Escuela Politécnica Superior

Plataforma MIDI virtual

Grado en Ingeniería en Sonido e Imagen en Telecomunicación

Trabajo Fin de Grado

Autor:
Alba Silvente Fuentes

Tutor/es:
José Manuel Iñesta Quereda

Septiembre 2017

Universitat d'Alacant
Universidad de Alicante

Universitat d'Alacant
Universidad de Alicante

ESCUELA POLITÉCNICA SUPERIOR

Escuela Politécnica Superior
Universidad de Alicante

Plataforma MIDI virtual
Conexión entre HTML5 y Csound

TRABAJO FIN DE GRADO

Grado en Ingeniería en Sonido e Imagen en Telecomunicación

Autor: Alba Silvente Fuentes

Tutor: Jose Manuel Iñesta Quereda

Curso 2016-2017

Gracias a mis padres por soportar mis cambios de humor a lo largo de estos meses.

Gracias a mis amigas por motivarme y recordarme que todo esfuerzo tiene su recompensa.

Gracias a los profesores que me han impartido clase, por enseñarme todo lo que hoy sé y por conseguir que los estudios hayan sido, y sean, la mejor decisión de mi vida.

Gracias a toda mi familia por confiar en cada cosa que hago y darme siempre un apoyo enorme.

Gracias a las personas que contribuyen a hacer que la gente despierte y se plantee quién es día a día.

Y gracias a las personas tolerantes y sin prejuicios que ven más allá de la piel.

Los vendedores de software quieren dividir a los usuarios y conquistarlos, haciendo que cada usuario decida no compartir su software con los demás. Me rehúso a romper la solidaridad con otros usuarios de esta manera.

Richard Stallman.

Resumen

La creación de un *Launchpad* o plataforma virtual reúne tanto una funcionalidad sonora especializada como una interfaz dirigida a la experiencia de usuario. Para poder llevar a cabo esta aplicación con código libre, es necesario tener conocimientos de los lenguajes estándar de maquetación disponibles y de la conexión con los lenguajes de programación especializados en audio. Este trabajo se centra en la conexión entre los lenguajes HTML5 y Csound. Inicialmente se investigará sobre las técnicas de maquetación y experiencia de usuario actuales. Después, se realizará un prototipo que permitirá experimentar los conocimientos adquiridos y demostrar las posibilidades que ofrece esta conexión.

Siendo HTML el estándar en el mundo web, Csound decidió introducir en su compilador (CsoundQt) la capacidad de generar una interfaz gráfica con este lenguaje. Debido a la falta de experimentación e información sobre HTML en Csound después de la versión 6.09, este proyecto pretende retomar esta línea de investigación para ampliar conocimientos y mejorar la capacidad de creación de aplicaciones mediante este lenguaje.

El desarrollo de este prototipo ha seguido un proceso que va desde la creación de la parte visual de la plataforma hasta la programación de su funcionalidad. Se ha llevado a cabo la estructura principal mediante HTML, aplicando el estilo y los efectos analizando los lenguajes CSS y JavaScript. Una vez conseguida la interfaz, el trabajo se centra en la conexión entre los lenguajes HTML5 y Csound a través de JS utilizando el objeto Csound, que le permite realizar esta conexión. Por otro lado, la declaración de instrumentos y la programación de controles y filtros se realiza de forma paralela a la conexión, según se ha ido desarrollando la funcionalidad.

La funcionalidad sonora no ha sido tratada en profundidad, sino que se han añadido ciertos ejemplos para comprobar el funcionamiento y la capacidad del *Launchpad*. De este modo, cualquier persona con conocimientos sobre Csound podrá añadir efectos y ampliar su utilidad.

Palabras clave: Síntesis de sonido, Música por ordenador, Launchpad, MIDI, HTML5, Tiempo real, Sonido digital

Abstract

The development of a *Launchpad* or virtual platform gathers both a specialized sound functionality as an interface thought to the UX (user experience). To be able to perform this application with free code, it is necessary to have knowledge of the standard layout languages available and the connection with the specialized audio programming languages. This work focuses on the connection between HTML5 and Csound languages. Initially, this work will be investigating current layout techniques and user experience. Afterwards, a prototype will be realized that will allow to practice the acquired knowledge and to demonstrate the possibilities offered by this connection.

Being HTML the standard in the WWW, Csound decided to introduce in its compiler (CsoundQt) the ability to generate a graphical interface with this language. Due to lack of experimentation and information about HTML in Csound after version 6.09, this project try to resume this line of research to expand knowledge and improve the capacity to create applications using this language.

The development of this prototype has followed a process that goes from the creation of the visual part of the platform to the programming of its functionality. The main structure has been implemented using HTML, applying style and effects by analyzing CSS and JavaScript languages. Once the interface is obtained, the work focuses on the connection between the HTML5 and Csound using JS with the Csound object, which allows us to make this connection. On the other hand, the creation of instruments and the programming of controls and filters is done parallel to the connection, as the functionality has been developed.

The sound functionality has not been dealt with in depth, but certain examples have been added to check the functionality and capacity of the *Launchpad*. In this way, anyone with knowledge about Csound can add effects and extend its usefulness.

Key words: Sound synthesis, Music by computer, Launchpad, MIDI, HTML5, Real time, Digital sound

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

Un sonido es la vibración del aire captada por los oídos. Esta vibración provoca la creación de una sinusoide que, dependiendo de la frecuencia a la que vibre, se obtendrá un tono u otro. De esta forma se producen las notas musicales que, a su vez, dependen de otros factores más complejos como el timbre para definirse.

Cuando se busca cambiar el sonido se suele utilizar un controlador MIDI (*Musical Instrument Digital Interface*). Es un dispositivo que sirve para enviar señales MIDI a un software de producción musical, que se instala en el ordenador. El controlador consiste en el equipo físico que permite reproducir los sonidos o instrumentos virtuales incluidos en el DAW (*Digital Audio Workstation*). Se encarga de enviar instrucciones que activan instrumentos ya creados con anterioridad.

Figura 1.1: Controlador MIDI modelo teclado (izquierda) y programa de producción musical (derecha)

El prototipo desarrollado en este proyecto se inspira, en algunos aspectos, en el uso del teclado MIDI. Se pretende realizar un software que permita un tratamiento del sonido por medio de información proporcionada por un usuario desde una interfaz. Los teclados MIDI, a diferencia de un teclado normal, no emiten sonido por sí mismos. Su funcionamiento consiste en recoger los datos que reciben de las teclas, transformarlos en información MIDI y transmitir a un generador qué nota debe sonar, con qué fuerza, hasta cuando, ...

La similitud que se busca entre el software a desarrollar y este tipo de plataforma es el control MIDI. Centrándonos en controles tales como:

Controles	Definición
tono	control básico de las notas MIDI, referido a la 'nota' que debe sonar. Siendo, el equivalente en este software, qué sonido previamente grabado quiere iniciarse
velocidad	parámetro de dinámica que hará que la nota suene con mayor o menor intensidad
frecuencia	siendo la cantidad de ciclos u ondas completas que se repiten en un determinado tiempo
volumen	aumento de la amplitud de la onda, desde la cresta (parte superior) al valle (inferior)
división por zonas	dividiendo el teclado en varias zonas de teclas, enviando datos a diferentes canales MIDI, independiente y asignada a un sonido específico. Siendo una característica principal de nuestro prototipo

Tabla 1.1: Tabla de controles MIDI. Muestra la relación entre el control en un teclado MIDI y su uso en el prototipo.

1.1 Objetivos del proyecto

A día de hoy existen miles de aplicaciones que permiten, al usuario, la manipulación y creación de melodías mediante sonidos almacenados previamente. Uno de los factores que se encuentra, en muy pocos casos, es el manejo de estos desde la interfaz de usuario y el mismo código, inclusive.

Por la necesidad expuesta, tanto de usuarios expertos en la materia como inexpertos, uno de los propósitos de este trabajo es compaginar la capacidad que tienen los Launchpad con un código abierto a modificaciones y explorar nuevos horizontes en este campo.

De esta manera, se consigue una plataforma totalmente independiente. Pero, para centrarnos en una serie de efectos sonoros y realizar un estudio más detallado, se pretende obtener resultados con sonidos percusivos.

El mayor reto planteado es la realización de una interfaz de usuario útil, sencilla y adaptable al compilador sonoro que se ha escogido. Se pretende unir las características del lenguaje HTML5 (*HyperText Markup Language*) con la funcionalidad de Csound6 desde CsoundQt, como se desarrollará en el capítulo 5.

El lenguaje de programación Csound6 se encarga de globalizar y conseguir llegar a un público más amplio la programación sonora, por ser libre y gratuito. Siendo este uno de los pilares más importantes con respecto al objetivo principal: construir una plataforma programable.

Aunque Csound esté especializado en programación funcional, se han desarrollado compiladores como CsoundQt y Cabbage que contienen *Widgets* para desarrollar la interfaz visual. A pesar de ello, existe una gran carencia en cuanto a la programación gráfica en este lenguaje.

Con el objetivo de mejorar la representación visual en Csound, se pretende explorar las opciones que ofrece el lenguaje HTML5 para el desarrollo de aplicaciones sonoras. Para conocer y ampliar las distintas posibilidades que se desarrollarán en este trabajo, se utilizará la extensión que ofrece CsoundQt para la unión entre HTML5 y Csound (desarrollado en el capítulo 5).

El objetivo principal es que todo el prototipo se desarrolle mediante software libre. Realizar un código que, por elección manifiesta de su autor, pueda ser copiado, estudiado, modificado, utilizado libremente con cualquier fin y redistribuido con o sin cambios o mejoras.

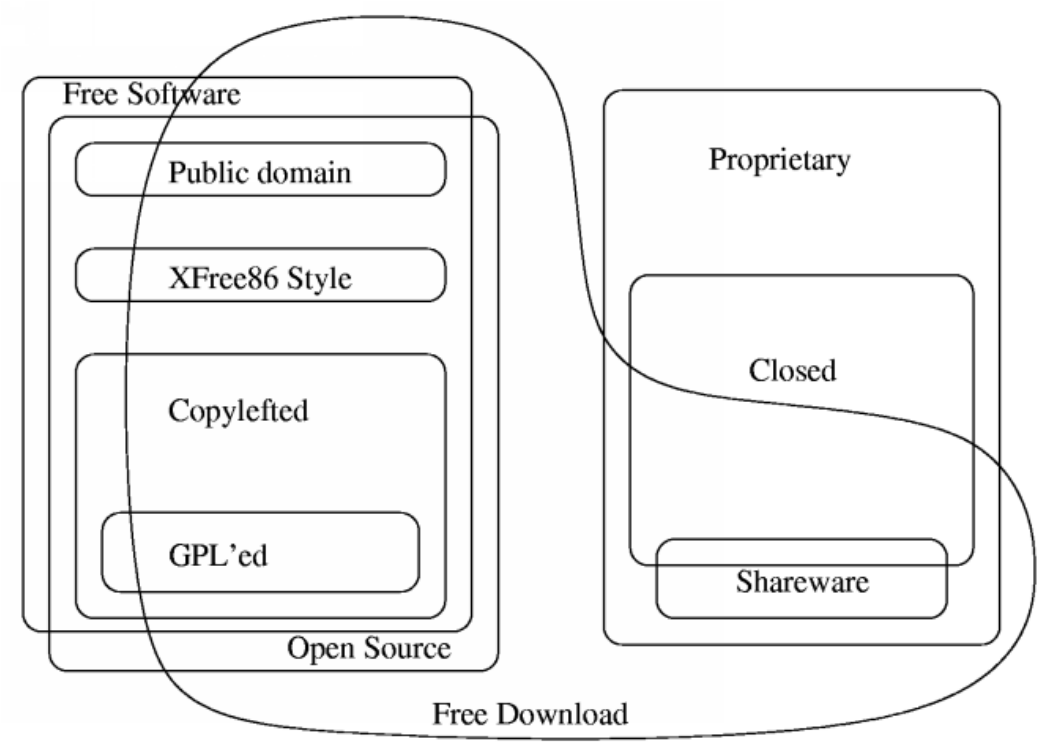

Figura 1.2: Características del Software libre

1.2 Referentes

Inicialmente se va a tratar con plataformas físicas con fines didácticos. Mostrando, después, ejemplos de software libre online, como referentes auténticos a la interfaz de usuario que se pretende conseguir.

