medio para poder mejorar su audición.

Como detalle interesante, saber que la transmisión ósea también es responsable de que escuchemos nuestro timbre de voz de manera diferente a como la escuchan el resto de personas.[3]

2.1.1.2 El sistema auditivo central

El sistema auditivo central se encuentra a continuación del oído interno y está formado por los nervios auditivos y los sectores de nuestro cerebro dedicados a la audición (fig. 2.4). Es, además, la parte del sistema auditivo que más se desconoce. Esto se deriva de nuestro escaso conocimiento sobre el cerebro y su funcionamiento en general.

A menudo ignorado, el sistema auditivo central se trata de una pieza fundamental en nuestra audición, ya que es donde se procesa la información recibida y se le asignan los significados a los sonidos percibidos. En un claro símil con un sistema de captación y procesamiento de audio el sistema auditivo periférico se asemejaría a la microfónica y demás elementos de la captación y el sistema auditivo central sería, en primer lugar las neuronas a modo de cables y el cerebro sería la mesa de mezclas donde se recibirían los diferentes sonidos y se procesarían.[4]

2.1.2 Umbrales de audición

Para el sistema de audición humano existen unos umbrales de audición, los cuales corresponden al mínimo nivel que un determinado estímulo debe tener para provocar así una reacción en el sujeto al cual se le está realizando el estudio.[5]

Normalmente son fáciles de determinar y medir y son una de las características más importantes de nuestro sistema auditivo. Existen dos tipos de umbrales:

- El umbral absoluto
- El umbral diferencial

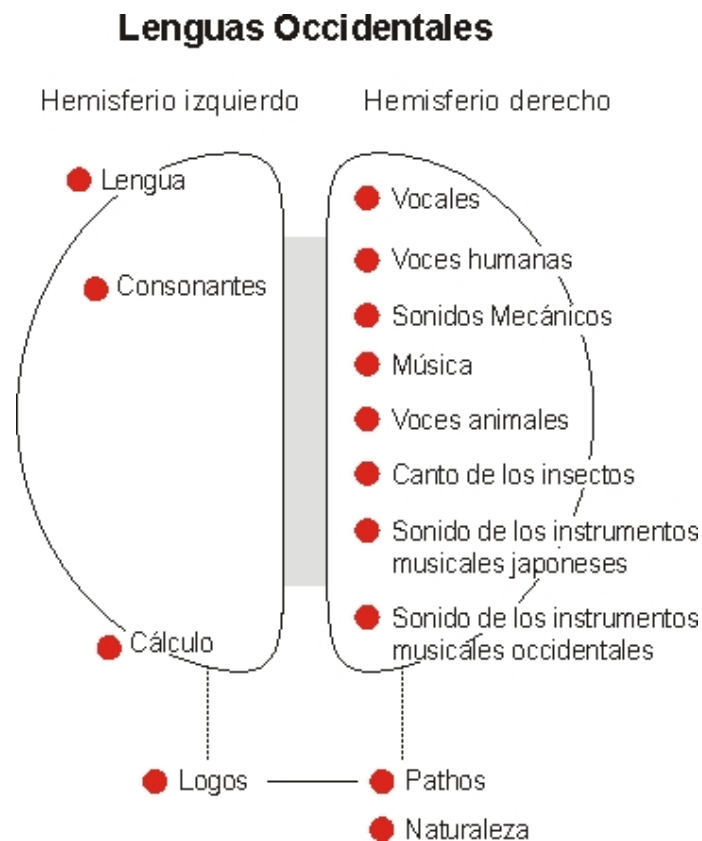


Figura 2.4: Distribución de las funciones a lo largo de los 2 hemisferios cerebrales para individuos con lengua materna occidental.[4]

2.1.2.1 El umbral absoluto

Los umbrales absolutos de la audición son aquellos valores de uno de los parámetros del estímulo físico a estudiar a partir del cual la sensación comienza o deja de producirse. Dicho umbral determina la mínima intensidad de un estímulo para la cual un 50 % de los intentos el sujeto considera que el estímulo está presente.

Para este tipo de umbral existe el umbral auditivo, fig. 2.5, el cual está definido por el valor mínimo de presión sonora para que un sonido pueda ser percibido. Además el umbral es dependiente de la frecuencia.[5][6][7]

2.1.2.2 El umbral diferencial

Los umbrales diferenciales de la audición marcan las mínimas variaciones de uno de los parámetros del estímulo físico que son necesarias para que se produzca un cambio en la sensación. Es decir, marcan la mínima diferencia que se debe producir en el valor de un estímulo para que dicha variación sea detectada por el sujeto. Al igual que para los estímulos absolutos, marcan la mínima intensidad con la que un estímulo debe exceder a otro para que el sujeto los reconozca como diferentes en, al menos, un 50 % de las pruebas.[5][7]

2.1.3 Métodos de medida

La psicoacústica es una disciplina empírica, es decir, está basada en la experiencia y en la observación de los hechos. Los resultados de los estudios se obtienen, por tanto, de manera estadística a partir de las respuestas a cada uno de los estudios. Si éstos resultados son muy diferentes, no se podrán sacar conclusiones del estudio.[8][9]

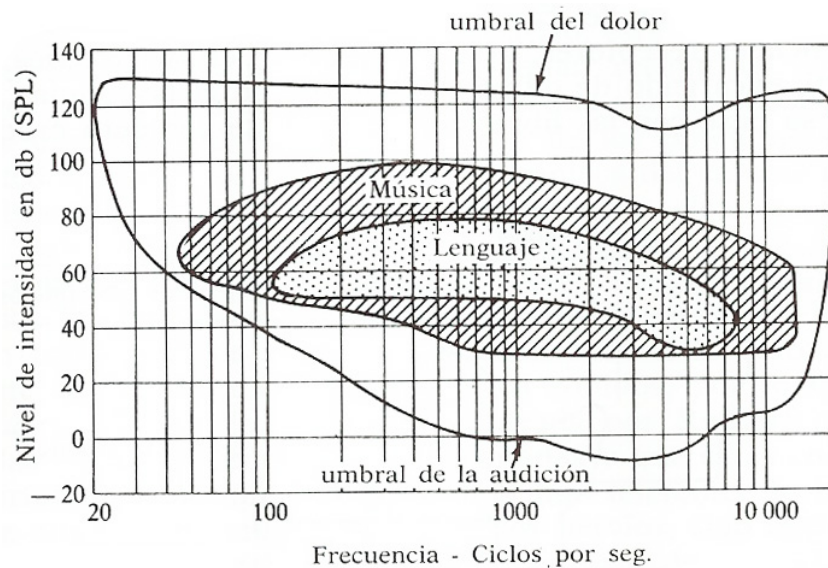


Figura 2.5: Gráfico con los umbrales de audición humanos.[7]

■ MÉTODOS CLÁSICOS

Gustav Theodor Fechner (1881-1887) elaboró una serie de procedimientos experimentales conocidos como "métodos psicofísicos clásicos", con la idea de poder determinar así los umbrales absoluto y diferencias, y de esta manera poder verificar sus hipótesis. Estos métodos son tres y se resumen de la siguiente manera:

- *Método de los límites:* También conocido como "Método de los cambios mínimos". En este método un experimentador es quien manipula la intensidad del estímulo a estudiar de manera sistemática, mientras que los sujetos del estudio se limitan únicamente a indicar si perciben o no el estímulo, además de indicar si perciben el estímulo en comparación de manera igual, menor o mayor al estímulo estándar.
- *Método de ajuste:* También conocido como "Método de error promedio". Es uno de los métodos más antiguos y fundamentales de la psicoacústica. Éste método recibe su nombre por la manera en la que realiza su estudio, es decir, el sujeto es quien debe ajustar la intensidad del estímulo hasta llegar a un punto en el que lo percibe. Otra variante se basa en que es el sujeto quien debe ajustar la intensidad del estímulo para intentar igualarla a un estímulo de referencia por comparación.
- *Método de estímulos constantes:* También conocido como "Método de los casos verdaderos y falsos". Al igual que sucedía con el método de los límites, el sujeto debe informar cuando el estímulo que recibe es igual, mayor o menor a un estímulo de referencia. La diferencia entre ambos métodos radica en que esta vez los estímulos se presentan de manera aleatoria y no en orden secuencial, tal y como ocurría en el método anteriormente visto. Al presentar los estímulos de manera aleatoria, se evitan errores de habituación y anticipación por parte del sujeto.

■ MÉTODOS ADAPTATIVOS

En los métodos adaptativos, tal y como su nombre indica, la manera de presentar el estímulo se adapta dependiendo de las respuestas anteriores del sujeto. Es decir, el nivel puede ser superior o inferior en cualquier momento. Algunos métodos clásicos, como el método de los límites y el método de ajuste pueden concordar con

esta afirmación, pero el término adaptativo debe entenderse como la convergencia de los métodos hacia un umbral de la curva psicométrica (50 % u otro punto de la misma). De este modo utilizamos menos estímulos fuera de la zona de interés del estudio.[10]

2.1.4 Bandas críticas

El ancho de banda crítico puede representarse como una medida de la selectividad frecuencial del oído. El ancho de banda crítico explica porque dado un tono de una frecuencia, una banda de ruido centrada en dicha frecuencia es capaz de producir la misma cantidad de enmascaramiento¹ sobre el tono que una banda ancha de ruido. Incluso cuando el nivel de la densidad espectral de ambos ruidos sea igual y la energía del ruido de banda estrecha sea menor (fig. 2.6). Si el ancho de la banda de ruido varía, para enmascarar el tono hará falta que la energía del ruido contenida en un intervalo de frecuencia alrededor del tono sea constante.

La energía efectiva de la señal enmascarante es la que está confinada en un intervalo, mientras que el resto no contribuye al enmascaramiento del tono. El ancho de este intervalo crítico se ha denominado como ancho de banda crítico.[12]

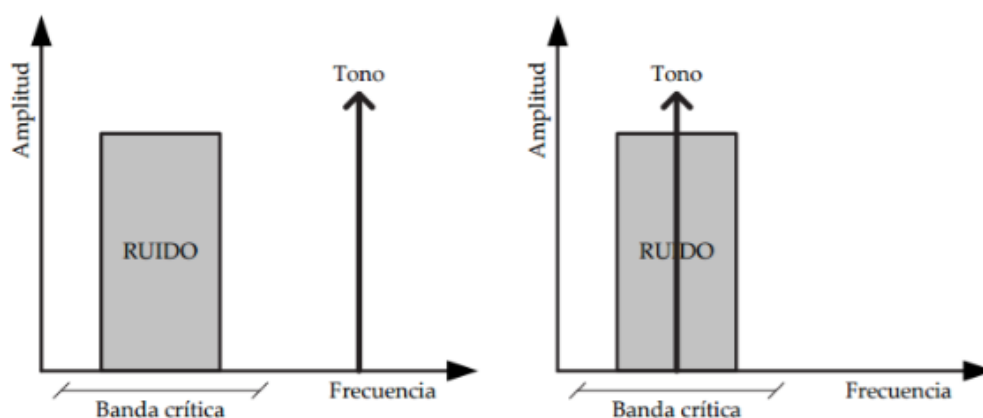


Figura 2.6: Diagrama de explicación para las bandas críticas.[12]

2.1.5 Sonoridad

Se trata de un atributo que nos permite ordenar sonidos en una escala del más fuerte al más débil.

La sonoridad² se trata de un atributo vinculado a la intensidad del sonido. No obstante, tal y como se vio en la sección de umbrales de audibilidad, la sonoridad no depende únicamente de la intensidad de un sonido, sino también de su frecuencia. Además, la sonoridad es dependiente de otras variables, como pueden ser el ancho de banda, el contenido frecuencial y la duración del sonido.

Para poder medir la sonoridad se estableció el parámetro conocido como “Nivel de sonoridad”, el cual determina cuán fuerte es escucha un sonido frente a otro. Es por tanto un parámetro psicoacústico y no físico, ya que mide cómo se detecta un sonido de fuerte frente a otro por el usuario.

¹Se habla de enmascaramiento cuando el umbral de audibilidad correspondiente a un sonido se eleva a causa de la presencia de otro. Es decir, el enmascaramiento ocurre cuando la presencia de un sonido impide la percepción de otro.

²En inglés: “loudness”

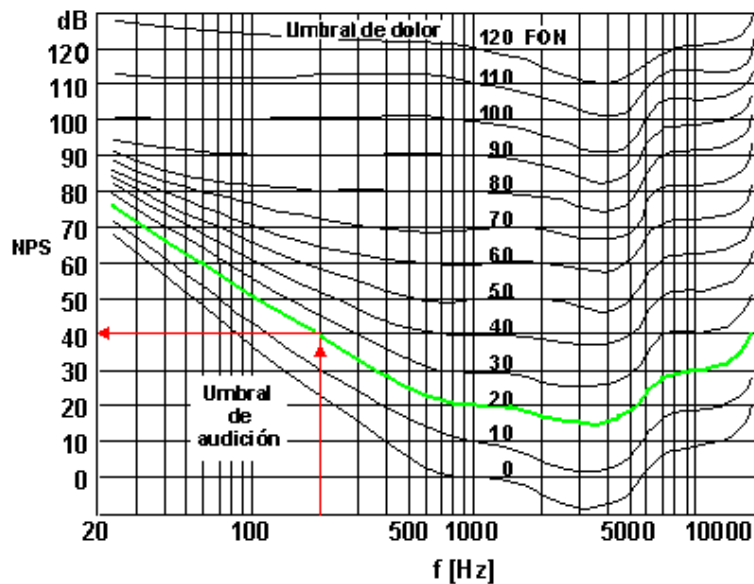


Figura 2.7: Curvas isofónicas o de sonoridad.[13]

En la fig. 2.7 se observan las curvas isófonas o isofónicas, utilizadas para determinar la sonoridad de un sonido senoidal puro. En el eje de las abscisas tenemos la frecuencia y en el eje de ordenadas observamos el nivel físico SPL. Para un punto dado entre ambos parámetros observamos que nos encontramos en una de las curvas isófonas, la cual nos indica el nivel de sonoridad del sonido.

La sonoridad está directamente relacionada con la sensibilidad del sistema auditivo frente al espectro audible. Por ejemplo, nuestro sistema auditivo es menos sensible en bajas frecuencias, por tanto necesitamos un nivel SPL mucho mayor que en medias y altas frecuencias para poder sentir el sonido como igual de fuerte. Es por tanto fácil determinar que todos los sonidos contenidos en una misma curva de sonoridad se percibirán como igual de fuertes aunque su nivel SPL sea diferente.

Además, a medida que aumenta el nivel de presión sonora las curvas se hacen más planas, es decir, la dependencia de la frecuencia es menor a medida que aumenta el nivel SPL.[13]

2.1.6 Altura

Se trata de una de las características más importantes para definir al sonido. Es dependiente de la frecuencia del mismo, lo cual se traduce en el número de vibraciones por segundo. A mayor número de vibraciones por segundo, mayor es la frecuencia y por tanto más agudo es el sonido, y viceversa, tal y como se observa en la fig. 2.8. Traducido a un ejemplo práctico, para el caso de un instrumento de cuerda, como bien puede ser una guitarra, cuando más fina, corta y tensa esté la cuerda, más agudo será el sonido y viceversa. Para un instrumento de viento, el factor determinante es el tamaño del orificio por el que se mueve la columna de viento, a mayor longitud, más grave será el sonido y viceversa.

La altura o frecuencia de un sonido se mide en hercios (Hz), de donde 1 Hz es igual a 1 vibración por segundo.

El ser humano, por su fisiología tiene un rango de audición limitado en bajas y altas frecuencias. El rango de audición se encuentra entre los 20 Hz hasta los 20000 Hz. Esto quiere decir que ni por debajo de 20 Hz (infrasonidos), ni por encima de 20000 Hz (ultrasonidos), seremos capaces de escuchar nada, si bien éste margen es una media estadística,

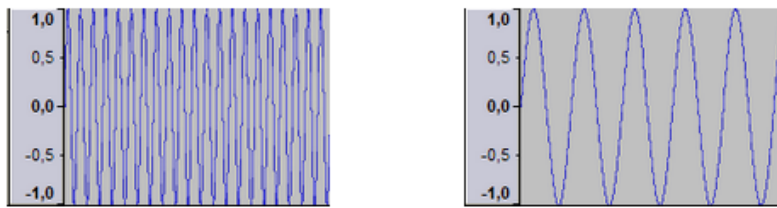


Figura 2.8: Sonido agudo (izquierda); Sonido grave (derecha)

por lo que no todo el mundo posee un margen tan amplio. Además, es importante mencionar que a medida que el ser humano se hace mayor, su rango audible se ve mermado, especialmente en altas frecuencias, disminuyendo así la frecuencia máxima audible.[14]

2.1.7 Duración

Hablando sobre la duración es algo claro pensar en el tiempo como característica principal que la define. Existe por tanto una duración objetiva, la cual se define como la duración de los sonidos que es posible medir físicamente, y su unidad suele ser el segundo. Por tanto, se trata del tiempo máximo de permanencia que pasa el sonido en vibración. Muchas veces queda limitado por las características de producción del sonido, como bien puede ser un instrumento musical acústico (no electrónico).

Además, también existe una duración mínima de los sonidos a partir de la cual, aunque un instrumento electrónico fuese capaz de generar sonidos muy breves, rápidos y consecutivos, el oído los integraría como un único sonido y, por tanto, los percibiría como simultáneos.

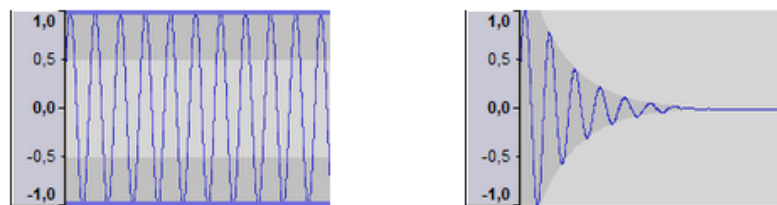


Figura 2.9: Sonido largo y sin extinción (izquierda); Sonido corto y con un decaimiento rápido (derecha)

En música por ejemplo existe otro tipo de forma para determinar la duración ya que la medición del tiempo de cada sonido no se realiza de manera individual, si no que se hace por comparación los unos con los otros. Aún así, para ésta referencia relativa de duraciones se necesita una referencia superior con la que establecer la duración absoluta. Para ello existe la indicación de metrónomo, la cual nos indica el número de “golpes” o “tiempos musicales” por minuto, expresadas en BPM¹. Por tanto, cuanto mayor sea el número de BPM, más rápida será la música y por tanto menor será cada tiempo de duración musical.[14]

2.1.8 Timbre

El timbre se trata de una cualidad que caracteriza al sonido desde un nivel perceptual. Gracias al timbre somos capaces de distinguir la voz de distintas personas por ejemplo, o un instrumento de otro aunque ambos estén sonando con la misma nota (altura).

¹BPM: Beats per minute. Golpes por minuto.

Hablando de una manera más técnica, el timbre es una cualidad relacionada con el espectro de cada sonido, tal y como se observa en la fig. 2.11. Es decir, la forma de onda que lo define. Si bien el espectro no es lo único que define al timbre, si es un factor muy importante, y tiene que ver con la cantidad y distribución de armónicos que posee un sonido respecto a su fundamental (fig. 2.10). Además, otros factores que influyen en el timbre pueden ser la envolvente temporal y espectral o el tipo de estructura de ruido que presente.

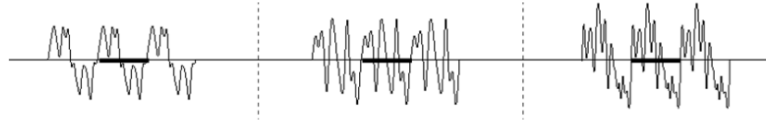


Figura 2.10: Distintos timbres para una misma altura y duración. Se observa la composición armónica de cada uno en su espectro.[16]

Por tanto, su caracterización viene dada por parámetros tanto espectrales como temporales. De especial importancia para la caracterización del timbre de un sonido son las formates, cuántos y en qué frecuencias del espectro aparecen.[16]

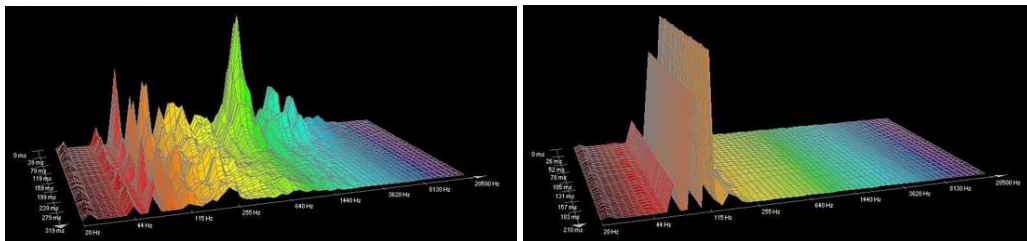


Figura 2.11: Análisis espectral de un sonido instrumental percusivo de "madera"(izquierda); Otro análisis espectral de un sonido simple (senoidal) estacionario(derecha).[17]

2.1.9 Localización espacial

Los seres humanos, al igual que multitud de seres vivos, disponemos de una pareja de oídos para escuchar los sonidos que nos rodean, por tanto disponemos de una audición binaural. La audición binaural nos permite determinar la posición de una fuente sonora gracias a la diferencia de tiempo y niveles de llegada de la señal de un oído a otro, tal y como se aprecia en la fig. 2.20

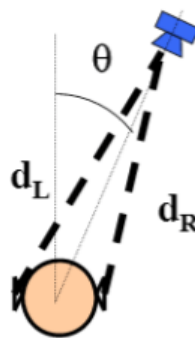


Figura 2.12: Diagrama para explicar la escucha binaural.[15]

Nuestros oídos forman dos canales receptores independientes entre sí, sin interferencia alguna entre ambos ni combinaciones entre las frecuencias recibidas por cada uno de

ellos. Por tanto, las señales se reciben independientemente por cada oído y se crean distintos efectos en diferentes partes del cerebro, donde se procesan. Por tanto, únicamente en el caso de que el nivel de la señal recibida sea extremadamente elevado, pequeñas partes de energía se transferirán de un oído a otro, debido a la transferencia provocada por los huesos craneales. El cerebro por tanto tomará la información recibida por ambos oídos y la procesará comparando ambos impulsos nerviosos, interpretando de cada uno los aspectos que caracterizarán al sonido incidente, entre ellos su posición espacial.[15]

2.2 Fundamentos del audio digital

En la presente sección se van a comentar una serie de características propias asociadas al audio digital. Para mayor comprensión de las secciones posteriores, será importante dejar claras algunas definiciones y conceptos asociados al audio en formato digital, tales como el teorema de *Nyquist*, el *dither*, etc.

Para ello, primero se debe comprender como se compone una cadena básica de conversión de audio analógico a digital, para, posteriormente, comentar que factores entran en juego en cada una de las etapas, así como posibles problemas que puedan aparecer.

2.2.1 Conversión AD/DA

La conversión AD/DA¹ es el proceso mediante el cuál una muestra de audio en formato analógico, es decir, definida por variaciones en su voltaje, puede ser convertida a formato digital, es decir, definida por unos y ceros ó código binario.

Dicha conversión nos proporciona una serie de ventajas adheridas al formato digital, tales como la posibilidad de la duplicación del material sin pérdidas tantas veces como se quiera, la perpetuidad del material en el mismo estado y sin deterioro a lo largo del tiempo, la facilidad en la edición y procesado, etc.

Para llevar a cabo dicha conversión, y su proceso inverso, hace falta una cadena de conversión, definida por un conversor AD. Dicho conversor se compone de 3 etapas fundamentales: muestreo, cuantificación y codificación.

2.2.1.1 Muestreo

La conversión de una señal analógica en una digital empieza por realizar el muestreo (*sampling*) de ésta. Este proceso consiste en tomar diferentes muestras de tensiones en diferentes puntos de la onda y a incrementos de tiempo predeterminados. La frecuencia a la que se realiza el muestreo se denomina tasa o frecuencia de muestreo y se mide en hertzios. Para audio, a mayor número de muestras tomadas, mayor será la calidad y fidelidad de la señal digital resultante.

Para determinar la frecuencia de muestreo mínima se utiliza el “Teorema de *Nyquist*”. En él, se expone que para que una señal de audio pueda ser reconstruida sin perder calidad, se debe utilizar una frecuencia de muestreo mayor al doble de la frecuencia máxima contenida en la señal de audio analógico. Es decir: $F_s > 2 \cdot F_{max}$ siendo F_s la frecuencia de muestreo y F_{max} la máxima frecuencia comprendida en la señal.

En caso de no cumplir dicho requisito, aparecerá *aliasing*, o ruido de muestreo. Este ruido está provocado por utilizar una frecuencia de muestreo menor al doble de la frecuencia máxima contenida en el audio, que suele ser 20 kHz, por lo que la frecuencia de muestreo mínima recomendada debería ser 40 kHz, siendo el mínimo estandarizado 44,1

¹ Analógico-Digital/Digital-Analógico

kHz. Esa pequeña diferencia sirve para asegurar un margen que nos permita no tener *aliasing*.

Durante el proceso de muestreo se asignan valores numéricos equivalentes a la tensión o voltaje existente en diferentes puntos de la senoide, con la finalidad de realizar a continuación el proceso de cuantización.

2.2.1.2 Cuantificación

Una vez hemos realizado el muestreo, se pasa al proceso de cuantificación. En él, se cogen los valores de señal continua de la sección de muestreo, y se convierten en series de valores numéricos decimales discretos, correspondientes a los diferentes niveles o variaciones de voltajes que contiene la señal analógica.

Por tanto, la cuantificación representa el componente de muestreo de las variaciones de valores de tensiones o voltajes tomados en diferentes puntos de la onda sinusoidal, que permite medirlos y asignarles sus correspondientes valores en el sistema numérico decimal, antes de convertir esos valores en sistema numérico binario.

Al igual que para el muestreo, para la cuantificación a mayor número de niveles en los que representar la señal analógica, mayor será la calidad de la conversión. Dicho número de niveles viene definido por la resolución del conversor, y se define en bits. El número de niveles será igual a $2^{n^{\text{bits}}}$, siendo 16 bits el estándar recomendado para un CD de audio, garantizando una conversión correcta y adecuada.

Cuando por ejemplo convertimos un audio con una resolución mayor a otra menor nos aparece lo que se conoce como “ruido de cuantificación”. Es un ruido muy tenue pero muy desagradable al oído humano. Para corregir dicho problema se añade un ruido *gaussiano* muy tenue también para enmascararlo, llamado *dither*. El *dither* puede parecer contraproducente, ya que al final se añade más ruido a la señal, pero éste es mucho más agradable que el ruido de cuantificación, por lo que el oído lo tratará como más aceptable y quedará enmascarado por la señal, la cuál estará a un nivel muy superior.

2.2.1.3 Codificación

Después de realizada la cuantificación, los valores de las tomas de voltajes se representan numéricamente por medio de códigos y estándares previamente establecidos. Lo más común es codificar la señal digital en código numérico binario.

2.3 Procesado y efectos de audio

La principal diferencia entre efectos y procesados de audio, está en que en los procesados de audio la señal cambia por completo, originando como resultado una nueva señal de audio, sin embargo en los efectos de audio la señal original se mezcla con la nueva. Aun tratándose de conceptos diferentes, las unidades de modelado y efectos no hacen distinción entre ellos y tienden a llamarlos simplemente “efectos”. Los diferentes tipos de efectos de audio que vamos a tratar para el procesado de audio son:

- Frecuenciales
- Dinámicos
- Temporales

2.3.1 Frecuenciales

Este tipo de efectos se basan en alterar el contenido en frecuencia de una señal de audio, logrando conseguir un sonido más o menos brillante, además permitimos modificar cualidades de una canción ya grabada según se desee. Algunos de los tipos de efectos basados en la alteración de la frecuencia son los ecualizadores y la distorsión.

2.3.1.1 Ecualizadores

Es un dispositivo electrónico que se encarga de modificar la respuesta en frecuencia de un sistema, ya sea aumentando o disminuyendo diferentes bandas; Para ello modifica en amplitud las frecuencias de la señal de audio, consiguiendo un aumento o disminución de volumen en la banda deseada. El objetivo de los ecualizadores es adaptar la señal sonora a los gustos del usuario, ya sea para atenuar o eliminar frecuencias molestas, ruidos o interferencias que se mezclan con el sonido, como para simplemente, obtener una salida que resulte más agradable o más adecuada al recinto.

2.3.1.2 Distorsión (*Overdrive*)

Este tipo de dispositivos se encargan de saturar la señal de audio a un determinado nivel, generando la aparición de multitudes de armónicos.

Se trata de una alteración en la onda del sonido que recorta las frecuencias convirtiendo la curvatura natural de la onda en una figura más plana y que, además altera los llamados armónicos (son frecuencias o tonos alternativos a la nota “pura”) generando disparidad en las ondas.

2.3.2 Dinámicos

Están basados en la variación de amplitud y volumen de una señal, generando una serie de efectos como el compresor o supresor o puerta de ruido.

2.3.2.1 Compresor

Se trata de realizar una reducción del rango dinámico de la señal que vamos a tratar, es utilizado generalmente para reducción de ruido, producción, etc.

2.3.2.2 Puerta de ruido

Una puerta de ruido o *noise gate* es un procesador dinámico de señal diseñado para eliminar los ruidos durante las pausas, pero también ofrece un efecto de barrido de volumen automático. La principal función de una puerta de ruido es cortar el paso de toda señal que no supere un umbral de referencia establecido.

2.3.3 Temporales

Este tipo de efectos son producidos por la introducción de muestras retardadas en la señal, produciendo alteraciones en determinados parámetros como la frecuencia, amplitud y fase. A continuación vamos a describir algunos de los diferentes tipos de efectos temporales más utilizados.

2.3.3.1 Eco

El efecto eco consiste en añadir al audio original múltiples versiones retardadas y atenuadas, imitando el eco original que se genera en la naturaleza. Para apreciar los retardos como eco, el tiempo del retardo debe ser superior a 50 milisegundos ($t > 50$ milisegundos); Las versiones se van atenuando hasta que se vuelven imperceptibles.

2.3.3.2 Reverb

Este tipo de efectos es utilizado principalmente para modificar una serie de parámetros que simulara un recinto acústico en concreto, pero además nos permite crear otro tipo de sonidos más personalizados o de mayor riqueza de sonido.

2.3.3.3 Delay

Este tipo de efectos, también conocidos como retardos, se basan en generar retardo en la señal, consiste en coger la señal de audio y volver a reproducirla después de un periodo de tiempo, mezclada y escalada con la señal actual.

2.4 Obras y apps de entrenamiento auditivo disponibles

2.4.1 *Golden Ears*

La primera de las aplicaciones que se ha tenido en cuenta para la realización de nuestro proyecto, y en la que nos hemos podido inspirar es la colección *Golden Ears* de *Moulton Laboratories*. Dicha compañía fue fundada por David Moulton, reputado ingeniero de audio, además, de productor, autor, compositor y profesor de acústica. A continuación una breve aunque interesante biografía del mismo:

David o Dave Moulton es un ingeniero y productor de audio, autor de numerosos artículos y obras relacionadas con el sector del audio, compositor y profesor de acústica. Comenzó a trabajar en el mundo del audio a finales de los años 60, fundando los estudios Dondisound de Red Hook, Nueva York (EEUU). Tras varios años dedicándose al mundo del audio profesional y a ampliar sus conocimientos académicos, en 1987 se convirtió en Presidente de Producción Musical e Ingeniería en el Berklee College of Music de Boston. Posteriormente renunció a Berklee en 1993 para dedicarse a tiempo completo a la grabación, la composición, la investigación y la escritura. En la actualidad, opera Moulton Laboratories / Digital Media Services en Groton. Además de todo esto, se ha especializado en la producción multicanal envolvente y ha sido nominado a los Grammy por su trabajo en la producción de un disco de George Crumb.

A través de *Moulton Laboratories*, David Moulton sacó al mercado una colección de muestras de audio con un manual de utilización llamado *Golden Ears*, destinado al entrenamiento y la mejora auditiva desde un punto de vista técnico. Se trata de un curso de entrenamiento auditivo para ingenieros de grabación, productores y músicos. Moulton comenzó a hacer dicho entrenamiento como un ejercicio educativo para sus estudiantes en el año 1969. Desde entonces lo ha utilizado con asiduidad, hasta que en 1992 *KIQ Productions* le pidió a Moulton que lo hiciese comercialmente disponible, a lo que éste aceptó. El resto es historia.

La colección *Golden Ears* viene distribuida en 4 volúmenes bien diferenciados y divididos según el ámbito de estudio de cada uno. A continuación se expone un breve resumen con la explicación de cada uno y en que consiste:

■ Vol. 1: Frecuencias

Ejercicios pensados para mejorar el reconocimiento e identificación en ligeras variaciones de nivel y cortes a lo largo de las diez octavas principales del espectro frecuencial. Se organiza en ejercicios progresivos en dificultad, desde simples aumentos en las diferentes octavas musicales hasta mínimas variaciones en octavas de ruido rosa. Gracias a la gran diferencia de dificultad entre los ejercicios más sencillos y los más complicados facilita el aprendizaje tanto de alumnos primerizos como de los más avanzados.

■ Vol. 2: Efectos y procesado

El segundo volumen está compuesto por 31 tipos de procesado de señal diferentes agrupados en familias: cambios de amplitud, distorsión bruta y sutil, compresión con liberación lenta y rápida, cambios de EQ, anomalías del campo estéreo y ajustes de *retardo/reverb*.