Los controladores MIDI pueden adoptar formas de instrumentos tradicionales como teclados, flautas, guitarras, arpas, etc. O aspectos que no se corresponden con ningún instrumento conocido como cañones de luz o sensores de movimiento.

Como existen cientos de representaciones se tratarán aquellas que han tenido gran auge entre el público.

1.2.1. Superficies de control

Controladores formados mediante *pads* (botones) sensibles a la velocidad de pulsación y a la presión ejercida. Según la velocidad o presión que se aplique a cada *pad* se accederá a bancos de sonidos diferentes y se reproducirán los sonidos de forma distinta. Esta plataforma sirve tanto para DJS y músicos como para programadores o productores.

Figura 1.3: Superficie de control MIDI: Novation - Launchpad PRO

Las versiones que se encuentran de estos controladores son relativas al número de *pads* que contienen, teniendo más opciones la que tenga un número más elevado de éstos.

1.2.2. Los teclados MIDI

Como se ha mencionado en la sección 1, son dispositivos que realizan la funcionalidad de un controlador MIDI por medio de un teclado físico.

Estos se forman de teclas que miden, al igual que en las superficies de control, la velocidad y el *aftertouch*. Aunque, también contienen *pads*, *faders*, entre otros.

Una de las características esenciales que diferencia un teclado MIDI de otro es el tipo de teclas que usan o la forma de ejecutar las acciones. La pulsación sobre la tecla envía impulsos MIDI distintos gracias a las resistencias y muelles que la forman, pero según la manera de captar esas acciones varía drásticamente la sensación transmitida al tocar.

Existen 3 tipos de tecla más conocidos de los que se habla a continuación:

- **La tecla común**

  Es el tipo común de un controlador MIDI de gama media-baja debido a los materiales usados y su poca complejidad. Esto permite que cualquier persona pueda obtener un teclado MIDI.

  Estas teclas fabricadas en plástico contienen un sistema de muelles que permite la pulsación retrocediendo a su posición inicial.

  Realizan las funciones por las que fueron construidas, por lo que para músicos expertos o grandes interpretaciones no será útil.

- **La tecla semicontrapesada**

  Los teclados que incluyen este tipo de tecla, a diferencia con la tecla normal, cuentan con un peso redudido que se añade para contrarrestar la acción producida por el muelle.

  Aunque sus teclas también estén realizadas con materiales de plástico, gracias al peso la sensación da mejores resultados.

- **La tecla contrapesada**

  El piano real contiene teclas construidas en madera que tienen un peso y comportamiento exactos. La tecla contrapesada, aunque siga siendo de plástico, imita este tipo de tacto y acción.

  El efecto que se consigue sigue contando con la acción del mecanismo de martillo[1].

  Además, este tipo de tecla, por su complejidad, se utiliza en los controladores de gama alta y tiene un precio más elevado a los tipos de tecla anteriores.

Aunque existan varios tipos tecla la finalidad es captar las acciones producidas por el compositor y procesarlas mediante un programa especializado.

Sabiendo como una plataforma mecánica transmite la información al DAW, ahora se verá como realizan esta función las aplicaciones online.

[1] Se trata del mecanismo que hace sonar las cuerdas en un piano mediante un martillo forrado de fieltro.

1.2.3. Online Launchpad virtual

La página web que se puede ver pulsando aquí se trata de una aplicación gratuita que contiene una base de datos con canciones divididas en 4 partes.

Cada una de esas partes contiene distintos efectos. Estas, a su vez, se muestran como fragmentos representados mediante un teclado ASCII común.

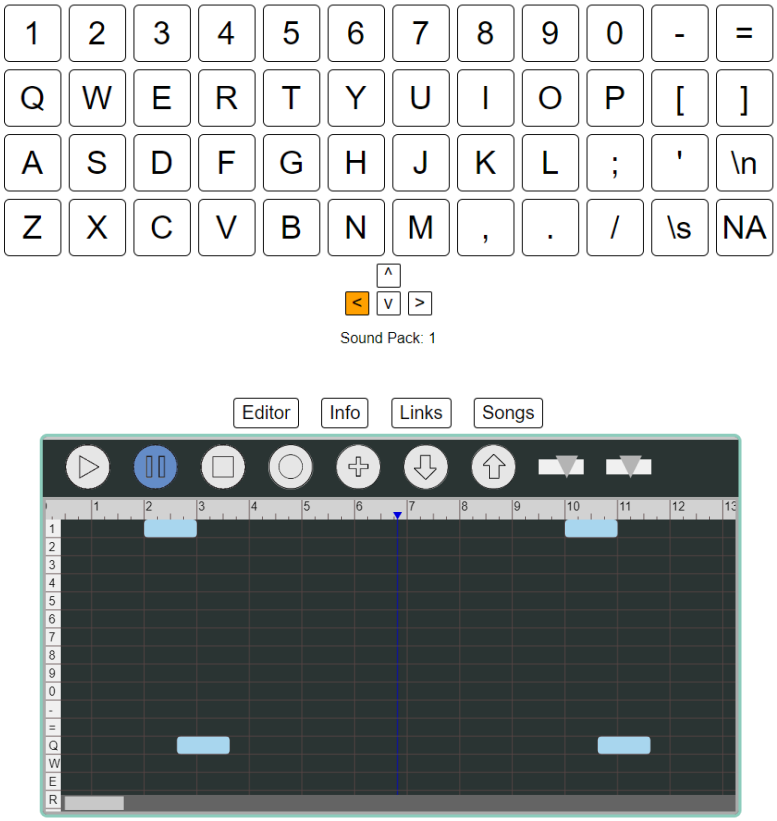

Figura 1.4: Interfaz visual del primer referente online: Online Launchpad virtual

El proceso que sigue este software virtual es:

Primero, se debe pulsar el botón que contiene la palabra *Songs*, para escoger la canción que se desee manejar. Una vez escogida, se pulsa el botón *Editor* con el fin de comenzar la modificación.

A continuación, pulsando la flecha de subida se selecciona el fragmento y aparecen los siguientes iconos:

- *Play*, *Pause*, *Stop* y *Record* que se encargan de realizar las funciones de escuchar, pausar (continuar desde el mismo segundo), parar (vuelve al inicio) y grabar el sonido, respectivamente.

- Símbolo **+**: se encarga de resetear, volver a empezar sin pista escogida.

- Flechas: bajada, guarda el archivo que se haya grabado, y subida, carga la parte de la canción escogida.

- Controles: existe un control de *ZOOM* para ver las notas más compactas o más detalladas, y *BPM* para controlar la velocidad de la canción.

Por último, para crear las melodías deseadas se debe utilizar el teclado que aparece en la misma pantalla. Pudiendo cambiar de fragmento con las flechas del teclado y de sonido con cada tecla utilizada.

Inspiración: Composición de bases mediante sonidos electrónicos y botones de control.

Mejoras: Sonidos modificables respecto a frecuencia, volumen y otros filtros.

1.2.4. Launchpad Intro

La aplicación online que contiene esta página http://intro.novationmusic.com/harry-coade, existe tanto para aplicaciones móviles como para cualquier navegador. Accediendo a la página web muestra los enlaces para su descarga gratuita en los sistemas operativos móviles, mientras que se visualiza la plataforma en la misma web si se accede desde un ordenador. Además, proporciona una guía de uso antes de comenzar a usarlo, consiguiendo el usuario un mejor resultado.

Este software, a diferencia del anterior, se divide en:

- *Pads*: se trata de los botones que se observan en la plataforma. Se separan en dos tipos: uno considerado bucle, que no deja de sonar desde que se pulsa hasta que se vuelve a seleccionar, y otro de una sola vez, finalizada después de un breve periodo de tiempo.

- Los *pads* en bucle tienen limitaciones con respecto a los de un solo sonido. Si existen varios en bucle agrupados en un mismo canal solo uno puede estar pulsado, mientras que el otro tipo de *pad* no tiene restricción.

- Los canales contienen a su vez dos botones genéricos para cualquier botón: *Stop* y *Mute*, que como indican se parará y borrará la sintonía conseguida y se silenciará, respectivamente.

Los sonidos que contiene este Launchpad están colocados por tipo en diferentes canales, dividiendo así sus funcionalidades.

Inspiración: Unión en tiempo real de sonidos, previamente creados, en bucle.

Mejoras: Creación de los bucles mediante sonidos cortos.

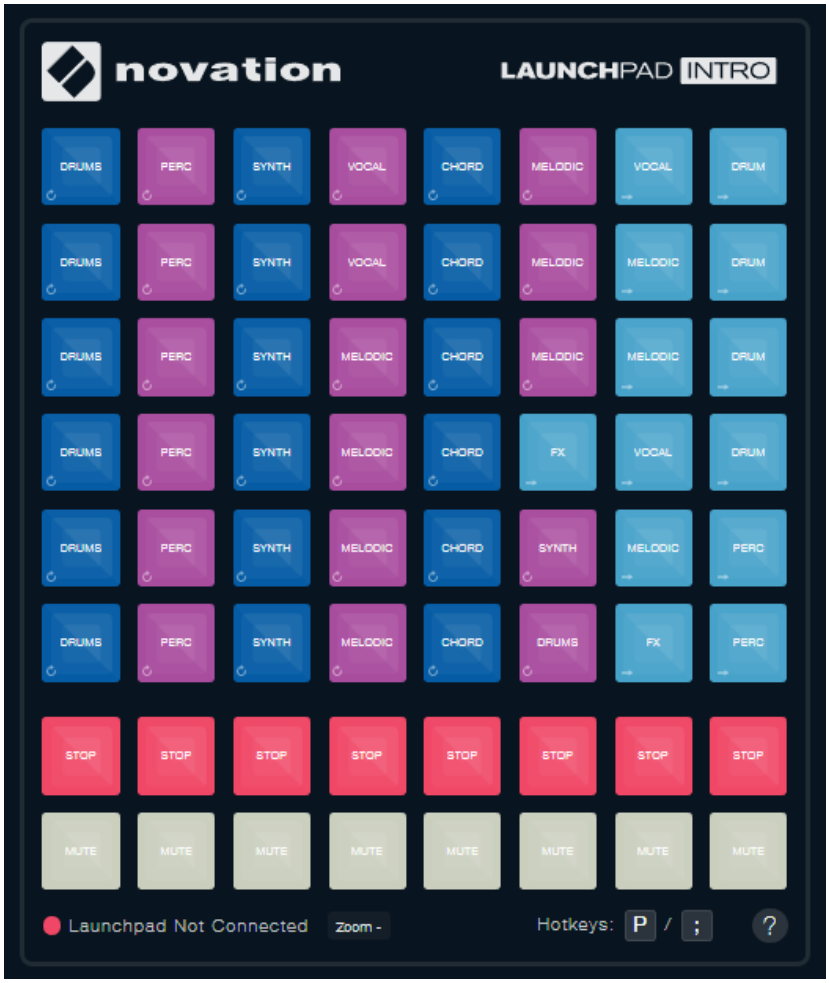

Figura 1.5: Interfaz visual del segundo referente online: Launchpad intro

1.2.5. Dubstep Cube

Esta plataforma se puede destacar como la más llamativa visualmente puesto que forma un cubo en 3D (*3-Dimensions*). Al igual que las dos anteriores se puede acceder desde esta página web dubstep-cube.com y utilizarla de forma gratuita con conexión a internet.

Su funcionamiento es bastante simple, no tiene nada que destacar, con respecto a las comentadas anteriormente. Simplemente, se pueden realizar combinaciones sonoras pulsando los pequeños cubos que se encuentran en las diferentes caras del cubo principal.

Es cierto que también incluye dos botones: uno, para parar la ejecución, y otro, para sincronizar las ondas creadas. Lo cual resultaba necesario en los casos anteriores y no fue mencionado.

Inspiración: Visualización llamativa de la interfaz. Sonidos sincronizados.

Mejoras: Facilidad de uso de la interfaz.