■ Vol. 3: retardos y Decays

Volumen enfocado a la comprensión y la correcta utilización de los tiempos de retardo para los efectos de *retardo*, tanto a nivel de segundos hasta incluso milisegundos. Además se complementa con efectos de panorámica/*slap*/amplitud en mono y estéreo, sonidos con *sustain* y transitorios y envolventes. Por último se completa con aprendizaje en los parámetros de reverberación, tiempos de decaimiento, etc.

■ Vol. 4: Frecuencias maestras

Ampliación de conocimientos en el campo de la ecualización para conseguir un control de EQ avanzado. Se pretende llegar a identificar los cortes y el aumento en tercios de octava. Además se amplía con la identificación de variaciones en la ecualización de dos bandas de octava simultáneamente modificadas.

A continuación el link a la página oficial del producto:

<http://www.moultonlabs.com/full/product01>

2.4.2 SoundGym

SoundGym, fig. 2.13, simula ser una especie de gimnasio online para el entrenamiento auditivo. Su objetivo es formar a productores de música e ingenieros de audio para mejorar su audición hasta poder llegar a detectar matices sonoros muy sutiles, desarrollar habilidades de escucha profesional y poder llegar así al objetivo de obtener mejores y más rápidas decisiones de cara a un entorno profesional de trabajo.

Para llevar a cabo todo lo anteriormente mencionado, *SoundGym* hace uso de una interfaz gráfica amigable, llevando a cabo los ejercicios propuestos en forma de divertidos juegos de sonido. Gracias a su forma de trabajo y enfoque didáctico, los miembros pueden mejorar así sus habilidades de escucha más importantes y necesarias, tales como la detección de frecuencias, compresión, coloración, detección de diferencias de ganancia, impresión espacial, etc.

Cada juego se centra en mejorar una habilidad acústica específica, ofreciendo así una experiencia de aprendizaje integral y efectiva. Además, y a diferencia del ejemplo visto anteriormente, *SoundGym* crea un perfil de usuario al comenzar a utilizarlo, de forma que el usuario puede seguir sus estadísticas de aprendizaje y el progreso en los ejercicios, además de seguir a amigos, comparar puntuaciones, lograr objetivos, ganar previos, etc.



Figura 2.13: Logo *SoundGym*

Por todo lo visto, uno de los puntos fuertes de *SoundGym* frente a *Golden Ears* es su forma de enfocar el aprendizaje como si fuese un juego, haciendo así el progreso mucho más amigable y sencillo.

A continuación el link a la página oficial del producto:

<https://www.soundgym.co/>

2.4.3 Train Your Ears

Train Your Ears EQ Edition, fig. 2.14, es un software disponible para PC y Mac para hacer llegar a entender a los ingenieros y productores cómo se comportan las frecuencias, como identificarlas y como trabajar con ellas.



Figura 2.14: Logo *Train Your Ears*

Las posibilidades que nos ofrece son todas ellas enfocadas al entrenamiento y afinación del oído en el ámbito frecuencial. La idea es acelerar el proceso de aprendizaje del oído a base de entrenamiento intensivo. Para ello, somete al alumno a cientos de ecualizaciones al azar y desconocidas, las cuáles el alumno tiene que adivinar. En caso de que la respuesta del alumno sea errónea, se le darán indicaciones sobre dónde ha sido su fallo y se le permitirá escuchar su respuesta y la opción correcta, para poder así contrastar una con la otra.

Gracias a su forma de trabajo, en poco tiempo se podrá desarrollar una especie de memoria frecuencial, con la que el ingeniero será capaz de poder “conectar” el sonido que imagina en su cabeza con los parámetros que necesita marcar en el ecualizador de manera rápida y efectiva.

Además, en la última actualización del software, éste incorpora un nuevo método de entrenamiento, el cual consiste en el proceso inverso. En vez de tener que adivinar los parámetros de una ecualización aleatoria dada, el alumno debe hacer correcciones sobre un audio “empeorado” para dejarlo en su estado original.

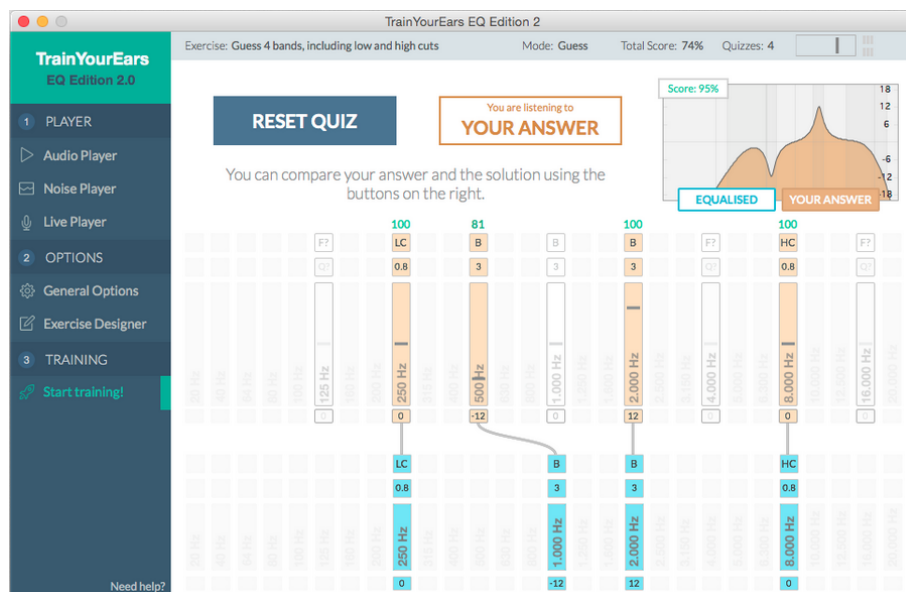


Figura 2.15: Interfaz gráfica *Train Your Ears EQ Edition*

Éste nuevo método fue sugerido a la empresa *Train Your Ears* por el gurú de masterización Bob Katz.

Además, en la última actualización se mejoró la interfaz gráfica, fig 2.15, adecuándola a un estilo moderno y limpio, con una nueva pantalla de entrenamiento asistido, diseñador de ejercicios, varios idiomas, etc.

A continuación el link a la página oficial del producto:

<https://www.trainyourears.com/>

2.4.4 iZotope Pro Audio Essentials

iZotope Pro Audio Essentials, fig. 2.17, es un software de entrenamiento auditivo desarrollado por la casa estadounidense *iZotope Inc*, fig. 2.16.

iZotope Inc. es una compañía de software de audio profesional con sede en Cambridge, Massachusetts, Estados Unidos. *iZotope* desarrolla software de audio para grabación, *broadcast*, mezcla, diseño sonoro y *mastering*. Todo su software es desarrollado para funcionar en los principales DAW¹ del mercado, tales como Pro Tools, Cubase y Nuendo, Logic Pro X, etc.



Figura 2.16: Logo *iZotope*

Además del segmento profesional, y gracias a toda la experiencia de la empresa, *iZotope* desarrolló una rama didáctica y académica con publicaciones educativas en revistas, libros e incluso un *blog* propio.

Partiendo de esta rama educativa, *iZotope* desarrolló su conjunto de aplicaciones software *Pro Audio Essentials*, enfocado al entrenamiento auditivo para técnicos, ingenieros y productores noveles.

La organización de dichas aplicaciones queda dividida en 3 grandes ramas:

- Ecualización
- Compresión
- Fundamentos básicos de audio digital.

A su vez cada una de las 3 familias queda subdividida en 4 partes:

- Aprendizaje.- Se compone de una serie de artículos y vídeos didácticos con explicaciones teóricas y prácticas para que el alumno asimile conceptos y llegue a entender los parámetros más importantes de cada ámbito.
- Explorar.- Consiste en una interfaz gráfica con el vocabulario técnico de cada ámbito enfocado a que el alumno entienda como funciona el audio y que características tiene.
- Practicar.- Interfaz gráfica a modo de juego con ejercicios sobre cada rama de estudio para que el alumno pueda practicar y entrenar el oído de cara al modo Desafío.
- Desafío.- Igual que el modo “Practicar” pero con un *ranking* de usuarios y registros de puntuaciones. Se enfoca desde un punto de vista competitivo para fomentar las ganas de mejora del alumno y sus ganas de practicar para aumentar su puntuación.

¹Digital Audio Workstation

Cabe destacar que por el momento las cuatro partes de aprendizaje sólo están disponibles en la familia de EQ, ya que en Compresión y Fundamentos del audio digital únicamente está disponible el apartado de Aprendizaje. Al parecer la empresa está todavía en desarrollo del resto de aplicaciones.

La aplicación posee una interfaz gráfica y un flujo de trabajo muy cómodo y agradable, aún tratándose de una aplicación *online*. Únicamente es necesario crearse una cuenta de usuario para acceder a todas las funcionalidades disponibles.

A continuación el link a la página oficial del producto:

<https://pae.izotope.com/>

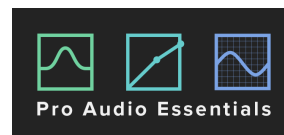


Figura 2.17: Logo Pro Audio Essentials

2.4.5 Quiztones



Figura 2.18: Logo Quiztones

Quiztones, fig. 2.18, es una aplicación nativa disponible para las plataformas *Mac OS X*, *iOS* y *Android* a un precio reducido de 7,99€ en su versión para *OS X*. Se trata, según el propio desarrollador, de una aplicación de entrenamiento auditivo enfocada para ingenieros de audio, productores y músicos aficionados y profesionales. La aplicación está enfocada en el apartado de frecuencias, utilizando tonos y ruidos alterados frecuencialmente, además de bucles musicales y material del propio usuario. A dichos audios se les modifica su espectro frecuencial según el nivel de dificultad y el ejercicio seleccionado por el usuario, y éste debe acertar que modificación se le ha aplicado.

En la aplicación existían 3 tipos de ejercicios con diferentes dificultades cada uno según las exigencias del usuario. Los ejercicios eran los siguientes:

- Tonos: Aplica al audio un tono puro a un nivel bastante reducido y el usuario debe adivinar la frecuencia del tono aplicado eligiendo entre 4 opciones aleatorias.
- EQ: La muestra de audio sufre una ecualización de pico o atenuación a una frecuencia concreta y el usuario debe adivinar en que frecuencia se ha aplicado dicha ecualización.
- Ganancia: Este ejercicio se desmarca un poco de los dos anteriores y ofrece al usuario la posibilidad de trabajar con diferencias sutiles de ganancia, teniendo que adivinar el usuario la atenuación que se ha aplicado a la muestra original.

Si bien la aplicación es bastante más básica que las anteriores vistas, ofrece la posibilidad de ser utilizada tanto en máquinas de sobremesa como en dispositivos móviles, añadiendo así un plus de funcionalidad y versatilidad. Además destaca por poseer una interfaz gráfica muy depurada y agradable, haciendo así mucho más ameno el entrenamiento.

A continuación el link a la página oficial del producto:

<https://quiztones.com/>

2.4.6 EarMaster



Figura 2.19: Logo EarMaster

EarMaster, fig. 2.19, es un programa disponible para PC y Mac OS X que va por su sexta edición. Se trata de una aplicación completa para el entrenamiento del oído. La diferencia clara de EarMaster respecto de las aplicaciones anteriores reside en que está enfocada desde un punto de vista mucho más musical. Su público potencial, según el propio desarrollador, son estudiantes de música, profesores de música, escuelas y coros y músicos profesionales y *amateurs*.

Si bien su precio es un tanto elevado, ronda los 60€, está disponible en versión Pro, y *Teacher*, dando así la facilidad a los profesores de disponer de una herramienta potente para el desarrollo académico.

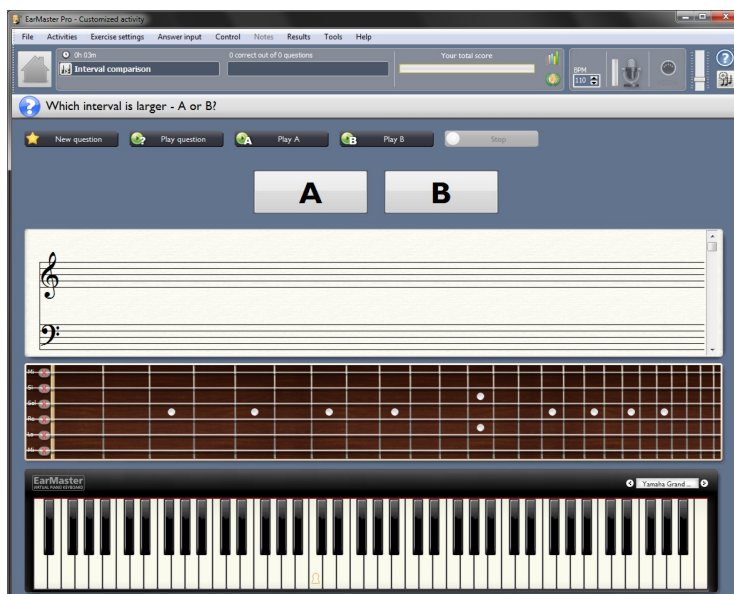


Figura 2.20: Interfas gráfica EarMaster

A continuación el link a la página oficial del producto:

<https://www.earmaster.com/es/>

2.5 Propuesta de este trabajo

En el siguiente proyecto se propone la investigación, desarrollo y documentación de una *suite* de aplicaciones para entrenamiento auditivo mediante *Csound* y *CsoundQt*. El público al que irán dirigidas las aplicaciones es principalmente un público especializado, especialmente técnicos e ingenieros de sonido. Si bien podrá utilizarse entre un público principiante, será necesario poseer ciertos conocimientos técnicos previos, por básicos que sean, para el correcto entendimiento de las aplicaciones y el fin que persiguen.

Cada una de las aplicación individuales que compondrán la *suite* estará enfocada a un campo del audio concreto, buscando así focalizar en cada una de ellas cada uno de los parámetros utilizados en el mundo del audio profesional y de esta manera centrar los esfuerzos en un punto concreto, siendo dichos focos de atención la localización espacial, la detección frecuencial, el ruido y la distorsión y por último la ganancia y la dinámica del audio.

El motivo por el que se ha buscado desarrollar esta serie de aplicaciones para entrenamiento auditivo existiendo ya alternativas comerciales como las vistas en la sección 2.4 es variado.

En primer lugar, la mayoría, y tal y como se ha dicho, son aplicaciones comerciales. Algunas incluyen pagos para ampliar funcionalidades, otras son completamente de pago, etc. En nuestro caso, se busca que sean gratuitas desde el primer momento y en su totalidad, facilitando así el acceso a quien desee probarlas.

En segundo lugar, están basadas en lenguaje *Csound*, es decir, software libre. Esto es un punto a su favor desde el punto de vista del desarrollo, ya que cualquiera que se muestre interesado, podrá hacer uso del código fuente original para mejorar o añadir funcionalidades, siendo todo esto legal y gratuito.

En vistas de lo anterior, estas son las premisas con las que se busca una diferenciación clara entre nuestras aplicaciones y las disponibles en la red, siendo pilares fundamentales la gratuidad de las mismas y el uso de software libre para su desarrollo.

2.6 Csound - Herramienta

2.6.1 Csound

Csound se trata de un lenguaje de programación en computador enfocado principalmente a la generación, edición y análisis de música y sonido. Su nombre deriva de su compilador, programado en lenguaje C, seguido de la palabra *sound* que es al fin y al cabo para lo que está desarrollado.

Está formado por una estructura en forma de instrumentos en un fichero de orquesta, especificados mediante operadores de síntesis y procesamiento de sonido. Dichos instrumentos son controlados y activados desde un segundo fichero denominado partitura. Para llevar a cabo su función, ambos ficheros (orquesta y partitura) son procesados por el compilador de *Csound*, obteniendo el resultado de la colaboración entre ambos archivos en forma de sonido. Cabe destacar que en la mayoría de los casos, y por comodidad a la hora de trabajar, tanto la orquesta como la partitura se agruparán en un mismo archivo de proyecto con la extensión *.csd*.

Csound es un lenguaje con una antigüedad notable, datando sus inicios en la década de 1970. Partiendo de esto es fácil pensar que en su momento era ejecutado en máquinas con una capacidad de procesamiento infinitamente menor a las máquinas actuales. Por tanto, funciona correctamente en máquinas pequeñas y lentas, por lo que, si se ejecuta en equipos actuales y potentes ofrece la posibilidad de utilizarlo en tiempo real, la creación de interfaces gráficas de usuario, así como la interconexión con otros sistemas de audio vía *drivers*, *MIDI* o red.

Csound es un software libre basado en código abierto, con licencia *LGPL*, totalmente modular y ampliable por el usuario. *Csound* está relacionado con el lenguaje para audio estructurado de *MPEG-4*, *SAOL*.

2.6.1.1 Funcionamiento de Csound

El funcionamiento en el que se basa *Csound* es, tal y como se ha mencionado anteriormente, trabajar con dos clases de objetos relevantes en la composición.

En primer lugar se debe crear la “orquesta”, la cual se compone de los instrumentos que se utilizarán para la composición.

En *Csound* hay que crear los instrumentos, es decir, realizar una descripción completa de cómo son y cómo funcionan: esto puede ir desde un oscilador que genere un tono puro

sinusoidal de un frecuencia determinada, a un instrumento complejo cuyo timbre varía estadísticamente.

```

1 ; EJEMPLO DE INSTRUMENTO EN CSOUND
2 instr 1                      ; instrumento 1
3   iamplitud = 10000          ; amplitud constante = 10000
4   ifrecuencia = p4           ; frecuencia controlada desde la partitura (parametro
5                               4)
6   itabla = 101               ; numero de tabla constante = 101
7   a1 oscil iamplitud, ifrecuencia, itabla
8   out a1
9   endin                      ; final de instrumento 1

```

Por otro lado está la partitura, que es el segundo objeto relevante. Esta no es más que una tabla o gráfica donde se especifica el orden de actuación de los instrumentos a lo largo del tiempo. Por ejemplo, el clarinete toca tal cosa mientras el timbal reverbera en una sala de tales características. Con *Csound*, el control sobre el sonido digital es absoluto (si se encuentra el programa adecuado).

```

1 ; EJEMPLO DE PARTITURA EN CSOUND
2 f 101 0 4096 10 1           ; tabla que contiene un ciclo de onda
3                               sinusoidal
4
5 ;instr inicio duracion frecuencia
6 ; p1      p2      p3      p4
7 i 1      0        4        440
8 i 1      +        2        880
9 i 1      +        1        1760
10 e        ; fin de la partitura

```

Desde la versión 5.0, se da un cambio significativo en código de *Csound*. Aparecen características como: mejora de la gestión en tiempo real, *API's* para otros lenguajes de programación como *Python* o *Lisp*, Interfaces gráficas definidas por el usuario, protocolos de *hosting* a *VST* y *LADSPA*, protocolo *OSC*, *opcodes* definidas por el usuario y un gran número de nuevos *opcodes* de orquesta. Además en la versión para Linux, se acopla perfectamente con *Jack*¹.

A continuación el link a la página oficial del producto:

<http://www.csounds.com/>

2.6.2 CsoundQt

Una vez llegados a este punto, cabe destacar la herramienta final utilizada para el desarrollo del presente proyecto. Tras varios intentos y pruebas con las herramientas descritas anteriormente, se acabó decidiendo utilizar la herramienta que ya habíamos testado anteriormente en clase de *Síntesis Digital de Sonido*, y que por tanto más se dominaba.

El motivo de la elección fue que, tras probar el resto de opciones de programación gráfica en lenguaje *Csound* disponibles en el mercado, *CsoundQt*, fig. 2.21, es de las más completas, y sobre todo, una de las que más información disponible tiene en la web. Esto es una prioridad a la hora de



Figura 2.21: Logo *CsoundQt*

¹Gestor de audio en *Linux*

desarrollar un proyecto de esta envergadura ya que las dudas son muchas y muy variadas. Partiendo de esta base, otra de las posibles elecciones para el desarrollo del proyecto fue *Cabbage*, pero se descartó precisamente por no disponer de suficiente información en la web como para poder desarrollar el proyecto de manera satisfactoria.

Una vez llegados a este punto, y sabiendo de la elección de *CsoundQt* como herramienta definitiva de cara a afrontar el proyecto, se hará una breve, pero concisa descripción de *CsoundQt* y de sus características y virtudes.

CsoundQt es un software *front-end* para lenguaje *Csound* que incorpora entre sus funcionalidades un editor de resaltado con autocompletado. Además, posee la posibilidad de incorporar widgets interactivos como parte de la edición GUI del programa, fig. 2.22, así como una ventana de ayuda integrada con asistencia para cada una de las funciones de *Csound*.

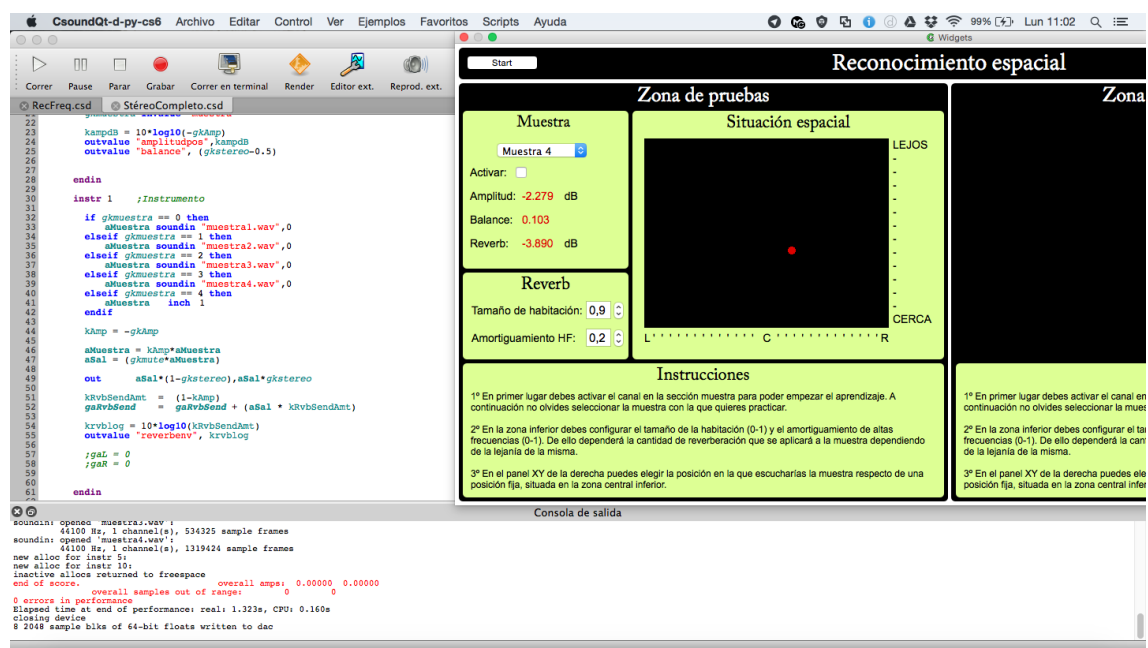


Figura 2.22: Interfaz gráfica *CsoundQt*

Otra de sus virtudes es que se trata de un software multiplataforma, disponible pues para los sistemas operativos *Microsoft Windows*, *Apple Mac OSX* y *Linux*. Pretende ser un entorno de desarrollo completo, sencillo, pero potente para *Csound*. Además permite abrir archivos creados por *MacCsound*.

CsoundQt pretende facilitar y acercar la potencia del lenguaje musical *Csound* a un grupo numeroso de gente. Su iniciativa se basa en reducir la curva de aprendizaje inicial de *Csound* mediante una serie de facilidades, tales como su interfaz gráfica, dando así a los usuarios un control inmediato de su sonido.

Por tanto se puede resumir en una herramienta sencilla pero potente, enfocada para usuarios desde un nivel básico como herramienta de práctica y aprendizaje, y para usuarios avanzados como herramienta de desarrollo con un control preciso y avanzado.

A continuación el link a la página oficial del producto:

<http://csoundqt.github.io/>

2.6.3 Otros *front-end*

Para el desarrollo de nuestro conjunto de aplicaciones para entrenamiento auditivo se valoraron inicialmente varias aplicaciones *front-end* diferentes a *CsoundQt*, si bien todas ellas trabajando bajo el lenguaje de programación *CSound*.

Las herramientas *front-end* que se valoraron inicialmente, y las cuáles se probaron en su totalidad antes de elegir la definitiva fueron *Cabbage*, *WinXound* y *Blue*. Cada una de ellas aportaba una característica diferenciadora que las hacía interesantes en su totalidad.

En el caso de *Cabbage*, destacaría su potencia respecto al desarrollo de interfaces gráficas, con una variedad y cantidad de *widgets* y posibilidades por encima del resto. Por contra, la información en la web era muy limitada, haciendo muy costoso trabajar con ella al carecer de documentación suficiente para ir resolviendo los problemas que nos iban surgiendo.

En el caso de *WinXound*, carecía de personalidad frente al resto de aplicaciones, ya que no aportaba nada que no nos diese *CsoundQt* por ejemplo. Frente a una limitación así, era fácil decantarse por *CsoundQt*, ya que la conocíamos previamente de asignaturas como síntesis digital de la señal.

Por último, se valoró el uso de *Blue*. Esta aplicación aportaba un enfoque diferente al desarrollo, ya que incluía una especie de “línea de tiempo” a través de la cuál se podían ir lanzando fragmentos de código de manera secuencial. Dichas funcionalidades no fueron suficientemente interesantes para nuestro proyecto, por lo que finalmente también se desechó el uso de *Blue*.

CAPÍTULO 3

Metodología

3.1 Visión general

Llegados a este punto había llegado la hora de comenzar a tomar decisiones relevantes sobre cómo iba a estar compuesto el proyecto. Decisiones de vital importancia como por ejemplo que aspectos del audio iban a ser entrenados, el flujo de trabajo que se iba a aplicar a los ejercicios, la organización de los mismos, etc.

En primer lugar, y basándonos en aplicaciones similares, se decidió que parámetros del audio íbamos a querer entrenar en nuestra suite de aplicaciones. Dicha decisión se debe tomar basándonos en dos factores muy importantes. En primer lugar se tuvo en cuenta la importancia y relevancia que tiene cada parámetro del audio en una mezcla para un técnico. Por ejemplo, no es lo mismo la importancia que tiene una panoramización o el espectro frecuencial de un instrumento que por ejemplo los tiempos de retardo que se aplican para conseguir un determinado efecto. Estaremos todos de acuerdo en que el primer parámetro es algo mucho más trivial y relevante que el segundo. Una mezcla puede estar sin retardo y sonar bien, pero no sin una panoramización o una ecualización adecuada.

En segundo lugar se tuvo en cuenta la dificultad de la implementación de cada aplicación deseada, así como su viabilidad en formato de ejercicio útil para el usuario. Por ejemplo, habrán ejercicios mucho más fáciles de implementar que otros, así como algunos que por sus características propias serán mucho más sencillos de entrenar que otros.

3.2 Apartados didácticos de la aplicación

Una vez claras las premisas que debemos tener en cuenta para el diseño y la implementación de nuestros ejercicios, a continuación se exponen cada uno de ellos con un pequeño resumen del mismo, así como los objetivos que se esperan conseguir con cada uno de ellos:

3.2.1 Reconocimiento espacial

El primero de nuestros objetivos es llevar a cabo un ejercicio enfocado al entrenamiento y la asimilación de los conceptos de reconocimientos espacial. Dentro del sonido existe lo que se conoce con el nombre de "planos sonoros". Esto es el lugar donde se desarrolla algún sonido dentro de una escena (ya sea real o imaginaria).

Un buen ejemplo de ello es una escena de cine. Dentro de la escena existe un escenario más o menos amplio en el cuál no sonará igual un personaje hablando a la cámara en primer plano, que alejado unos metros por ejemplo. Tampoco sonará igual un personaje situado fuera de plano en un lateral, que otro en el centro de la escena.

En la música sucede algo similar con la colocación espacial de los instrumentos. Según el género musical (ya sea música clásica, contemporáneas como el rock o el pop, etc) la colocación de los instrumentos dentro del plano espacial, ya sea izquierda-derecha o delante-detrás, es diferente. Para conseguir dicha colocación y dar la sensación de espacialidad tal y como se situarían los músicos en un escenario por ejemplo, se utilizan técnicas como la panoramización de instrumentos, el aumento de la reverberación, etc.

Como ejemplo claro tenemos una batería acústica. Para una batería acústica en géneros de música moderna como pop o rock, y utilizando un estándar de 8 micrófonos para sopnorrizarla, podemos tener el siguiente esquema de localización estéreo:

- Bombo - Centro
- Caja - Centro
- Charles - 20 % R¹
- Tom 1 - 10 % R
- Tom 2 - 10 % L²
- Tom base - 20 L
- Aéreo L - 100 % L
- Aéreo R - 100 % R

Como vemos, al final se trata de simular en la mezcla la colocación natural de los instrumentos en el espacio, de manera que no se amontonen todo en el centro de la mezcla. Los porcentajes anteriormente expuestos son sólo un ejemplo de como pueden ir colocadas las panoramizaciones de cada elemento. Dicha tabla se puede modificar a criterio personal de cada uno siempre que ofrezca buenos resultados.

Para el resto de instrumentos se hará algo similar, colocando elementos principales como la voz o el bajo al centro, y abriendo otros como las guitarras y los coros a izquierda y derecha, de manera que tengamos un estéreo rico y amplio, pero siempre manteniendo lo principal de la canción en el centro de la mezcla.

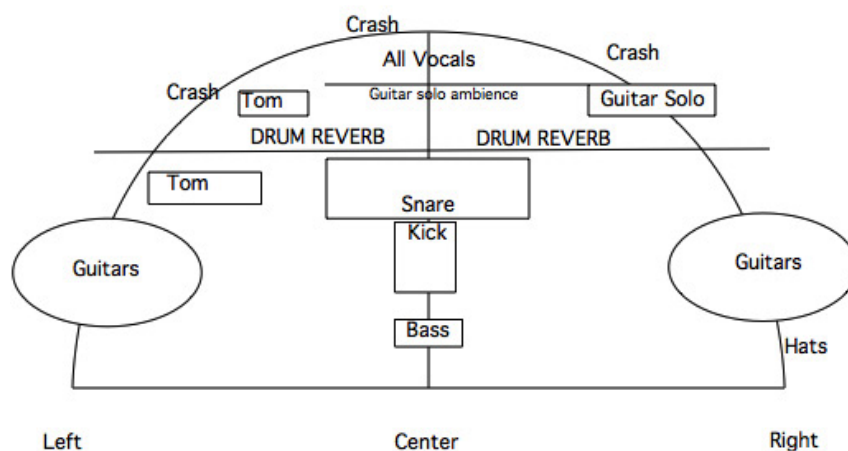


Figura 3.1: Ejemplo de colocación espacial de una banda estándar.[19]

¹R.- Right (Derecha).

²L.- Left (Izquierda).

Tal y como vemos en la fig. 3.1, se puede ver otro ejemplo de panoramización de los instrumentos. Además se aprecia que además de la panoramización, existe también una importancia relevante en la reverberación que se aplica a cada instrumento, así como su nivel en la mezcla. Ambos detalles ayudan a definir la cercanía del instrumento al oyente.

Por ejemplo, si nos situamos en una sala, a mayor distancia, el sonido se atenúa, pierden las altas frecuencias, y aumenta la sensación de reverberación debido a que nos llega una relación de sonido directo y reflejado diferente. Si la fuente se encuentra a escasos metros de nosotros, recibiremos en su mayoría sonido directo, y por tanto la sensación de reverberación será menor.

Con todo lo visto, se pretende realizar un programa que sea capaz de implementar todo lo explicado y que por tanto facilite el entrenamiento y la asimilación de los conceptos de planos sonoros, colocación espacial de la fuente de sonido, y, en definitiva, facilite al alumno el entendimiento del plano estéreo y la distancia de la fuente de sonido al oyente.

3.2.2 Reconocimiento frecuencial

El segundo de nuestros objetivos es realizar una aplicación enfocada en el espectro frecuencial del sonido. El usuario deberá ser capaz de poder afinar su oído desde un punto de vista frecuencial (que no musical). De esta manera, deberá ser capaz de aprender a reconocer frecuencias de oído, así como diferentes tipos de formas de onda para una misma frecuencia, etc.

Aunque de primeras pueda parecer bastante básico lo que se pide en este ejercicio, el objetivo final no es que el usuario se vuelva un maestro en reconocer frecuencias puras o diferentes tipos de onda, ya que éstas no se encuentran de manera natural en los sonidos acústicos, y por tanto, difícilmente va a enfrentarse en la vida a real a la necesidad de ser capaz de identificarlos. El objetivo final es que mediante este tipo de ejercicios y esta metodología, afine el oído progresivamente, de forma que llegue un momento en el que sea tal el nivel de afinación que el usuario sea capaz de identificar frecuencias fundamentales en un instrumento, fig. 3.2, por ejemplo, o sea capaz de determinar la frecuencia de un acople¹ en una aplicación de directo.

3.2.3 Ruido y distorsión

En el siguiente lugar, y uno de los parámetros que muy pocas veces se tienen en cuenta pero que pueden provocar verdaderos quebraderos de cabeza a un técnico son los ruidos y distorsiones. Estos suelen ser provocados por diversos motivos, todos ellos muy variados, como pueden ser interferencias electromagnéticas, malos conectores, cableado defectuoso, etc. Si además añadimos elementos inalámbricos en la cadena, el riesgo de que acaben apareciendo ruidos en la señal de audio aumenta considerablemente.