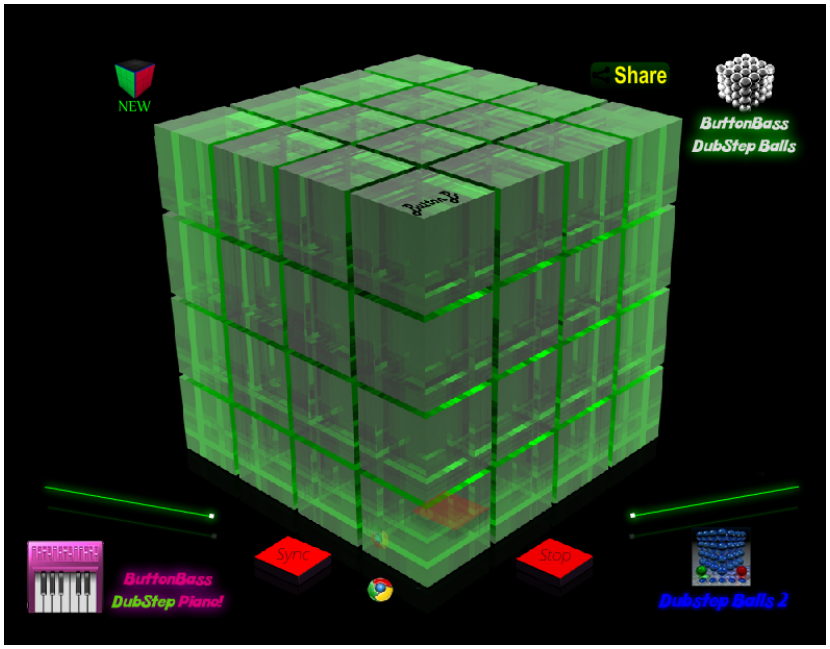

Figura 1.6: Interfaz visual del tercer referente online: Dubstep Cube

Estos referentes han servido como inspiración para la realización del trabajo. Dando una perspectiva más clara de la funcionalidad que se pretende llevar a cabo y la forma visual que se desea conseguir.

Las mejoras que se han propuesto, al realizar el estudio de mercado de estos dispositivos, han sido:

- La composición con posibles cambios de efecto panorámico, reverberación, volumen y filtrados.
- Ser una plataforma 100 % personalizable con respecto al resto de gratuitas.
- Los sonidos electrónicos se consiguen mediante un banco de sonidos propio, lo cual lo hace más práctico y sencillo de utilizar.
- La interfaz se desarrolla de manera intuitiva añadiendo iconos que describan su funcionalidad. Además de interacciones con JavaScript que facilitan al usuario el proceso, teniendo total control de sus elecciones en cada instrumento.

CAPÍTULO 2
Fundamentos de los lenguajes de programación utilizados

En este apartado se pretende exponer las características y usos de los lenguajes de programación que se trabajan.

2.1 HTML5

2.1.1. Indroducción histórica

El origen del lenguaje HTML (*HyperText Markup Language*) se remonta a 1980 cuando el físico *Tim Berners-Lee* propone una nueva forma de compartir documentos mediante un sistema de *hipertexto*. Él, junto a *Robert Cailliau*, llevaron más allá esa propuesta extendiendola al mundo de internet y formando así la *WorldWideWeb*.

Más adelante, por la gran cantidad de información que se transmitía a través de Internet, con el objetivo de darle un sentido y estructurarla, todos los datos que se exponían en este medio de información empezaron a contar con el estándar HTML 2.0. Este estándar fue la primera propuesta aprobada por el IETF (*Internet Engineering Task Force*) en 1995, a pesar de ya ser utilizado con regularidad los años anteriores.

Las siguientes versiones del estándar las publica el W3C (*World Wide Web Consortium*). La 4ª versión de HTML se publicó en Abril de 1998 que supone un gran salto puesto que introdujo las hojas de estilos CSS, programas o scripts en la misma página y mejoras en los formularios.

Desde la revisión del HTML 4.0 publicada en 1999 hasta 2004 W3C dejó de lado este lenguaje. Por este hecho, Apple, Mozilla y Opera se organizaron en la asociación llamada WHATWG (*Web Hypertext Application Technology Working Group*).

Aunque, al ver la fuerza de los cambios que fueron publicando el año 2007 el grupo WHATWG, W3C retomó la actividad que realizaba hasta entonces.

2.1.2. Versión actual

Ahora mismo la versión en la que se está programando es la 5ª, sacada en 2014 como lenguaje de WWW (*World Wide Web*). Aunque, para llegar a ello se ha estado desarrollando durante más una década la revisión del lenguaje oficial en el año 2000, el HTML4.

Figura 2.1: Logo del lenguaje HTML5

Como todas las versiones, especifica dos variantes de sintaxis: para HTML (text/html), la "clásica", utiliza HTML5; y para la variante XHTML (application/xhtml+xml), la "sintaxis", XHTML5 que deberá servirse con sintaxis XML.

A diferencia del resto, HTML y XHTML se han desarrollado en paralelo.

HTML5 domina el desarrollo en internet, por el interés que puso Google con su navegador Chrome y por Adobe, que eliminó el soporte de Flash para Android para dejar lugar al HTML5.

2.1.3. Características

A continuación, se expondrán las características que destacan de esta versión del lenguaje y, así, entender mejor su importancia.

Pero antes, es necesario saber que a pesar de estructurar el contenido, este lenguaje es, a su vez, texto. Se trata de un texto 'especial', llamado "etiquetas" (*Markups*), que se añade al resto de información para darle una forma. Por lo tanto, las etiquetas no serán leídas por el usuario, sino que se encargarán de dar las instrucciones de diseño a un programa especializado que las entienda, como los navegadores (Google, Mozilla, ...).

1. **Misma función con menos recursos**

   Lo primero que HTML5 ha mejorado, con respecto a la última versión existente, es que proporciona otras posibilidades para realizar las mismas estructuras

usando menos recursos, siendo más simples y lógicas. Además de provocar el desuso del formato XHTML, puesto que ya no se requiere su implementación.

El sistema que utiliza se encarga de formatear el layout de las páginas, pudiendo reorganizar y ajustar el contenido sin necesidad de hacer uso de tablas u otros elementos engorrosos. Además, HTML5 consigue unificar la programación en la mayoría de navegadores, conociendo la manera de mostrar una determinada página web en cualquiera de los navegadores.

Por lo tanto, hasta el momento, se considera que esta versión contiene un nivel de sofisticación del código más elevado con respecto a sus predecesores.

2. **Nuevas etiquetas semánticas**

   Una de las características que más cabe destacar son los elementos y atributos, etiquetas, que se han añadido en esta versión. Son similares a otras ya existentes pero cuentan con un significado semántico, lo cual se trata a continuación.

   Como bien expresa su nombre, las etiquetas definidas como semánticas, o que se encuentran en una Web 3.0 (Web semántica), describen el significado de lo que contienen.

   Así bien, las etiquetas y corresponden a versiones anteriores, ya que no expresan lo que se encuentra en su interior. Mientras que para esta versión se cuenta con <header>, que hace referencia a las cabeceras; <article>, que estructura un *post*; <nav>, el sistema de navegación (menú); y <footer>, el pie de página, entre otras.

   HTML5, además, se encarga de estructurar este contenido en un documento semántico de formato '*.html*'.

   Es aconsejable, para leer estas nuevas etiquetas, actualizar el navegador que se esté utilizando.

3. **Inserción multimedia y diseño Responsive**

   Otra de las grandes ventajas que presenta es la inserción de elementos multimedia en los sitios web como: audio, vídeo, canvas 2D y 3D, ..., mediante etiquetas especiales. Incluyendo tanto códecs de tipo libre (formatos: WebM y VP8) como privado (formatos: H.264/MPEG-4 AVC).

   Aunque no sea utilizado en este proyecto, es necesario conocer la mejora que realiza con respecto a los formularios, creando nuevos tipos de datos (e-mail y number entre otros) y facilitando la validación de los mismos sin utilizar JS (*JavaScript*).

   Por último, una de las novedades más importantes se enfoca en la ampliación del campo de las aplicaciones móviles. Ya que, ha conseguido trabajar tanto

en aplicaciones propias de sistemas operativos para móviles de forma offline, como en estructuras *Responsive* que se adaptan al tamaño de pantalla del visor.

2.1.4. Sintaxis

Antes de continuar con el siguiente lenguaje es necesario conocer la sintaxis que será útil en este proyecto:

- Las etiquetas van siempre entre los símbolos: < y >.
- Si se desea escribir comentarios a lo largo de la estructura se debe introducir entre: <!– y –>.

2.2 CSS3

2.2.1. Indroducción histórica

Antes de que se generalizara el uso del lenguaje que vamos a definir, los diseñadores utilizaban etiquetas HTML especiales para modificar el aspecto de los elementos de la página, como en el ejemplo siguiente:

```
1 <h1><font color="blue" face="Calibri" size="3">Titular</font></h1>
```

Listing 2.1: Antigua forma de dar estilo al texto

siendo color, face y size los atributos de la etiqueta <font>.

Debido a la enorme cantidad de atributos que se debían definir en una sola página, para cada etiqueta creada, esta forma no tiene cabida en un tipo de web renovado y ágil, en la Web 3.0.

Existe una forma intermedia entre las hojas de estilo, que se explican a continuación, y los atributos en <font> de HTML, conocida como '*definición en línea*'. Esta consiste, simplemente, en definir los atributos del elemento en él mismo, pero de una forma mucho más compacta:

```
1 <h1 style="color: blue; size: 14px;">
```

Listing 2.2: Estilos CSS en línea

Aun simplificando el proceso, continúa siendo engorroso y muy poco intuitivo. Por este motivo CSS (*Cascading Style Sheets*) cambia el concepto de estilo como se verá a continuación.

2.2.2. Versión

Al igual que el lenguaje HTML5, a CSS lo regula el organismo W3C. Esta organización definió la primera versión CSS1 en 1996, por la problemática que surgió al no estar estandarizado su uso entre los navegadores y desarrolladores. Anteriormente a esta revisión, el código se podía escribir de diferentes formas haciendo más complicado su desarrollo.

La estandarización conseguida por W3C permitió que los navegadores no interpretaran de maneras distintas una misma instrucción de estilo y mostraran el mismo resultado.

A lo largo de los años se realizaron distintas revisiones teniendo actualmente vigente la 3ª, llamada CSS3. Esta versión va ligada con HTML5 facilitando el mantenimiento de nuestra página al máximo.

Figura 2.2: Logo del lenguaje CSS3

Por otro lado, poco a poco, se ha modificado su uso facilitando la creación de los archivos '*.css*' mediante preprocesadores como LESS y SASS, que no se desarrollan en este trabajo pero son de gran utilidad en el mundo web.

2.2.3. Características

CSS se define como un lenguaje de hojas de estilos que controla el aspecto visual de los documentos estructurados con HTML.

Este lenguaje procura separar los contenidos estructurados de la página, de su diseño y forma de presentación. Para ello, divide estos dos conceptos en dos documentos distintos:

- *.html*, que expone el significado semántico de cada elemento de la página como párrafo, sección, lista, etc. como ya hemos descrito anteriormente en el apartado 2.1 y,
- *.css*, que define el aspecto de cada elemento que compone la estructura: color, tamaño y tipo de letra del texto, etc.

De esta manera se consigue crear páginas web de mayor complejidad de una forma más compacta.

Esta separación proporciona al programador numerosas ventajas, ya que, al obligar a crear archivos HTML por separado, estos están bien definidos y tienen un significado completo (documentos semánticos en el apartado 2.1). De esta forma, además, se consigue una mejora en el acceso al documento, se facilita el mantenimiento de este y se permite visualizarlo en infinidad de dispositivos.

Además, este lenguaje consigue agrupar el estilo de un elemento, repetido varias veces en la estructura HTML5, en una sola definición. A continuación, se muestra un ejemplo simple:

```
<style type="text/css">
  h1 { color: red;  font-family: Arial;   font-size: large;  }
</style>
<body>
  <h1>Titular</h1>
</body>
```

Listing 2.3: Estilo CSS agrupado entre las etiquetas <style>

Se puede observar, que escribiendo entre las etiquetas <style></style>, se define el estilo de miles de elementos en una sola regla, como en el caso anterior para la etiqueta <h1>.

Por otro lado, esto solo sería posible si todas las etiquetas a las que se quiere hacer referencia están en el mismo documento, en el que se define la etiqueta <style>. Si no es así, surge una nueva necesidad.