Por tanto se va a implementar un sistema de entrenamiento para ruidos aleatorios (procesos estocásticos), donde se podrá trabajar con ruido blanco y rosa para poder aprender a detectar éste tipo de ruidos en una señal de audio degradada, fig. 3.3. De esta forma, el usuario podrá entrenar la capacidad de detectar posibles degradaciones aleatorias provocadas por diferentes fallos en un sistema de audio.

Además, también nos pueden aparecer ruidos diversos relacionados con el audio digital, como bien puede ser ruido de cuantificación, el cual aparece al reducir el número de bits o resolución al digitalizar una muestra de audio analógica en un conversor ADC. Dicho ruido, si no es muy grande, como el que aparece en procesos de masterización al

¹ Acople.- También conocido en inglés como *feedback*, consiste en una realimentación acústica entre la micrófona y el sistema de refuerzo sonoro, manifestándose como un sonido creciente e intenso a una frecuencia determinada, denominada frecuencia de resonancia.

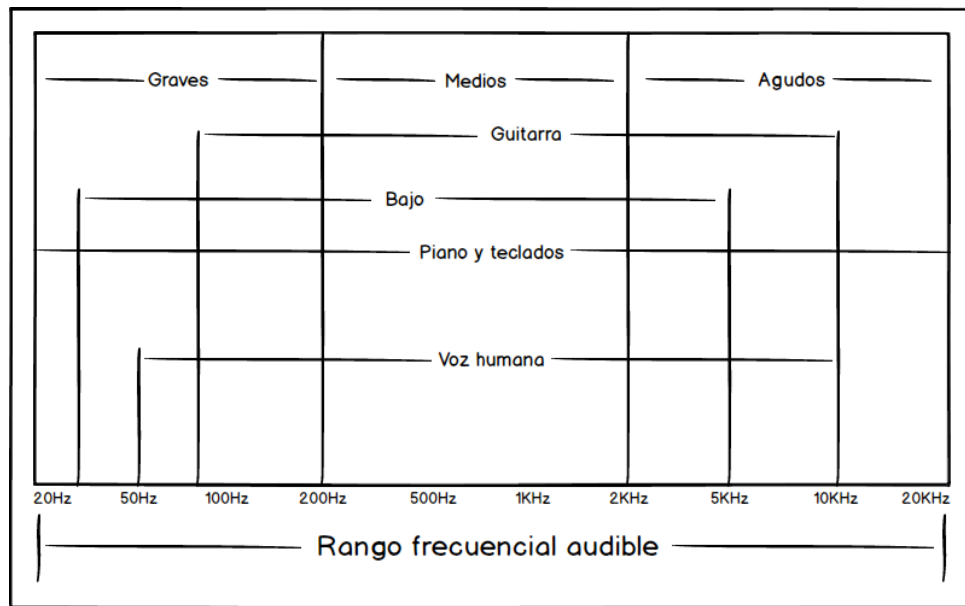


Figura 3.2: Instrumentos con su colocación frecuencial a lo largo del espectro audible.

reducir de 24 bits a 16 bits, se suele enmascarar con un *dither* digital. En cualquier caso, es interesante trabajar sobre este concepto con el objetivo de llegar a ser capaces de identificarlo de oído, de manera que podamos ser un instrumento más de detección y por tanto ser menos dependientes de herramientas externas.

Otro de los ruidos que suelen aparecer es el *aliasing*, causado por trabajar a una frecuencia de muestreo inferior a 44.100 Hz, que es el estándar mínimo al que se garantiza que la frecuencia de muestreo será mayor al doble de la frecuencia máxima audible. En el momento en que reducimos la frecuencia por debajo de la anteriormente citada, empieza a aparecer ruido de *aliasing*. Todo esto viene explicado en el teorema de Nyquist-Shannon¹.

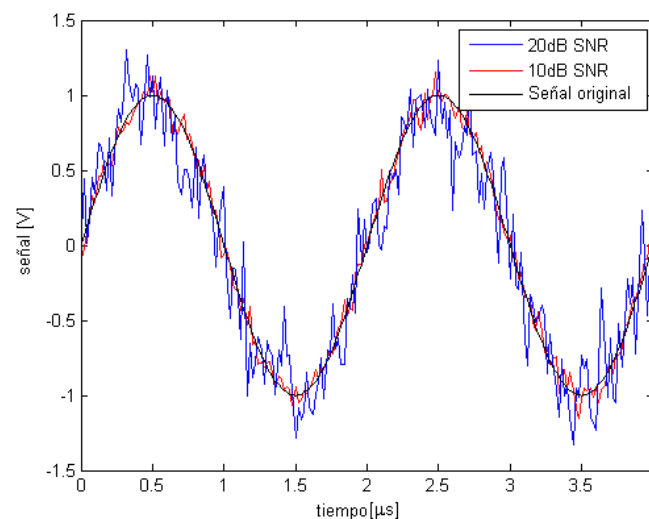


Figura 3.3: Ejemplo de una señal sin ruido blanco agregado, con ruido leve (10 dB SNR) y con ruido intenso (20dB SNR).

¹El teorema de Nyquist-Shannon dice que $F_s > 2 \cdot F_{max}$ siendo F_s la frecuencia de muestreo y F_{max} la máxima frecuencia comprendida en la señal.

3.2.4 Volumen y dinámica

Por último se pretende desarrollar una aplicación enfocada a uno de los parámetros más importantes del sonido, el volumen. Además, y para añadir alguna funcionalidad extra relacionada con el ámbito del volumen, se ha implementado la posibilidad de trabajar además la dinámica del sonido.

En primer lugar, y como base en la que se apoya la aplicación, está el tema del volumen. Se trata de algo primordial en el mundo del audio. Define la intensidad con la que un sonido es emitido o recibido y se trata de un parámetro importantísimo para un técnico o ingeniero de sonido. Para llevar a cabo por tanto el entrenamiento de volúmenes, se deberá afinar el oído para que éste llegue a ser más sensible a los cambios de presión, de manera que llegue a diferenciar cambios mínimos en el volumen, es decir, aumente su sensibilidad.

En segundo lugar, y tal y como se ha comentado previamente, se desea complementar el trabajo con volúmenes con el trabajo con la dinámica del sonido. Ambos parámetros están relacionados, o más bien el segundo depende del primero, y por tanto se ha decidido aunar ambos en una misma aplicación. A medida que la música ha ido evolucionando, ésta ha entrado en una especie de "*Loudness war*" o "Guerra de nivel", y es que cada vez existe menos rango dinámico en la música en pro de un mayor volumen. Por otro lado tenemos la música clásica, la cual posee una gran dinámica, y por tanto una capacidad enorme de crear diferentes pasajes sonoros y de crear en el oyente unas sensaciones muy marcadas. Se pretenda mezclar música clásica o moderna es importante trabajar éste aspecto, ya que actualmente es algo fundamental en cualquier mezcla que se precie y por tanto deberá trabajarse la capacidad para poder diferenciar y dominar la dinámica del audio a través de procesadores de efectos tales como compresores, puertas de ruido, etc.

CAPÍTULO 4

Diseño del sistema

En este capítulo nos vamos a meter de lleno en el desarrollo de las propias aplicaciones en sí, hablando un poco de las decisiones que se han tomado previas a empezar a escribir código propiamente dicho.

Estas decisiones son importantes ya que nos marcarán las pautas a seguir a la hora de programar. Serán las encargadas de definirnos factores tan importantes como el flujo de trabajo que seguirá el usuario mientras realiza los ejercicios, las indicaciones que deberemos darles para un correcto entendimiento del ejercicio así como un seguimiento de sus resultados, etc.

Además de hablar de cómo están constituidas las aplicaciones y del método de funcionamiento, se expondrán decisiones relevantes acerca de las decisiones tomadas acerca de la interfaz gráfica, etc.

4.1 Modos de trabajo

En primer lugar vamos a hablar de lo que se ha denominado “Modos de trabajo”. Apoyándonos en iZotope PAE, referente visto anteriormente en [2.4.4](#), se ha decidido implementar dos formas de trabajar dentro de cada una de las aplicaciones anteriormente vistas.

La decisión de implementar dos formas de trabajo diferentes ha sido el poder ampliar las funcionalidades de las mismas, a la par que fomentamos no sólo el entrenamiento propiamente dicho acerca de una materia concreta, si no también la posibilidad de entender y asimilar los conceptos y los parámetros que la definen.

4.1.1 Modo de prueba

El “Modo de prueba” pretende ser, para la gran mayoría de los usuarios, la puerta de entrada a cada una de las aplicaciones. La idea es que cuando un usuario arranque por primera vez una de las 4 aplicaciones que componen la *suite*, éste empiece por este modo.

El “Modo de prueba” permite al usuario, de una manera sencilla, experimentar de forma libre con todos los parámetros que definen el fenómeno al que va enfocado la aplicación, de manera que pueda observar y escuchar en qué afecta cada uno de dichos parámetros al sonido. De esta forma, podrá afianzar conceptos y entender cada uno de ellos.

Para una mayor comprensión del modo en el que nos encontramos, se va a ver a modo de ejemplo el “Modo de prueba” de la aplicación de “Reconocimiento frecuencial”, fig. [4.1](#), expuesta a continuación:

Tal y como vemos en el título superior de la fig. [4.1](#), estamos en la “Zona de pruebas” de la aplicación “Reconocimiento frecuencial”. En ella disponemos de todos los paráme-

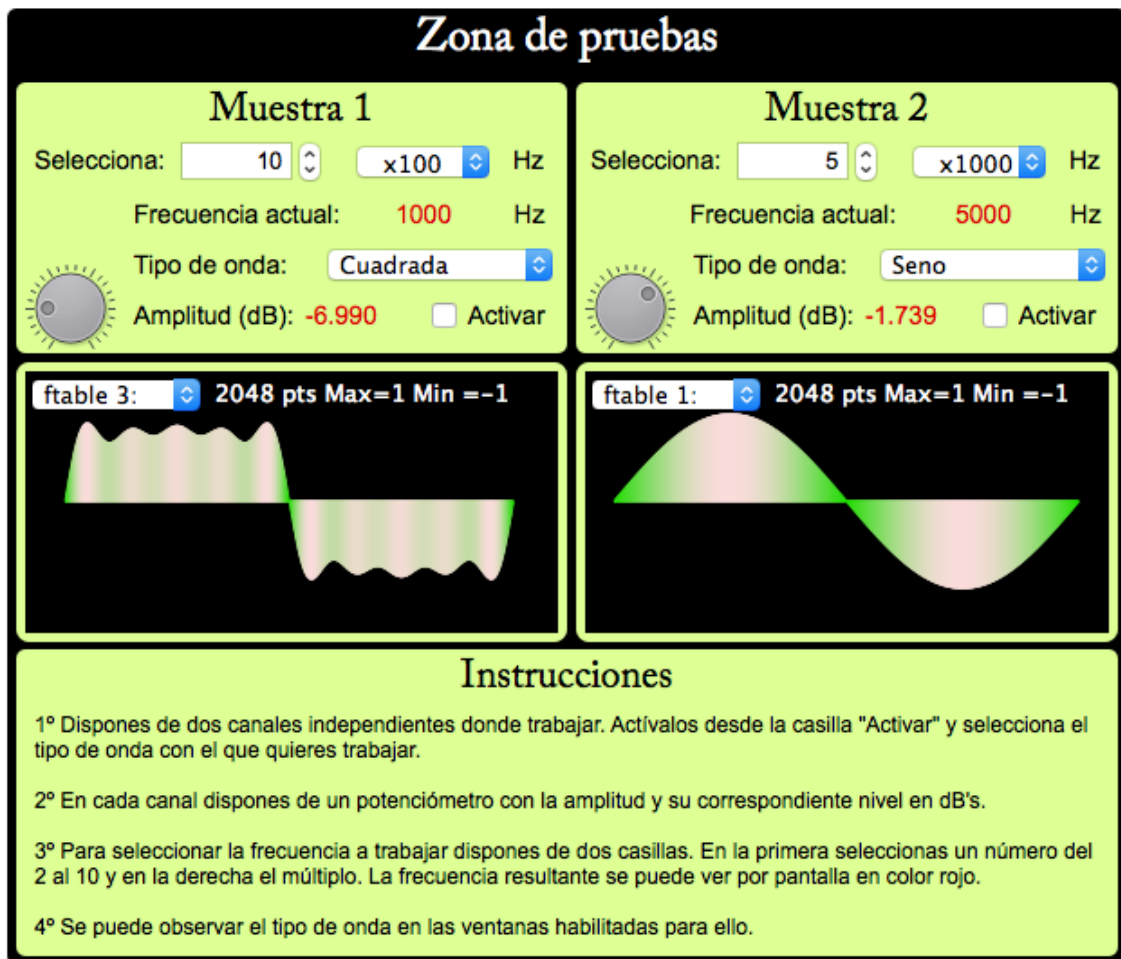


Figura 4.1: Ventana del “Modo de prueba” de la aplicación de “Reconocimiento frecuencial”.

tros con los que trabaja internamente la aplicación disponibles para el usuario. En ella el usuario puede, y debe, practicar con ellos de forma que llegue a entender en qué afecta cada uno de dichos parámetros al sonido.

De esta manera, podrá practicar para entender cómo funciona cada uno de dichos parámetros y en qué y cuánto afecta su modificación en el sonido resultante.

Por ejemplo, en la imagen podemos observar cómo disponemos de dos osciladores independientes, dónde es posible modificar parámetros tan diversos como la frecuencia de los mismos, el tipo de onda o la amplitud. Además, disponemos de *displays* para que el usuario pueda observar, además de escuchar, lo que está modificando.

4.1.2 Modo de entrenamiento

Como complemento al “Modo de prueba”, y parte principal en la que se sustenta cada una de las aplicaciones desarrolladas, está el “Modo de entrenamiento”. Aquí el usuario pondrá a prueba los conocimientos adquiridos en el “Modo de prueba” y deberá entrenar el oído para mejorar y afinar el mismo.

En esta zona, al contrario que en el “Modo de prueba”, no todos los parámetros están disponibles, ya que se trata de una zona en la que el usuario deberá “adivinar” dichos parámetros. Es decir, el programa generará unos parámetros aleatorios y desconocidos para el usuario. Dichos parámetros se aplicarán a una muestra de audio la cuál sonará

Zona de entrenamiento

Generar

Muestra 1

Amplitud (dB): -3.979 ☐ Activar

Muestra 2

Amplitud (dB): -6.198 ☐ Activar

Resultados

El número de fallos es de: 00

Respuestas - Muestra 1

Selecciona frecuencia:

Selecciona múltiplo:

Selecciona onda:

Respuestas - Muestra 2

Selecciona frecuencia:

Selecciona múltiplo:

Selecciona onda:

Instrucciones

1º En primer lugar pulsa "Generar" para generar una muestra aleatoria.

2º Tras esto, deberás decidir si trabajar con una o dos muestras simultáneas. Para ello activa la que desees.

3º Intenta adivinar de que frecuencia se trata, así como su múltiplo y el tipo de onda.

4º En el apartado "Resultados" puedes consultar en todo momento el número de fallos de cada intento. Si no eres capaz de adivinar los valores de cada muestra, pulsa en el botón "Soluciones" para visualizar los resultados.

Figura 4.2: Ventana del “Modo de entrenamiento” de la aplicación de “Reconocimiento frecuencial”.

por la salida convencional de audio del ordenador. El usuario deberá entonces adivinar de oído dichos parámetros.

Al principio será complicado y costoso, pero a medida que el usuario entrene con la aplicación y vaya puliendo su oído en el campo correspondiente, será capaz de interpretar mejor lo que escucha y adivinar que tipo de modificación ha sufrido la muestra de audio, es decir, lo que está sonando.

Como ejemplo, y al igual que en el apartado anterior, tenemos la aplicación de “Reconocimiento frecuencial”. En dicha aplicación, la “Zona de entrenamiento”, la cual se puede ver en la fig. 4.2. En ella se aprecia que ya no están disponibles todos los parámetros. Ahora el objetivo es que el usuario genere una muestra. Dicha muestra sonará y el usuario deberá adivinar que parámetros contiene y colocarlos en la zona de “Respuestas”.

De esta forma, y repitiendo el proceso, el oído estará cada vez más entrenado y será capaz de reconocer al instante qué está escuchando.

4.1.3 Visión general de diseño

Por tanto, y en vista de los dos modos de trabajo comentados, podemos observar el funcionamiento de cada uno como partes independientes y necesarias en el camino hacia

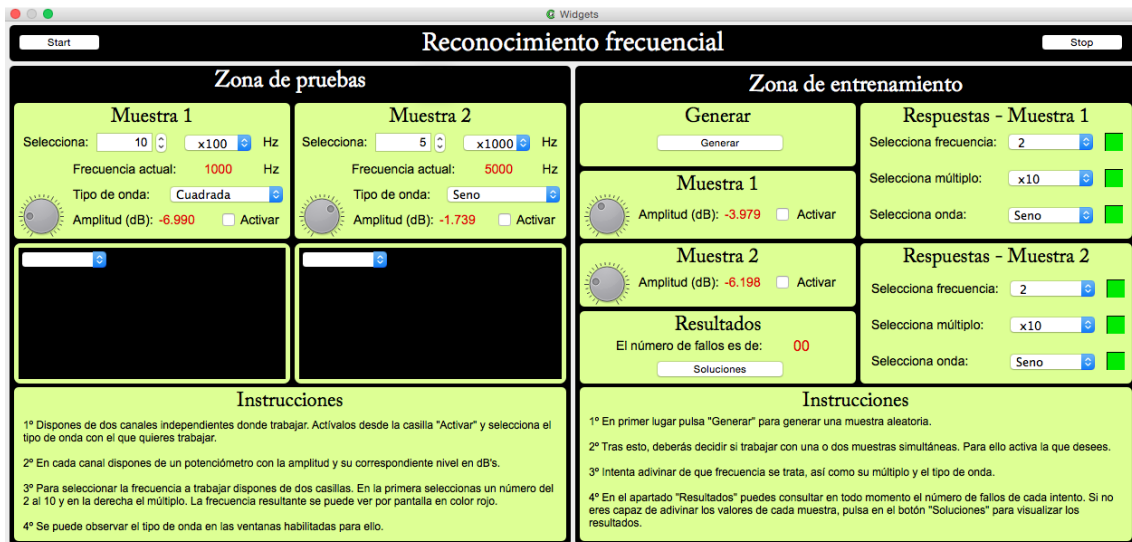


Figura 4.3: Vista general de la aplicación de “Reconocimiento frecuencial”.

el entrenamiento y la comprensión total de cada uno de los campos del audio contenidos en cada una de las aplicaciones.

El objetivo final es que el usuario pueda, en primer lugar, practicar y aprender, y en segundo lugar aplicar los conocimientos adquiridos para entrenar y afinar el oído.

Es fácil pensar entonces en los pasos a seguir por un usuario nuevo:

- 1º Arrancar la aplicación y leer las instrucciones del “Modo de prueba”.
- 2º Comenzar a practicar con dicho modo y observar cómo se comporta cada parámetro y en qué afecta dicha modificación al sonido resultante.
- 3º Una vez comprendidos todos los parámetros, procederá a leer las instrucciones del “Modo de entrenamiento”.
- 4º Comenzará el entrenamiento, repitiendo el proceso tantas veces desee hasta conseguir resultados satisfactorios en cada ejercicio.

4.2 Interfaz gráfica

Antes de comenzar con la explicación sobre nuestra propia interfaz gráfica, primero debemos explicar las herramientas con las que disponemos. Dichas herramientas están contenidas dentro de *CsoundQt*, el *front-end* de *Csound* elegido para este proyecto, tal y como se ha visto anteriormente.

Por ello vamos a ver en primer lugar dichas herramientas y a continuación una visión profunda y detallada sobre nuestro diseño gráfico propio.

4.2.1 Widgets

Una vez se decidió que se iba a utilizar *CsoundQt* en lugar de *Cabbage* como programa *front-end* para realizar nuestro proyecto, tal y como ya se vio en la sección 2.6.2, las posibilidades gráficas para interfaz se limitaron a lo que nos ofrecía únicamente *CsoundQt*.

CsoundQt gestiona la posibilidad de trabajar con elementos gráficos a través de unos elementos denominados *widgets*. Los *widgets* permiten al usuario interactuar con el programa en tiempo real y de una forma gráfica e intuitiva, lo cual constituirá la interfaz gráfica del programa o GUI¹.

CsoundQt dispone de un total de 14 *widgets*, todos ellos editables gráficamente a través de sus propiedades. Además, una de las mayores ventajas es que *CsoundQt* no añade código extra por cada *widget*, simplificando así el código y dejando la atención del programador únicamente en el programa, y no en la interfaz, la cuál es editable de manera gráfica desde la ventana *widget*.

Existen varios tipos de *widget* dependiendo de la función que realizan. Por ejemplo, existen *widgets* de salida de texto (*Label* y *Display*), de entrada de texto (*Line Edit*), entrada numérica (*Slider*, *Knob*, *Scroll Number* o *Spin Box*), botones (*Button*), menús desplegables (*Menu*), además de diversos elementos gráficos que nos facilitan el desarrollo de una interfaz gráfica completa y actual.

4.2.1.1 Presets

Otra funcionalidad que nos ofrece *CsoundQt* es la poder crear, guardar y cargar *presets* desde la propia interfaz gráfica. Ahora bien, primero vamos a ver que es un *preset*.

Un *preset* es una configuración con unos valores determinados de uno o varios *widgets*. Es decir, podemos guardar una configuración concreta de los *widgets* que deseemos para poder así cargar dicha configuración cuando deseemos.

Para gestionar los *presets*, *Csound* hace uso de una serie de canales reservados para gestionar la lectura de los mismos, por ejemplo.

Para poder crear un *preset* en *CsoundQt* primero debemos crear todos los *widgets* que queremos que formen parte de él. Una vez tengamos todos creados, debemos colocarlos en la posición que deseemos que se guarden. Una vez tengamos todo colocado como deseamos, deberemos pulsar en la ventana *widgets* con el botón derecho del ratón, y a continuación en la opción *Nuevo Preset*. Una vez se nos habrá la ventana, deberemos elegir un nombre y un número que identifiquen al *preset*, y de esta forma ya tendríamos almacenado nuestro *preset*.

Para cargar un *preset* de manera sencilla, podremos volver a pulsar con el botón derecho sobre la ventana *widgets* y a continuación en la opción *Cargar preset*. Se nos abrirá una ventana en donde podremos seleccionar con doble click sobre el *preset* que deseemos cargar.

Ahora sí, hablaremos sobre los canales reservados para el uso de *presets*. Un canal reservado es, en esencia, un identificador (el cual viene precedido por el símbolo “_”) que implementa una función específica y propia de *CsoundQt*.

Por tanto, si creamos un *widget* y le colocamos en su nombre de canal un canal reservado de *CsoundQt*, éste realizará una función específica asociada a dicho canal reservado.

Existen 3 canales reservados en *CsoundQt* relacionados con el uso de *presets*, los cuáles se describen a continuación:

- `_GetPresetName`: Obtiene el nombre del *preset* que está cargado en ese momento. Se utiliza por ejemplo para mostrarlo a través de un *display* y que el usuario sepa en todo momento sobre que *preset* está trabajando.
- `_GetPresetNumber`: Similar al anterior pero en vez de devolver el nombre, devuelve el número asociado al *preset* en uso. Por tanto, en vez devolver una variable tipo *String* la devuelve de carácter numérico.

¹GUI: Graphical Unit Interface.

- `_SetPresetIndex/_SetPreset`: Establece el número de *preset* que se va a cargar. Es muy útil para utilizarlo por ejemplo con un *spin box* y que el usuario pueda elegir que *preset* desea cargar directamente desde la interfaz gráfica, y sin necesidad de abrir el menú de carga de *presets* anteriormente explicado.

4.2.2 Nuestro diseño

Una vez vistos los aspectos técnicos y las herramientas que nos facilita *CsoundQt* vamos a comentar las decisiones tomadas acerca de nuestra interfaz gráfica(GUI): estructura general, gama de colores y una breve visión general que nos de una idea general de cómo se estructuran y componen nuestras aplicaciones.

4.2.2.1 Estructura general

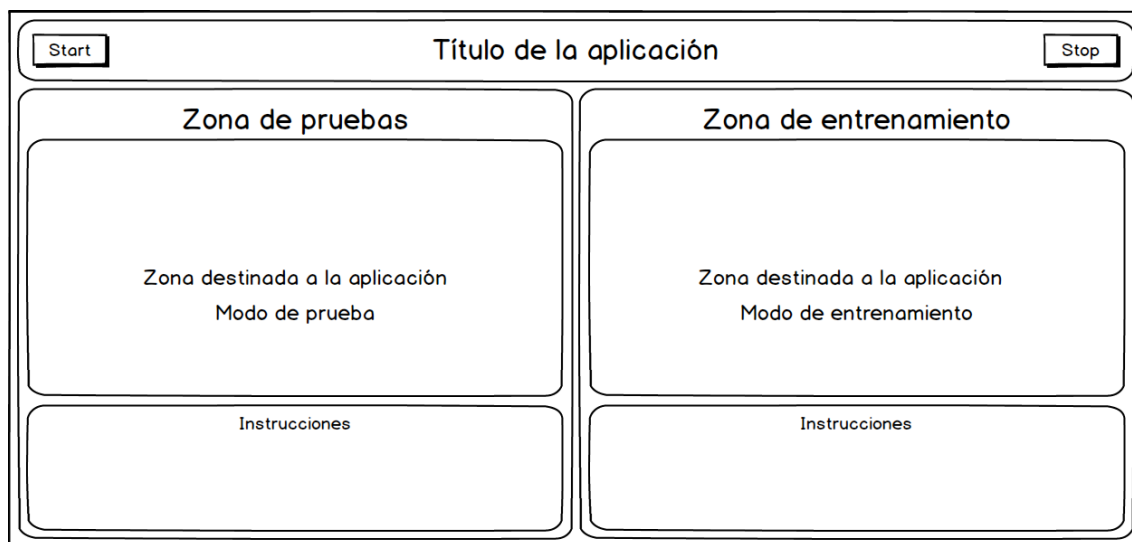


Figura 4.4: Esquema de la estructura general asociada a las 4 aplicaciones.

Tal y como vemos en la fig. 4.4, se ha diseñado una estructura general y clara sobre la que irán diseñadas cada una de las 4 aplicaciones que componen nuestra *suite*. Dicho esquema¹ define claramente las secciones y los botones comunes a todas las aplicaciones.

Empezando por la parte superior observamos una barra general que contiene un botón llamado *Start*, el título de la aplicación en cuestión y otro botón complementario con el rótulo de *Stop*. Ambos botones gestionan el arranque y la detención de *CsoundQt* a través de dos canales reservados, de manera que el usuario no tenga que tocar la interfaz de *CsoundQt*, ya que puede ser autónomo directamente desde nuestra interfaz. Dichos canales reservados son `_Play` y `_Stop`, y nos facilitan el poder arrancar y detener el programa desde un *widget* de nuestra interfaz.

A continuación, y separadas claramente por dos secciones independientes, tenemos la “Zona de pruebas” y la “Zona de entrenamiento”, comentadas anteriormente en la sección 4.1. Tal y como vemos en el esquema, ambas zonas están compuestas por su rótulo identificativo en la parte superior, una amplia zona destinada directamente a implementar cada una de las secciones y por último, pero no menos importante, encontramos en la parte inferior las “Instrucciones” pertenecientes a cada sección.

¹Ha sido diseñado con el *software* **Balsamiq Mockups 3**.

4.2.2.2 Gama de colores

Para la paleta de colores, se mantuvieron el blanco y el negro como colores neutros, ayudando así a crear las secciones dentro de la ventana general, así como para las tipografías. Para añadir la personalidad se utilizó un color amarillo verdoso claro como color principal, el cual, acompañado por el negro y el blanco anteriormente citados, crean una combinación bastante elegante y sin ser demasiado agresiva, fig. 4.5.

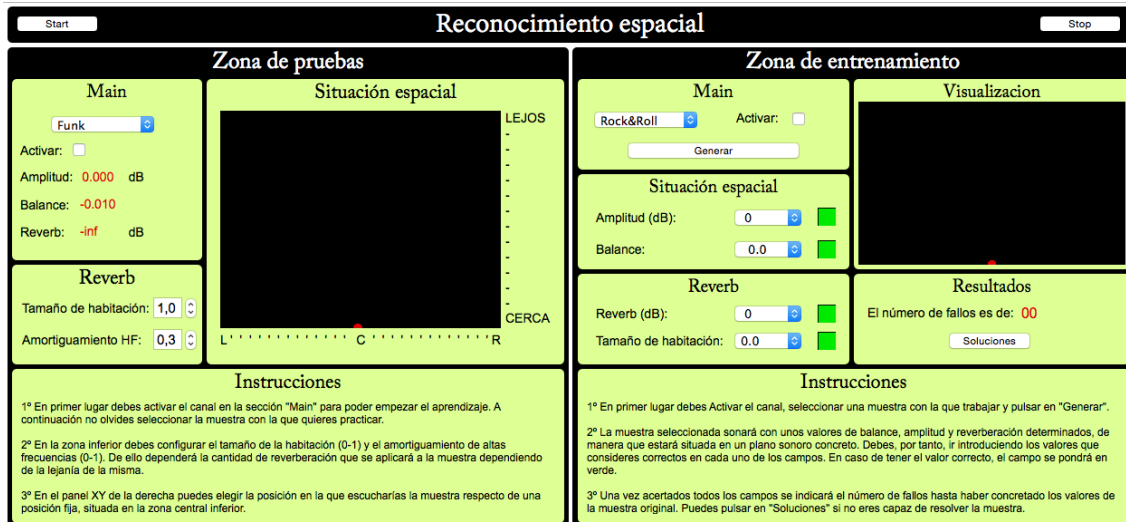


Figura 4.5: Ejemplo de diseño final, con la composición de colores y tipografías definitivas.

Como podemos observar, la combinación de colores se hace bastante cómoda a la vista, aún con el paso del tiempo, apoyada siempre por una tipografía agradable para los títulos y estándar para el resto. Además, el contraste que genera el color amarillo claro con la tipografía negra y roja (para los *displays* únicamente) hace que la lectura sea sencilla y rápida, algo importante de cara la experiencia de usuario.

4.3 Evaluación de los ejercicios

En esta sección hablaremos cómo se esperan evaluar los resultados obtenidos por el usuario dentro de cada aplicación. Esto es importante porque no debemos olvidar que el presente proyecto consiste en desarrollar una *suite* de aplicaciones para el entrenamiento auditivo, es decir, para afinar y mejorar el oído. Dicha mejora se debe valorar de una manera subjetiva por el usuario pero, ¿cómo evalúa el usuario sus resultados dentro de la aplicación?

Inicialmente se pensó en implementar un sistema de niveles de dificultad progresivos similar a los que se encuentran en cualquier juego *software* de pantallas. El problema vino por las limitaciones a nivel gráfico que nos encontramos en *CsoundQt* y *Csound*. *Csound* como lenguaje de programación está enfocado en todos sus aspectos al audio, y si bien en éste apartado es muy potente y saca a relucir todas sus virtudes, no está pensado para otras tareas como la programación visual, donde destacan otros lenguajes mucho más potentes y avanzados.

Por tanto, y llegados a este punto, se desechó la idea de implementar niveles de dificultad, por lo que se buscó una alternativa con un compromiso entre utilidad y sencillez de implementación. Para ello se pensó inicialmente en uno de los factores principales: a quién van enfocadas las aplicaciones. Tal y como se ha mencionado anteriormente en repetidas ocasiones, se ha enfocado a un usuario técnico y con ciertos conocimientos pre-

vios sobre audio, lo que nos limita inicialmente a un usuario adulto, es decir, no se espera que un usuario infantil haga uso de las mismas. Partiendo de esta premisa se ha decidido implementar un sistema de auto evaluación.

4.3.1 Auto evaluación

El concepto de “Auto evaluación” viene habitualmente asociado a campos como la pedagogía, ya que es en ésta donde saca especialmente a relucir todas sus virtudes. Es aquí donde se deja que el alumno sea quien evalúe y juzgue sus progresos respecto a una determinada materia. Entran por tanto en juego valores como la honestidad, la auto crítica y la auto superación[20]. Tal y como hemos visto, el público que se espera es un público adulto, con la suficiente sensatez como para ser honestos consigo mismos, por lo que se espera que sea el alumno quien desee realizar los ejercicios de manera correcta y sin trampas, y a su vez evalúe su trabajo y mejoría.

Para que todo esto se pueda lleva a cabo de una manera eficiente y adecuada, se han implementado una serie de herramientas en cada una de las aplicaciones, buscando así que el usuario pueda controlar su progreso.

En primer lugar se ha implementado un contador de fallos. Dicho contador se inicia a 0 al arrancar la aplicación y, además, se *resetea* a 0 de nuevo cada vez que pulsamos en el botón “Generar” para generar una nueva muestra de entrenamiento. De esta forma, contará los fallos que tengamos en cada una de las muestras del entrenamiento. Debe ser por tanto el usuario quien controle su número de fallos y decida bajarlo a base de entrenamiento, pudiendo alcanzar rondas de 0 fallos de manera habitual una vez haya alcanzado un nivel alto en cada una de las aplicaciones.

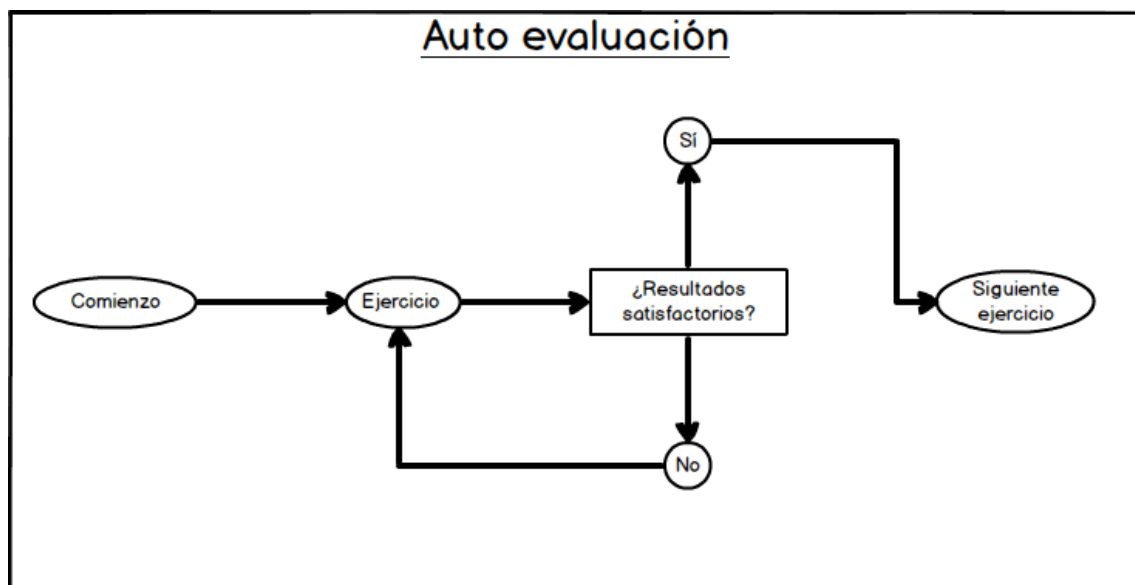


Figura 4.6: Esquema del flujo de trabajo mediante auto evaluación.

En caso de que una de las muestras se resista mucho por algún motivo y no seamos capaces de acertar alguna de las respuestas, se ha implementado un botón “Soluciones”, donde si se pulsa se colocará automáticamente en la zona de respuestas la respuesta correcta a cada parámetros. De esta manera, cuando el usuario comience a entrenar y no sea capaz de acertar alguno de los parámetros, podrá saber cual es la respuesta para poder mejorar.

La idea final es que sea el usuario quien se auto evalúe a base de rondas de entrenamiento, buscando siempre la auto superación y la perfección en cada uno de los ejercicios antes de pasar al siguiente, tal y como se observa en el flujo de trabajo de la fig. [4.6](#).

CAPÍTULO 5

Implementación

Llegados a este punto ya podemos asumir que tenemos un sistema diseñado y con unas especificaciones claras y concisas, por lo que se deduce que a llegado el momento de implementarlo. Para su implementación, tal y como se ha mencionado anteriormente en el apartado 2.6.2, se utilizará *CsoundQt* como herramienta de desarrollo.

Antes de comenzar a describir la implementación específica de cada una de las cuatro aplicaciones se pretende dejar claro que la secuencia de desarrollo no fue directa, es decir, no se diseñó todo el sistema de aplicaciones y luego se desarrolló, si no que se fueron complementando ambos procesos de manera paralela. De esta forma de trabajo, podíamos ir desarrollando la siguiente aplicación habiendo aprendido de los errores cometidos en la anterior y habiendo aprendido nuevas técnicas, pudiendo abrir las posibilidades. Por tanto, la secuencia de desarrollo fue en primer lugar la aplicación de “Reconocimiento Espacial”, seguida de “Reconocimiento Frecuencial”, tras la cuál se desarrolló una primera versión de “Ruido y Degradación” a la que siguió “Ganancia y Dinámica”. Llegados a este punto se realizaron modificaciones notables a “Ruido y Degradación” para añadir funcionalidades a la misma, recomendadas por el tutor.

Por tanto, y vista la secuencia de trabajo seguida durante todo el desarrollo, vamos a ver la implementación de cada una de las cuatro aplicaciones en orden cronológico a su desarrollo.

5.1 Visión general

Para una mayor sencillez a la hora de escribir cada uno de los programas y facilitar así su entendimiento, se ha definido previo a su desarrollo una estructura general de composición para el código, de manera que cada uno de los cuatro programas contengan la misma estructura general.

De esta manera, y al tratarse de bloques cerrados, estos son muy similares entre sí y nos facilita la declaración de cada uno de ellos. Es importante dejar claro que la estructura es algo general y que por tanto está sujeta a modificaciones dependiendo de la necesidad de las mismas. Por ejemplo, en el esquema general no se han definido los bloques dentro de cada instrumentos, los cuáles se verán más adelante con más detalle. Únicamente se pretende ver cómo se van a componer cada una de nuestras aplicaciones, de forma que podamos tener una idea previa clara sobre lo que es cada bloque o instrumento dentro de nuestro código.

En primer lugar vamos a tener una cabecera en la cuál vendrá definida la información correspondiente a nuestra aplicación, así como la declaración de los *flags* de *Csound* dentro del bloque *CsOptions*. Tras esta parte se abre el bloque *CsInstruments*, en el cuál vendrán definidas en primer lugar los parámetros de audio encargados de definir nues-

tro proyecto, como son la frecuencia de muestreo (FS), el número de muestras por periodo de control, el número de canales o si hubiera normalización a 0 dB.

A continuación definiremos e inicializaremos las variables globales que utilizarán durante el programa. Es importante realizar este paso ya que *Csound* funciona de una manera diferente a lenguajes como C o C++. En *Csound* el programa está ejecutando el código de los instrumentos en bucle, por lo que las variables que declaremos dentro de éstos se *resetearán* en cada iteración. Para evitar eso debemos definir las como variables globales fuera de los instrumentos.

Tras la definición de las variables globales se declara el instrumento 10. Dicho instrumento es un tanto especial ya que no manejará nada de audio, únicamente será el encargado de hacer de puente de comunicación entre los *widgets* de la *GUI* y el programa. Se ha decidido realizarlo de esta forma por simplicidad en la programación y para conseguir un control mayor a partir de un código más limpio y ordenado. Es importante recordar que éste instrumento también debemos activarlo en la partitura (*CsScore*).

Después le siguen la declaración de los instrumentos 1 y 2, los cuáles definen la “Zona de pruebas” y la “Zona de prácticas” respectivamente. En algún caso particular como en el programa “Reconocimiento frecuencial” se ha necesitado definir 2 instrumentos para cada una de las zonas nombradas, pero la estructura general es similar a la explicada, por lo que se deja como algo meramente anecdótico.

```

1 ; CABECERA
2 <CsoundSynthesizer>
3 <CsOptions>
4 ; OPCIONES GENERALES DE CSOUND
5 </CsOptions>
6 <CsInstruments>
7
8 sr = 44100 ; FS
9 ksmps = 128 ; NÚMERO DE MUESTRAS POR PERIODO DE CONTROL
10 nchnls = 2 ; NÚMERO DE CANALES
11 Odbfs = 1.0 ; NORMALIZACIÓN DEL NIVEL 0dB
12
13 ; DECLARACIÓN E INICIALIZACIÓN DE VARIABLES GLOBALES
14
15 instr 10 ; INTERFAZ GRÁFICA GUI
16 ; INSTRUMENTO 10 ES EL ENCARGADO DE LA COMUNICACIÓN GENERAL ENTRE LOS
17 ; WIDGETS GRÁFICOS Y EL PROGRAMA.
18 endin
19
20 instr 1 ; SECCIÓN DE PRUEBA
21 ; INSTRUMENTO 1 DEFINE LA SECCIÓN DE PRUEBA.
22 endin
23
24 instr 2 ; SECCIÓN DE PRÁCTICAS
25 ; INSTRUMENTO 2 DEFINE LA SECCIÓN DE PRÁCTICAS.
26 endin
27
28 ; INSTRUMENTOS ADICIONALES SI FUESEN NECESARIOS
29 </CsInstruments>
30 <CsScore>
31 ; DEFINICIÓN DE TABLAS DE ONDA
32 ; INICIALIZACIÓN DE LOS INSTRUMENTOS
33 i 1 0 10000
34 i 2 0 10000
35 i 10 0 10000
36 e
37 </CsScore>
38 </CsoundSynthesizer>

```

Por último se define la partitura en la que vienen activados cada uno de los instrumentos definidos. Se activan en el instante 0 y se mantienen durante un largo periodo de tiempo de manera que difícilmente se vaya a detener mientras el usuario trabaja con el.

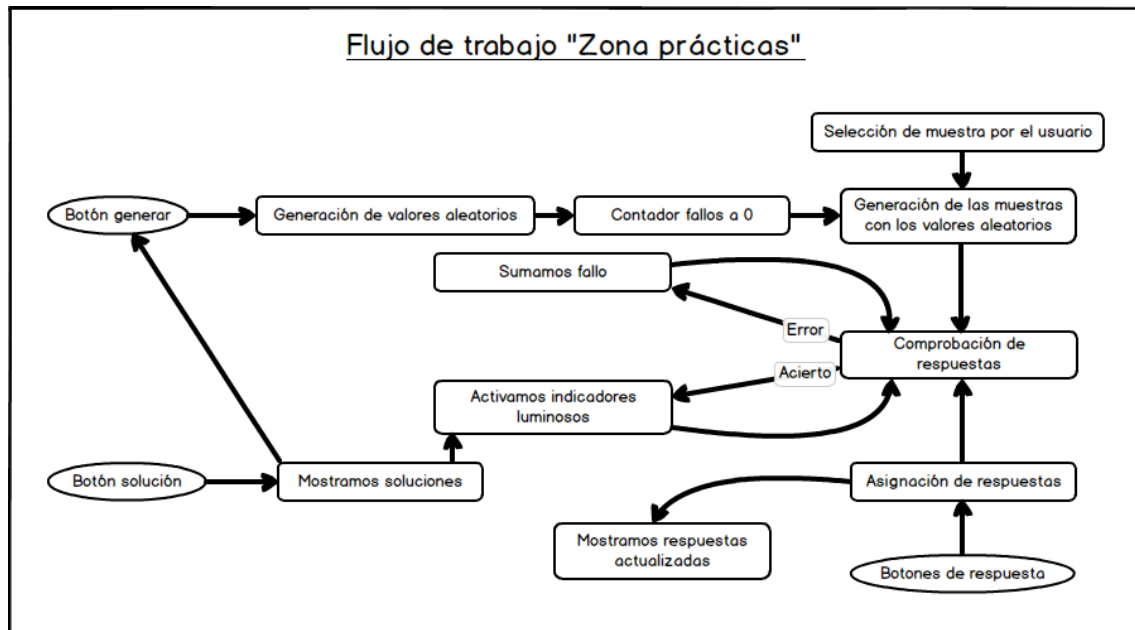


Figura 5.1: Esquema del flujo de funcionamiento dentro de la “Zona de prácticas”.

Por otro lado, en la fig. 5.1 podemos observar el flujo de trabajo general dentro de cada una de las aplicaciones, siendo éste flujo para la “Zona de prácticas”. El esquema para la “Zona de pruebas” es abierto, ya que el usuario podrá modificar cada parámetro de manera independiente, por lo que no se ha considerado mostrarlo por no poseer ninguna utilidad. Por tanto, el esquema mostrado en la figura y perteneciente al flujo de trabajo dentro de la “Zona de prácticas” será general y común a cada una de las cuatro aplicaciones, asentando así una base de trabajo estándar que sea común a todas ellas.

5.2 Reconocimiento espacial

Llegados a este punto es hora de adentrarnos de lleno en la implementación de cada una de las aplicaciones. En primer lugar, y para seguir con la cronología que se ha mencionado anteriormente, vamos a hablar de la aplicación Reconocimiento espacial”.

Empezó como la primera en ser desarrollada más bien por pura casualidad, una debía ser la primera elegida y fue esta. Se punto de atención es el enorme *PAD XY*, fig. 5.2, de la sección de pruebas, el cuál se utiliza para “mover” el sonido por el espacio y escuchar cómo se comporta éste al sufrir dicha alteración. Una de las características que definen esta aplicación es su sencillez a la hora de trabajar.

5.2.1 Zona de pruebas

En esta sección vamos a entrar de lleno en el funcionamiento y los objetivos a los que está enfocada la “Zona de pruebas” dentro de la aplicación “Reconocimiento espacial”. Para ello iremos viendo cada una de sus secciones con detenimiento y comentando tanto sus características, como su funcionamiento y cómo se han desarrollado.

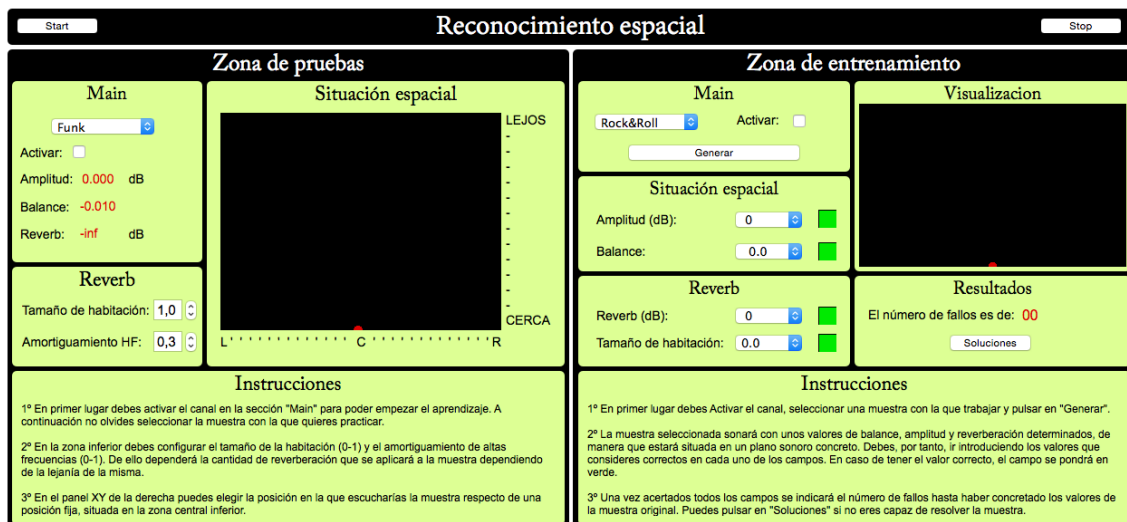


Figura 5.2: Vista general de la aplicación “Reconocimiento espacial”.

5.2.1.1 Main

En esta sección encontramos la información principal y más importante a la hora de trabajar en la “Zona de pruebas”. Podemos elegir con que tipo de audio deseamos trabajar, pudiendo elegir entre una lista de lo más variada (Rock&Roll, Instrumental, Locución, Funk, Cine y muestra Live). Para desarrollar este menú desplegable se ha hecho uso del *widget Menu* y una serie de condicionales *if* para detectar que muestra ha sido elegida por el usuario y su asignación a una variable de audio concreta. Para las muestras hace uso de varios archivos *wav* los cuáles deben estar contenidos dentro de la carpeta del proyecto. Además hay varios detalles que comentar. Por ejemplo, para cargar los archivos se utilizó la función *soundin*, la cuál reproduce en bucle la muestra seleccionada, facilitando así el trabajo al usuario, ya que de esta manera no dejará al programa nunca en silencio. Además, y como funcionalidad extra, se ha añadido la posibilidad de trabajar con el audio entrante por el micrófono del ordenador con el que se está trabajando. Es una funcionalidad extra que se decidió añadir aún con el ligero problema de tener una latencia de trabajo considerable.

Por otro lado nos encontramos un con la posibilidad de activar o desactivar la “Zona de pruebas” desde un *checkbox*. Dicho *checkbox* vale 1 o 0 dependiendo de si está o no activado, y a través de un sencillo condicional *if* podemos activar o no la salida de audio, activando así la sección.

Para terminar tenemos una serie de *displays* que nos muestran los valores de amplitud, balance y cantidad de *reverb*. Además, se consideró interesante la opción de implementar la cantidad de *reverb* y la amplitud en formato logarítmico, es decir, en decibelios. Esto viene justificado por el factor de que a la hora de trabajar en una mesa o programa de audio, siempre se trabaja en éste formato, y por tanto se acerca mucho más a la realidad que si fuese en escala lineal.

5.2.1.2 Reverb

En esta ventana podemos modificar dos parámetros característicos de la *reverb* que se aplica al sonido a medida que éste se aleja de la fuente. Para entender esto es fácil pensar en una habitación en la que, si estamos cerca de la fuente sonora, la cantidad de sonido directo que nos llega será mucho mayor que la de sonido reflejado. A medida que nos alejamos de la fuente, ésta relación disminuye y por tanto la sensación de sonido reverberado será mayor.

Para ello se ha implementado una *reverb* sencilla en la que únicamente podemos modificar dos parámetros: el tamaño de la habitación y el amortiguamiento en altas frecuencias.

Para implementar la reverberación se ha llevado un proceso similar a cómo se trabaja en audio profesional. La reverberación no se aplica directamente en el audio, si no que éste se manda a un canal independiente y se genera dicha reverberación. De ésta manera aumentamos la cantidad de la misma únicamente aumentando el nivel de dicho canal. En nuestro caso en vez un canal se ha creado un instrumento aparte, el cuál es el encargado de generar la reverberación. Dependiendo del nivel de señal que le enviemos, el nivel de la reverberación será mayor o menor.

Para la salida de audio utilizamos el opcode `out`, teniendo en cuenta en todo momento la cantidad de panoramización que debemos aplicar a cada uno de los dos canales estéreo, para lo cual utilizamos la fórmula `out aSal*(1-gkstereo), aSal*gkstereo`, asegurando así el estéreo de manera correcta a la salida de audio. Tras esto, gestiona la cantidad de reverberación, la cual será directamente proporcional a lo lejos que estemos de la fuente. Una vez esto, genera una variable global llamada `gaRvbSend`, que será el envío de señal que hacemos a la reverberación, la cual está en otro instrumento.

Al final se trata de tener una señal *seca*¹ sin reverberación y otra que contenga la reverberación. De esta manera jugaremos con los niveles de ambas para añadir más o menos cantidad de reverberación a la salida general de audio.

El instrumento que define la *reverb* recibe los parámetros de la sección “Reverb” de la interfaz gráfica, y con ellos y la variable global explicada anteriormente, genera la reverberación y la saca por la salida de audio. Además, debemos tener en cuenta el estéreo a la hora de sacar el sonido por la salida de audio, tanto para la señal *wet* como para la señal *dry*.

5.2.1.3 Situación espacial

En esta sección se encuentra la parte más importante y característica de la aplicación, el *PAD XY*. Desde este *widget* podemos tener control de 2 variables diferentes a la vez, en nuestro caso serán la amplitud y el balance. Además, como la reverberación será inversamente proporcional a la amplitud, se modificarán 3 variables de manera simultánea desde éste panel.

A través de una serie de marcas, podemos marcar al usuario dónde está el punto central del estéreo, así como la zona más cercana y más lejana a la “fuente sonora”, tal y como se puede observar en la fig. 5.2. Por tanto, moviendo el punto del *PAD*, el cuál simboliza al oyente, podremos escuchar cómo varía el sonido a lo largo de la “habitación” ficticia.

5.2.2 Zona de prácticas

5.2.2.1 Main

En la sección “Main” nos encontramos, al igual que en la “Zona de pruebas”, con la posibilidad de elegir la muestra con la que queremos trabajar, así como la opción para activar o desactivar el entrenamiento. Además, se ha incluido el botón “Generar”. A través de este botón, el usuario será capaz de generar las muestras de entrenamiento y resetear los menús del ejercicio anterior.

¹En las reverberaciones profesionales, a la señal sin reverberación se le llama *seca* o *dry*, mientras que a la señal con la reverberación se le denomina *húmeda* o *wet*.

Para llevar a cabo la generación, se incluye un sencillo condicional `if`, mediante el cuál se entra dentro si el botón se pulsa. Una vez dentro, y a través de las funciones `randh`, `abs` y `round`, somos capaces de generar números aleatorios, positivos y redondeados a la unidad. Estos números nos definirán, en función de su valor, la amplitud, balance, etc.

La generación de los valores aleatorios se realiza una vez para cada uno de los parámetros que el usuario va a tener que acertar, de manera que éstos serán aleatorios en cada generación nueva. Además, se resetean los menús y el contador de fallos para poder empezar una nueva muestra cada vez que se pulse en “Generar”.

Como detalle, destacar que el amortiguamiento en altas frecuencias de la reverberación se mantiene fijo a 0.5, de manera que sea neutro. Esto se hace por la enorme dificultad de poder acertarlo en una muestra aleatoria. Al valorar al principio que se hacía muy difícil su acierto, se decidió colocarlo fijo y que no fuese necesario acertarlo en los ejercicios.

5.2.2.2 Situación espacial

Esta sección corresponde a la zona de respuestas para la amplitud y el balance. En ella, el usuario deberá responder en cada nueva muestra aleatoria cuál cree que es la amplitud y el balance de la muestra que está escuchando. Si acierta, un indicador luminoso de color verde le indicará la validez de su respuesta. En cambio, si falla, un fallo se añadirá al contador de fallos.

Para el sistema de validación de respuestas, en primer lugar debemos colocar un condicional `if`, el cual comprueba si la respuesta del usuario es igual a la muestra aleatoria generada. Si son iguales, pone la variable de control `gkValidA` a 1, la cual se usará más adelante para mostrar el indicador luminoso al usuario. Por contra, si no son iguales, coloca la variable a 0 para que el indicador siga apagado y continúa a un nuevo condicional. Dicho condicional comprueba en primer lugar si la respuesta del usuario es igual a una variable auxiliar llamada `gkres1`, la cual contiene la respuesta del usuario en la interacción anterior del programa. Si son iguales, quiere decir que no ha cambiado su respuesta a pesar de estar mal, y por tanto no suma ningún fallo. En cambio, si no son iguales, significa que éste sí que ha cambiado su respuesta y sigue estando mal, por lo que ha cometido un nuevo fallo y por tanto debemos añadirlo al contador.

El funcionamiento para el resto de comprobaciones en las respuestas del usuario funcionan exactamente igual a la explicada, y por tanto no se considera necesario hacer una explicación particular para cada uno.

5.2.2.3 Reverb

Al igual que en la sección anterior, esta también es una sección para las respuestas del usuario. La diferencia radica en que en esta zona el usuario deberá responder la cantidad de reverberación y el tamaño de la habitación que escucha en la muestra aleatoria. El funcionamiento de los indicadores luminosos y el sistema de fallos funciona igual que en el apartado anterior.

5.2.2.4 Visualización

Esta sección se añadió como un apoyo visual. Lo único que hace es mostrar en un *PAD XY* similar al de la “Zona de pruebas” las respuestas que el usuario coloca en los menús de la sección “Situación espacial” de respuestas. De esta manera, puede hacerse una idea gráfica de dónde está situado el oyente en la sala.

La modificación manual del *PAD* no afecta a ningún parámetro del programa y por tanto únicamente funciona como *widget* de salida, y no de entrada.

5.2.2.5 Resultados

En esta sección, y tal y como aclara su nombre, tenemos los resultados de cada ejercicio. Para ello, en primer lugar tenemos un *display* que nos muestra el número de fallos de cada ejercicio. Para contabilizar los fallos se utiliza el sistema explicado en la sección 5.2.2.2. Al llegar al final de una vuelta del código, se almacenan en las variables auxiliares de control el valor de la respuesta del usuario para cada uno de los menús de repuestas. De esta manera, podremos comprobar en la siguiente vuelta si dicha respuesta ha sido modificada, y en función de eso, sumar un fallo al contador, tal y como ya se ha mencionado previamente. A continuación, y mediante la función *sprintfk*, convertimos la variable *gkfallos*, la cual tiene el formato 1.000, a una cadena de texto con el formato 1. De esta manera, a la hora de mostrar el número de fallos, éste será un número entero y sin decimales, los cuáles no nos interesan a la hora de dar un número de fallos.

Por último, y para terminar con la sección y la aplicación, nos encontramos con el botón “Resultados”. Si el usuario pulsa en el, éste devuelve a los menús las repuestas a cada variable. Se ha facilitado dicho botón para ocasiones en las que no se consiga acertar alguna de las respuestas, poder conocer su solución y aprender para la próxima vez. Para devolver las respuestas, únicamente gestionamos mediante un condicional *if* si el botón ha sido pulsado, y mediante una pequeña adaptación de las variables aleatorias para que coincidan con su posición en el menú de respuestas, éstas se devuelven a través de un *outvalue*.

Para consultar el código fuente completo de la aplicación ver el Anexo A.

5.3 Reconocimiento frecuencial

Como siguiente aplicación, y continuando con el sistema explicado anteriormente en la aplicación de “Reconocimiento espacial”, nos encontramos ante una nueva aplicación, denominada “Reconocimiento frecuencial”, y destinada a afinar el oído desde un punto de vista de frecuencias. Para ello, nos haremos uso de una variada y rica cantidad de ondas diferentes, tal y como se ve en la fig. 5.3, así como de frecuencias discretas para un mayor control por parte del usuario.

Cabe destacar que a partir de este punto no se mostrará tanto código fuente por ser redundante con el mencionado en la sección 5.2. De manera que únicamente se mostrará código fuente que se crea conveniente y novedoso, ya que en muchos casos, el sistema de realizar tareas como las comprobaciones de las respuestas por ejemplo, es el mismo en todas las aplicaciones. Por ello, y para no redundar en el mismo código una y otra vez, únicamente se mostrará aquel que sea nuevo e interesante de mencionar.

5.3.1 Zona de pruebas

En esta sección vamos a entrar de lleno en el funcionamiento y los objetivos a los que está enfocada la “Zona de pruebas” dentro de la aplicación “Reconocimiento frecuencial”. Para ello iremos viendo cada una de sus secciones con detenimiento y comentando tanto sus características, como su funcionamiento y cómo se han desarrollado.



Figura 5.3: Vista general de la aplicación “Reconocimiento frecuencial”.

5.3.1.1 Muestra 1

En primer lugar, y como elemento característicos de esta aplicación respecto del resto, existe la diferencia en que existen cuatro instrumentos, y no dos. Dos instrumentos son los que definen cada una de las secciones de la “Zona de pruebas”, y los otros dos para las secciones de “Muestra 1” y “Muestra 2” de la “Zona de prácticas”.

Una vez mencionado dicho detalle, y ya en la sección “Muestra 1”, nos encontramos con un oscilador común, definido por el opcode `oscilikt`, al cual se le pasa por parámetros de entrada la amplitud, la frecuencia y la tabla de onda que se desea utilizar, y éste devuelve una variable de tipo audio con la señal resultante.

Para controlar sus parámetros de entrada se disponen de varios elementos de control a disposición del usuario. En primer lugar, y para la amplitud, éste dispone de un *widget* tipo *knob* o rotor para aumentar o disminuir la amplitud de la muestra resultante.

Por otro lado, y para la frecuencia, dispone de un sistema un tanto peculiar. En primer lugar, dispone de un *spinbox* con unos valores que van del 1 al 10 en pasos de unidad, y por otro lado, un menú con valores de $\times 10$, $\times 100$ y $\times 1000$. La frecuencia resultante entonces será, el número del *spinbox* multiplicado por la potencia elegida en el menú. De esta manera, el usuario dispone de una paleta de frecuencias discreta y controlada, al igual que bien diferenciadas entre sí. Además, dispone de un *display* en el que se muestra la frecuencia resultante y la amplitud.

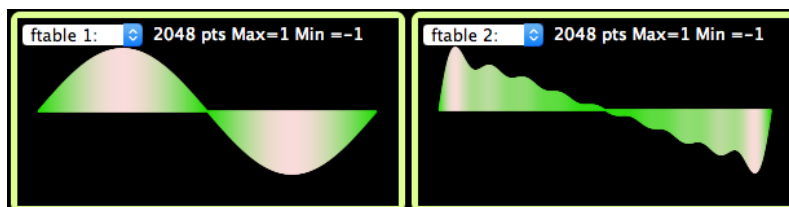


Figura 5.4: *Displays* encargados de mostrar la onda elegida por el usuario.

Para terminar existe otro menú con un listado de tipos de onda, entre las que se incluyen sinusoidal, diente de sierra, cuadrada, pulso, segmento y sigmoide. Cada una tiene unas características armónicas diferentes y variadas, ofreciendo así una mayor riqueza y variedad sonora al usuario. Para visualizar los tipos de onda se dispone de una pantalla de visualización de un ciclo de onda en la parte inferior, tal y como se puede observar en

la fig. 5.4. De esta forma, no sólo podemos escuchar, si no que también podemos “visualizar” lo que suena.

Para la generación de los tipos de onda se hace uso de la función *GEN10* de *Csound*, la cuál nos permite crear una tabla de onda a partir de sus armónicos parciales, tal y cómo vemos en el código inferior.

```

1 ;Declaración de las tablas de onda a partir de la función GEN10 de CSound
2 f 1 0 2048 10 1 ;Sinusoidal
3 f 2 0 2048 10 1 0.5 0.3 0.25 0.2 0.167 0.14 0.125 0.111 ;Diente de sierra

```

Dichas tablas se declaran en la partitura. En el ejemplo únicamente se han mostrado la declaración de la onda sinusoidal y diente de sierra para una mayor simplificación, pero habría que declarar cada una de las 6 ondas disponibles en la aplicación.

Para consultar el código fuente completo de la aplicación ver el Anexo B.

5.3.1.2 Muestra 2

La sección que nos acontece funciona y está declarada exactamente igual a la sección “Muestra 1”, ya que su funcionamiento es idéntico. La diferencia radica en que cada una está declarada en un instrumento diferente e independiente, de manera que podemos trabajar con uno, otro, o ambos de manera simultánea. Esto es muy útil para añadir dificultad, mezclando dos tonos diferentes para complicar la identificación de cada uno.

Para activar o desactivar cada uno de ellos se ha implementado un *spinbox* en cada uno a modo de botón de activación.

5.3.2 Zona de prácticas

5.3.2.1 Generar

Esta sección únicamente contiene el botón “Generar”. Su funcionamiento es igual al comentado en la aplicación de “Reconocimiento espacial”. Su tarea es, mediante un condicional *if*, comprobar cuando ha sido pulsado, y si éste recibe un valor 1, generará los valores aleatorios para cada uno de los parámetros que el usuario deberá adivinar más tarde, además de poner en blanco los menús de respuestas de usuario y el contador de fallos.

Para esta aplicación, al tener que implementar una sección de control para cada una de las dos muestras, se decidió crear una sección para generar muestras independiente al resto, en la que únicamente se encontrase el botón “Generar”.

5.3.2.2 Muestra 1 y Muestra 2

En este punto se han agrupado las dos secciones de la aplicación denominadas “Muestra 1” y “Muestra 2” por ser iguales y no caer en la redundancia. En ellas disponemos de los controles básicos para cada una de las muestras en la “Zona de prácticas”.

En primer lugar tenemos un botón tipo *spinbox* para activar o desactivar cada una de las muestras. De esta forma, el usuario podrá entrenar con una muestra o ambas, según quiera añadir más o menos dificultad. Cabe destacar que en cualquier caso se generan los valores aleatorios y, por tanto, cada una de las muestras. Desde este control únicamente las activamos por los auriculares o no.

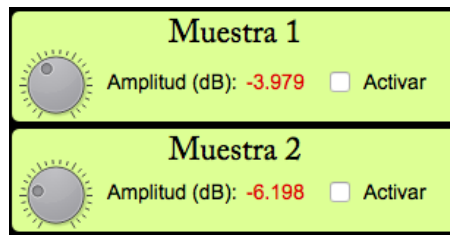


Figura 5.5: Sección de control de ambas muestras para la “Zona de prácticas”.

Además, se incorpora un control tipo rotor o *knob* para el control de la amplitud de cada muestra. Por otro lado, y tal y como se puede observar en la fig. 5.5, disponemos de un *display* en cada una de las secciones para visualizar la amplitud de cada una de ellas. Dicha amplitud se muestra en decibelios, referida siempre a los 0 dB como nivel máximo.

5.3.2.3 Respuestas - Muestra 1 y Muestra 2

En estas dos secciones nos encontramos la zona de respuestas en las que el usuario debe responder cuáles cree que son los parámetros que caracterizan la muestra aleatoria que está sonando. El sistema de respuestas y verificación de las mismas es igual en todas las aplicaciones, el cual ha sido explicado en la aplicación de “Reconocimiento espacial”.



Figura 5.6: Sección de respuestas de ambas muestras para la “Zona de prácticas”.

Para hacer más dinámica a la aplicación se han filtrado las respuestas en función de si esa muestra está activada. Es decir, el contador de fallos no sumará fallos si el usuario intenta responder los parámetros de una muestra que no esté activa, porque se sobre entiende que no está trabajando con ella. Por tanto, para que el contador de fallos funcione, y por tanto, la verificación de respuesta correcta, dicha muestra debe estar activa desde su sección de control.

5.3.2.4 Resultados

Por último, tenemos la zona de “Resultados”, la cual funciona de manera similar a la anteriormente mencionada en la aplicación anterior. Disponemos de un *display* para mostrar los fallos que lleva el usuario en cada ejercicio, el cual únicamente sumará si la muestra que está respondiendo está activa, evitando así sumar fallos por error de una muestra con la que no se esté trabajando. Por otro lado disponemos de un botón de soluciones, el cual devuelve en cualquier momento las soluciones a los menús de respuesta. Dicho botón, y al igual que el contador de fallos, está filtrado según la muestra que esté

activa, y, por tanto, únicamente devolverá las soluciones de la muestra que el usuario tenga activa.