Para definir un estilo común a todas las subpáginas que contenga la página web se implementará un archivo *.css* independiente que se hará referencia en el <head> común, mediante:

```
  <link rel="stylesheet" type="text/css" href="/css/estilos.css" media="
     screen" />
```

Listing 2.4: Sentencia HTML para cargar la hoja de estilos 'estilos.css'

Siendo el archivo al que se hace referencia: *estilos.css*, definiendo los siguientes parámetros:

- **rel**: tipo de relación que existe entre el recurso enlazado (el archivo CSS) y la página HTML. '*Stylesheet*', i.e. hoja de estilos.
- **type**: tipo de recurso enlazado. Un texto de tipo CSS.

- **href**: indica la ubicación del archivo CSS que contiene los estilos. Puede ser interna, dentro de nuestro proyecto (css/estilos.css), o externa, una URL.
- **media**: medio al que se aplican. Estos estilos se aplicarán a la pantalla.

Al poner este enlace se está haciendo referencia a las hojas de estilo que el navegador debe leer y, así, se consigue que el lector descargue este archivo y lo cargue, al igual que realiza con la estructura HTML.

2.2.4. Sintaxis

Al igual que en el apartado 2.1 es preferible conocer de antemano de qué forma se debe programar. Para ello, se enumeran las distintas maneras que tiene un elemento HTML de ser nombrado en la hoja de estilos:

- **Etiqueta**

  De uso común en HTML, este tipo se nombra poniendo la nomenclatura que se encuentra entre < y > sin ningún tipo de símbolo añadido, ya que nombra a todos los elementos de un mismo tipo.

  Si la etiqueta es <h1> su equivalente en CSS será h1{ 'definiendo aquí los atributos' };

- **Identificador** (selector único)

  Esta forma de nombrar a un elemento es única. Sólo puede haber un identificador con ese nombre, independientemente del tipo de elemento no se debe repetir.

  Para un elemento como <section id="section-book"> se define el estilo mediante #section-book{ };

- **Clase** (selector múltiple)

  Las clases, a diferencia del identificador, se pueden definir cuantas veces se quiera en los elementos que se necesite.

  Una clase va definida dentro del atributo 'class' en la etiqueta como: . Y, para ello, se define .container{ }; en el archivo '*.css*'.

- **Comentarios**: entre /* y */.

Además, los atributos tienen siempre la misma estructura, siendo esta: "propiedad (color, size, ...)": "valor (blue, 14px, ...)";

Todos los valores, etiquetas y propiedades que se describan estarán en inglés para globalizar su uso.

2.3 JavaScript

2.3.1. Indroducción histórica

El JavaScript surge en 1997 por la necesidad del HTML de crear nuevas animaciones y efectos que, a su vez, consigan mejores experiencias en las webs y ampliar sus prestaciones. Hasta su creación, HTML se centraba en actuar exclusivamente sobre el contenido y su diseño, pero tras su llegada experimentó grandes cambios.

Gracias a este lenguaje de desarrollo, HTML aumentó sus servicios con eventos como abrir una ventana emergente o transmitir un mensaje de error, validación de un formulario.

Este lenguaje de programación, a diferencia de otros, se ha convertido en un estándar con soporte en todos los navegadores actuales, y no tan solo funcional en navegadores de las firmas creadoras.

Netscape fue la unidad donde Brendan Eich inició la implementación de JavaScript, aunque anteriormente no fuese conocido por ese nombre sino por LiveScript. Posteriormente, Netscape y Sun, empresa fundadora del lenguaje Java, se unieron permitiendo que JavaScript se impulsara, bautizándose con ese nombre por el auge de Java en esos momentos. Además de definirse como una variante de Java más fácil de utilizar.

Otra versión diferente, pero con el mismo fin de este tipo de lenguaje, es JScript, que fue desarrollada por Microsoft reduciendo su uso a ese sistema operativo.

Al principio se pensaba que JavaScript era una modificación de Java más simple, pero lo cierto es que en lo único que se asemejan es en la sintaxis. Fue creado con el propósito de tener un lenguaje auxiliar para HTML, no teniendo uso fuera del mundo web.

2.3.2. Versión

La versión actual, la cual será utilizada, corresponde a la tercera edición del estándar ECMA-262, organización que regula este lenguaje. Ésta fue publicada en 1999, pero en estos momentos la cuarta versión se encuentra en desarrollo, por lo que se sigue utilizando.

2.3.3. Características

Ahora se hablará de las características y uso que se hace de este lenguaje de programación, para tener más clara su utilidad en este proyecto.

Figura 2.3: Logo del lenguaje JavaScript

- **Interpretación línea a línea**

  La forma en la que un navegador lee el código desarrollado en JavaScript es interpretando línea a línea su contenido. Esta lectura se realiza en el momento de carga de la página a la que pertenece.

- **Entorno de aplicación**

  Al igual que CSS, JavaScript tiene la opción de realizar sus acciones en el entorno de la página HTML donde está escrito, por medio de las etiquetas <script></script>; o en varias páginas, poniendo la llamada al archivo desde el <head>.

- **Lenguaje orientado a objetos**

  Se conoce como un lenguaje orientado a objetos, ya que la mayoría de instrucciones que se emplean son llamadas a propiedades y métodos de objetos del navegador e, incluso, del propio lenguaje.

2.3.4. Usos

Como se ha resumido anteriormente, JavaScript es embebido en la página donde se coloca el elemento <script>, en el <head> de la misma exactamente. La forma de ubicarlo se muestra en este ejemplo:

```
1  <html>
2    <head>
3      <script language="JavaScript">
4        // methods
5      </script>
6    </head>
7    <body>
8    </body>
9  </html>
```

Listing 2.5: Formas de colocar las sentencia JS dentro de una página

Pero de esta forma se tiene que modificar el archivo que contiene tanto el HTML como el JS. Para no tener que desarrollar dos tipos de lenguaje en un mismo archivo, se puede crear el JS en un archivo externo, tan sólo es necesario hacer referencia al mismo:

```
1 <script type="text/javascript" src="/js/codigo.js"></script>
```

Listing 2.6: Sentencia HTML para cargar el archivo 'codigo.js'

Por cada archivo externo que se haya desarrollado se debe repetir esta línea con la URL correspondiente, o ruta del archivo, en src. El número de archivos puede ser infinito, pero hay que tener en cuenta que para cada uno habrá que definir la sentencia anterior.

2.3.5. Desventajas

Aunque el JavaScript sea un complemento para el lenguaje estándar de la web (HTML), este no siempre cumple la norma. El mayor problema es que tiene diferentes versiones y según el navegador que lo lea necesitará una versión u otra. Por culpa de este hecho, algunos eventos de la página serán leídos y realizados para unos navegadores, pero en otros podrían obstaculizar el paso del usuario no mostrando al completo su funcionalidad y/o contenido.

Para conseguir acabar con este hándicap se tiene que probar el código desarrollado en la mayoría de navegadores en los se quiera hacer uso de la web, programando de la forma más estandarizada posible.

Aunque se consiga desarrollar un código adaptado a cualquier navegador sigue habiendo otros problemas.

Uno de los aspectos que más obstaculiza el uso de este lenguaje en las páginas web es que el usuario no siempre tiene activado el JavaScript de su navegador. Por ello, no se debe basar el funcionamiento de la página en los eventos cargados mediante este lenguaje.

2.3.6. Sintaxis

Las variables de JS no cuentan con un símbolo previo, sino que el propio nombre que se adjudique a la variable será esta en sí misma. Pero hay que tener en cuenta que para inicializarla se debe incluir esta sentencia: `var variable = 0;`

La ventaja de esta sintaxis es que no es necesario definir el tipo de variable de la que se trata, puesto que al darle valor, automáticamente, se convierte en ese tipo. Esto trae infinidad de beneficios.

Si se necesita escribir comentarios se pondrán los elementos // seguido del texto, pero si se necesita utilizar varias líneas se introducirá el contenido entre /* y */.

Este lenguaje a diferencia de los dos anteriores sí que entra, en cierto modo, en el *backend* de la página pudiendo desarrollar funciones, estructuras con iteraciones como for, while, entre otras. Todo ello similar a los lenguajes PHP, C, C++ y un largo etc.

2.4 Csound6

Csound se trata de un lenguaje de programación destinado a la creación, edición y análisis de cualquier tipo de sonido o melodía.

Este lenguaje cuenta con un compilador programado en C, por ello contiene la C en su nombre. Además, contiene la palabra sound que hace referencia a la finalidad de composición sonora del mismo.

Este lenguaje es un software libre basado en código abierto con estructura modular. Para garantizar la libertad de compartir y modificar su software cuenta con licencia LGPL (Lesser General Public License, i.e. Licencia Pública General Reducida).

Además, permite su ampliación por el usuario, pudiendo extender su funcionalidad hasta los límites que se deseen.

2.4.1. Introducción histórica

Csound deriva de los lenguajes de generación de sonido creados por Max Matthews, que se encargó de recopilar algoritmos y crear Music, incluidas todas sus versiones hasta la 4ª.

Csound fue formado en 1970 para cubrir las necesidades que tenían los lenguajes que le precedían. Como los ordenadores iban avanzando y los lenguajes Music se iban quedando obsoletos, se tenía que adaptar cada versión a los nuevos sistemas de computación.

Después de varias versiones de Music, Barry Vercoe desarrolló un sistema para el IBM System/360, el Music11. Fue el primer programa sonoro desarrollado para microcomputadores y antecesor directo de Csound.

Al inicio de su creación trabajaba en sistemas operativos como MAC OS X, MS-Windows, Unix/Linux y DOS/os. Las personas que formaron este lenguaje eran compositoras interesadas por la música y la informática, mezclando ambos conceptos se consiguió desarrollar el proyecto.

Como ya se ha comentado, Barry Vercoe contribuyó ampliamente en su creación junto con Richard Boulanger del Berklee College of Music (de Boston), Dan Ellis y Bill Gardiner.

Este lenguaje destaca respecto a los anteriores por la gran flexibilidad que proporciona. Por otro lado, permite su funcionamiento en cualquier sistema operativo que contenga un compilador en C, siendo totalmente compatible.

En la actualidad, se estima que más de 1200 operadores dan funcionalidad al lenguaje Csound. Cada cierto tiempo, con regularidad, aparecen nuevas versiones publicadas que incluyen ampliaciones y mejoras de los algoritmos que lo forman. Todo ello realizado por una comunidad de programadores que día a día investigan sobre este lenguaje.

2.4.2. Estructura de Csound y funcionamiento de la misma

- **Ficheros orquesta**

  La estructura por la que está formado se compone de instrumentos que se almacenan en un fichero de orquesta. Para la formación de los instrumentos cuenta con operadores de síntesis y procesamiento del sonido, siendo la funcionalidad principal de este lenguaje.

- **Ficheros partitura**

  Para el control de los instrumentos contiene un fichero de partitura donde se pueden activar, pausar o pasar parámetros.

La forma en la que el compilador consigue extrapolar el resultado sonoro de la programación es procesando los archivos mencionados, orquesta y partitura, para realizar las instrucciones programadas.

A continuación, se expone cómo trabajaCsound con las dos clases de objetos mencionados anteriormente, que son orquesta y partitura.

1. Se crea la «orquesta» (*orchestra*) junto con los instrumentos que se utilizarán en la composición.

   A la hora de crear un instrumento hay que describirlo, explicar cómo es y cómo suena. El instrumento puede componerse tanto de un oscilador que genere una sinusoide de una frecuencia determinada, tono puro, como de un instrumento complejo cuyo timbre varía estadísticamente.

2. En segundo lugar, se forma la partitura ((*score*). Es, en si misma, una tabla o gráfica en la que se expone el orden de activación de los instrumentos en el tiempo que dure la reproducción.

Además del orden de reproducción se pueden controlar parámetros del sonido como la frecuencia, la velocidad y la amplitud, entre otros. La forma en que se transmiten estos parámetros se muestra en el apartado sintaxis de la sección 2.4.5. Mientras el sonido sea digital Csound podrá controlarlo al completo.

Desde la versión 5.0, Csound da un gran cambio en código añadiendo mejoras como la posibilidad de escribir la orquesta y la partitura en un único fichero compacto (.csd) estructurado mediante *tags* tipo XML.

2.4.3. Versión

La versión de Csound que se utiliza en el presente proyecto es la 6.0.9.

Figura 2.4: Logo del lenguaje Csound

2.4.4. Usos

Csound tiene una variedad inmensa de tratamientos de audio siendo conocido por el trato que da al formato MIDI. Este lenguaje es capaz de leer, crear, modificar, escribir, reproducir cualquier nota, frecuencia, duración o velocidad que sea de tipo MIDI. Estas acciones las realiza mediante *opcodes* ya creados como puede ser **midiin**, que lee contenido midi, o **midiout**, que realiza la acción contraria. En definitiva, es capaz de obtener información desde cualquier controlador MIDI o enviarla a la tarjeta de sonido o a archivos de audio o texto.

El uso principal que se le da en este proyecto es para tiempo real, siendo necesario para su ejecución un procesador con buenas prestaciones. No obstante, en un ordenador portátil también es posible su uso.

Contiene múltiples posibilidades para la creación de interfaces gráficas, lo que facilita la tarea de crear aplicaciones que se puedan utilizar de manera ágil e intuitiva para el usuario.

2.4.5. Sintaxis

Ahora se mostrará un ejemplo simple para entender la sintaxis que utiliza este lenguaje.

Para entender como un mismo archivo .csd contiene tanto el instrumento como la partitura se expone un caso sobre la generación de un tono sinusoidal simple.

```
instr prueba                ; instrumento llamado prueba
  iamplitud = 21000  ; amplitud de la sinusoide (volumen) = 21000
  ifrecuencia = p4   ; frecuencia de la partitura (cuarto item)
  itabla = 101       ; tabla que lee la funcion oscil
  aout oscil iamplitud, ifrecuencia, itabla ; operador que genera la
     sinusoide
  out aout
endin                ; final del instrumento
```

Listing 2.7: Código Csound para la definición de un instrumento

Esta parte del programa es la referente a la orquesta. En ella se encuentra un instrumento definido con el identificador de prueba, ya que en las últimas versiones proporciona tanto número como letras para su definición.

En el interior de un instrumento, entre las etiquetas de inicialización y fin, se incluyen los parámetros internos a utilizar en la creación de la sinusoide, que son respectivamente: amplitud, frecuencia y tabla de forma de lectura de la sinusoide.

Los tipos en Csound se definen por la letra con la que empiezan los identificadores de las variables. Así, una variable de tipo 'a' empieza por a y corresponde a una variable que contiene una señal de audio representada por un array de muestras. Por ello, la variable que contiene el audio de salida se nombra como 'aout' y se ejecuta mediante el *opcode* 'out'. Siendo 'aout' el resultado que proporciona la ejecución de 'oscil'.

Por otro lado, cualquier contenido escrito después del carácter ';' es un comentario y no será ejecutado.

```
f 101  0  4096  10  1     ; tabla que contiene un ciclo de onda
   sinuosidal
;instr     inicio  duracion frecuencia
; p1         p2        p3       p4
i prueba     0         2        440
i prueba     +         .        880
i prueba     +         .        1760
e     ; fin de la partitura
```

Listing 2.8: Código Csound para la definición de una partitura

Una vez definido el instrumento para que la lectura cobre sentido se deben pasar los parámetros que utiliza. Para ello, se ha creado la partitura que activa y controla el instrumento prueba.

Una vez se ejecuta el programa, las llamadas desde la partitura al instrumento permitirán que cree la sinusoide con la frecuencia que se encuentra en el 4º elemento.

Dependiendo de la configuración con la que se ejecute Csound, este programa puede crear un archivo sonoro y almacenarlo o enviarlo a la tarjeta de sonido y ejecutarlo.

CAPÍTULO 3

Características de los programas de desarrollo

Una vez se conocen los lenguajes y objetivos de este proyecto, no está de más hablar de los programas que permiten su desarrollo. Así, como de las páginas que proporcionarán parte del contenido visual y sonoro.

En este apartado se desarrollarán las aplicaciones que se han utilizado.

3.1 Paint

Paint es el programa de dibujo simple que aparece incluido en cualquier versión del sistema operativo Windows, desde 1985 hasta el momento, pero que dejará de estar disponible en la siguiente versión del OS (*Operating System*).

Figura 3.1: Logo del programa Paint

Este software sirve para realizar bocetos simples, rápidos y de peso reducido, utilizando pocos KB. En él se incluyen multitud de formatos para almacenar los archivos producidos.

3.1.1. Ventajas

Una de las virtudes de este programa de edición es la facilidad que ofrece a la hora de realizar cualquier tarea. Si se busca una funcionalidad más compleja este

editor no tiene suficientes prestaciones, pero para una plantilla simple y estructurada es más que suficiente.

Además, al ser gratuito y poder trabajar de manera offline (en Windows) ha conseguido una gran popularidad entre los estudiantes, pudiendo utilizarlo para crear esquemas y editar de forma básica imágenes, sin tener que ser todo un experto.

3.1.2. Usos

En este proyecto se ha utilizado con el fin de crear una plantilla fácil y visual para después poder plasmarla mediante código HTML5 y estilo CSS.

3.2 Flaticon

Figura 3.2: Logo de la web Flaticon

flaticon.com se trata de una página web que reúne gran variedad de iconos desarrollados por diseñadores y/o artistas.

A la hora de conseguir estos iconos existen gratuitos y premium, para lo que se necesitaría estar registrado y pagar una cantidad económica al mes.

3.2.1. Ventajas

Uno de los puntos fuertes de esta página web es su buscador. Desde el buscador se puede buscar cualquier tipo de palabra, escrita en inglés, para la que aparecerán todos los iconos relacionados.

Por ejemplo, si se pretende encontrar un icono de interrogación, poniendo: '*question*' aparecerán todos aquellos que estén relacionados.

Además, los iconos se pueden encontrar en forma de paquetes que contendrán una colección temática con diferentes símbolos, pero con el mismo estilo. Esto es útil para realizar un proyecto con un estilo homogéneo.

No obstante, si todavía no está claro el estilo que se busca, lo mejor es encontrar el icono deseado y ver, en los resultados, todas las variantes de este.

Otra de las características que resalta de esta web, es que estos iconos se pueden obtener en diferentes tamaños y formatos, contando con el formato de moda .svg, que hoy en día es muy útil en la velocidad de repuesta de las webs y en la versión responsive.

3.2.2. Usos

Para el uso que se le va a dar en este trabajo la modalidad gratuita es suficiente.

El formato que más se adapta a la necesidad de nuestro prototipo es .png, ya que es el más conocido y contiene transparencias.

3.3 Freesound

freesound.org es una página web con 4 millones de usuarios que suben y bajan sonidos. De esta forma se ha creado una de las bases de datos más completa de este ámbito, contando con más de 230000 sonidos.

Figura 3.3: Logo de la web Freesound

3.3.1. Características

En su repositorio se pueden encontrar muestras de audio de cualquier tipo de sonido. Para encontrar muestras específicas contiene un buscador con varios niveles de filtrado, así consigue acercarse al resultado más exacto. Esto es gracias al etiquetado que se adjudica a cada muestra, proporcionado por folksonomic.

Una de las grandes ventajas de esta comunidad colaborativa es que tiene licencia por *Creative Commons*, lo que hace que tanto músicos como oyentes tengan confianza en ella.

Otra es el uso extendido que se hace de esta página, su contenido se analiza mediante la herramienta de análisis de audio Essentia. Esta herramienta cuenta con un código abierto que analiza, sintetiza y describe audio potenciando la de búsqueda de similitud de Freesound.

Además, cuenta con una API que permite el acceso de otras aplicaciones a su base de datos.

3.3.2. Usos

Los sonidos utilizados para este prototipo son grabados sin edición. Aunque Freesound cuente hasta con efectos de sonidos sintetizados, solo se usarán de este tipo.

3.4 PhpStorm

Esta aplicación es conocida como el mejor entorno de desarrollo interactivo (IDE: *Integrated Development Environment*) de PHP. La finalidad de este programa es facilitar al programador el desarrollo de software que esté realizando. Aunque, en un principio, su uso está destinado al lenguaje PHP contiene editores de texto extendido para los tres lenguajes web que se utilizan, HTML, CSS y JavaScript.

Figura 3.4: Logo del IDE PhpStorm

Se encarga de prevenir contra errores instantáneamente, como puede ser detectar un error en la sintaxis. Además, es capaz de autocompletar el código mientras se escribe y de refactorizalo.

3.4.1. Características

A continuación, se explica brevemente las características que hacen de esta plataforma el entorno de desarrollo más útil:

- **Análisis**

  Principalmente, el análisis que realiza sobre la calidad del código le da un papel muy importante en el desarrollo de cualquier aplicación. PhpStorm inspecciona el código cientos de veces para verificar, como ya se ha comentado incluso mientras se está escribiendo, que esté exento de posibles errores. Esto permite solucionar los errores antes de seguir programando.

- **Búsqueda avanzada**

  Otra de las ventajas respecto a un editor de texto común es que permite realizar una búsqueda exhaustiva. Contiene elementos de búsqueda y navegación

sencillos que ayudan a encontrar de una forma eficiente en cada plantilla, carpetas o en el proyecto completo. Dentro de los elementos se incluyen búsquedas tanto nombres de archivos como variables, funciones, líneas de código o textos simples, entre otras.

- **Editor especializado en HTML, CSS y JavaScript**

  Como se ha mencionado, contiene un editor específico HTML/CSS/JavaScript que incluye todas las características de estos tres lenguajes. La parte encargada de HTML y CSS permite ver los cambios realizados en el navegador sin refrescar la página, instantáneamente. Para el lenguaje JavaScript cuenta con herramientas que permiten finalizar, validar y perfeccionar el código, entre otras.

3.4.2. Usos

Este programa es crucial para el desarrollo del front-end del prototipo realizado. La parte visual del launchpad se realizará, inicialmente sin funcionalidad, en este IDE.

3.5 CsountQt

CsoundQt (nombrado QuteCsound hasta 2011) es un programa gratuito que permite programar código en Csound siendo compilado desde el mismo, i.e. editor de código principal del lenguaje Csound. Se trata del frontend, parte visual gráfica, que permite la creación y ejecución de código de una forma más sencilla, ya que cuenta con botones para la realización de las funciones principales sin ejecutarlas desde línea de comandos, como Run y Stop.

Figura 3.5: Logo del programa CsoundQt

Además, incluye el concepto de multi-plataforma. Contiene soluciones que se utilizan en diferentes entornos tecnológicos, creando aplicaciones que funcionan en diversos sistemas operativos, aparte de en una variedad inmensa de dispositivos. De esta forma, con un solo archivo, se puede leer el código en cualquier SO, evitando tener que crear un ejecutable para cada uno.

3.5.1. Características

- **Código de colores para la sintaxis de Csound**

  Este programa cuenta con destacado de palabras para variables, instrumentos y partitura, entre las más importantes. Gracias a este formato de escritura se consigue un código más limpio e intuitivo, ya que se distingue claramente qué palabras hacen referencia a qué funcionalidad.

- **Ejemplos de código integrados**

  CsoundQt cuenta con una gran base de datos de ejemplos, incluido en el mismo programa o descargable en GitHub. Se compone de múltiples ejemplos de código, pasando desde tutoriales de escritura básica hasta sintetizadores o códigos específicos complejos.

  Además, cuenta con un manual de ayuda para cualquier duda sintáctica que surja.

- **Panel de widgets**

  Para el control de Csound, CsoundQt proporciona una gran variedad de widgets. Contiene opcodes (operation codes) que permiten la transmisión de información del widget a la plataforma, y viceversa.

  Gracias a estos se permite trabajar más cómodamente en tiempo real.

- **Detección de código HTML y JS**

  En las últimas versiones CsoundQt ha contado con los lenguajes web, que se encuentran en auge, para su plataforma. El código HTML es detectado cuando el archivo .csd contiene el elemento <html>, encontrándose embebido dentro.

  Una vez compila el código, lo muestra en la GUI (Graphical User Interface) de HTML5 que tiene implementada.

  Para el control de Csound desde la interfaz implementada en HTML5 se debe implementar código en JS, ya que contiene opcodes que relacionan y controlan los instrumentos de Csound y el *frontend* desarrollado.

3.5.2. Usos

Los usos que se hacen de este programa se desarrollarán en profundidad en los capítulos 5 y 6, pero el objetivo principal de CsoundQt es desarrollar código Csound, *backend* y conectarlo con HTML5, el *frontend*, que se ha utilizado en este proyecto.