La devolución de resultados funciona por pasos. En primer lugar se comprueba si el botón “Soluciones” ha sido pulsado, y en el caso de que valga 1, se comprueba si la muestra está activa. Únicamente en el caso en que ambas variables valgan 1 se mostrarán las soluciones en los menús correspondientes.

5.4 Ruido y Distorsión

Empezando con la tercera aplicación de la *suite* nos encontramos con la aplicación denominada “Ruido y Distorsión”, fig. 5.7. Tal y como su nombre indica, está enfocada a lo que se conoce como degradaciones del audio. Se ha enfocado de una manera muy general y se han ido creando secciones más específicas como pueden ser el ruido, *down-sampling* o la reducción de *bit depth*. Todo esto se verá de manera más específica en cada una de las secciones específicas destinadas a ello.

Por otro lado, comentar que es la aplicación más completa y a la vez compleja de las cuatro, debido a las posibilidades de interacción por parte del usuario que se han incorporado en ella. Inicialmente se hizo mucho más sencilla, pero por recomendación del tutor se añadieron funcionalidades tales como la aleatoriedad de la degradación aplicada en la “Zona de prácticas” o la posibilidad de elegir entre 3 niveles de dificultad. Todo esto se verá más adelante en profundidad desde su sección.



Figura 5.7: Vista general de la aplicación “Ruido y Distorsión”.

5.4.1 Zona de pruebas

5.4.1.1 Main

La sección “Main” de la zona de pruebas hace las veces de zona de control general de la “Zona de pruebas”, al igual que en las aplicaciones anteriores. Desde esta sección, el usuario puede seleccionar con qué muestra desea trabajar (se trata de las mismas muestras que en la aplicación de “Reconocimiento espacial”), así como control y visualización de la amplitud general desde un *slider* y un *display* respectivamente. Además, se incluye un control de visualización y control de la panoramización general, y un *spinbox* para activar la “Zona de pruebas”.

5.4.1.2 Ruido

En esta sección se trabaja con diferentes ruidos, con el objetivo de que el usuario sea capaz de identificar cada uno de ellos. Se han incorporado los ruidos más habituales dentro de una instalación de audio, como pueden ser ruido blanco, ruido rosa o ruido eléctrico. Obviamente, en una instalación compleja nos pueden aparecer diferentes tipos de ruido que nos puedan ensuciar la señal, pero sí es cierto que todos giran en torno a estos tres mencionados anteriormente, por lo que se ha considerado suficiente incorporar estos tres como tipos de ruido principales para trabajar.

Para la generación del ruido blanco y ruido rosa se han utilizado los opcodes propios de *Csound* `noise` y `pinkish`. A través de ellos, y únicamente pasando por entrada la amplitud, éstos nos devuelven una variable de tipo audio con el ruido generado. Para decidir cada uno de ellos únicamente debemos comprobar cuál está seleccionado en el menú por parte del usuario, previa comprobación de si está activada la sección “Ruido”. Por otro lado, y para la generación del ruido eléctrico, inicialmente se pensó en generarlo de manera artificial. Al encontrarnos con problemas a la hora de crear un espectro armónico rico y similar al ruido eléctrico real se decidió por introducir el ruido a través de una muestra de audio externa, siempre buscando que el ruido fuese similar al que nos podemos encontrar en una instalación real.

Para generar el audio de ruido eléctrico se montó una pista de grabación en el *software Logic Pro X*, y, a través de una entrada de audio de la tarjeta de sonido externa se conectó un cable, el cual se dejó al aire en el otro extremo. Al tocar dicho extremo con un metal de algún elemento conectado a la corriente eléctrica, éste induce dicha señal en el cable y podemos grabar la señal eléctrica. A través el previo de micrófono de la tarjeta podemos conseguir un nivel adecuado para su posterior procesamiento.

Para realizar este proceso, tuvimos que realizarlo con mucho cuidado para proteger el material de posibles subidas de tensión o problemas que pudieran deteriorar algún circuito, pero ya sabíamos que funcionaría por experiencias anteriores, por lo que no fue mayor problema. Tras esto, tratamos la señal grabada para adecuarla a niveles válidos para nuestra aplicación.

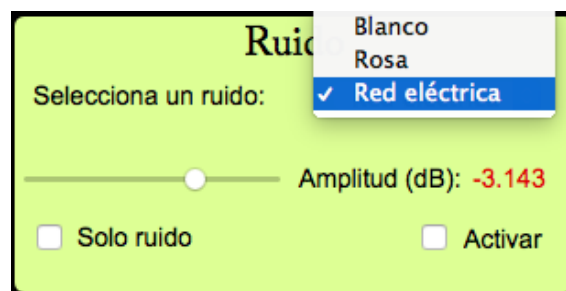


Figura 5.8: Vista general de la sección “Ruido”.

Además, se incorpora un botón de “Solo ruido”, en el que el usuario puede escuchar únicamente el ruido, sin muestra, para poder identificarlo de manera correcta y poder escucharlo con mucho más detalle. Se incorpora también un control de amplitud independiente para el ruido, así como un *display* de visualización y por último una casilla de activación del ruido.

5.4.1.3 Cuantización

En esta sección se puede trabajar con la cantidad de bits de cuantización de la muestra. Para ello, debemos mencionar que todas las muestras están en calidad 44.100 Hz y 16 bits, es decir, calidad de CD de audio estándar.

Partiendo de esa base, en esta sección podemos reducir la cantidad de bits desde los 16 originales hasta 4, es decir, pasamos de $2^{16} = 65536$ niveles de cuantización a únicamente $2^4 = 16$ niveles. Obviamente, dicha reducción es muy notoria, y por tanto, la calidad del audio se ve mermada de forma notoria también. Para poder reducir la cantidad de bits se ha colocado un *slider* con una rotulación en la parte inferior que nos indica la escala entre la que nos movemos, tal y como se observa en la fig. 5.9.

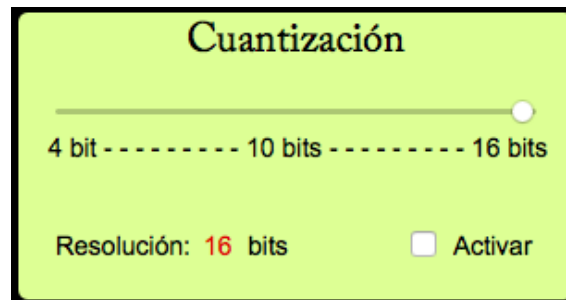


Figura 5.9: Vista general de la sección “Cuantización”.

Además, se incorpora el ya clásico *spinbox* para activar la sección y un *display* de visualización de la cantidad de bits con la que estamos trabajando.

Entrando ya en la parte técnica, para la reducción de bits se ha consultado el código `BitDepthReduction.csd` de la categoría “Distortion” de la colección “McCurdy” en los ejemplos de *Csound*.

Apoyándonos en ese código, hemos realizado las modificaciones pertinentes para adaptarlos a nuestra aplicación.

Para realizar la reducción de bits de manera artificial en primer lugar comprobamos si la sección está activa. Una vez hecho esto, mediante el opcode `pow` generamos la cantidad de niveles de cuantización a partir del primer parámetro que se le pasa, el cual es la base, y el segundo, la potencia, quedando `kvalue pow 2, 16`. Por un lado generamos el número de niveles a partir de los bits seleccionados por el usuario, es decir, $2^{8\text{kbtsprof}}$, y por otro 2^{16} , que serán el número de niveles totales.

Una vez hecho esto, utilizamos el opcode `downsamp` para realizar un *downsampling* a la muestra, antes de calcular el redondeo de cada muestra de forma independiente. Tras esto, utilizamos la función `interp` para interpolar los valores en una señal de tipo audio.

Es importante detallar que debido al hecho de tener que realizar los cálculos muestra a muestra, y a la naturaleza de funcionamiento del código de reducción de bits, ésta aplicación se tuvo que realizar con un valor de `ksmps = 1`, ya que si no, no funcionaba correctamente y mostraba fallos de *buffer underrun* y *buffer overrun*, es decir, fallos en el búfer de lectura y escritura.

5.4.1.4 Muestreo

Para terminar con la “Zona de pruebas” tenemos la sección dedicada al muestreo. tal y como se ha mencionado antes, las muestras originales de la aplicación están a una frecuencia de muestreo de 44100Hz. Pues bien, en esta sección podemos reducir dicha frecuencia hasta un mínimo de 4410Hz, es decir, 10 veces menos. Para ello, y al igual en que la sección de “Cuantización”, el usuario dispone de un *slider* con una rotulación inferior, en donde puede seleccionar la frecuencia de muestreo deseada para la muestra. Además, dispone de un *spinbox* de activación y un *display* para visualizar la frecuencia actual de trabajo.

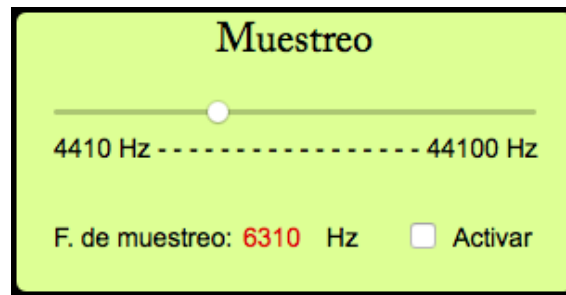


Figura 5.10: Vista general de la sección “Cuantización”.

Para realizar la reducción de la frecuencia de muestreo se ha utilizado el opcode `fold`. Dicha función de *Csound* recibe por entrada la muestra original de audio y una variable numérica, la cuál indica el cociente por el que se va a dividir la frecuencia de muestreo original. Por tanto, en caso de que le introduzcamos un 10, devolverá una muestra de audio con una frecuencia de muestreo de 4410 Hz, por ejemplo.

Para poder llevar a cabo este proceso de forma correcta, el *slider* trabaja en una escala de -10 a -1, de manera que a la función `fold` le introducimos la variable que devuelve el *slider* con un signo menos delante, invirtiendo así su signo y pasándole un número entre 10, para una frecuencia de muestreo de 4410 Hz, y un 1 para una de 44100 Hz, es decir, la original.

Cabe destacar un ligero detalle, y es que el *slider* trabaja de forma continua y no discreta, por lo que entre el múltiplo 10 y el 1 habrán infinitos valores, de forma que la reducción de la frecuencia de muestreo de la muestra será progresiva y no por pasos.

5.4.2 Zona de prácticas

5.4.2.1 Main

Una vez en la “Zona de prácticas”, y al igual que en la aplicación de “Reconocimiento espacial”, disponemos de una sección de control general desde donde podemos activar el entrenamiento, controlar y visualizar el volumen general de esta sección, seleccionar la muestra con la que deseamos trabajar o generar una nueva muestra.

El funcionamiento es igual al mencionado para la aplicación de “Reconocimiento espacial”, por lo que no se considera necesario volver a explicar cada una de las funciones de esta sección. Para conocer los detalles, consultar la sección [5.2.2.1](#).

5.4.2.2 Dificultad

Llegados a este punto comienza la complejidad de esta aplicación, la cual se diferencia del resto en que el usuario puede elegir una dificultad concreta. Para ello se ha colocado un menú en el que el usuario puede elegir la cantidad de degradaciones de las anteriormente explicadas que quiere que se apliquen a la muestra del ejercicio.

La dificultad reside en que elige cuántas, pero no cuáles. Para realizar esto primero se generan todos los parámetros aleatorios tal y como ya se ha mencionado anteriormente, además de un parámetro aleatorio extra entre 0 y 2. Dicho parámetro se utilizará para decidir que degradaciones se aplican dependiendo de la dificultad seleccionada.

Después se crean todas las degradaciones, cada una en una muestra de audio diferente, y, dependiendo de la dificultad elegida y del número aleatorio, se aplicarán a la muestra final las degradaciones correspondientes. Por ejemplo, en caso de elegir una degradación sólo, comprobamos el número aleatorio que nos dice que degradación aplicar, si dicho número vale 0 aplicamos ruido, si vale 1 cuantización y si vale 2, reducción de

la frecuencia de muestreo. En el caso de elegir 2 degradaciones, dicho número nos marca qué degradación no aplicamos, siendo la numeración de las degradaciones la misma. En el caso de que elijamos 3 degradaciones aplicaremos las 3.

Es importante tener en cuenta que al sumar varias señales de audio, deberemos dividir las para no saturar la muestra de audio final.

Además, en cada caso, es decir, en cada conjunción entre dificultad y degradaciones aplicadas, se genera un *flag* de control del 0 al 6, el cuál nos va a indicar en casos futuros con qué estamos trabajando. Es decir, para una dificultad de 1 y ruido tendremos la variable $gkflag = 0$, para dificultad 2 y degradaciones de ruido y cuantización tendremos $gkflag = 5$, etc. Gracias a este *flag*, podremos realizar una comprobación de resultados u otra, o devolver unos resultados u otros en caso de querer visualizar las soluciones.

5.4.2.3 ¿Qué degradación hay?

Aquí tenemos la primera sección de respuestas para el usuario. En esta zona el usuario debe marcar qué degradación o degradaciones se han aplicado de forma aleatoria a la muestra de salida. Como en aplicaciones anteriores, los fallos en esta sección también suman, y por tanto deberemos estar seguros antes de responder. El único caso en el que no se podrán cometer fallos es que el usuario elija 3 degradaciones, ya que sabrá de antemano que las 3 están aplicadas. Por contra, las posibilidades de fallar después serán mayores, ya que deberá acertar más variables aleatorias.

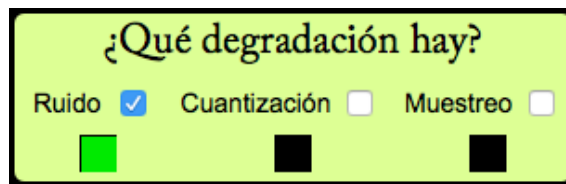


Figura 5.11: Vista general de la sección “¿Qué degradación hay?”.

El sistema de comprobación de resultados es igual al mencionado en las aplicaciones anteriores, si bien debemos comprobar mediante el *flag* anteriormente mencionado en qué caso estamos para poder saber los resultados de manera exacta.

5.4.2.4 Respuestas - Ruido, Cuantización y Muestreo

Estas tres secciones se han agrupado al ser comunes entre sí. En ellas el usuario debe responder los parámetros de cada una de las degradaciones que se hayan aplicado a la muestra, fig. 5.12.

Para ello, las respuestas se han filtrado a través del *flag*, de manera que únicamente contará los fallos de la muestra que se haya aplicado. Por tanto, primero deberá saber que degradaciones se han aplicado y después deberá adivinar los valores de éstas.

En primer lugar comprobamos el *flag* para saber qué degradación debemos comprobar. Una vez hecho esto comprobamos si el usuario ha respondido bien a la degradación que existe en la muestra, y dependiendo de ello mostramos verificación o no. Si el usuario pulsa otro de los checks debemos sumar un fallo al no ser éste correcto. Para la comprobación de las respuestas, únicamente comprobamos la de la degradación que hay en ese *flag*.

Destacar un problema que nos surgió. En ocasiones, el contador de fallos no funcionaba bien en la primera muestra generada, y a partir de ahí daba errores en el contador de fallos. Para ello, se colocó un contador y se hizo que en la primera pulsación al botón “Generar” valiese 4410, de forma que filtramos ese valor y no sumamos fallo. De esta ma-

nera arreglamos el fallo y evitamos que contase fallos en la primera pulsación de manera errónea, solucionando así el *bug*.

Para ver el código fuente completo se puede consultar el Anexo C.



Figura 5.12: Vista general de la sección de respuestas.

5.4.2.5 Resultados

Por último tenemos la sección de resultados, en la cuál, y al igual que en todas las aplicaciones, se mostrará el número de fallos cometido en cada ejercicio, además del ya conocido botón de soluciones.

Para mostrar las soluciones de manera adecuada, y tal y como ya se ha comentado antes, se hace uso del *flag* que nos indica que variables están entrando en juego en cada ejercicio.

Por tanto, una vez habiendo comprobado si se ha pulsado el botón de soluciones, comprobamos el *flag* en el que estamos. Dependiendo de su valor, devolveremos los valores a los *checks* superiores, activando únicamente el de la degradación aplicada en ese ejercicio y poniendo a 0 los que no. Por otro lado, deberemos mostrar las soluciones en la zona de respuestas para los parámetros que forman parte de ese *flag*.

Deberemos realizar lo mismo para cada uno de los 6 *flags*, de manera que en cualquiera de los casos se devuelvan las soluciones de manera correcta y sin mostrar respuestas para parámetros que no entran en juego en cada ejercicio.

5.5 Ganancia y Dinámica

Y por fin llegamos a la última de las cuatro aplicaciones que componen nuestra colección para entrenamiento auditivo psicoacústico. Se trata de la aplicación “Ganancia y Dinámica”, y en ella se tocan dos aspectos del audio bien diferenciados pero que van ligados entre sí. El motivo de haberlos juntado en una misma aplicación es que cada uno de ellos de manera individual hubiese quedado una aplicación muy pobre, y por tanto se ha decidido agruparlos en una misma aplicación, si bien cada uno de ellos se trabaja en zonas bien diferenciadas.

Inicialmente se iban a realizar únicamente tres aplicaciones, pero al ver que existía tiempo suficiente para implementar una cuarta, se apostó por realizar la aplicación en la que nos encontramos como forma de trabajar un poco aspectos del audio como las diferencias de ganancia, los niveles de la señal de audio y, no menos importante, la dinámica de éste. Para ello se agrupó en una misma aplicación una sección de ganancia y niveles y otra de dinámica.

Es importante destacar que para la zona de dinámica se permite trabajar en la “Zona de pruebas” con un procesador de dinámica completo, al cual se le han implementado dos *presets*, un compresor y una puerta de ruido. Para el modo de entrenamiento el usua-

rio únicamente deberá trabajar con ambos *presets* y no deberá adivinar cada uno de los parámetros que lo componen. Esto se ha decidido así por la extrema dificultad que supondría adivinar parámetros tan dispares como el codo del procesador, el ataque o el decaimiento.

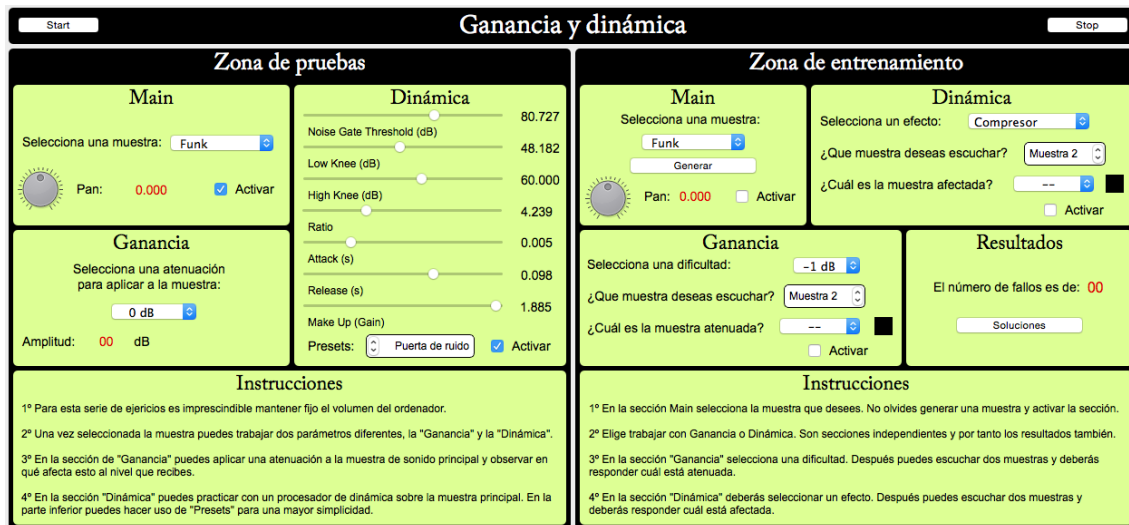


Figura 5.13: Vista general de la aplicación “Ganancia y Dinámica”.

5.5.1 Zona de pruebas

5.5.1.1 Main

Ya metidos en la “Zona de pruebas”, y dejando a un lado la introducción, nos encontramos, cómo no, con la sección “Main”. Desde ella, y tal y como hemos visto ya en repetidas ocasiones, podremos elegir la muestra deseada con la que trabajar, un rotor para controlar la panoramización de la misma, así como un *display* en el que visualizar la cantidad de panoramización y por último, un *spinbox* para activar la “Zona de pruebas”.

Todo esto es de sobra conocido a estas alturas del proyecto, por lo que no se hará mayor hincapié en ello.

5.5.1.2 Ganancia

Entramos en la sección “Ganancia”. Como se habrá podido observar, en la sección “Main” de esta aplicación no existe posibilidad de controlar el volumen de la muestra. Esto es debido a esta sección. En ella, el usuario dispone de un menú desplegable en el que puede elegir una de las siete ganancias establecidas, las cuáles son 0 dB, -1 dB, -2 dB, -3 dB, -6 dB, -9 dB, -12 dB. El motivo de que se hayan elegido las ganancias de manera discreta y preestablecidas es porque así podremos aplicar dichas ganancias en la “Zona de entrenamiento” y que puedan ser identificables. Si se hubiera utilizado un *slider* con valores de ganancia infinitos, sería muy difícil preguntar al usuario por qué ganancia hay aplicada a una muestra, o que seleccionase un nivel de dificultad a partir de la ganancia aplicada.

Además, y para completar la sección, se ha aplicado un *display* para visualizar la ganancia seleccionada. Es información redundante, ya que en el mismo menú de selección se puede consultar, pero se ha creído conveniente para seguir con los sistemas de visualización de información que se vienen utilizando a lo largo de todas las aplicaciones, manteniendo así la coherencia en el diseño.

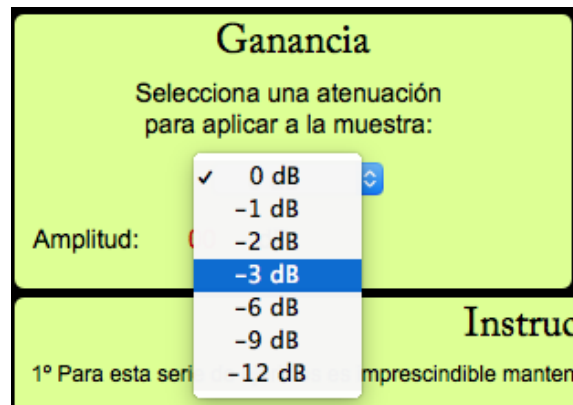


Figura 5.14: Vista general de la sección “Ganancia”.

5.5.1.3 Dinámica

Para terminar la “Zona de pruebas” tenemos la sección destinada a la dinámica. En ella, el usuario dispone de un completo procesador de dinámica, en el cuál puede modificar uno a uno cada uno de sus parámetros. Cada parámetro, tal y como se puede observar en la fig. 5.15, dispone de un *slider* para controlarlo, así como un *label* en la zona inferior con el parámetro al que corresponden y un *display* a la derecha para visualizar el valor del parámetro que hay en cada momento.

Además, en la zona inferior disponemos de un *spinbox* para activar el procesador, el cuál se aplicará a la muestra seleccionada, independientemente de la ganancia que hayamos puesto en la sección “Ganancia”.

Por otro lado, disponemos de un combo de *spinbox* y *display* para poder seleccionar y visualizar cada uno de los dos *presets* disponibles, compresor y puerta de ruido.



Figura 5.15: Vista general de la sección “Dinámica”.

Ahora entraremos de lleno en la parte técnica, es decir, el código. Para poder implementar el procesador de dinámica nos apoyamos en primer lugar del código `compress.csd` de la colección *McCurdy* de los ejemplos, en la sección *DynamicsProcessing*. Cuando observamos cómo funcionaba el opcode `compress`, implementamos nuestro propio procesador dinámico, utilizando únicamente del código mencionado las tres líneas para convertir los

sliders lineales en logarítmicos. Dicho código se puede observar en la parte inferior, y, aplicado a al *opcode invalue* de cada *slider* de entrada, convierte los valores que éste introduce en escala logarítmica, de forma que para parámetros como el *Ratio*, nos pueden ser muy útiles.

Mencionar que al principio tuvimos múltiples problemas con éstas líneas ya que en la generación de la variable *giExp100*, utilizada para el *Ratio*, tal y como se observa en el código inferior, ésta comprendía los valores entre 0 y 100, y no entre 1 y 100, como se muestra en el código inferior. Esto nos ocasionó múltiples problemas, ya que al aplicar al procesador valores para el *ratio* de compresión menores a 1, la salida de audio se disparaba a valores incontrolables. Para solucionarlo, tuvimos que detectar la errata en el código y subsanarla.

```

1 ;Declaración de tablas para crear sliders exponenciales
2 giExp1 ftgen 0, 0, 129, -25, 0, 0.001, 128, 1.0
3 giExp2 ftgen 0, 0, 129, -25, 0, 0.001, 128, 2.0
4 giExp100 ftgen 0, 0, 129, -25, 0, 1, 128, 100.0
5
6 kratio invalue "ratio"
7 gkratio tablei kratio, giExp100, 1

```

Una vez tuvimos configurados los *sliders* de manera adecuada, nos apoyamos en el *opcode compress* de *Csound* para generar el procesado dinámico de la muestra activa. Para ello, se le introducen los canales L y R de la muestra (en nuestro caso se introduce la misma muestra al ser ésta mono), así como el nivel de umbral, los valores para el codo bajo y alto del procesador, el ratio de compresión, el ataque, y el decaimiento, así como un parámetro fijo llamado *ilook*, el cuál siempre vale 0.5.

Una vez procesada la muestra, ésta quedaba con un volumen muy bajo, por lo que se introdujo un *slider* de corrección de ganancia¹, dejando la muestra procesada y corregida, lista para ser sacada por la salida de audio de nuestro sistema.

5.5.2 Zona de prácticas

5.5.2.1 Main

Una vez en la “Zona de prácticas”, y al igual que en la aplicación de “Reconocimiento espacial” y “Ruido y Distorsión”, disponemos de una sección de control general desde donde podemos activar el entrenamiento, seleccionar la muestra con la que deseamos trabajar o generar una nueva muestra, así como un control y visualización de la panoramización. Al igual que en la “Zona de pruebas”, aquí tampoco se incorpora control de la ganancia, ya que ésta forma parte de los ejercicios.

Por tanto, y tal y como se menciona en las instrucciones de la aplicación, se recomienda trabajar siempre con un volumen del ordenador fijo, ya que de ello dependerá que seamos capaces de reconocer los cambios de volumen en la muestra.

El funcionamiento es igual al mencionado para las aplicaciones anteriores, por lo que no se considera necesario volver a explicar cada una de las funciones de esta sección. Para conocer los detalles, consultar la sección 5.2.2.1.

5.5.2.2 Ganancia

Centrándonos ahora en el entrenamiento de la ganancia, éste se ha realizado de manera similar a aplicaciones comerciales conocidas y vistas en la sección 2.4. Para ello,

¹En inglés: *Make up gain*, disponible en cualquier procesador de dinámica.

el usuario debe seleccionar una dificultad, la cual viene caracterizada por la cantidad de atenuación de la muestra. El ejercicio consistirá en identificar cuál, de dos muestras aleatorias, es la atenuada y cuál posee el volumen original. Por tanto, a mayor atenuación aplicada, más sencillo será identificarla. Este es el motivo por el cuál en la “Zona de pruebas” las ganancias están preestablecidas, para poder utilizarlas como niveles de dificultad en esta sección.

Por tanto, el usuario deberá, en primer lugar, seleccionar el nivel de atenuación, y por tanto de dificultad, con el que quiere trabajar. tras esto, tendrá un *spinbox* asociado a un *display* en el que podrá elegir que muestra desea escuchar, pudiendo elegir entre “Muestra 1” y “Muestra 2”, tal y como observamos en la fig. 5.16.

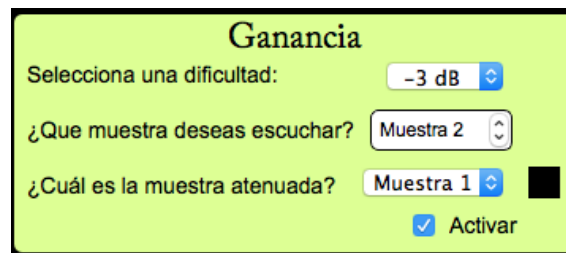


Figura 5.16: Vista general de la sección “Ganancia”.

Por tanto, se trata de un entrenamiento a ciegas, en las que el oído será la única herramienta para acertar la respuesta atenuada. Dicha respuesta se deberá indicar en un menú, el cuál mostrará un identificador luminoso en caso de acierto, tal y como ya venimos viendo a lo largo de todo el proyecto.

Para generar una muestra atenuada, hacemos uso de un número aleatorio entre 0 y 1, creado en la sección del código donde se generan el resto de valores aleatorios. Dicho número nos indicará que muestra de las dos es la atenuada en cada ejercicio.

En primer lugar debemos comprobar si la sección de entrenamiento está activada. Si es así, convertimos la atenuación elegida en el nivel de dificultad de decibelios a escala lineal. Tras esto, comprobaremos que muestra está seleccionada por el usuario para escuchar, y, en función de eso y del número aleatorio que nos indica qué muestra estará atenuada, aplicaremos la atenuación a la correspondiente y dejaremos la otra con su ganancia original.

Para la comprobación de resultados de esta sección tuvimos un problema similar al que apareció en la aplicación de “Ruido y Distorsión” con el contador de fallos. Para solucionarlo llevamos a cabo una medida similar a la tomada anteriormente, con la diferencia de que para el caso que nos acontece existe un pequeño *bug* que no ha sido posible solucionar, pero que no tiene mucha importancia.

El *bug* consiste en que, si recién arrancado el programa, en los siguientes 2 segundos, el usuario pulsa el botón “Generar” muy rápido, el contador de fallos puede llegar a mostrar un error en los resultados. Por tanto, se sugiere arrancar el programa y dejar que corra 2-3 segundos antes de pulsar el botón para generar una muestra aleatoria.

No se ha considerado un fallo muy notable ya que el flujo habitual de trabajo de un usuario es arrancar el programa, probar en la “Zona de pruebas” y, tras esto, comenzar los ejercicios, por lo que no sería habitual que dicho *bug* apareciese.

5.5.2.3 Dinámica

En la siguiente sección, se trata el entrenamiento de la parte de dinámica. Esta sección fue difícil de implementar con unos resultados satisfactorios debido a la dificultad de entrenar parámetros asociados a un procesador de dinámica. Por ello, y para facilitar y

simplificar el entrenamiento y la aplicación, se decidió hacer uso de los *presets* de compresor y puerta de ruido vistos en la “Zona de pruebas”, para utilizarlos como base del entrenamiento.

Esta sección funciona y está estructurada igual que la sección de entrenamiento de ganancia, si bien cada una son independientes y activables de manera individual.

En primer lugar, y como se puede observar en la fig. 5.17, visualmente es casi idéntica a la sección de entrenamiento para la ganancia. como aspecto diferente, en primer lugar el usuario no deberá elegir un nivel de atenuación para trabajar, si no el *preset*, es decir, el efecto que desea entrenar.

A continuación, podrá seleccionar entre dos muestras a ciegas para escuchar cada una de ellas y, en función de lo que escuche, responder cuál de las dos muestras cree que es la afectada por el procesado. El funcionamiento de acierto y error es igual al visto anteriormente, por lo que no se va a comentar en esta sección.

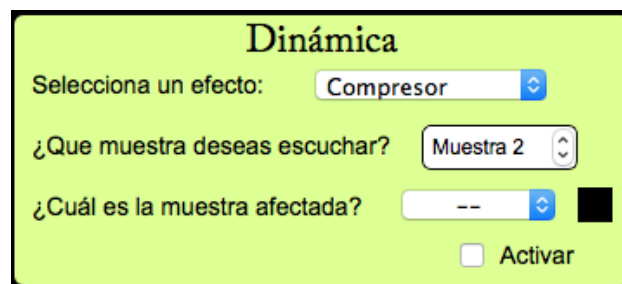


Figura 5.17: Vista general de la sección “Dinámica”.

El código fuente para la generación de cada muestra en función de la aleatoriedad generada y del efecto deseado es demasiado extenso para exponerlo aquí, por lo que si se desea, se puede consultar el Anexo D con el código fuente completo de la aplicación, si bien la filosofía de funcionamiento es la misma que para la sección de “Ganancia” de esta misma aplicación.

5.5.2.4 Resultados

Para terminar nos encontramos con la sección de “Resultados”. En ella, y tal y como se viene viendo en todas las aplicaciones, se muestran los fallos cometidos por el usuario en cada uno de los ejercicios realizados, poniendo a 0 el contador cada vez que pulsa en el botón “Generar” para comenzar un nuevo ejercicio.

Además, se incorpora el botón “Soluciones” a través del cuál se devuelven a los menús las respuestas de cada ejercicio. Como ya viene siendo habitual, dichas respuestas están filtradas, de manera que si por ejemplo estamos trabajando con ganancias, únicamente se devolverán las respuestas de la ganancia, y no de la dinámica.