CAPÍTULO 4

Front-end - Maquetación HTML5, estilo CSS y efectos JavaScript

En este capítulo, se hablará del método seguido para la realización del aspecto visual de la plataforma.

Inicialmente, se ha llevado a cabo una plantilla en Paint (programa descrito en la sección 3.1) que ha servido como modelo para la estructura básica del Front-end:

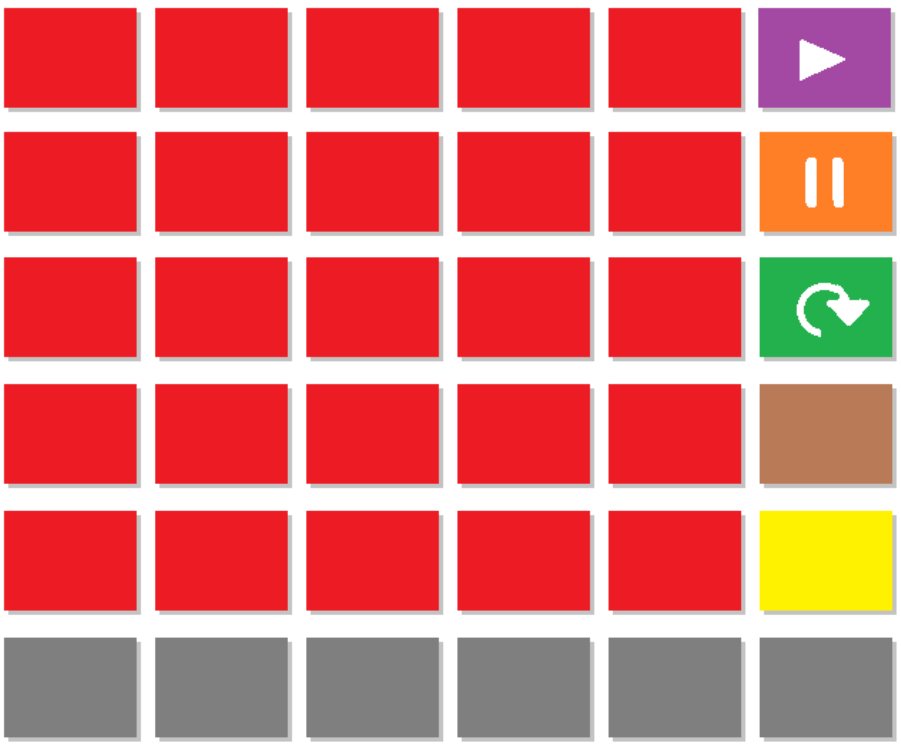

Figura 4.1: Plantilla inicial realizada en Paint

A continuación, se desarrollarán los puntos necesarios para comprender la maquetación que se extrapola de la plantilla anterior.

4.1 HTML5

En este apartado se pretende mostrar, parte por parte, el código HTML5 desarrollado para llevar a cabo la interfaz visual del prototipo propuesto. No obstante, existen diversas opciones al realizar una estructura de este tipo.

La vista se divide en dos plantillas claramente diferenciadas. Primero se observa la plantilla inicial, delimitada por las etiquetas <section>, donde se muestran los instrumentos de los que dispone la plataforma y los botones básicos de control.

4.1.1. Primera plantilla

Esta primera plantilla se define por el identificador 'first-page':

```
<section id="first-part">
// Content
</section>
```

Listing 4.1: Estructura de la primera plantilla

En el interior se encuentra la representación de esta primera parte, la cual se divide en tres secciones diferenciadas:

4.1.1.1. Primera zona

Se compone de 5 filas x 5 columnas, lo que hacen un total de 25 celdas. El tamaño de esta zona va fijado por las clases 'row-5' y 'col-xs-5' de CSS, lo cual se visualizará en el apartado CSS.

```
<div class="row-5 col-xs-5">
// celdas
</div>
```

Listing 4.2: Estructura de la matriz de instrumentos en HTML

En estas celdas se intercalarán dos tipos de botones: unos sin utilidad y otros representados mediante instrumentos con la función de activar la segunda pantalla, la equivalente a cada uno de ellos.

Para definir el botón correspondiente a una celda vacía se utiliza la siguiente estructura:

```
<div class="col-xs-3 style-button violet">
    <figure class="style-figure hide-figure">
        <img width="48" height="48">
    </figure>
</div>
```

Listing 4.3: Botón sin uso y sin imagen

No se utiliza la etiqueta <button>, ya que define inicialmente un estilo. Para evitar condicionantes a la hora de dar estilo o funcionalidad a un elemento se ha utilizado la etiqueta , que cumple con todos los requisitos.

El tamaño del botón se define por la clase 'col-xs-3', mientras que el color y la forma la definen las clases 'violet' y 'style-button' (pie de página: sección CSS), respectivamente.

En su interior se encuentra una imagen vacía. Esto sirve para delimitar el tamaño del botón igualándolo al valor de los botones que si contienen una imagen. Para evitar la representación de una imagen sin encontrar se añade la clase 'hide-figure'.

Este tipo de estructura se puede observar a continuación:

Figura 4.2: Botón vacío de la primera zona de la plantilla principal del Launchpad

Por otro lado, la estructura de los botones con funcionalidad será similar a la descrita anteriormente, pero con una ruta a la imagen correspondiente y la llamada a una función cuando se pulsa sobre este.

A continuación se muestra un ejemplo:

```
<div class="col-xs-3 style-button violet" onclick="instrument('
    instrumento')">
    <figure class="style-figure">
        <img src="img/percusion/instrumento.png" width="48" height="48">
    </figure>
</div>
```

Listing 4.4: Botón que llama a la segunda plantilla con el instrumento que muestre en la imagen

siendo 'instrumento' sustituido por cada uno de los que se observan en el prototipo siguiente:

Figura 4.3: Botón con instrumento de la primera zona de la plantilla principal del Launchpad

Figura 4.4: Primera zona de la plantilla principal del Launchpad

4.1.1.2. Segunda zona

La segunda estructura está definida como la última columna del Launchpad. En ella se encuentran los controles generales.

Para darle forma a la columna y contener en ella las celdas correspondientes se utilizan las clases: 'row-5' y 'col-xs-1', en el algoritmo siguiente se muestran definidas:

```
1 <div class="row-5 col-xs-1 colocate">
2 //celdas
3 </div>
```

Listing 4.5: Estructura de la columna de controles

Para adaptarla al resto de la interfaz se añade la clase 'colocate' que se especificará en el apartado CSS.

El contenido de las celdas se estructura principalmente de la siguiente manera:

```
<div class="row">
    <div class="col-xs-3 style-button green" onclick="commonControl()">
        <figure class="style-figure">
            <img src="img/control.png" width="48" height="48">
        </figure>
    </div>
</div>
```

Listing 4.6: Estructura de los botones de control del instrumento

Como se observa en la figura siguiente es prácticamente idéntico al botón de instrumento, modificando la clase que proporciona el color y la función y la imagen a las que llama.

Figura 4.5: Botón de control de la segunda zona de la plantilla principal del Launchpad

Al igual que con la palabra 'instrumento', aquí se sustituye la palabra 'control' por los controles definidos en el apartado JS: play, pause y stop.

Y la forma visual que se consigue junto con el estilo que se lleva a cabo en CSS

4.1.1.3. Tercera zona

Por último, aunque en esta plantilla no se utilice la última fila la estructura debe definirse.

Para su forma se define una fila de 6 columnas en este caso, ya que el botón inferior derecho está incluido en ella. Las clases que utiliza son 'row-1' y 'col-xs-6', respectivamente.

```
<div class="row-1 col-xs-6">
//estructuras sin nada
</div>
```

Listing 4.7: Estructura de la fila inferior de la primera plantilla

Figura 4.6: Segunda zona de la plantilla principal del Launchpad

A la hora de definir una estructura vacía tan solo definimos el tamaño y su forma y aspecto. Por eso, se repite el código que se muestra a continuación tantas veces como celdas vacías existan:

```
<div class="col-xs-3 style-button green">
</div>
```

Listing 4.8: Estructura de los botones sin utilidad

Figura 4.7: Botón vacío de la tercera zona de la plantilla principal del Launchpad

Para el botón que permitirá el reinicio de la aplicación se diseñará con el mismo aspecto que un elemento de control, pero cambiando la imagen y su función asociada:

```
<div class="col-xs-3 style-button spaced-first green" onclick="reset()">
<figure class="style-figure">
<img src="img/back.png" width="48" height="48">
</figure>
</div>
```

Listing 4.9: Estructura del botón de reseteo

Figura 4.8: Tercera zona de la plantilla principal del Launchpad

De esta forma, finaliza la explicación de la primera parte de la interfaz visual y se obtiene como resultado:

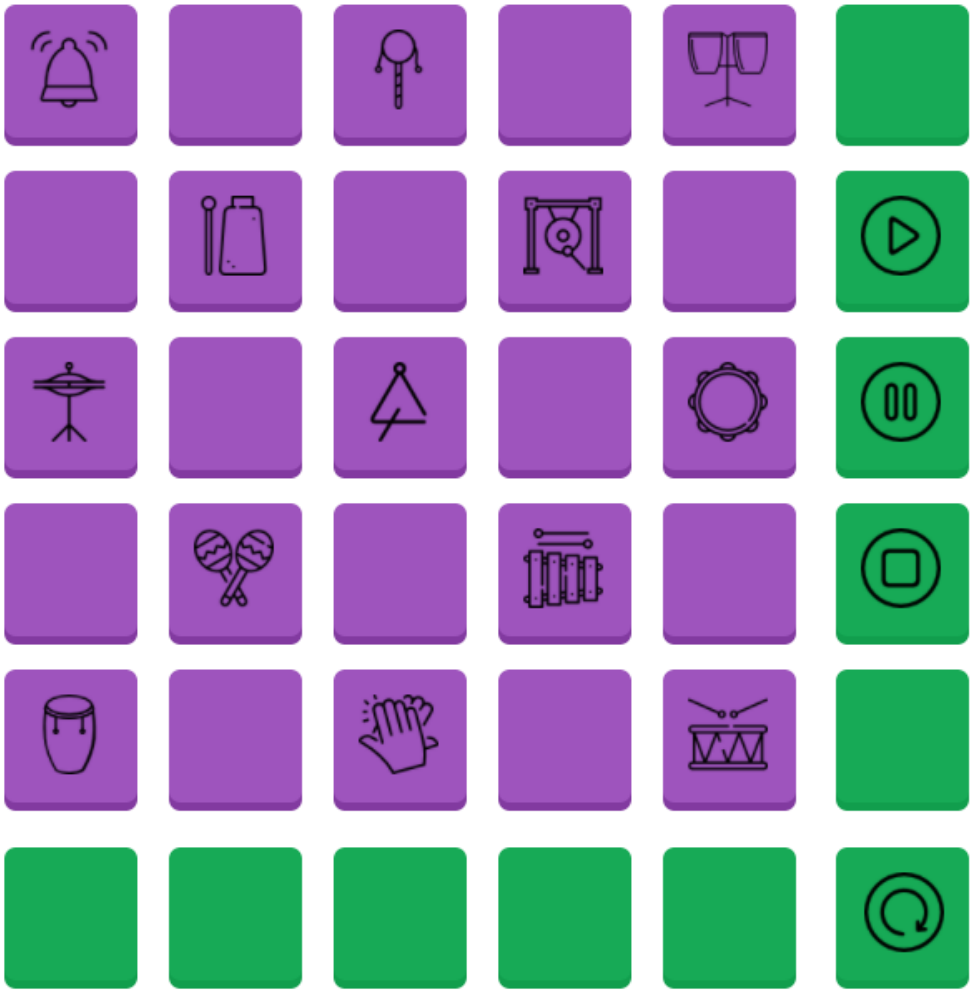

Figura 4.9: Plantilla principal del Launchpad

4.1.2. Segunda plantilla

La segunda plantilla se corresponde a la sección que contiene la clase 'instrument', que inicialmente no se visualizará. Programando el código siguiente se consigue ocultar esta parte de nuestra plantilla, temporalmente:

```
<section class="instrument" style="display: none;">
//contenido segunda cara
</section>
```

Listing 4.10: Estructura de la plantilla del instrumento

Esta sección se define por las 3 formas que existen en la primera plantilla, pero con otra funcionalidad. A continuación, se exponen los cambios realizados:

En esta matriz cuadrada de 25 celdas se pretende modificar la lectura del instrumento mediante los filtros y efectos que contiene cada columna.