Por tanto, y en primer lugar, deberemos comprobar si estamos trabajando con ganancias o con dinámica. Una vez hecho este paso, deberemos comprobar el valor del número aleatorio que nos indica que muestra está atenuada o afectada por el procesador de dinámica, y, dependiendo de en que caso estemos, devolver a sus menús la respuesta correspondiente. Para este caso, y como únicamente debemos indicar si la afectada es la muestra 1 o 2, pasamos por `outvalue` el valor de 1 o 0 al canal correspondiente, simplificando así el código al no depender de ninguna variable extra, ya que es posible pasar el valor numérico de la respuesta directamente.

CAPÍTULO 6

Conclusiones y trabajos futuros

6.1 Conclusiones

Csound constituye una potente y versátil herramienta de desarrollo aplicada al mundo del audio y la síntesis de sonido, tal y como se ha visto a lo largo del presente proyecto y gracias a sus opcodes propios.

Como segundo matiz a destacar, tenemos *CsoundQt*, sin el cual no habría sido posible la interacción entre *widgets* gráficos y el lenguaje *Csound*. Esto nos ha facilitado la posibilidad de desarrollar una serie de aplicaciones con interfaz gráfica bastante completa en cuanto funcionalidades y potencial.

Si bien las aplicaciones para entrenamiento auditivo son algo real y existente en la actualidad, con propuestas profesionales que se mejoran y ofrecen por mucho respecto a las nuestras, el objetivo del presente proyecto no era desarrollar una serie de aplicaciones comerciales, si no una base beta de las mismas sobre la que apoyarnos en futuros proyectos.

Además, para su desarrollo se ha desarrollado un sistema completo de ejercicios, todos ellos basados en el entrenamiento por repetición y mediante muestras de audio aleatorias. Gracias a ello, el usuario deberá realizar un entrenamiento “a ciegas”. Éste método de entrenamiento está de sobra validado por numerosas aplicaciones disponibles en el mercado, tanto para entrenamiento de audio, como en otros muchos ámbitos.

En base al sistema de ejercicios desarrollado, como a la exploración de las funcionalidades que nos ofrecen tanto *Csound* como *CsoundQt*, y, finalmente, al desarrollo de cuatro aplicaciones para entrenamiento auditivos psicoacústico, se pueden dar por concluidos los objetivos iniciales del proyecto.

Quien desee, puede utilizar, modificar y distribuir el código fuente desarrollado en el presente proyecto para lo que considere oportuno, especialmente si es para fines didácticos y de aprendizaje.

6.2 Propuestas de trabajo futuras

Gracias a la naturaleza del presente proyecto se pueden vislumbrar varias e interesantes líneas de trabajo futuras que complementen todo lo desarrollado. A continuación se proponen una serie de líneas de trabajo, bien para nuevos trabajos de fin de grado como nuevas líneas de investigación.

- Evaluación y depuración del sistema de ejercicios de la *suite* de Aplicaciones para entrenamiento auditivo psicoacústico.

Puede ser interesante desarrollar una evaluación exhaustiva para verificar la utilidad de las presentes aplicaciones desarrolladas en este proyecto. Para ello, se debe-

rá centrar la atención especialmente en el punto psicoacústico de estas y, mediante sujetos voluntarios, evaluar la utilidad de las mismas y la posible mejoría en su audición.

- Ampliación y optimización de las aplicaciones desarrolladas en el presente proyecto (versión 2.0).

Depuración de las funcionalidades ya implementadas como de la interfaz gráfica. Además, se presenta interesante la posibilidad de añadir nuevas funcionalidades y utilidades dentro de cada aplicación, ampliando así su potencial y utilidad.

- Conversión del formato *Csound* a formato *Android* o *iOS*.

Con el potencial actual de los *smartphones* y las *tablets*, se considera muy oportuno y con un potencial extremo la posibilidad de migrar las *suite* de aplicaciones desarrolladas a formatos para los dos principales sistemas operativos móviles del mercado, *Android* e *iOS*.

- Creación de un *plugin VST standalone* a partir de las aplicaciones en *Csound*.

Se propone como opción interesante la posibilidad de migrar las aplicaciones desarrolladas en *Csound* al formato universal VST, buscando especialmente la opción de que éstas sean *standalone*, es decir, autónomas. De esta manera, no necesitarán de ningún DAW externo para poder ejecutarlas en el ordenador.

- Migración de formato *Csound* a C++

Una de las limitaciones encontradas en *Csound* ha sido su potencial gráfico, entre otras cosas. Para solucionar este problema, y hacer de las aplicaciones una herramienta mucho más universal, se propone migrarlas a lenguaje C++. Apoyándonos en C++, podremos implementar una interfaz gráfica mucho más elaborada, además de una serie de posibilidades *software* imposibles en *Csound*.

Bibliografía

- [1] AZNAR Casanova, J.A., *La psicofísica clásica y la contemporánea*, 2017. Universidad de Barcelona (Barcelona, España).
Disponible en <http://www.ub.edu/pa1/node/66>
- [2] WIKIPEDIA, *Fisiología humana*, 2017.
Disponible en https://es.wikipedia.org/wiki/Fisiología_humana
- [3] MAGGIOLO, D., *El sistema auditivo periférico*, 2003. Escuela universitaria de música (Montevideo, Uruguay).
Disponible en <http://www.eumus.edu.uy/docentes/maggiolo/acuapu/sap.html>
- [4] MAGGIOLO, D., *El sistema auditivo central*, 2003. Escuela universitaria de música (Montevideo, Uruguay).
Disponible en <http://www.eumus.edu.uy/docentes/maggiolo/acuapu/sac.html>
- [5] RODRÍGUEZ, A., *Conceptos básicos de la Psicoacústica*, 2005. Instituto de Ingeniería Eléctrica IIE, Facultad de Ingeniería, UDELAR (Montevideo, Uruguay).
Disponible en http://iie.fing.edu.uy/investigacion/grupos/gmm/audio/seminario/seminariosviejos/2005/charlas2005/charla4_Informe.pdf
- [6] PÉREZ Cano, M., *La audición humana 4.1.Umbrales psicológicos*, 2004. Universidad e Valladolid (Castilla y León, España).
Disponible en https://www.lpi.tel.uva.es/~nacho/docencia/ing_ond_1/trabajos_04_05/io1/public_html/marcos.htm
- [7] MAGGIOLO, D., *Umbrales de audición*, 2003. Escuela universitaria de música (Montevideo, Uruguay).
Disponible en <http://www.eumus.edu.uy/docentes/maggiolo/acuapu/umb.html>
- [8] MAGGIOLO, D., *Psicoacústica*, 2003. Escuela universitaria de música (Montevideo, Uruguay).
Disponible en <http://www.eumus.edu.uy/docentes/maggiolo/acuapu/int.html>
- [9] AZNAR Casanova, J.A., *Los métodos psicofísicos directos o escalas psicofísicas*, 2017. Universidad de Barcelona (Barcelona, España).
Disponible en <http://www.ub.edu/pa1/node/73>
- [10] MORENO Ibarra, G., *Estudio psicoacústico y estadístico de la audición en la profesión del DJ. Características y pérdidas auditivas de los individuos*, 2012. Escuela Politécnica Superior de Gandía, Universidad Politécnica de Valencia (Valencia, España).
Disponible en <https://riunet.upv.es/bitstream/handle/10251/14234/memoria.pdf?sequence=1>
- [11] MAGGIOLO, D., *Enmascaramiento*, 2003. Escuela universitaria de música (Montevideo, Uruguay).
Disponible en <http://www.eumus.edu.uy/docentes/maggiolo/acuapu/enm.html>

- [12] MARTÍN Rocamora, L. J., *Psicoacústica*, 2011. Escuela universitaria de música (Montevideo, Uruguay).
Disponible en <http://www.eumus.edu.uy/eme/ensenanza/acustica/apuntes/Psicoacustica.pdf>
- [13] MAGGIOLO, D., *Sonoridad*, 2003. Escuela universitaria de música (Montevideo, Uruguay).
Disponible en <http://www.eumus.edu.uy/docentes/maggiolo/acuapu/son.html>
- [14] MINISTERIO DE EDUCACIÓN, *Sonido y música con ordenador - Cualidades del sonido*, 2017. Ministerio de Educación (España).
Disponible en http://www.ite.educacion.es/formacion/materiales/60/cd/02_elsonido/2_cualidades_del_sonido.html
- [15] ROYUELA del Val, J. y DE LA PARRA García, C., *Escucha 3D y holofonía*, 2. *Escucha binaural*, 2004. Universidad de Valladolid (Castilla y León, España).
Disponible en https://www.lpi.tel.uva.es/~nacho/docencia/ing_ond_1/trabajos_04_05/io1/public_html/marcos.htm
- [16] IÑESTA Quereda, J. M., *Síntesis Digital del Sonido*, 2016. Departamento de Lenguajes y Sonidos Informáticos, Universidad de Alicante (Alicante, España).
- [17] MAGGIOLO, D., *Timbre*, 2003. Escuela universitaria de música (Montevideo, Uruguay).
Disponible en <http://www.eumus.edu.uy/docentes/maggiolo/acuapu/tbr.html>
- [18] Así funciona la conversión analógico-digital, *Así funciona*, 2010.
Disponible en http://www.asifunciona.com/electronica/af_conv_ad/conv_ad_5.htm
- [19] Panear, conceptos básicos, *Alternative Silence*, 2013.
Disponible en <http://alternativesilence.blogspot.com.es/2011/10/panear-conceptos-basicos.html>
- [20] Definición ABC, *Definición ABC*, 2017.
Disponible en <https://www.definicionabc.com/social/autoevaluacion.php>
- [21] HEINTZ, J. y MCCURDY, I., *Csound FLOSS Manual*, 2005. FLOSS Manuals.
Disponible en <http://www.write.flossmanuals.net/csound/preface/>
- [22] WIKIPEDIA, *Csound*, 2017.
Disponible en <https://es.wikipedia.org/wiki/Csound>

APÉNDICE A

Código fuente - Reconocimiento espacial

```
1 ;-----
2 ;RECONOCIMIENTO ESPACIAL v.1.0
3 ;-----
4 ; Código: Implementación de una aplicación para el entrenamiento auditivo
5 ; enfocada al reconocimiento espacial y la imagen estéreo.
6 ;-----
7 ; Autor: Fernando Ayelo Sánchez
8 ;-----
9 ; Fecha: 6 de Junio de 2017
10 ;-----
11
12 <CsoundSynthesizer>
13
14 ;Sección Opciones de Csound
15 <CsOptions>
16 --displays ;Activamos este flag para visualizar los gráficos del interfaz.
17 </CsOptions>
18 ;Diseño de la orquesta
19 <CsInstruments>
20 ;Definimos las variables globales
21 sr = 44100 ;Frecuencia de muestreo
22 ksmps = 128 ;Número de muestras calculadas por periodo de control
23 nchnls = 2 ;Número de canales
24 Odbfs = 1 ;Amplitud de referencia (0dB)
25
26 ;Declaración e inicialización de las variables globales
27 gaRvbSend init 0
28 gaRvbSend2 init 0
29 gkAmple init 0
30 gkfallos init 0
31 gkamprand init 0
32 gkbal init 0
33 gktr init 0
34 gkhf init 0
35
36 ;inicialización de las variables de control para el sistema de fallos
37 gkres1 init 0
38 gkres2 init 0
39 gkres3 init 0
40 gkres4 init 0
41
42 ;inicializamos las variables aleatorias a 0
43 gkstereopr init 0
44 gkroompr init 0
45 gkhfpr init 0
46 gkamppr init 0
47 gkresbalmenu init 5
48
49 ;inicializamos las variables de validación a 0
```

```

50 gkValidA init 0
51 gkValidB init 0
52 gkValidR init 0
53 gkValidH init 0
54
55 instr 10 ;Instrumento de comunicación con la interfaz gráfica (GUI)
56 ;instrumento de prueba
57 gkAmp invalue "amplitud"
58 gkmute invalue "muteoriginal"
59 gkstereo invalue "stereo"
60 gkmuestra invalue "muestra"
61
62 ;instrumento de práctica
63 gkmuestra2 invalue "muestra2"
64 gkmute2 invalue "muteoriginal2"
65 gkAmp2 invalue "amplitud2"
66 gkstereo2 invalue "stereo2"
67 gkroom2 invalue "room2"
68 gkhf2 invalue "hf2"
69 gkgenerar invalue "generar"
70 gksoluc invalue "soluc"
71
72 ;valores para mostrar
73 kampdB = 10*log10(-gkAmp)
74 outvalue "amplitudpos",kampdB
75 outvalue "balance", (gkstereo-0.5)
76 endin
77
78 instr 1 ;Instrumento de prueba
79 ;Carga de la muestra de audio según decida el usuario
80 if gkmuestra == 0 then
81     aMuestra soundin "muestra1.wav",0
82 elseif gkmuestra == 1 then
83     aMuestra soundin "muestra2.wav",0
84 elseif gkmuestra == 2 then
85     aMuestra soundin "muestra3.wav",0
86 elseif gkmuestra == 3 then
87     aMuestra soundin "muestra4.wav",0
88 elseif gkmuestra == 4 then
89     aMuestra soundin "muestra5.wav",0
90 elseif gkmuestra == 5 then
91     aMuestra inch 1
92 endif
93
94 ;Adaptación de las variables entregadas por los widgets
95 kAmp = -gkAmp
96
97 aMuestra = kAmp*aMuestra ;Ajuste de la amplitud general
98 aSal = (gkmute*aMuestra) ;Mute para la muestra general
99
100 out aSal*(1-gkstereo),aSal*gkstereo ;Salida de la muestra con la
    posibilidad de estéreo
101
102 kRvbSendAmt = (1-kAmp) ;Ajuste de la cantidad de reverb
103 gaRvbSend = gaRvbSend + (aSal * kRvbSendAmt) ;Envío de la señal
    para el instrumento de reverb
104
105 krvblog = 10*log10(kRvbSendAmt) ;Adaptación a dB de la cantidad de
    reverb
106 outvalue "reverbenv", krvblog ;Envío de la cantidad de reverb a la
    interfaz GUI
107 endin
108
109 instr 2 ;Instrumento de practica

```

```

110 ;Carga de la muestra de audio según decida el usuario
111 if gkmuestra2 == 0 then
112     aMuestra2 soundin "muestra1.wav",0
113 elseif gkmuestra2 == 1 then
114     aMuestra2 soundin "muestra2.wav",0
115 elseif gkmuestra2 == 2 then
116     aMuestra2 soundin "muestra3.wav",0
117 elseif gkmuestra2 == 3 then
118     aMuestra2 soundin "muestra4.wav",0
119 elseif gkmuestra2 == 4 then
120     aMuestra2 soundin "muestra5.wav",0
121 elseif gkmuestra2 == 5 then
122     aMuestra2 inch 1
123 endif
124
125 ;Asignación de las respuestas a su variable correspondiente
126
127 ;Variable de amplitud
128 if gkAmp2 == 0 then
129     kamplitud = 0
130 elseif gkAmp2 == 1 then
131     kamplitud = -1
132 elseif gkAmp2 == 2 then
133     kamplitud = -2
134 elseif gkAmp2 == 3 then
135     kamplitud = -3
136 elseif gkAmp2 == 4 then
137     kamplitud = -4
138 elseif gkAmp2 == 5 then
139     kamplitud = -5
140 elseif gkAmp2 == 6 then
141     kamplitud = -6
142 elseif gkAmp2 == 7 then
143     kamplitud = -7
144 elseif gkAmp2 == 8 then
145     kamplitud = -8
146 elseif gkAmp2 == 9 then
147     kamplitud = -9
148 elseif gkAmp2 == 10 then
149     kamplitud = -10
150 elseif gkAmp2 == 11 then
151     kamplitud = -11
152 endif
153
154 ;Variable de balance
155 if gkstereo2 == 0 then
156     kbalance = -0.5
157 elseif gkstereo2 == 1 then
158     kbalance = -0.4
159 elseif gkstereo2 == 2 then
160     kbalance = -0.3
161 elseif gkstereo2 == 3 then
162     kbalance = -0.2
163 elseif gkstereo2 == 4 then
164     kbalance = -0.1
165 elseif gkstereo2 == 5 then
166     kbalance = 0.0
167 elseif gkstereo2 == 6 then
168     kbalance = 0.1
169 elseif gkstereo2 == 7 then
170     kbalance = 0.2
171 elseif gkstereo2 == 8 then
172     kbalance = 0.3
173 elseif gkstereo2 == 9 then

```

```

174     kbalance = 0.4
175     elseif gkstereo2 == 10 then
176     kbalance = 0.5
177     elseif gkstereo2 == 11 then
178     kbalance = 0.8
179     endif
180
181 ;Variable del tamaño de la habitación
182     if gkroom2 == 0 then
183         ktam = 0.0
184     elseif gkroom2 == 1 then
185         ktam = 0.1
186     elseif gkroom2 == 2 then
187         ktam = 0.2
188     elseif gkroom2 == 3 then
189         ktam = 0.3
190     elseif gkroom2 == 4 then
191         ktam = 0.4
192     elseif gkroom2 == 5 then
193         ktam = 0.5
194     elseif gkroom2 == 6 then
195         ktam = 0.6
196     elseif gkroom2 == 7 then
197         ktam = 0.7
198     elseif gkroom2 == 8 then
199         ktam = 0.8
200     elseif gkroom2 == 9 then
201         ktam = 0.9
202     elseif gkroom2 == 10 then
203         ktam = 1.0
204     elseif gkroom2 == 11 then
205         ktam = 1.5
206     endif
207
208 ;Variable de la cantidad de reverb
209     if gkhf2 == 0 then
210         kagudos = 0
211     elseif gkhf2 == 1 then
212         kagudos = -1
213     elseif gkhf2 == 2 then
214         kagudos = -2
215     elseif gkhf2 == 3 then
216         kagudos = -3
217     elseif gkhf2 == 4 then
218         kagudos = -4
219     elseif gkhf2 == 5 then
220         kagudos = -5
221     elseif gkhf2 == 6 then
222         kagudos = -6
223     elseif gkhf2 == 7 then
224         kagudos = -7
225     elseif gkhf2 == 8 then
226         kagudos = -8
227     elseif gkhf2 == 9 then
228         kagudos = -9
229     elseif gkhf2 == 10 then
230         kagudos = -10
231     elseif gkhf2 == 11 then
232         kagudos = -11
233     endif
234
235 ;Generación de valores aleatorios y reseteo de menús de usuario
236

```

```

237 ;Si se pulsa generar, se generan los valores aleatorios y se ponen a 0
    los menús
238 if gkgenerar == 1 then
239 ;Creamos el aleatorio de la amplitud
240     krnd randh 10, 10, 10
241     krnd = abs(krnd)
242     kamprand = round(krnd)
243     gkamprand = -kamprand
244
245 ;Creamos el aleatorio del balance
246     krnd2 randh 5, 10, 10
247     krnd2 = round(krnd2)
248     gkbal = krnd2/10
249
250 ;Creamos el aleatorio del tamaño de la habitación
251     krnd3 randh 10, 10, 10
252     krnd3 = abs(krnd3)
253     krnd3 = round(krnd3)
254     gktr = krnd3/10
255
256 ;Creamos valor fijo de amortiguamiento, no es necesario acertarlo, la
    reverb va directamente relacionada con la amplitud  $R = 1 - A$ 
257     gkhf = 0.5
258
259 ;Ponemos a 0 los menús para el usuario
260     outvalue "amplitud2", 11
261     outvalue "stereo2", 11
262     outvalue "room2", 11
263     outvalue "hf2", 11
264
265 ;Reseteamos el contador de fallos
266     gkfallos = 0
267     endif
268
269 ;Generamos la señal de audio
270
271     klog = gkamprand/10
272     gkAmple = 10^(klog)
273
274     kAmple = gkAmple
275     gkbaladap = gkbal + 0.5
276
277     aMuestra2 = kAmple*aMuestra2
278     aSal2 = (gkmute2*aMuestra2)
279
280     out      aSal2*(1-gkbaladap),aSal2*gkbaladap
281
282     kRvbSendAmt2 = (1-kAmple)
283     gaRvbSend2 = gaRvbSend2 + (aSal2 * kRvbSendAmt2)
284
285     krvblog2 = 10*log10(kRvbSendAmt2)
286     krvblog2 = round(krvblog2)
287
288 ;Comprobación de resultados
289
290 ;Comprobación de la amplitud
291 if gkamprand == kamplitud then
292     gkValidA = 1
293 else
294     gkValidA = 0
295
296     if gkres1 == kamplitud then
297         gkfallos = gkfallos
298     else

```

```

299     gkfallos = gkfallos + 1
300   endif
301
302 endif
303
304 ;Comprobación del balance
305 if gkbal == kbalance then
306   gkValidB = 1
307 else
308   gkValidB = 0
309
310   if gkres2 == kbalance then
311     gkfallos = gkfallos
312   else
313     gkfallos = gkfallos + 1
314   endif
315
316 endif
317
318 ;Comprobación del balance
319 if gktr == ktam then
320   gkValidR = 1
321 else
322   gkValidR = 0
323
324   if gkres3 == ktam then
325     gkfallos = gkfallos
326   else
327     gkfallos = gkfallos + 1
328   endif
329
330 endif
331
332 ;Comprobación del balance
333 if krvblog2 == kagudos then
334   gkValidH = 1
335 else
336   gkValidH = 0
337
338   if gkres4 == kagudos then
339     gkfallos = gkfallos
340   else
341     gkfallos = gkfallos + 1
342   endif
343
344 endif
345
346 ;Devolvemos indicación luminosa al usuario
347 outvalue "validAmp", gkValidA
348 outvalue "validBal", gkValidB
349 outvalue "validTH", gkValidR
350 outvalue "validHF", gkValidH
351
352 ;Mostramos los resultados en el pad XY en tiempo real
353 ;Adaptamos los valores antes de mostrarlos
354 krespannuevo = kbalance + 0.5
355 klog2 = kamplitud/10
356 kAmple2 = 10^(klog2)
357 kresamp = -kAmple2
358 ;Mostramos valores en el pad XY
359 outvalue "resulpan", krespannuevo
360 outvalue "resulamp", kresamp
361

```

```

362 ;Si pulsamos en Solución, colocamos los menús con los valores de la
    solución
363 ;Primero debemos asignar cada variable aleatoria a una variable con el
    valor equivalente en el menú, para que cuando toque dar la solución
    , se muestre la que toque
364 gkresbalmenu = (gkbal+0.5)*10
365 kresthmenu = gktr*10
366
367 ;Si se pulsa solución, devuelve los valores a los menús
368 if gksoluc == 1 then
369
370     outvalue "amplitud2", -gkamprand
371     outvalue "stereo2", gkresbalmenu
372     outvalue "room2", kresthmenu
373     outvalue "hf2", -krvblog2
374
375 endif
376
377 ;Contador de fallos al final del ciclo
378 gkres1 = kamplitud
379 gkres2 = kbalance
380 gkres3 = ktam
381 gkres4 = kagudos
382 ;Creamos una cadena de texto con el número de fallos y lo mostramos en
    la interfaz gráfica
383 Sdisp sprintfk "%02d", gkfallos
384 outvalue "fallos", Sdisp
385
386 endin
387
388 ;Generación de las reverberaciones de prueba y de práctica
389 instr 4 ;Reverb práctica
390     kstereo3 = gkbal+0.5
391
392     kroomsize2 = gktr                ;Tamaño de habitación (0-1)
393     kHFDamp2   = gkhf                ;Amortiguamiento en altas frecuencias (0-1)
394
395     aRvbL2,aRvbR2 freeverb gaRvbSend2, gaRvbSend2,kroomsize2,kHFDamp2
396         outs          aRvbL2*(1-kstereo3), aRvbR2*kstereo3
397
398     clear          gaRvbSend2
399 endin
400
401 instr 5 ;Reverb prueba
402     kstereo2 invaline "stereo"
403
404     kroomsize      invaline "room"          ;Tamaño de habitación (0-1)
405     kHFDamp         invaline "hf1"          ;Amortiguamiento en altas
        frecuencias (0-1)
406
407     aRvbL,aRvbR freeverb gaRvbSend, gaRvbSend,kroomsize,kHFDamp
408         outs          aRvbL*(1-kstereo2), aRvbR*kstereo2
409
410     clear          gaRvbSend
411     endin
412
413 </CsInstruments>
414 ;Diseño de partitura
415 <CsScore>
416     ;Llamada a los instrumentos
417     i 1  0 10000 ;Instrumento de prueba
418     i 2  0 10000 ;Instrumento de práctica
419     i 4  0 10000 ;Reverb instrumento de práctica
420     i 5  0 10000 ;Reverb instrumento de prueba

```

```
421 i 10 0 10000 ;Intefaz gráfica
422 e
423 </CsScore>
424
425 </CsoundSynthesizer>
```

APÉNDICE B

Código fuente - Reconocimiento frecuencial

```
1 ;-----
2 ;RECONOCIMIENTO FRECUENCIAL v.1.0
3 ;-----
4 ; Código: Implementación de una aplicación para el entrenamiento auditivo
5 ; enfocada al reconocimiento frecuencial y al espectro auditivo.
6 ;-----
7 ; Autor: Fernando Ayelo Sánchez
8 ;-----
9 ; Fecha: 6 de Junio de 2017
10 ;-----
11
12 <CsoundSynthesizer>
13
14 ;Sección Opciones de Csound
15 <CsOptions>
16 --displays ;Activamos éste flag para visualizar los gráficos del interfaz.
17 </CsOptions>
18 ;Diseño de la orquesta
19 <CsInstruments>
20 ;Definimos las variables globales
21 sr = 44100 ;Frecuencia de muestreo
22 ksmps = 128 ;Número de muestras calculadas por periodo de control
23 nchnls = 1 ;Número de canales
24 Odbfs = 1 ;Amplitud de referencia (0dB)
25
26 ;Declaración e inicialización de las variables globales
27 gkFrecu init 0
28 gkampltudres init 0
29 gkFrecu2 init 0
30 gkampltudres2 init 0
31 gkAmp3 init 0
32 gkFrecu3 init 0
33 gkfallos init 0
34 gkAmp4 init 0
35 gkFrecu4 init 0
36
37 ;inicialización de las variables de control para el sistema de fallos
38 gkres1 init 0
39 gkres2 init 0
40 gkres3 init 0
41 gkres12 init 0
42 gkres22 init 0
43 gkres32 init 0
44
45 ;inicializamos las variables de validación a 0
46 gkValidF init 0
47 gkValidM init 0
48 gkValid0 init 0
49 gkValidF2 init 0
```

```

50 gkValidM2 init 0
51 gkValidO2 init 0
52
53 ;inicializamos las variables aleatorias a 0
54
55 gkfr1 init 0
56 gkmulti1 init 0
57 gkonda init 0
58 gkfr2 init 0
59 gkmulti2 init 0
60 gkonda2 init 0
61
62 instr 10 ;Instrumento de comunicación con la interfaz gráfica (GUI)
63 ;Instrumento de prueba 1
64 gkAmp invalue "amplitud"
65 gkmute invalue "muteoriginal"
66 gkFrec invalue "frecuencia"
67 gkonda invalue "onda"
68 gkmulti invalue "multiplo"
69
70 ;Instrumento de prueba 2
71 gkAmp2 invalue "amplitud2"
72 gkmute2 invalue "mute2"
73 gkFrec2 invalue "frecuencia2"
74 gkonda2 invalue "onda2"
75 gkmulti2 invalue "multiplo2"
76
77 ;Instrumento de entrenamiento 1
78 gkAmp3 invalue "amplitud3"
79 gkmute3 invalue "muteoriginal2"
80 gkfrec1res invalue "frec1res"
81 gkmulti1res invalue "multi1res"
82 gkonda1res invalue "onda1res"
83
84 gkgenerar1 invalue "generar1"
85 gksoluc invalue "solucion"
86
87 ;Instrumento de entrenamiento 2
88 gkAmp4 invalue "amplitud4"
89 gkmute4 invalue "mute3"
90 gkfrec2res invalue "frec2res"
91 gkmulti2res invalue "multi2res"
92 gkonda2res invalue "onda2res"
93 endin
94
95 instr 1 ;Instrumento de prueba 1
96 ;Primero valoramos el múltiplo de la frecuencia
97
98 if gkmulti == 0 then
99 gkFrecu = gkFrec * 10
100 elseif gkmulti == 1 then
101 gkFrecu = gkFrec * 100
102 elseif gkmulti == 2 then
103 gkFrecu = gkFrec * 1000
104 endif
105
106 Sdisp4 sprintfk "%02d", gkFrecu ;Creamos un string con el valor de la
    frecuencia y lo mostramos en la interfaz GUI
107 outvalue "kFrecu", Sdisp4
108
109 gkampludres = 10*log10(gkAmp) ;Adaptamos la amplitud a dB y mostramo en
    la interfaz GUI
110 outvalue "amplitudres", gkampludres
111

```

```

112 ;Dependiendo del tipo de onda se genera la señal necesaria a partir de
    las tablas de onda declaradas en la partitura
113 if gkonda == 0 then
114     aSal  oscilikt  gkAmp, gkFrecu, 1
115 elseif gkonda == 1 then
116     aSal  oscilikt  gkAmp, gkFrecu, 2
117 elseif gkonda == 2 then
118     aSal  oscilikt  gkAmp, gkFrecu, 3
119 elseif gkonda == 3 then
120     aSal  oscilikt  gkAmp, gkFrecu, 4
121 elseif gkonda == 4 then
122     aSal  oscilikt  gkAmp, gkFrecu, 5
123 elseif gkonda == 5 then
124     aSal  oscilikt  gkAmp, gkFrecu, 6
125 endif
126
127 ;Sacamos el resultado controlando si está o no en mute. Dividimos por
    dos para que no sature al tratarse de una señal mono
128 out  (gkmute*aSal)/2
129 endin
130
131 ;-----Instrumento de prueba muestra 2
132
133 instr 2 ;Instrumento de prueba 1
134 ;Primero valoramos el múltiplo de la frecuencia
135
136 if gkmulti2 == 0 then
137     gkFrecu2 = gkFrec2 * 10
138 elseif gkmulti2 == 1 then
139     gkFrecu2 = gkFrec2 * 100
140 elseif gkmulti2 == 2 then
141     gkFrecu2 = gkFrec2 * 1000
142 endif
143
144 Sdisp3    sprintfk  "%02d", gkFrecu2 ;Creamos un string con el valor de
    la frecuencia y lo mostramos en la interfaz GUI
145 outvalue "kFrecu2", Sdisp3
146
147 gkampltdres2 = 10*log10(gkAmp2) ;Adaptamos la amplitud a dB y mostramo
    en la interfaz GUI
148 outvalue "amplitudres2", gkampltdres2
149
150 ;Dependiendo del tipo de onda se genera la señal necesaria a partir de
    las tablas de onda declaradas en la partitura
151 if gkonda2 == 0 then
152     aSal2  oscilikt  gkAmp2, gkFrecu2, 1
153 elseif gkonda2 == 1 then
154     aSal2  oscilikt  gkAmp2, gkFrecu2, 2
155 elseif gkonda2 == 2 then
156     aSal2  oscilikt  gkAmp2, gkFrecu2, 3
157 elseif gkonda2 == 3 then
158     aSal2  oscilikt  gkAmp2, gkFrecu2, 4
159 elseif gkonda2 == 4 then
160     aSal2  oscilikt  gkAmp2, gkFrecu2, 5
161 elseif gkonda2 == 5 then
162     aSal2  oscilikt  gkAmp2, gkFrecu2, 6
163 endif
164
165 ;Sacamos el resultado controlando si está o no en mute. Dividimos por dos
    para que no sature al tratarse de una señal mono
166 out  (gkmute2*aSal2)/2
167 endin
168
169 instr 3 ;Instrumento de práctica 1

```

```

170 ;Asignación de las respuestas a su variable correspondiente
171
172 ;Variable de frecuencia
173
174 if gkfreclres == 0 then
175     kfreclres = 2
176 elseif gkfreclres == 1 then
177     kfreclres = 3
178 elseif gkfreclres == 2 then
179     kfreclres = 4
180 elseif gkfreclres == 3 then
181     kfreclres = 5
182 elseif gkfreclres == 4 then
183     kfreclres = 6
184 elseif gkfreclres == 5 then
185     kfreclres = 7
186 elseif gkfreclres == 6 then
187     kfreclres = 8
188     elseif gkfreclres == 7 then
189         kfreclres = 9
190 elseif gkfreclres == 8 then
191     kfreclres = 10
192 elseif gkfreclres == 9 then
193     kfreclres = 15
194 endif
195
196 ;Variable de múltiplo
197
198 if gkmulti1res == 0 then
199     kmulti1res = 10
200 elseif gkmulti1res == 1 then
201     kmulti1res = 100
202 elseif gkmulti1res == 2 then
203     kmulti1res = 1000
204 elseif gkmulti1res == 3 then
205     kmulti1res = 100000
206 endif
207
208 ;Variable de onda
209
210 if gkondalres == 0 then
211     kondalres = 1
212 elseif gkondalres == 1 then
213     kondalres = 2
214 elseif gkondalres == 2 then
215     kondalres = 3
216 elseif gkondalres == 3 then
217     kondalres = 4
218 elseif gkondalres == 4 then
219     kondalres = 5
220 elseif gkondalres == 5 then
221     kondalres = 6
222 elseif gkondalres == 6 then
223     kondalres = 10
224 endif
225
226 ;Generación de valores aleatorios y reseteo de menú de usuario
227
228 ;Si se pulsa generar, se generan los valores aleatorios y se ponen a 0
    los menús
229 if gkgenerar1 == 1 then
230 ;Creamos el aleatorio de la frecuencia
231 krnd randh 8, 10, 10
232 krnd = abs(krnd)