4.1.2.1. Primera zona

En el interior de se encuentran 5 estructuras de este tipo:

```
<div id="level" class="row">

    <!-- Button de control de Amplitud -->
    <div class="col-xs-3 style-button cian" onclick="volumen(level)"></
        div>

    <!-- Filtros -->
    <div class="col-xs-3 style-button orange"></div>
    <div class="col-xs-3 style-button orange"></div>

    <!-- Button de control de Panorama -->
    <div class="col-xs-3 style-button blue" onclick="panoramica(level)"><
        /div>

    <!-- Button de control de Rever -->
    <div class="col-xs-3 style-button green-lighter" onclick="
        reverberacion(level)"></div>
</div>
```

Listing 4.11: Estructura de una fila en la plantilla del instrumento

La relación entre el nivel (level) y la posición del botón va desde 1, en el nivel inferior, hasta 5, en el superior. Esta variable será sustituida según la fila en la que se encuentre.

Cada una de las funciones de JS a las que llaman obtendrán el valor del nivel y actuarán según el mismo. Tanto el efecto visual de la plataforma como el efecto sonoro producido según este valor, se explicarán en los apartados CSS, JS y Csound.

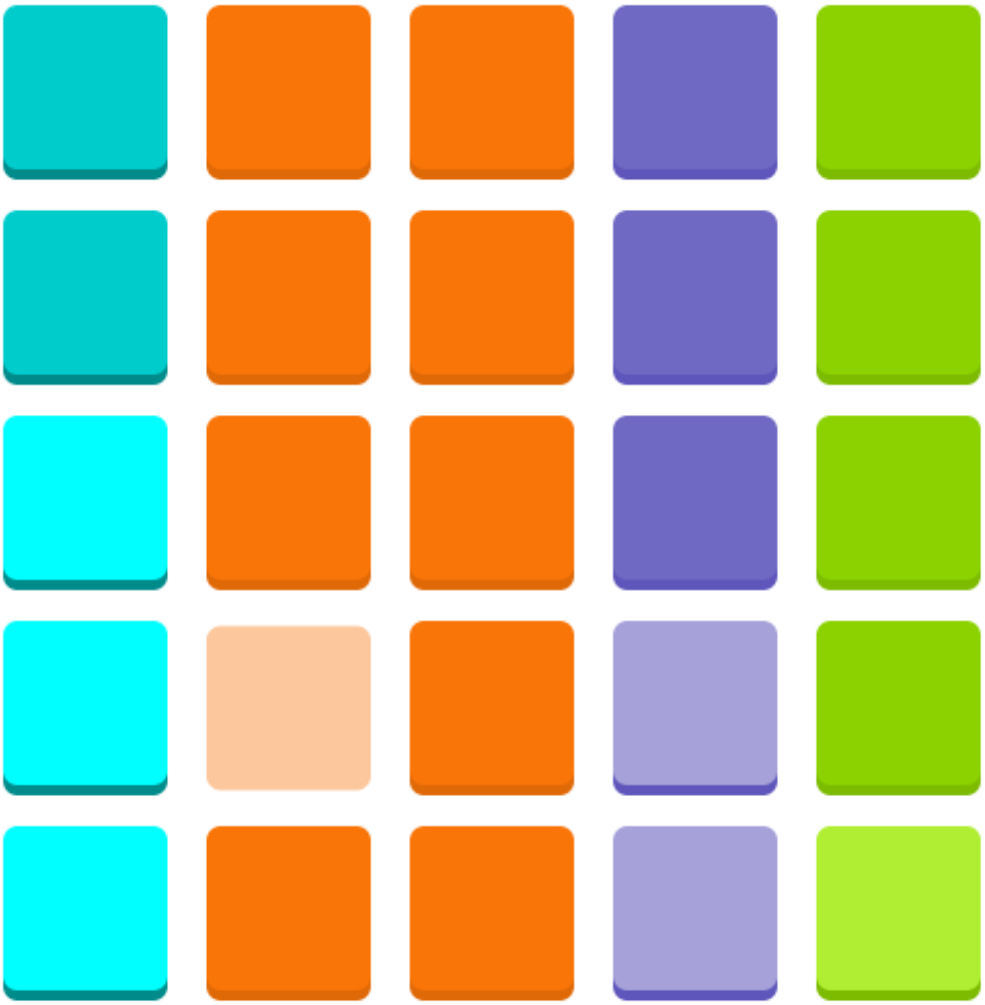

Figura 4.10: Primera zona de la plantilla principal del Instrumento

4.1.2.2. Segunda zona

Este apartado es idéntico al explicado en la primera parte del HTML, a diferencia de que las funciones a las que llaman dejan de ser comunes y, por lo tanto, su nomenclatura es distinta:

```
<div class="row">
    <div class="col-xs-3 style-button green" onclick="insControl()">
        <figure class="style-figure">
            <img src="img/control.png" width="48" height="48">
        </figure>
    </div>
</div>
```

Listing 4.12: Estructura de un botón de control en la plantilla del instrumento

Siendo insControl() sustituido por cada control: insPlay(), insPause() e insStop().

Figura 4.11: Segunda zona de la plantilla principal del Instrumento

4.1.2.3. Tercera zona

En la última fila, aunque ya no estén vacías las estructuras, seguirán sin tener funcionalidad.

```
<div class="col-xs-3 style-button green">
    <figure class="style-figure">
        <img src="img/efecto.png" width="48" height="48">
    </figure>
</div>
```

Listing 4.13: Estructura de un botón con imagen pero sin utilidad

Cada una de las imágenes representa el efecto que se produce en su columna, explicado en el primer punto de esta plantilla. Y la vista que genera es:

Figura 4.12: Tercera zona de la plantilla principal del Instrumento

Al final el resultado es:

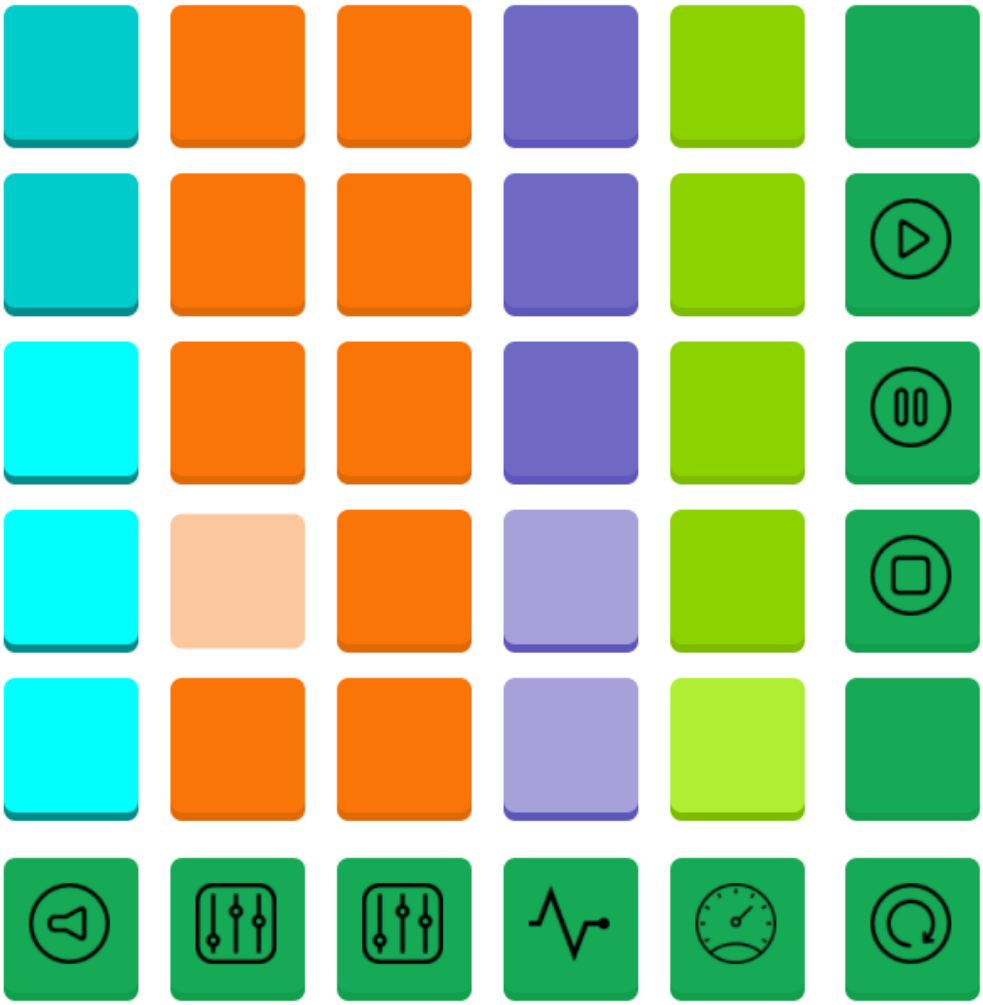

Figura 4.13: Plantilla principal del Instrumento

4.2 CSS

Antes de empezar a explicar el contenido de las clases, mencionadas en el apartado HTML5, es necesario conocer que método se ha escogido para implementar las propiedades CSS.

Como ya se explicó en la sección CSS del apartado LENGUAJES, existen diversas formas de cargar el contenido. En este caso, se ha escogido el método más práctico, que conlleva la separación en un archivo externo, para mejorar la programación.

El archivo ha sido almacenado en una carpeta denominada CSS. El nombre que se ha utilizado es 'launchpad.css', en él se contendrán todas las clases a las que hagamos referencia en el código HTML5.

Para cargar este archivo externo simplemente se debe definir entre las etiquetas <head> de nuestro prototipo:

```
<head>
<!-- Sentencia para cargar el estilo -->
<link rel=StyleSheet href="css/launchpad.css" type="text/css">
</head>
```

Listing 4.14: Cabecera de llamada al archivo CSS

En el link se incluyen:

- El tipo de texto que se encuentra en su interior, definido en lenguaje CSS;
- y la relación que guarda este archivo con el HTML donde se lee. Para este caso se ha introducido 'StyleSheet', ya que se utiliza como estilo principal.

Si se deseara crear un archivo añadido, para no interferir en este diseño, la relación debería ser 'Alternate StyleSheet', que como su nombre indica será alternativo al estilo principal.

Una vez cargado el contenido, se puede comenzar a detallar el estilo creado. La hoja de estilos que se ha creado se divide en 4 secciones importantes, las cuales se irán explicando una a una.

4.2.1. Estructura

En esta sección de encuentran las propiedades que definen la estructura general de nuestro Launchpad, como son los tamaños de filas y columnas.

Inicialmente, se encuentra la clase *container*:

```
.container{
    width: 600px;
    height: 600px;
    display: block;
}
```

Listing 4.15: Clase CSS que define el contenedor principal

En ella se define la propiedad *display* con el valor *block*, para especificar que se trata de un bloque independiente en sí mismo. El tamaño de este bloque se define con las propiedades de ancho y alto, *width* y *height*, respectivamente, que contienen un valor de 600px para formar una matriz cuadrada.

Una vez especificado cómo será la estructura que contendrá las filas y columnas de la interfaz, se pueden desarrollar el resto de clases.

- Para conocer mejor la forma del prototipo se van a desarrollar primero las propiedades que definen las filas (*rows*):

```
.row{
    height: 100px;
}
.row-5{
    height: 500px;
}
.row-1{
    height: 100px;
    margin-top: 7.5px!important;
    display: inline-block;
}
```

Listing 4.16: Clases CSS para la forma de las filas

Como se puede observar en este código una fila será simplemente la definición de una altura.

Sabiendo que el contenedor medirá 600px de alto y teniendo en cuenta que queremos definir 6 filas, la altura de cada una será de 100px. Por ello, si definimos una sola fila (*row*) tendrá la propiedad *height* con el valor 100px. Mientras que para la definición de 5 filas (*row-5*) el valor de esa propiedad será 500px, multiplicando por 5 la altura de cada fila.