```

```

233     kfrecprem = round(krnd)
234
235     ;Creamos el aleatorio del múltiplo
236     krnd2 randh 2, 10, 10
237     krnd2 = abs(krnd2)
238     kmulti = round(krnd2)
239
240     ;Creamos el aleatorio del tipo de onda
241     krnd3 randh 5, 10, 10
242     krnd3 = abs(krnd3)
243     ktipo = round(krnd3)
244
245     ;Ponemos a 0 los menús para el usuario
246     outvalue "frec1res", 9
247     outvalue "multi1res", 3
248     outvalue "onda1res", 6
249
250     ;Reseteamos el contador de fallos
251     gkfallos = 0
252
253     endif
254
255     ;Asignamos los valores reales para la frecuencia aleatoria. Desde el
        aleatorio no se pueden asignar tal y como queremos, por lo que
        debemos adaptarlos
256
257     if kfrecprem == 0 then
258         gkfr1 = 2
259     elseif kfrecprem == 1 then
260         gkfr1 = 3
261     elseif kfrecprem == 2 then
262         gkfr1 = 4
263     elseif kfrecprem == 3 then
264         gkfr1 = 5
265     elseif kfrecprem == 4 then
266         gkfr1 = 6
267     elseif kfrecprem == 5 then
268         gkfr1 = 7
269     elseif kfrecprem == 6 then
270         gkfr1 = 8
271     elseif kfrecprem == 7 then
272         gkfr1 = 9
273     elseif kfrecprem == 8 then
274         gkfr1 = 10
275     endif
276
277     ;Asignamos los valores reales de los múltiplos y por tanto sacamos la
        frecuencia aleatoria final
278
279     if kmulti == 0 then
280         gkmulti1 = 10
281     elseif kmulti == 1 then
282         gkmulti1 = 100
283     elseif kmulti == 2 then
284         gkmulti1 = 1000
285     endif
286
287     ;Asignamos los valores para los tipos de onda ya que el aleatorio no lo
        genera como nosotros queremos
288
289     if ktipo == 0 then
290         gkonda1 = 1
291     elseif ktipo == 1 then
292         gkonda1 = 2

```

```

293 elseif ktipo == 2 then
294     gkonda1 = 3
295 elseif ktipo == 3 then
296     gkonda1 = 4
297 elseif ktipo == 4 then
298     gkonda1 = 5
299 elseif ktipo == 5 then
300     gkonda1 = 6
301 elseif ktipo == 6 then
302     gkonda1 = 7
303 endif
304
305 gkFrecu3 = gkfr1*gkmulti1 ;Multiplicamos la frecuencia creada por su múltiplo para obtener la frecuencia final
306
307 ;Dependiendo del tipo de onda se genera la señal necesaria a partir de las tablas de onda declaradas en la partitura
308 aSal3 oscilikt gkAmp3, gkFrecu3, gkonda1
309
310 ;Sacamos el resultado controlando si está o no en mute. Dividimos por dos para que no sature al tratarse de una señal mono
311 out (gkmute3*aSal3)/2
312
313 ;Pasamos la amplitud a formato dB y mostramos en la interfaz GUI
314 gkampltudres3 = 10*log10(gkAmp3)
315 outvalue "amplitudres3", gkampltudres3
316
317 ;Comprobación de resultados
318
319 ;comprobación de la frecuencia
320 if gkfr1 == kfrecre then
321     gkValidF = 1
322 else
323     gkValidF = 0
324
325     if gkres1 == kfrecre then
326         gkfallos = gkfallos
327     else
328         gkfallos = gkfallos + 1
329     endif
330 endif
331
332 ;comprobación del múltiplo
333 if gkmulti1 == kmultires then
334     gkValidM = 1
335 else
336     gkValidM = 0
337
338     if gkres2 == kmultires then
339         gkfallos = gkfallos
340     else
341         gkfallos = gkfallos + 1
342     endif
343 endif
344
345 ;comprobación del tipo de onda
346 if gkonda1 == kondares then
347     gkValidO = 1
348 else
349     gkValidO = 0
350
351     if gkres3 == kondares then
352         gkfallos = gkfallos
353     else

```

```

354         gkfallos = gkfallos + 1
355     endif
356 endif
357
358 ;Devolvemos indicación luminosa al usuario
359 outvalue "validFrec", gkValidF
360 outvalue "validMulti", gkValidM
361 outvalue "validOnda", gkValidO
362
363 ;Si pulsamos en Solución, colocamos los menús con los valores de la
    solución
364 if gksoluc == 1 then
365
366     outvalue "frec1res", kfreqrem
367     outvalue "multi1res", kmulti
368     outvalue "onda1res", ktipo
369
370 endif
371
372 ;Contador de fallos al final del ciclo
373 gkres1 = kfreqres
374 gkres2 = kmultires
375 gkres3 = kondares
376 ;Creamos una cadena de texto con el número de fallos y lo mostramos en
    la interfaz gráfica
377 Sdisp sprintfk "%02d", gkfallos
378 outvalue "fallos", Sdisp
379
380 endin
381
382 instr 4 ;Instrumento de práctica 2
383 ;Asignación de las respuestas a su variable correspondiente
384
385 ;Variable de frecuencia
386
387 if gkfrec2res == 0 then
388     kfreqres = 2
389 elseif gkfrec2res == 1 then
390     kfreqres = 3
391 elseif gkfrec2res == 2 then
392     kfreqres = 4
393 elseif gkfrec2res == 3 then
394     kfreqres = 5
395 elseif gkfrec2res == 4 then
396     kfreqres = 6
397 elseif gkfrec2res == 5 then
398     kfreqres = 7
399 elseif gkfrec2res == 6 then
400     kfreqres = 8
401 elseif gkfrec2res == 7 then
402     kfreqres = 9
403 elseif gkfrec2res == 8 then
404     kfreqres = 10
405 elseif gkfrec2res == 9 then
406     kfreqres = 15
407 endif
408
409 ;Variable de múltiplo
410
411 if gkmulti2res == 0 then
412     kmultires = 10
413 elseif gkmulti2res == 1 then
414     kmultires = 100
415 elseif gkmulti2res == 2 then

```

```

416     kmultires = 1000
417 elseif gkmulti2res == 3 then
418     kmultires = 100000
419 endif
420
421 ;Variable de onda
422
423 if gkonda2res == 0 then
424     kondares = 1
425 elseif gkonda2res == 1 then
426     kondares = 2
427 elseif gkonda2res == 2 then
428     kondares = 3
429 elseif gkonda2res == 3 then
430     kondares = 4
431 elseif gkonda2res == 4 then
432     kondares = 5
433 elseif gkonda2res == 5 then
434     kondares = 6
435 elseif gkonda2res == 6 then
436     kondares = 10
437 endif
438
439 ;Generación de valores aleatorios y reseteo de menú de usuario
440
441 ;Si se pulsa generar, se generan los valores aleatorios y se ponen a 0
    los menús
442
443 if gkgenerar1 == 1 then
444 ;Creamos el aleatorio de la frecuencia
445 krnd randh 8, 10, 10
446 krnd = abs(krnd)
447 kfrecprem = round(krnd)
448
449 ;Creamos el aleatorio del múltiplo
450 krnd2 randh 2, 10, 10
451 krnd2 = abs(krnd2)
452 kmulti = round(krnd2)
453
454 ;Creamos el aleatorio del tipo de onda
455 krnd3 randh 5, 10, 10
456 krnd3 = abs(krnd3)
457 ktipo = round(krnd3)
458
459 ;Ponemos a 0 los menús para el usuario
460 outvalue "frec2res", 9
461 outvalue "multi2res", 3
462 outvalue "onda2res", 6
463
464 ;Reseteamos el contador de fallos
465 gkfallos = 0
466 endif
467
468 ;Asignamos los valores reales para la frecuencia aleatoria. Desde el
    aleatorio no se pueden asignar tal y como queremos, por lo que
    debemos adaptarlos
469
470 if kfrecprem == 0 then
471     gkfr2 = 2
472 elseif kfrecprem == 1 then
473     gkfr2 = 3
474 elseif kfrecprem == 2 then
475     gkfr2 = 4
476 elseif kfrecprem == 3 then

```

```

477     gkfr2 = 5
478 elseif kfrecprem == 4 then
479     gkfr2 = 6
480 elseif kfrecprem == 5 then
481     gkfr2 = 7
482 elseif kfrecprem == 6 then
483     gkfr2 = 8
484 elseif kfrecprem == 7 then
485     gkfr2 = 9
486 elseif kfrecprem == 8 then
487     gkfr2 = 10
488 endif
489
490 ;Asignamos los valores reales de los múltiplos y por tanto sacamos la
    frecuencia aleatoria final
491
492 if kmulti == 0 then
493     gkmulti2 = 10
494 elseif kmulti == 1 then
495     gkmulti2 = 100
496 elseif kmulti == 2 then
497     gkmulti2 = 1000
498 endif
499
500 ;Asignamos los valores para los tipos de onda ya que el aleatorio no lo
    genera como nosotros queremos
501
502 if ktipo == 0 then
503     gkonda2 = 1
504 elseif ktipo == 1 then
505     gkonda2 = 2
506 elseif ktipo == 2 then
507     gkonda2 = 3
508 elseif ktipo == 3 then
509     gkonda2 = 4
510 elseif ktipo == 4 then
511     gkonda2 = 5
512 elseif ktipo == 5 then
513     gkonda2 = 6
514 elseif ktipo == 6 then
515     gkonda2 = 7
516 endif
517
518
519 gkFrecu4 = gkfr2*gkmulti2 ;Multiplicamos la frecuencia creada por su mú
    ltiple para obtener la frecuencia final
520
521 ;Dependiendo del tipo de onda se genera la señal necesaria a partir de
    las tablas de onda declaradas en la partitura
522 aSal4 oscilikt gkAmp4, gkFrecu4, gkonda2
523
524 ;Sacamos el resultado controlando si está o no en mute. Dividimos por dos
    para que no saturate al tratarse de una señal mono
525 out (gkmute4*aSal4)/2
526
527 ;Pasamos la amplitud a formato dB y mostramos por interfaz GUI
528 gkampludres4 = 10*log10(gkAmp4)
529 outvalue "amplitudres4", gkampludres4
530
531 ;Comprobación de resultados
532
533 ;comprobación de la frecuencia
534 if gkfr2 == kfrecres then
535     gkValidF2 = 1

```

```

536     else
537         gkValidF2 = 0
538
539         if gkres12 == kfrecres then
540             gkfallos = gkfallos
541         else
542             gkfallos = gkfallos + 1
543         endif
544
545     endif
546
547     ;comprobación del múltiplo
548     if gkmulti2 == kmultires then
549         gkValidM2 = 1
550     else
551         gkValidM2 = 0
552
553         if gkres22 == kmultires then
554             gkfallos = gkfallos
555         else
556             gkfallos = gkfallos + 1
557         endif
558
559     endif
560
561     ;comprobación del tipo de onda
562     if gkonda2 == kondares then
563         gkValidO2 = 1
564     else
565         gkValidO2 = 0
566
567         if gkres32 == kondares then
568             gkfallos = gkfallos
569         else
570             gkfallos = gkfallos + 1
571         endif
572
573     endif
574
575     ;Devolvemos indicación luminosa al usuario
576     outvalue "validFrec2", gkValidF2
577     outvalue "validMulti2", gkValidM2
578     outvalue "validOnda2", gkValidO2
579
580     ;Si se pulsa solución, devuelve los valores a los menús con la solución
581     if gksoluc == 1 then
582
583         outvalue "frec2res", kfrecprem
584         outvalue "multi2res", kmulti
585         outvalue "onda2res", ktipo
586
587     endif
588
589     ;Contador de fallos al final del ciclo
590     gkres12 = kfrecres
591     gkres22 = kmultires
592     gkres32 = kondares
593     ;Creamos una cadena de texto con el número de fallos y lo mostramos en la
594     interfaz gráfica
595     Sdisp sprintfk "%02d", gkfallos
596     outvalue "fallos1", Sdisp
597
598     endin

```

```

599 </CsInstruments>
600 ;Diseño de partitura
601 <CsScore>
602 t 0 60
603 ;Declaración de las tablas de onda a partir de la GEN10 de CSound
604 f 1 0 2048 10 1
;
Sinusoidal
605 f 2 0 2048 10 1 0.5 0.3 0.25 0.2 0.167 0.14 0.125
.111 ;Diente de sierra
606 f 3 0 2048 10 1 0 0.3 0 0.2 0 0.14 0
.111 ;Cuadrada
607 f 4 0 2048 10 1 1 1 1 0.7 0.5 0.3 0.1
;Pulso
608 f 5 0 8192 7 -1 8192 1 ;segmento lineal de -1 a 1
609 f 6 0 128 8 -1 16 [2/(1+2.7183^(-10*(-0.75)))-1] 16
[2/(1+2.7183^(-10*(-0.5)))-1] 16 \
610 [2/(1+2.7183^(-10*(-0.25)))-1] 33 [2/(1+2.7183^(-10*(0.25)))-1] 16
\
611 [2/(1+2.7183^(-10*(0.5)))-1] 16 [2/(1+2.7183^(-10*(0.75)))-1] 16 1
;Sigmoide
612 ;Llamada a los instrumentos
613 i 1 0 10000 ;Instrumento prueba 1
614 i 2 0 10000 ;Instrumento prueba 2
615 i 3 0 10000 ;Instrumento prácticas 1
616 i 4 0 10000 ;Instrumento prácticas 2
617 i 10 0 10000 ;Interfaz gráfica
618 e
619 </CsScore>
620 </CsoundSynthesizer>

```

APÉNDICE C

Código fuente - Ruido y distorsión

```
1 ;-----
2 ;RUIDO Y DISTORSIÓN v.1.0
3 ;-----
4 ; Código: Implementación de una aplicación para el entrenamiento auditivo
5 ; enfocada al ruido y la distorsión en el audio.
6 ;-----
7 ; Autor: Fernando Ayelo Sánchez
8 ;-----
9 ; Fecha: 6 de Junio de 2017
10 ;-----
11
12 <CsoundSynthesizer>
13
14 ;Sección "Opciones" de Csound
15 <CsOptions>
16 --displays ;Activamos éste flag para visualizar los gráficos del interfaz.
17 </CsOptions>
18 ;Diseño de la orquesta
19 <CsInstruments>
20 ;Definimos las variables globales
21 sr = 44100 ;Frecuencia de muestreo
22 ksmps = 1 ;Número de muestras calculadas por periodo de control
23 nchnls = 2 ;Número de canales
24
25 ;Inicialización de variables globales
26 gkstereo = 0.5
27 gkbitsprof init 0
28 gkfallos init 0
29 gkAmple init 0
30 gkAmplitud init 0
31 gkgenerar init 0
32 gkcontador init 0
33
34 ;inicializamos las variables de validación a 1 (encendidas)
35 gkValidA init 1
36 gkValidN init 1
37 gkValidB init 1
38 gkValidS init 1
39
40 gkValidRuido init 1
41 gkValidCuant init 1
42 gkValidMues init 1
43
44 ;inicialización de las variables de control para el sistema de fallos
45 gkres1 init 0
46 gkres2 init 0
47 gkres3 init 0
48 gkres4 init 0
49 gkres5 init 1
```

```

50 gkres6 init 1
51 gkres7 init 1
52
53 ;Inicializamos a 0 para que empiece por la dificultad inicial, 1
    degradación
54 outvalue "dificultad",0
55
56 instr 10 ;Instrumento de comunicación con la interfaz gráfica (GUI)
57 ktrig metro 10 ;Al necesitar por muestras independientes, utilizamos la
    función metro para controlar las interacciones por pasos
58 if (ktrig == 1) then
59
60     ;Instrumento de prueba
61     ;Main
62     gkAmp invalue "amplitud"
63     gkmute invalue "muteoriginal"
64     gkstereo invalue "stereo"
65     gkmuestra invalue "muestra"
66
67     outvalue "stereo", gkstereo
68
69     gkamplitudres = 10*log10(gkAmp)
70     outvalue "ampres", gkamplitudres
71     outvalue "panres", gkstereo
72
73     ;Profundidad de bits
74     gkbitsprof invalue "bitsprof"
75     gkbitsmute invalue "bitsmute"
76
77     Sbits sprintfk "%02d", gkbitsprof
78     outvalue "bitsres", Sbits
79
80     ;Ruido
81     gknoise invalue "noise"
82     gknoisemute invalue "noisemute"
83     gknoiseamp invalue "ampnoise"
84     gksolo invalue "solo"
85
86     outvalue "ampnoiseres", gknoiseamp
87
88     ;Sampling
89     gksampmute invalue "sampmute"
90     gksamp invalue "samp"
91
92     Sfrecu sprintfk "%02d", 44100/-gksamp
93     outvalue "sampres", Sfrecu
94
95     ;Instrumento de prácticas
96     ;Main
97     gkmuestra2 invalue "muestra2"
98     gkmute2 invalue "muteoriginal2"
99     gkAmp2 invalue "amplitud2"
100    gkgenerar invalue "generar"
101
102    gkamplitudres2 = 10*log10(gkAmp2)
103    outvalue "ampres2", gkamplitudres2
104
105    ;Dificultad
106    gkdificultad invalue "dificultad"
107
108    ;Soluciones
109    gksoluc invalue "soluciones"
110
111    ;¿Que degradación hay?

```

```

112 gkcheckruido invalue "checkruido"
113 gkcheckcuant invalue "checkcuant"
114 gkcheckmuestreo invalue "checkmuestreo"
115
116 outvalue "validRuidoC", gkValidRuido
117 outvalue "validCuantC", gkValidCuant
118 outvalue "validMuesC", gkValidMues
119
120 ;Ruido
121 gkruidores invalue "ruidores"
122 gkruidoampres invalue "ruidoampres"
123
124 outvalue "validnoise", gkValidN
125 outvalue "validnoiseamp", gkValidA
126
127 ;Cuantización
128 gkbitsres invalue "bitsres2"
129
130 outvalue "validbits", gkValidB
131
132 ;Muestreo
133 gksampres invalue "sampres2"
134
135 outvalue "validsamp", gkValidS
136
137 endif
138 endin
139
140 instr 1 ;Instrumento de prueba
141 ;Carga de la muestra de audio según decida el usuario
142 if gkmuestra == 0 then
143 aMuestra disk2 "muestra1.wav",1,0,1
144 elseif gkmuestra == 1 then
145 aMuestra disk2 "muestra2.wav",1,0,1
146 elseif gkmuestra == 2 then
147 aMuestra disk2 "muestra3.wav",1,0,1
148 elseif gkmuestra == 3 then
149 aMuestra disk2 "muestra4.wav",1,0,1
150 elseif gkmuestra == 4 then
151 aMuestra disk2 "muestra5.wav",1,0,1
152 elseif gkmuestra == 5 then
153 aMuestra inch 1
154 endif
155
156 kstereo = gkstereo + 0.5 ;Adaptación de la variable estéreo
157
158 ;Seccion de ruidos
159 klogi = gknoiseamp/10 ;Pasamos la amplitud a formato dB
160 gkAmplitud = 10^(klogi)
161
162 if gknoisemute == 1 then ;Comprobamos si la sección de ruido está
    activa. Si está, generamos el ruido correspondiente
163
164     if gknoise == 0 then
165         aNoise noise gkAmplitud*15000, 0 ;Sección para crear ruido blanco
166     elseif gknoise == 1 then
167         aNoise pinkish gkAmplitud*15000 ;Sección para crear ruido rosa
168     elseif gknoise == 2 then
169         aNoise disk2 "noise.wav",1,0,1 ;Para el ruido eléctrico hacemos
            uso de una muestra de audio
170         aNoise = aNoise * gkAmplitud
171     endif
172
173 endif

```

```

174
175 ;Seccion de resolucio de bits
176 if gkbtsmute == 1 then ;Comprobamos si la secci3n de cuantizaci3n est3
    activa. Si est3, generamos la muestra con la cuantizaci3n
    correspondiente
177
178     kvalues pow 2, gkbtsprof
179     k16bit pow 2, 16
180     ksig downsamp aMuestra
181     ksig = int(ksig * (kvalues/k16bit))
182     ksig = ksig * (k16bit/kvalues)
183     aMuestra interp ksig
184
185 endif
186
187 ;Secci3n de frecuencia de muestreo
188 kincr = -gksamp ;Adaptamos la variable del m3ltiplo para la reducci3n
    de la frecuencia de muestreo
189
190 if gksampmute == 1 then ;Comprobamos si la secci3n de muestreo est3
    activa. Si est3, generamos la muestra con la reducci3n de Fs
    correspondiente
191     aMuestra fold aMuestra, kincr
192 endif
193
194 ;Procesamos aMuestra y la sacamos por salida de audio aplicando las
    degradaciones correspondientes
195
196 aMuestra = gkAmp*(aMuestra+aNoise)
197 aSal = (gkmute*aMuestra)
198
199     if gksolo == 1 then
200     outs aNoise/2, aNoise/2
201     else
202     outs aSal*(1-kstereo), aSal*kstereo
203     endif
204 endin
205
206 instr 2 ;Instrumento de entrenamiento
207 ;Carga de la muestra de audio seg3n decida el usuario
208 if gkmuestra2 == 0 then
209     aMuestra2 disk2 "muestra1.wav",1,0,1
210 elseif gkmuestra2 == 1 then
211     aMuestra2 disk2 "muestra2.wav",1,0,1
212 elseif gkmuestra2 == 2 then
213     aMuestra2 disk2 "muestra3.wav",1,0,1
214 elseif gkmuestra2 == 3 then
215     aMuestra2 disk2 "muestra4.wav",1,0,1
216 elseif gkmuestra2 == 4 then
217     aMuestra2 disk2 "muestra5.wav",1,0,1
218 elseif gkmuestra2 == 5 then
219     aMuestra2 inch 1
220 endif
221
222 ;Asignaci3n de las respuestas a su variable correspondiente
223 ;Variable de ruido
224 if gkruidores == 0 then
225     kruidousuario = 0
226 elseif gkruidores == 1 then
227     kruidousuario = 1
228 elseif gkruidores == 2 then
229     kruidousuario = 2
230 elseif gkruidores == 3 then
231     kruidousuario = -2

```

```
232     endif
233
234     ;Variable de amplitud
235     if gkruidoampres == 0 then
236         kamplitud = 0
237     elseif gkruidoampres == 1 then
238         kamplitud = -1
239     elseif gkruidoampres == 2 then
240         kamplitud = -2
241     elseif gkruidoampres == 3 then
242         kamplitud = -3
243     elseif gkruidoampres == 4 then
244         kamplitud = -4
245     elseif gkruidoampres == 5 then
246         kamplitud = -5
247     elseif gkruidoampres == 6 then
248         kamplitud = -6
249     elseif gkruidoampres == 7 then
250         kamplitud = -7
251     elseif gkruidoampres == 8 then
252         kamplitud = -8
253     elseif gkruidoampres == 9 then
254         kamplitud = -9
255     elseif gkruidoampres == 10 then
256         kamplitud = -10
257     elseif gkruidoampres == 11 then
258         kamplitud = 20
259     endif
260
261     ;Variable de bits
262     if gkbitsres == 0 then
263         kbitsusuario = 4
264     elseif gkbitsres == 1 then
265         kbitsusuario = 5
266     elseif gkbitsres == 2 then
267         kbitsusuario = 6
268     elseif gkbitsres == 3 then
269         kbitsusuario = 7
270     elseif gkbitsres == 4 then
271         kbitsusuario = 8
272     elseif gkbitsres == 5 then
273         kbitsusuario = 9
274     elseif gkbitsres == 6 then
275         kbitsusuario = 10
276     elseif gkbitsres == 7 then
277         kbitsusuario = 11
278     elseif gkbitsres == 8 then
279         kbitsusuario = 12
280     elseif gkbitsres == 9 then
281         kbitsusuario = 13
282     elseif gkbitsres == 10 then
283         kbitsusuario = 14
284     elseif gkbitsres == 11 then
285         kbitsusuario = 40
286     endif
287
288     ;Variable de sampling
289     if gksampres == 0 then
290         ksampusuario = 10
291     elseif gksampres == 1 then
292         ksampusuario = 9
293     elseif gksampres == 2 then
294         ksampusuario = 8
295     elseif gksampres == 3 then
```

```

296 ksampusuario = 7
297 elseif gksampres == 4 then
298 ksampusuario = 6
299 elseif gksampres == 5 then
300 ksampusuario = 5
301 elseif gksampres == 6 then
302 ksampusuario = 4
303 elseif gksampres == 7 then
304 ksampusuario = 3
305 elseif gksampres == 8 then
306 ksampusuario = 2
307 elseif gksampres == 9 then
308 ksampusuario = 20
309 endif
310
311
312 ;Generación de valores aleatorios y reseteo de menús de usuario
313 ;si se pulsa generar, se generan los valores aleatorios y se ponen a 0
    los menús
314 if gkgenerar == 1 then
315     ;Creamos el aleatorio del ruido
316     krnd2 randh 2,10,10
317     krnd2 = abs(krnd2)
318     kvar1 = round(krnd2)
319     gknoiserand = kvar1
320
321     ;Creamos el aleatorio de la amplitud
322     krnd randh 10, 10, 10
323     krnd = abs(krnd)
324     kamprand = round(krnd)
325     gkamprand = -kamprand
326
327     ;Creamos el aleatorio de bits
328     krnd3 randh 10,10,10
329     krnd3 = abs(krnd3)
330     kbitsrand = round(krnd3)
331     gkbitsrand = kbitsrand+4
332
333     ;Creamos el aleatorio de frecuencia de muestreo
334     krnd4 randh 8,10,10
335     krnd4 = abs(krnd4)
336     ksamprand = round(krnd4)
337     gksamprand = ksamprand+2
338
339     ;Ponemos a 0 los menús para el usuario
340     gkValidA = 0
341     gkValidN = 0
342     gkValidB = 0
343     gkValidS = 0
344
345     outvalue "ruidores", 3
346     outvalue "ruidoampres", 11
347     outvalue "bitsres2", 11
348     outvalue "sampres2", 9
349
350     ;Reseteamos la zona superior de checks de degradaciones para evitar
        trampas
351     gkValidRuido = 0
352     gkValidCuant = 0
353     gkValidMues = 0
354
355     outvalue "checkruido", 0
356     outvalue "checkmuestreo", 0
357     outvalue "checkcuant", 0

```

```

358
359 ;Aleatorios para repartir el tipo de degradaciones
360 ;Genero un valor aleatorio entre 0 y 2 (1 a 3) que nos servirá para
    aplicar una de las tres degradaciones de manera aleatorias
361 knum randh 2,10,10
362 knum = abs(knum)
363 knumer = round(knum)
364
365 ;Reseteamos el contador de fallos
366 gkfallos = 0
367
368 ;Ponemos el contador a 1 para saber el número de rondas que lleva
369 gkcontador = gkcontador+1
370
371 endif
372
373 ;Generamos el resultado dependiendo del tipo de onda
374 ;Seccion de resolucio n de bits
375 kvalues pow 2, gkbitsrand
376 k16bit pow 2, 16
377 ksig downsamp aMuestra2
378 ksig = int(ksig * (kvalues/k16bit))
379 ksig = ksig * (k16bit/kvalues)
380 aMuestrabits interp ksig
381
382 ;Seccion de ruidos
383 klog = gkamprand/10
384 gkAmple = 10^(klog)
385
386 if gknoiserand == 0 then
387     aNoise noise gkAmple*15000, 0
388 elseif gknoiserand == 1 then
389     aNoise pinkish gkAmple *15000
390 elseif gknoiserand == 2 then
391     aNoise disk2 "noise.wav",1,0,1
392     aNoise = aNoise * gkAmple
393 endif
394
395 ;Sección de frecuencia de muestreo
396 aMuestramuestreo fold aMuestra2, gksamprand
397
398 ;Dependiendo de la dificultad, aplicamos una, dos o tres degradaciones
399
400 if gkdificultad == 0 then
401     ;Dependiendo del número, aplico una degradación u otra a la salida 1-
        Ruido 2- Cuantización 3- Muestreo
402     if knumer == 0 then ;Ruido
403         aMuestrafin = aMuestra2+aNoise
404         gkflag = 0
405     elseif knumer == 1 then ;Cuantizacion
406         aMuestrafin = aMuestrabits
407         gkflag = 1
408     elseif knumer == 2 then ;Muestreo
409         aMuestrafin = aMuestramuestreo
410         gkflag = 2
411     endif
412
413 elseif gkdificultad == 1 then
414     ;Dependiendo del número, NO APLICO esa degradación a la salida 1- Ruido
        2- Cuantización 3- Muestreo
415     if knumer == 0 then ;Ruido
416         aMuestrafin = (aMuestrabits/2)+(aMuestramuestreo/2)
417         gkflag = 3
418     elseif knumer == 1 then ;Cuantizacion

```

```

419     aMuestraFin = ((aMuestra2+aNoise)/2)+(aMuestramuestreo/2)
420     gkflag = 4
421 elseif knumer == 2 then ;Muestreo
422     aMuestraFin = (aMuestrabits/2)+((aMuestra2+aNoise)/2)
423     gkflag = 5
424 endif
425
426 elseif gkdificultad == 2 then
427     aMuestraFin = ((aMuestra2+aNoise)/3)+(aMuestrabits/3)+(aMuestramuestreo
428         /3) ;Debemos tener en cuenta que al sumar varias señales, estas
429         deben ser divididas para no saturar
430     gkflag = 6
431 endif
432
433 ;Procesamos aMuestra y la sacamos por salida de audio, teniendo en
434 cuenta la amplitud, el mute y sacando estéreo
435 aMues = gkAmp2 *aMuestraFin
436 aSal = gkmute2*aMues
437 outs      aSal/2,aSal/2
438
439 ;Comprobación de resultados
440 ;Dependiendo del flag, sabremos que tipo de degradación tenemos, y
441 comprobaremos en base a eso cada uno de los parámetros presentes en
442 el programa
443
444 if gkflag == 0 then
445     if gkcheckruido == 1 then ;Primero comprobamos la respuesta para
446         saber si debemos encender indicador luminoso
447         gkValidRuido = 1
448     else
449         gkValidRuido = 0
450     endif
451 endif
452
453 if gkcheckcuant == 1 then ;Segundo comprobamos si alguno de los checks
454     que no van en este flag se pulsan. Si se pulsan debemos hacer la
455     comprobación para sumar un fallo al contador
456     if gkcheckcuant == gkres5 then
457         gkfallos = gkfallos
458     else
459         gkfallos = gkfallos + 1
460     endif
461 endif
462
463 if gkcheckmuestreo == 1 then ;Segundo comprobamos si alguno de los
464     checks que no van en este flag se pulsan. Si se pulsan debemos
465     hacer la comprobación para sumar un fallo al contador
466     if gkcheckmuestreo == gkres6 then
467         gkfallos = gkfallos
468     else
469         if gkcontador == 4410 then ;Condicionante para solucionar bug en el
470             contador de fallos. De ésta manera no suma en la primera
471             generación de muestras
472             gkfallos = gkfallos
473         else
474             gkfallos = gkfallos +1
475         endif
476     endif
477 endif
478
479 ;comprobación del ruido
480 if gknoiserand == kruidousuario then
481     gkValidN = 1
482 else
483     gkValidN = 0
484

```

```

471     if gkres1 == kruidousuario then
472         gkfallos = gkfallos
473     else
474         if gkcontador == 4410 then ;Condicionante para solucionar bug en
            el contador de fallos. De ésta manera no suma en la primera
            generación de muestras
475             gkfallos = gkfallos
476         else
477             gkfallos = gkfallos +1
478         endif
479     endif
480 endif
481
482 ;comprobación de la amplitud
483 if gkamprand == kamplitud then
484     gkValidA = 1
485 else
486     gkValidA = 0
487     if gkres2 == kamplitud then
488         gkfallos = gkfallos
489     else
490         if gkcontador == 4410 then ;Condicionante para solucionar bug en
            el contador de fallos. De ésta manera no suma en la primera
            generación de muestras
491             gkfallos = gkfallos
492         else
493             gkfallos = gkfallos +1
494         endif
495     endif
496 endif
497
498 elseif gkflag == 1 then
499     if gkcheckmuestreo == 1 then
500         gkValidMues = 1
501     else
502         gkValidMues = 0
503     endif
504
505 if gkcheckcuant == 1 then ;Segundo comprobamos si alguno de los checks
    que no van en este flag se pulsan. Si se pulsan debemos hacen la
    comprobación para sumar un fallo al contador
506 if gkcheckcuant == gkres5 then
507     gkfallos = gkfallos
508 else
509     if gkcontador == 4410 then ;Condicionante para solucionar bug en el
        contador de fallos. De ésta manera no suma en la primera
        generación de muestras
510         gkfallos = gkfallos
511     else
512         gkfallos = gkfallos +1
513     endif
514 endif
515 endif
516
517 if gkcheckruido == 1 then ;Segundo comprobamos si alguno de los checks
    que no van en este flag se pulsan. Si se pulsan debemos hacen la
    comprobación para sumar un fallo al contador
518 if gkcheckruido == gkres7 then
519     gkfallos = gkfallos
520 else
521     if gkcontador == 4410 then ;Condicionante para solucionar bug en el
        contador de fallos. De ésta manera no suma en la primera
        generación de muestras
522         gkfallos = gkfallos

```

```

523         else
524             gkfallos = gkfallos +1
525         endif
526     endif
527 endif
528
529 ;comprobación de frec. muestreo
530 if gksamprand == ksampusuario then
531     gkValidS = 1
532 else
533     gkValidS = 0
534     if gkres4 == ksampusuario then
535         gkfallos = gkfallos
536     else
537         if gkcontador == 4410 then ;Condicionante para solucionar bug en
                    el contador de fallos. De ésta manera no suma en la primera
                    generación de muestras
538             gkfallos = gkfallos
539         else
540             gkfallos = gkfallos +1
541         endif
542     endif
543 endif
544
545 elseif gkflag == 2 then
546     if gkcheckcuant == 1 then
547         gkValidCuant = 1
548     else
549         gkValidCuant = 0
550     endif
551
552 if gkcheckruido == 1 then ;Segundo comprobamos si alguno de los checks
                    que no van en este flag se pulsan. Si se pulsan debemos hacen la
                    comprobación para sumar un fallo al contador
553 if gkcheckruido == gkres7 then
554     gkfallos = gkfallos
555 else
556     if gkcontador == 4410 then ;Condicionante para solucionar bug en el
                    contador de fallos. De ésta manera no suma en la primera
                    generación de muestras
557         gkfallos = gkfallos
558     else
559         gkfallos = gkfallos +1
560     endif
561 endif
562 endif
563
564 if gkcheckmuestreo == 1 then ;Segundo comprobamos si alguno de los
                    checks que no van en este flag se pulsan. Si se pulsan debemos
                    hacen la comprobación para sumar un fallo al contador
565 if gkcheckmuestreo == gkres6 then
566     gkfallos = gkfallos
567 else
568     if gkcontador == 4410 then ;Condicionante para solucionar bug en el
                    contador de fallos. De ésta manera no suma en la primera
                    generación de muestras
569         gkfallos = gkfallos
570     else
571         gkfallos = gkfallos +1
572     endif
573 endif
574 endif
575
576 ;comprobación de bits

```

```

577     if gkbitsrand == kbitsusuario then
578         gkValidB = 1
579     else
580         gkValidB = 0
581         if gkres3 == kbitsusuario then
582             gkfallos = gkfallos
583         else
584             if gkcontador == 4410 then ;Condicionante para solucionar bug en
                    el contador de fallos. De ésta manera no suma en la primera
                    generación de muestras
585                 gkfallos = gkfallos
586             else
587                 gkfallos = gkfallos +1
588             endif
589         endif
590     endif
591
592 elseif gkflag == 3 then
593     if gkcheckcuant == 1 then
594         gkValidCuant = 1
595     else
596         gkValidCuant = 0
597     endif
598
599     if gkcheckmuestreo == 1 then
600         gkValidMues = 1
601     else
602         gkValidMues = 0
603     endif
604
605     if gkcheckruido == 1 then ;Segundo comprobamos si alguno de los checks
                    que no van en este flag se pulsan. Si se pulsan debemos hacen la
                    comprobación para sumar un fallo al contador
606         if gkcheckruido == gkres7 then
607             gkfallos = gkfallos
608         else
609             if gkcontador == 4410 then ;Condicionante para solucionar bug en el
                    contador de fallos. De ésta manera no suma en la primera
                    generación de muestras
610                 gkfallos = gkfallos
611             else
612                 gkfallos = gkfallos +1
613             endif
614         endif
615     endif
616
617     ;comprobación de frec. muestreo
618     if gksamprand == ksampusuario then
619         gkValidS = 1
620     else
621         gkValidS = 0
622         if gkres4 == ksampusuario then
623             gkfallos = gkfallos
624         else
625             if gkcontador == 4410 then ;Condicionante para solucionar bug en
                    el contador de fallos. De ésta manera no suma en la primera
                    generación de muestras
626                 gkfallos = gkfallos
627             else
628                 gkfallos = gkfallos +1
629             endif
630         endif
631     endif
632

```

```

633 ;comprobación de bits
634 if gkbitsrand == kbitsusuario then
635     gkValidB = 1
636 else
637     gkValidB = 0
638     if gkres3 == kbitsusuario then
639         gkfallos = gkfallos
640     else
641         if gkcontador == 4410 then ;Condicionante para solucionar bug en
            el contador de fallos. De ésta manera no suma en la primera
            generación de muestras
642             gkfallos = gkfallos
643         else
644             gkfallos = gkfallos +1
645         endif
646     endif
647 endif
648
649 elseif gkflag == 4 then
650     if gkcheckruido == 1 then
651         gkValidRuido = 1
652     else
653         gkValidRuido = 0
654     endif
655
656     if gkcheckmuestreo == 1 then
657         gkValidMues = 1
658     else
659         gkValidMues = 0
660     endif
661
662     if gkcheckcuant == 1 then ;Segundo comprobamos si alguno de los checks
        que no van en este flag se pulsan. Si se pulsan debemos hacen la
        comprobación para sumar un fallo al contador
663         if gkcheckcuant == gkres5 then
664             gkfallos = gkfallos
665         else
666             if gkcontador == 4410 then ;Condicionante para solucionar bug en el
                contador de fallos. De ésta manera no suma en la primera
                generación de muestras
667                 gkfallos = gkfallos
668             else
669                 gkfallos = gkfallos +1
670             endif
671         endif
672     endif
673
674     ;comprobación del ruido
675     if gknoiserand == kruidousuario then
676         gkValidN = 1
677     else
678         gkValidN = 0
679         if gkres1 == kruidousuario then
680             gkfallos = gkfallos
681         else
682             if gkcontador == 4410 then ;Condicionante para solucionar bug en
                el contador de fallos. De ésta manera no suma en la primera
                generación de muestras
683                 gkfallos = gkfallos
684             else
685                 gkfallos = gkfallos +1
686             endif
687         endif
688

```

```

689         endif
690
691         ;comprobación de la amplitud
692         if gkamprand == kamplitud then
693             gkValidA = 1
694         else
695             gkValidA = 0
696             if gkres2 == kamplitud then
697                 gkfallos = gkfallos
698             else
699                 if gkcontador == 4410 then ;Condicionante para solucionar bug en
                    el contador de fallos. De ésta manera no suma en la primera
                    generación de muestras
700                     gkfallos = gkfallos
701                 else
702                     gkfallos = gkfallos +1
703                 endif
704             endif
705         endif
706
707         ;comprobación de frec. muestreo
708         if gksamprand == ksampusuario then
709             gkValidS = 1
710         else
711             gkValidS = 0
712             if gkres4 == ksampusuario then
713                 gkfallos = gkfallos
714             else
715                 if gkcontador == 4410 then ;Condicionante para solucionar bug en
                    el contador de fallos. De ésta manera no suma en la primera
                    generación de muestras
716                     gkfallos = gkfallos
717                 else
718                     gkfallos = gkfallos +1
719                 endif
720             endif
721         endif
722
723         elseif gkflag == 5 then
724             if gkcheckruido == 1 then
725                 gkValidRuido = 1
726             else
727                 gkValidRuido = 0
728             endif
729
730             if gkcheckcuant == 1 then
731                 gkValidCuant = 1
732             else
733                 gkValidCuant = 0
734             endif
735
736             if gkcheckmuestreo == 1 then ;Segundo comprobamos si alguno de los
                    checks que no van en este flag se pulsan. Si se pulsan debemos
                    hacen la comprobación para sumar un fallo al contador
737                 if gkcheckmuestreo == gkres6 then
738                     gkfallos = gkfallos
739                 else
740                     if gkcontador == 4410 then ;Condicionante para solucionar bug en el
                        contador de fallos. De ésta manera no suma en la primera
                        generación de muestras
741                         gkfallos = gkfallos
742                     else
743                         gkfallos = gkfallos +1
744                     endif

```

```

745     endif
746 endif
747
748     ;comprobación del ruido
749 if gknoiserand == kruidousuario then
750     gkValidN = 1
751 else
752     gkValidN = 0
753     if gkres1 == kruidousuario then
754         gkfallos = gkfallos
755     else
756         if gkcontador == 4410 then ;Condicionante para solucionar bug en
            el contador de fallos. De ésta manera no suma en la primera
            generación de muestras
757             gkfallos = gkfallos
758         else
759             gkfallos = gkfallos +1
760         endif
761     endif
762 endif
763
764     ;comprobación de la amplitud
765 if gkamprand == kamplitud then
766     gkValidA = 1
767 else
768     gkValidA = 0
769     if gkres2 == kamplitud then
770         gkfallos = gkfallos
771     else
772         if gkcontador == 4410 then ;Condicionante para solucionar bug en
            el contador de fallos. De ésta manera no suma en la primera
            generación de muestras
773             gkfallos = gkfallos
774         else
775             gkfallos = gkfallos +1
776         endif
777     endif
778 endif
779
780     ;comprobación de bits
781 if gkbitsrand == kbitsusuario then
782     gkValidB = 1
783 else
784     gkValidB = 0
785     if gkres3 == kbitsusuario then
786         gkfallos = gkfallos
787     else
788         if gkcontador == 4410 then ;Condicionante para solucionar bug en
            el contador de fallos. De ésta manera no suma en la primera
            generación de muestras
789             gkfallos = gkfallos
790         else
791             gkfallos = gkfallos +1
792         endif
793     endif
794 endif
795
796 elseif gkflag == 6 then
797     if gkcheckruido == 1 then
798         gkValidRuido = 1
799     else
800         gkValidRuido = 0
801     endif
802

```

```

803     if gkcheckcuant == 1 then
804         gkValidCuant = 1
805     else
806         gkValidCuant = 0
807     endif
808
809     if gkcheckmuestreo == 1 then
810         gkValidMues = 1
811     else
812         gkValidMues = 0
813     endif
814
815         ;comprobación del ruido
816     if gknoiserand == kruidousuario then
817         gkValidN = 1
818     else
819         gkValidN = 0
820         if gkres1 == kruidousuario then
821             gkfallos = gkfallos
822         else
823             if gkcontador == 4410 then ;Condicionante para solucionar bug en
824                 el contador de fallos. De ésta manera no suma en la primera
825                 generación de muestras
826                 gkfallos = gkfallos
827             else
828                 gkfallos = gkfallos +1
829             endif
830         endif
831     endif
832
833     ;comprobación de la amplitud
834     if gkamprand == kamplitud then
835         gkValidA = 1
836     else
837         gkValidA = 0
838         if gkres2 == kamplitud then
839             gkfallos = gkfallos
840         else
841             if gkcontador == 4410 then ;Condicionante para solucionar bug en
842                 el contador de fallos. De ésta manera no suma en la primera
843                 generación de muestras
844                 gkfallos = gkfallos
845             else
846                 gkfallos = gkfallos +1
847             endif
848         endif
849     endif
850
851     ;comprobación de frec. muestreo
852     if gksamprand == ksampusuario then
853         gkValidS = 1
854     else
855         gkValidS = 0
856         if gkres4 == ksampusuario then
857             gkfallos = gkfallos
858         else
859             if gkcontador == 4410 then ;Condicionante para solucionar bug en
860                 el contador de fallos. De ésta manera no suma en la primera
861                 generación de muestras
862                 gkfallos = gkfallos
863             else
864                 gkfallos = gkfallos +1
865             endif
866         endif
867     endif

```

```

861     endif
862
863     ;comprobación de bits
864     if gkbitsrand == kbitsusuario then
865         gkValidB = 1
866     else
867         gkValidB = 0
868         if gkres3 == kbitsusuario then
869             gkfallos = gkfallos
870         else
871             if gkcontador == 4410 then ;Condicionante para solucionar bug en
                el contador de fallos. De ésta manera no suma en la primera
                generación de muestras
872                 gkfallos = gkfallos
873             else
874                 gkfallos = gkfallos +1
875             endif
876         endif
877     endif
878 endif
879
880 ;Si se pulsa solución, devuelve los valores a los menús
881 if gksoluc == 1 then
882     ;Corrección de la variable de f. de muestreo para mostrar correctamente
        la solución
883     if gksamprand == 2 then
884         kresultadofrecu = 8
885     elseif gksamprand == 3 then
886         kresultadofrecu = 7
887     elseif gksamprand == 4 then
888         kresultadofrecu = 6
889     elseif gksamprand == 5 then
890         kresultadofrecu = 5
891     elseif gksamprand == 6 then
892         kresultadofrecu = 4
893     elseif gksamprand == 7 then
894         kresultadofrecu = 3
895     elseif gksamprand == 8 then
896         kresultadofrecu = 2
897     elseif gksamprand == 9 then
898         kresultadofrecu = 1
899     elseif gksamprand == 10 then
900         kresultadofrecu = 0
901     endif
902
903     ;Soluciones dependiendo del flag en el que estemos
904     if gkflag == 0 then
905         outvalue "checkruido", 1
906         outvalue "checkmuestreo", 0
907         outvalue "checkcuant", 0
908
909         outvalue "ruidores", gknoiserand
910         outvalue "ruidoampres", -gkamprand
911     elseif gkflag == 1 then
912         outvalue "checkruido", 0
913         outvalue "checkmuestreo", 1
914         outvalue "checkcuant", 0
915
916         outvalue "sampres2", kresultadofrecu
917     elseif gkflag == 2 then
918         outvalue "checkruido", 0
919         outvalue "checkmuestreo", 0
920         outvalue "checkcuant", 1
921

```

```

922     outvalue "bitsres2", gkbitsrand-4
923     elseif gkflag == 3 then
924         outvalue "checkruido", 0
925         outvalue "checkmuestreo", 1
926         outvalue "checkcuant", 1
927
928     outvalue "sampsres2", kresultadofrecu
929     outvalue "bitsres2", gkbitsrand-4
930     elseif gkflag == 4 then
931         outvalue "checkruido", 1
932         outvalue "checkmuestreo", 1
933         outvalue "checkcuant", 0
934
935     outvalue "sampsres2", kresultadofrecu
936     outvalue "ruidores", gknoiserand
937     outvalue "ruidoampres", -gkamprand
938     elseif gkflag == 5 then
939         outvalue "checkruido", 1
940         outvalue "checkmuestreo", 0
941         outvalue "checkcuant", 1
942
943     outvalue "ruidores", gknoiserand
944     outvalue "ruidoampres", -gkamprand
945     outvalue "bitsres2", gkbitsrand-4
946     elseif gkflag == 6 then
947         outvalue "checkruido", 1
948         outvalue "checkmuestreo", 1
949         outvalue "checkcuant", 1
950
951     outvalue "ruidores", gknoiserand
952     outvalue "ruidoampres", -gkamprand
953     outvalue "sampsres2", kresultadofrecu
954     outvalue "bitsres2", gkbitsrand-4
955     endif
956 endif
957
958 ;Memorias para la comprobación
959
960 gkres1 = kruidousuario
961 gkres2 = kamplitud
962 gkres3 = kbitsusuario
963 gkres4 = ksampusuario
964
965 gkres5 = gkcheckcuant
966 gkres6 = gkcheckmuestreo
967 gkres7 = gkcheckruido
968
969 ;Creamos una cadena de texto con el número de fallos y lo mostramos en la
970     interfaz gráfica
971 Sfallos sprintfk "%02d", gkfallos
972 outvalue "fallos", Sfallos
973
974     endin
975
976 </CsInstruments>
977 ;Diseño de partitura
978 <CsScore>
979     ;Llamada a los instrumentos
980     i 1 0 10000 ;Instrumento de prueba
981     i 2 0 10000 ;Instrumento de prácticas
982     i 10 0 10000 ;Interfaz gráfica
983 </CsScore>
984 </CsoundSynthesizer>

```

APÉNDICE D

Código fuente - Ganancia y dinámica

```
1 ;-----
2 ;GANANCIA Y DINÁMICA v.1.0
3 ;-----
4 ; Código: Implementación de una aplicación para el entrenamiento auditivo
5 ; enfocada a la ganancia y la dinámica del sonido.
6 ;-----
7 ; Autor: Fernando Ayelo Sánchez
8 ;-----
9 ; Fecha: 6 de Junio de 2017
10 ;-----
11
12 <CsoundSynthesizer>
13
14 ;Sección Opciones de Csound
15 <CsOptions>
16 --displays ;Activamos éste flag para visualizar los gráficos del interfaz.
17 </CsOptions>
18 ;Diseño de la orquesta
19 <CsInstruments>
20 ;Definimos las variables globales
21 sr = 44100 ;Frecuencia de muestreo
22 ksmps = 16 ;Número de muestras calculadas por periodo de control
23 nchnls = 2 ;Número de canales
24
25 ;Declaración de tablas para crear sliders exponenciales
26 giExp1 ftgen 0, 0, 129, -25, 0, 0.001, 128, 1.0
27 giExp2 ftgen 0, 0, 129, -25, 0, 0.001, 128, 2.0
28 giExp100 ftgen 0, 0, 129, -25, 0, 1, 128, 100.0
29
30 ;Declaración e inicialización de las variables globales
31 gkdifi init 0
32 gkresgain init 0
33 gkamplitud init 0
34 gkamprand init 0
35 gkcontador2 init 0
36 gkgenerar init 0
37
38 ;inicializamos las variables de validación a 0
39 gkValidG init 1
40 gkValidD init 1
41 gkfallos init 0
42
43 ;inicialización de las variables de control para el sistema de fallos
44 gkres1 init 0
45 gkres2 init 0
46
47 ;Al empezar colocamos el panorama en el centro
48 outvalue "stereo", 0
49 outvalue "stereo2", 0
```

```

50
51 instr 10 ;Instrumento de comunicación con la interfaz gráfica (GUI)
52 ktrig metro 10 ;Al necesitar por muestras independientes, utilizamos la
    función metro para controlar las interacciones por pasos
53 if (ktrig == 1) then
54
55     ;Instrumento de prueba
56     ;Zona Main
57     gkmute invalue "muteoriginal"
58     gkstereo invalue "stereo"
59     gkmuestra invalue "muestra"
60
61     outvalue "panres", gkstereo
62
63     ;Zona Ganancia
64     gkaten invalue "aten"
65
66     Sgain    sprintfk "%02d", gkamplitud
67     outvalue "amplitud", Sgain
68
69     ;Zona Dinámica
70     gkmutecomp invalue "mutecomp"
71     gkthresh invalue "thresh"
72     gklowknee invalue "knee"
73     gkhiknee invalue "hiknee"
74
75     kratio invalue "ratio"
76     gkratio tablei kratio, giExp100, 1
77     outvalue "ratiores", gkratio
78
79     katt      invalue "att"
80     gkatt      tablei katt, giExp1, 1
81     outvalue "attres", gkatt
82
83     krel      invalue "rel"
84     gkrel      tablei krel, giExp1, 1
85     outvalue "resres", gkrel
86
87     kgain      invalue "makeup"
88     gkgain      tablei kgain, giExp2, 1
89     outvalue "makeres", gkgain
90
91 ;Instrumento de entrenamiento
92 ;Zona Main
93 gkmute2 invalue "muteoriginal2"
94 gkstereo2 invalue "stereo2"
95 gkmuestra2 invalue "muestra2"
96 gkgenerar invalue "generar"
97
98 outvalue "panres2", gkstereo2
99
100 ;Zona Gain
101 gkdifi invalue "dificultad"
102 gkselecc invalue "muestralisten"
103 gkresgain invalue "muestrares"
104
105 if gkselecc == 0 then
106     S1 = "Muestra 1"
107     outvalue "muestramostrar", S1
108 elseif gkselecc == 1 then
109     S2 = "Muestra 2"
110     outvalue "muestramostrar", S2
111 endif
112

```

```

113 outvalue "solgan", gkValidG
114
115 ;Zona Dinámica
116 gkefectosel invalue "efectosel"
117 gkselecc2 invalue "muestralis"
118 gkresefec invalue "muestrares2"
119 gkmutedin invalue "mutedin"
120 gkmutegain invalue "mutegain"
121
122 if gkselecc2 == 0 then
123     S3 = "Muestra 1"
124     outvalue "muestramos", S3
125 elseif gkselecc2 == 1 then
126     S4 = "Muestra 2"
127     outvalue "muestramos", S4
128 endif
129
130 outvalue "soldin", gkValidD
131
132 ;Resultados
133 gksoluc invalue "soluciones"
134
135 endif
136 endin
137
138 instr 1 ;Instrumento de prueba
139 ;Carga de la muestra de audio según decida el usuario
140 if gkmuestra == 0 then
141     aMuestra disk2 "muestra1.wav",1,0,1
142 elseif gkmuestra == 1 then
143     aMuestra disk2 "muestra2.wav",1,0,1
144 elseif gkmuestra == 2 then
145     aMuestra disk2 "muestra3.wav",1,0,1
146 elseif gkmuestra == 3 then
147     aMuestra disk2 "muestra4.wav",1,0,1
148 elseif gkmuestra == 4 then
149     aMuestra disk2 "muestra5.wav",1,0,1
150 elseif gkmuestra == 5 then
151     aMuestra inch 1
152 endif
153
154 kstereo = gkstereo + 0.5 ;Adaptación de la variable estéreo
155
156 ;Zona de ganancia
157
158 ;Asignación de la atenuación seleccionada en el widget a un formato ú
    til para aplicar
159 if gkaten == 0 then
160     gkamplitud = 0
161 elseif gkaten == 1 then
162     gkamplitud = -1
163 elseif gkaten == 2 then
164     gkamplitud = -2
165 elseif gkaten == 3 then
166     gkamplitud = -3
167 elseif gkaten == 4 then
168     gkamplitud = -6
169 elseif gkaten == 5 then
170     gkamplitud = -9
171 elseif gkaten == 6 then
172     gkamplitud = -12
173 endif
174
175 klogi = gkamplitud/10 ;Pasamos la ganancia formato dB

```

```

176     kamp = 10^(klogi)
177
178     aMuestra = kamp*aMuestra ;Aplicamos la ganancia a la muestra
179     aSal = gkmute*aMuestra ;Controlamos si está activada la muestra o no
180
181     ;Zona de dinámica
182     if gkmutecomp == 1 then ;Si la opción de dinámica está activada,
        aplicamos el efecto
183     aSal compress aSal, aSal,gkthresh, gklowknee, gkhiknee, gkratio,
        gkatt, gkrel, 0.05 ;Uso de la función compress para aplicar
        Compresor o Puerta de ruido
184     aSal = aSal*gkgain ;Aplicamos la ganancia de corrección ya que el
        procesado nos disminuye demasiado la amplitud
185     endif
186
187     ;Sacamos la señal generada previamente en formato estéreo
188     outs      aSal*(1-kstereo),aSal*kstereo
189
190     endin
191
192     instr 2 ;Instrumento de entrenamiento
193     ;Carga de la muestra de audio según decida el usuario
194     if gkmuestra2 == 0 then
195     aMuestra2 diskin2 "muestra1.wav",1,0,1
196     elseif gkmuestra2 == 1 then
197     aMuestra2 diskin2 "muestra2.wav",1,0,1
198     elseif gkmuestra2 == 2 then
199     aMuestra2 diskin2 "muestra3.wav",1,0,1
200     elseif gkmuestra2 == 3 then
201     aMuestra2 diskin2 "muestra4.wav",1,0,1
202     elseif gkmuestra2 == 4 then
203     aMuestra2 diskin2 "muestra5.wav",1,0,1
204     elseif gkmuestra2 == 5 then
205     aMuestra2 inch 1
206     endif
207
208     kstereo2 = gkstereo2 + 0.5 ;Adaptación de la variable estéreo
209
210     ;Asignación de las respuestas a su variable correspondiente
211     ;Zona Gain
212     ;Dificultad
213     if gkdifi == 0 then
214     katen = -1
215     elseif gkdifi == 1 then
216     katen = -2
217     elseif gkdifi == 2 then
218     katen = -3
219     elseif gkdifi == 3 then
220     katen = -6
221     elseif gkdifi == 4 then
222     katen = -9
223     elseif gkdifi == 5 then
224     katen = -12
225     endif
226
227     ;Respuesta
228     if gkresgain == 0 then
229     kresgain = 0
230     elseif gkresgain == 1 then
231     kresgain = 1
232     elseif gkresgain == 2 then
233     kresgain = 6
234     endif
235

```

```

236 ;Zona Dinámica
237 ;Efecto
238 if gkefectosel == 0 then
239     kefecto = 0
240 elseif gkefectosel == 1 then
241     kefecto = 1
242 endif
243
244 ;Respuesta
245 if gkresefec == 0 then
246     kresdin = 0
247 elseif gkresefec == 1 then
248     kresdin = 1
249 elseif gkresefec == 2 then
250     kresdin = 6
251 endif
252
253 ;Generación de valores aleatorios y reseteo de menús de usuario
254
255 ;Si se pulsa generar, se generan los valores aleatorios y se ponen a 0
    los menús
256 if gkgenerar == 1 then
257     ;Creamos el aleatorio de que muestra será la atenuada
258     krnd randh 1, 10, 10
259     krnd = abs(krnd)
260     gkmuesrand = round(krnd)
261
262     ;Creamos el aleatorio para la muestra que sufrirá el proceso dinámico
263     krnd2 randh 1, 10, 10
264     krnd2 = abs(krnd2)
265     gkmuesrand2 = round(krnd2)
266
267     ;Ponemos a 0 los menús para el usuario
268     outvalue "muestrares", 2
269     outvalue "muestrares2", 2
270     gkValidG = 0
271     gkValidD = 0
272
273     ;Reseteamos el contador de fallos
274     gkfallos = 0
275
276     ;Ponemos el contador a 1 para saber el número de rondas que lleva
277     gkcontador2 = (gkcontador2)+1
278
279     endif
280
281     ;Generamos la señal de audio
282     ;Generación de la muestra para la sección de Ganancia
283     if gkmutegain == 1 then ;Debemos controlar si está activada la sección
        n de ganancia
284         klog = katen/10 ;Adaptamos la ganancia de formato logarítmico a
            lineal
285         kAmp = 10^(klog)
286
287         if gkselecc == 0 then ;Dependiendo de que muestra esté seleccionada
            para sonar, la muestra atenuada será una u otra en función del nú
            mero aleatorio creado
288             if gkmuesrand == 0 then
289                 aMuestra2 = aMuestra2*kAmp
290             elseif gkmuesrand == 1 then
291                 aMuestra2 = aMuestra2
292             endif

```

```

293     elseif gkselecc == 1 then ;Dependiendo de que muestra esté
        seleccionada para sonar, la muestra atenuada será una u otra en
        función del número aleatorio creado
294     if gkmuesrand == 0 then
295         aMuestra2 = aMuestra2
296     elseif gkmuesrand == 1 then
297         aMuestra2 = aMuestra2*kAmple
298     endif
299     endif
300
301     endif
302     ;Generación de la muestra para la sección de Dinámica
303     if gkmutedin == 1 then ;Debemos controlar si está activada la sección
        de dinámica
304         if kefecto == 0 then ;Debemos controlar si está activado el efecto
            Compresor
305             if gkselecc2 == 0 then ;Dependiendo de que muestra esté seleccionada
                para sonar, la muestra procesada será una u otra en función del
                número aleatorio creado
306             if gkmuesrand2 == 0 then
307                 aMuestra2 compress aMuestra2, aMuestra2, 0, 48.182, 60, 5.091,
                    0.005, 0.101, 0.05 ;Aplicamos directamente los valores del
                    preset Compresor
308                 aMuestra2 = aMuestra2*1.885 ;Aplicamos la corrección de ganancia
309             elseif gkmuesrand == 1 then
310                 aMuestra2 = aMuestra2*0.8 ;Aplicamos la corrección de ganancia
311             endif
312         elseif gkselecc2 == 1 then ;Dependiendo de que muestra esté
            seleccionada para sonar, la muestra procesada será una u otra en
            función del número aleatorio creado
313             if gkmuesrand2 == 0 then
314                 aMuestra2 = aMuestra2*0.8 ;Aplicamos la corrección de ganancia
315             elseif gkmuesrand == 1 then
316                 aMuestra2 compress aMuestra2, aMuestra2, 0, 48.182, 60, 5.091,
                    0.005, 0.101, 0.05 ;Aplicamos directamente los valores del
                    preset Compresor
317                 aMuestra2 = aMuestra2*1.885 ;Aplicamos la corrección de ganancia
318             endif
319         endif
320         elseif kefecto == 1 then ;Debemos controlar si está activado el
            efecto Puerta de ruido
321             if gkselecc2 == 0 then ;Dependiendo de que muestra esté seleccionada
                para sonar, la muestra procesada será una u otra en función del
                número aleatorio creado
322             if gkmuesrand2 == 0 then
323                 aMuestra2 compress aMuestra2, aMuestra2, 80.727, 48.182, 60,
                    1.017, 0.005, 0.098, 0.05 ;Aplicamos directamente los valores
                    del preset Puerta de ruido
324                 aMuestra2 = aMuestra2*1.885 ;Aplicamos la corrección de ganancia
325             elseif gkmuesrand == 1 then
326                 aMuestra2 = aMuestra2*0.8 ;Aplicamos la corrección de ganancia
327             endif
328         elseif gkselecc2 == 1 then ;Dependiendo de que muestra esté
            seleccionada para sonar, la muestra procesada será una u otra en
            función del número aleatorio creado
329             if gkmuesrand2 == 0 then
330                 aMuestra2 = aMuestra2*0.8 ;Aplicamos la corrección de ganancia
331             elseif gkmuesrand == 1 then
332                 aMuestra2 compress aMuestra2, aMuestra2, 80.727, 48.182, 60,
                    1.017, 0.005, 0.098, 0.05 ;Aplicamos directamente los valores
                    del preset Puerta de ruido
333                 aMuestra2 = aMuestra2*1.885 ;Aplicamos la corrección de ganancia
334             endif
335         endif

```

```

336     endif
337 endif
338
339     aSal2 = (gkmute2*aMuestra2) ;Controlamos si está activada la sección
        de entrenamiento
340     outs      aSal2*(1-kstereo2),aSal2*kstereo2 ;Sacamos la salida en
        formato estéreo
341
342 ;Comprobación de resultados
343 ;comprobación de ganancia
344 if gkmuesrand == 0 then
345     if kresgain == 0 then
346         gkValidG = 1
347     else
348         gkValidG = 0
349         if gkres1 == kresgain then
350             gkfallos = gkfallos
351         else
352             if gkcontador2 == 551 then ;Condicionantes para solucionar bug en
                el contador de fallos. De ésta manera no suma en la primera
                generación de muestras
353                 gkfallos = gkfallos
354             elseif gkcontador2 == 0 then
355                 gkfallos = gkfallos
356             else
357                 gkfallos = gkfallos +1
358             endif
359         endif
360     endif
361
362 elseif gkmuesrand == 1 then
363     if kresgain == 1 then
364         gkValidG = 1
365     else
366         gkValidG = 0
367         if gkres1 == kresgain then
368             gkfallos = gkfallos
369         else
370             if gkcontador2 == 551 then ;Condicionantes para solucionar bug en
                el contador de fallos. De ésta manera no suma en la primera
                generación de muestras
371                 gkfallos = gkfallos
372             else
373                 gkfallos = gkfallos +1
374             endif
375         endif
376     endif
377 endif
378
379 ;Comprobación Dinámica
380 if gkmuesrand2 == 0 then
381     if kresdin == 0 then
382         gkValidD = 1
383     else
384         gkValidD = 0
385         if gkres2 == kresdin then
386             gkfallos = gkfallos
387         else
388             if gkcontador2 == 551 then ;Condicionantes para solucionar bug en
                el contador de fallos. De ésta manera no suma en la primera
                generación de muestras
389                 gkfallos = gkfallos
390             elseif gkcontador2 == 0 then
391                 gkfallos = gkfallos

```

```

392         else
393             gkfallos = gkfallos +1
394         endif
395     endif
396 endif
397
398 elseif gkmuesrand2 == 1 then
399     if kresdin == 1 then
400         gkValidD = 1
401     else
402         gkValidD = 0
403         if gkres2 == kresdin then
404             gkfallos = gkfallos
405         else
406             if gkcontador2 == 551 then ;Condicionantes para solucionar bug en
                                     el contador de fallos. De ésta manera no suma en la primera
                                     generación de muestras
407                 gkfallos = gkfallos
408             else
409                 gkfallos = gkfallos +1
410             endif
411         endif
412     endif
413 endif
414
415 ;Si pulsamos en Solución, colocamos los menús con los valores de la
416 solución
417 ;Si se pulsa solución, devuelve los valores a los menús
418 if gksoluc == 1 then
419
420     if gkmutegain == 1 then
421         if gkmuesrand == 0 then
422             outvalue "muestrares", 0
423         elseif gkmuesrand == 1 then
424             outvalue "muestrares", 1
425         endif
426     endif
427 ;Debemos filtrar si estamos en la sección ganancia o dinámica y a su
428 vez devolver la solución dependiendo del aleatorio
429 if gkmutedin == 1 then
430
431     if gkmuesrand2 == 0 then
432         outvalue "muestrares2", 0
433     elseif gkmuesrand2 == 1 then
434         outvalue "muestrares2", 1
435     endif
436 endif
437
438 endif
439
440 ;Contador de fallos al final del ciclo
441 gkres1 = kresgain
442 gkres2 = kresdin
443
444 ;Creamos una cadena de texto con el número de fallos y lo mostramos en
445 la interfaz gráfica
446 Sdisp sprintfk "%02d", gkfallos
447 outvalue "fallos", Sdisp
448
449 printk 1, gkcontador2, 50
450 endin
451 </CsInstruments>
452 ;Diseño de partitura

```

```
451 <CsScore>
452   ;Llamada a los instrumentos
453   i  1 0 10000 ;Instrumento de prueba
454   i  2 0 10000 ;Instrumento de práctica
455   i 10 0 10000 ;Interfaz gráfica
456
457 </CsScore>
458 </CsoundSynthesizer>
```