Como la fila inferior del launchpad va añadida en última instancia se han realizado unos ajustes por medio de su margen superior y su relación con respecto al resto de filas. Todos estos cambios están definidos por medio de la clase *row-1*

- Por otro lado, para la estructuración de columnas se ha dividido en la necesidad de cada fila.

```
.col-6{
    width: 600px;
}
.col-5{
    width: 500px;
    float: left;
}
.col-1{
    width: 100px;
    float: right;
}
```

Listing 4.17: Clases CSS para la forma de las columnas

Al igual que en las filas, según el número de columnas, variará la propiedad principal de tamaño que las define. En este caso, se trata del ancho, por lo que *width* va cambiando de valor según la clase en la que se encuentre.

Al ser una matriz cuadrada su contenedor para el caso de desear que la fila contenga 6 columnas se definirá el ancho total de la plantilla: 600px. En cambio, si se desea obtener el espacio para 5 celdas su valor será de 500px. Esto sucede porque el valor de una sóla columna es de 100px, como se observa en la clase *col-1*.

- Por último, para definir cada celda se tendrá en cuenta el tamaño que permita tener una separación entre ellas y poder definir un sombreado que la realce.

  La clase que permite esto es:

```
.celda{
    width: 80px;
    display: inline-block;
    height: 80px;
}
```

Listing 4.18: Clases CSS para la forma de las celdas

Teniendo un valor de 80px, 10px menos por cada lado del cuadrado, lo que cumple los requisitos propuestos.

4.2.2. Espaciado y ajustes

Para aquellas filas o columnas que se han desajustado por estar en estructuras separadas se han preparado dos clases que lo solucionan:

```
.spaced{
    margin-left: 13px!important;
}
.colocate{
    margin-left: -7.5px;
}
```

Listing 4.19: Clases CSS para ajustar la plantilla

Mediante estos márgenes se ajustan las celdas, filas y columnas al 100 % del tamaño de la plataforma.

Pero para la separación entre celdas también es necesario contar con una clase que lo realice. Además de añadir la forma del botón que tendrá bordes redondeados. Se trata de *style-button*:

```
.style-button{
    margin: 7.5px;
    border-radius: 7px;
}
```

Listing 4.20: Clase CSS para dar forma a los botones

4.2.3. Figuras

Cuando las estructuras ya están formadas y bien definidas, es el momento de incluir las imágenes. Para ello, se deben crear las siguientes clases para adaptar el contenido de las figuras a las celdas:

```
.style-figure{
    padding: 15px;
    margin: 0;
}
.style-figure:hover{
    cursor: pointer;
}
.spaced-first .style-figure{
    float: right;
}
```

Listing 4.21: Clases CSS para colocar las imágenes en los botones

Para la figura se ha configurado una propiedad *padding* de valor 15px. Se encarga de añadir, a cada figura que contenga esta clase, un relleno de 15px y así centrar la imagen en el botón.

El selector *:hover* añadido es, simplemente, para establecer que el cursor sea un puntero y así hacer más intuitivo cuando se trata de un botón con funcionalidad y cuando no.

Por último, en el caso de que un botón no se coloque como es debido, como pasa en el caso del botón inferior derecho, se ha definido la clase *spaced-first* que añadida a *style-figure* ubica de forma flotante su posición.

Para las estructuras que necesitan la imagen para adaptarse, pero no necesitan mostrarla, se utiliza la siguiente clase:

```
.hide-figure{
    visibility: hidden;
}
```

Listing 4.22: Clase CSS para ocultar el icono cuando no hay imagen

Utiliza la propiedad *visibility* con el valor *hidden*, para simplemente ocultar el contenido y no eliminarlo como haría *display: none;*.

Con esta propiedad también oculta las propiedades definidas en el *:hover*, así que se mostrará el cursor por defecto.

4.2.4. Colores

A la hora de definir el color y su relieve se utilizan clases que se iran añadiendo a las celdas correspondientes.

Además, existen colores que cambiarán dependiendo de que celda esté pulsada.

Esta sección de divide en dos partes esenciales:

- Los colores estáticos son aquellos que se definen en las celdas que no variarán siendo pulsadas, siempre tendrán el mismo tono.

```
.violet{
    background-color: #9e54bd;
    box-shadow: 0 5px #823aa0;
}
.green{
    background-color: #17aa56;
    box-shadow: 0 5px #119e4d;
}
.orange {
    background-color: #f97508;
    box-shadow: 0 5px #e06907;
}
```

Listing 4.23: Clases CSS para dar color a los botones

  Las dos propiedades básicas que dan estilo a la celda son *background-color* y *box-shadow*, rellenando de color y dándole relieve, respectivamente.

- Por otro lado, si se trata de los botones de efectos los colores variarán si están pulsados o no. Para ello, se definen las clases 'color'-light, donde 'color' será sustituido por el color correspondiente.

```
.orange-light {
    background-color: #fcc79c;
    box-shadow: none;
    transform: translateY(2.5px);
}
.green-lighter{
    background-color: #8bd200;
    box-shadow: 0 5px #7cbb00;
}
.green-lighter-light {
    background-color: #afee32;
}
.cian{
    background-color: #00cccc;
    box-shadow: 0 5px #008B8B;
}
.cian-light{
    background-color: #00ffff;
}
.blue {
    background-color: #7069c3;
    box-shadow: 0 5px #5e56bb;
}
.blue-light {
    background-color: #a6a1d9;
}
```

Listing 4.24: Clases CSS para dar color a los botones con efectos

Y como se puede observar, solo cambia el color de fondo a unos tonos más claros.

4.3 JavaScript

Ahora se van a explicar las funciones que permiten el funcionamiento visual de la interfaz, aunque la mayoría también se utilizarán para la funcionalidad sonora, explicada en el capítulo CSOUND.

Este apartado se centra en todo lo que ha sido necesario para dar funcionalidad al prototipo visual. Por lo tanto, se definirán las funciones en orden lógico.

4.3.1. Función inicial del instrumento

Esta función, como se pudo observar en el apartado HTML5, se llama cada vez que se pulsa un instrumento de la plantilla inicial.

Según el instrumento que se pulse, 'elementSlug' tendrá un valor u otro, el que se le especifique en la llamada.

```
function instrument(elementSlug) {
    // Funcionalidad desarrollada
}
```

Listing 4.25: Función JS al pulsar un instrumento

Dentro de la función se ejecutarán una serie de sentencias cada vez que se haga clic en un botón de instrumento.

4.3.1.1. Sentencias básicas

```
document.getElementById("first-part").style.display = 'none';
```

Listing 4.26: Sentencia JS para ocultar la primera plantilla

En esta sentencia se oculta la plantilla inicial que se estaba observando antes de clicar. Para ello, se ha seguido el siguiente proceso:

- Primero se debe conocer la palabra 'document', ya que con ella se puede llamar a cualquier función de JS que exista.
- Para detectar elementos HTML existen diversas funciones, las cuales se tratarán durante la explicación del código. En este caso, necesitamos reconocer la sección principal mediante su identificador. Por ello, se utilizará la función: getElementById(), introduciendo el id definido 'first-part'.
- Al haber obtenido el objeto se le aplica el estilo 'display' con valor 'none' para que se oculte.

```
document.querySelector(".instrument").classList.add("instrument-"+
    elementSlug);
```

Listing 4.27: Sentencia JS para añadir una clase con el instrumento actual a la segunda plantilla

Como la plantilla que se cargará posteriormente, la relativa al instrumento, no contiene un identificador del mismo, esta sentencia se encarga de definirlo. Para ello:

- Desde 'document' se nombrará la función que selecciona cualquier elemento usando clases, identificadores, etiquetas, ... misma nomenclatura que en archivo CSS.

- Una vez seleccionada la segunda sección se accede a su listado de clases mediante 'classList' y se añade una nueva con la palabra 'instrument', seguida de un guión y el 'slug' escogido para el instrumento actual.

```
1 document.getElementById("replay").classList.add(elementSlug);
```

Listing 4.28: Sentencia JS para añadir el instrumento a las clases del botón de retroceder

En esta sentencia se obtiene el botón de retroceder y, al igual que en el caso anterior, se accede a su lista de clases para añadir una nueva. En este caso será el identificador del instrumento en si mismo.

```
1 document.querySelector(".instrument-"+elementSlug).style.display = 'block
    ';
```

Listing 4.29: Sentencia JS para mostrar la segunda plantilla

Por último, se ejecutará la sentencia que permite mostrar la segunda plantilla. Se accederá a ella mediante el selector clase que contiene el nombre del instrumento, para asegurarse de que los cambios se han almacenado.

4.3.1.2. Sentencias de carga de valores

```
1 addControl(addEfecto);
2 removeControl(removeEfecto);
```

Listing 4.30: Sentencias JS para restaurar los colores de efecto en los botones pulsados para el instrumento actual

Para que, al entrar en un instrumento, la representación de la plantilla corresponda con lo seleccionado anteriormente, se leerán los vectores que se explicarán en el siguiente punto.

Las llamadas a las funciones addControl y removeControl permiten que se pinten los efectos, o se borren, según el instrumento en el que se encuentren. La palabra Efecto será sustituida por Volumen, Panoramica y Reverberacion, realizando tres veces cada llamada.

4.3.2. Controles básicos

Las funciones que llevan a cabo los efectos de cada instrumento siguen la misma estructura. Por ello, se va a desarrollar una de ellas sabiendo que la palabra 'Volumen' deberá sustituirse por Panorámica y Reverberación para los otros dos casos.

```
var addVolumen = new Array();
var removeVolumen = new Array();
```

Listing 4.31: Creación de variables JS para almacenar el estado del volumen del instrumento

Inicialmente, contiene la definición de dos nuevos vectores vacíos globales. Estos Arrays almacenarán de forma asociativa el estado de cada instrumento, relacionándolos mediante su identificador (slug). Además, también contendrán acceso al color de las celdas del efecto en el que se encuentren almacenando sus clases.

La función que hace posible la detección del nivel en el que el usuario ha clicado y, así, poder modificar el estilo de la plantilla en cada instrumento, es la que se muestra a continuación:

```
function volumen(id){

    var addVolumenInstrument = new Array();
    var removeVolumenInstrument = new Array();

    Slug = document.getElementById("replay").classList;

    var indice = 0;
    for(contador=id+1;contador<=5;contador++){
        removeVolumenInstrument[indice]= [contador,'.cian','cian-light'];
        indice++;
    }
    removeVolumen[Slug] = removeVolumenInstrument;
    removeControl(removeVolumen);

    indice = 0;
    for(contador=id;contador>=1;contador--){
        addVolumenInstrument[indice]= [contador,'.cian','cian-light'];
        indice++;
    }
    addVolumen[Slug] = addVolumenInstrument;
    addControl(addVolumen);
}
```

Listing 4.32: Función JS para el tratamiento del volumen en la segunda plantilla

Para entender su funcionamiento se explicará línea por línea a continuación.

- Lo primero que sucede al hacer clic en un botón que contenga la función descrita, es que obtiene el nivel en el que se encuentra.

- Después, define e inicializa dos nuevos Arrays vacíos, pero esta vez no serán globales, sino que solo podrán ser utilizados en el interior de esta función.

```
1 var addVolumenInstrument = new Array();
2 var removeVolumenInstrument = new Array();
```

Listing 4.33: Arrays JS para almacenar estados temporales de volumen

- A continuación, consigue el identificador del instrumento leyendo la clase del botón de volver. Este será utilizado para asociar los datos que se almacenen en los Arrays internos en los equivalentes globales.

- Después de definir las variables necesarias, se desarrollan dos estructuras similares para obtener qué celdas estarán marcadas, de la primera a la pulsada, y cuáles no, de la siguiente hasta el final.

  Aquellas que se consideren pulsadas se almacenarán en addVolumenInstrument y el resto en removeVolumenInstrument.

  Para realizar esta separación se realizan dos bucles (for):

  - **for** que estudia las celdas que no serán modificadas

    El contador de este for se inicializa con el valor del nivel más uno (id+1). En cada iteración sumará uno (++) hasta que llegue al valor máximo, que es el 5 que se observa en la condición.

    En el interior del bucle se almacenarán los niveles de los botones que no deberán representarse como pulsados. Además de las clases que se eliminarán más adelante.

    Para asegurar de qué forma se almacenan los datos, en el vector interno, se ha creado un índice que comenzará en cero y finalizará con el número de vueltas ejecutado. Este se utiliza tanto para este bucle como para el siguiente.

  - **for** que estudia las celdas que serán modificadas

    Este contador a diferencia del anterior se inicializa en el nivel actual (id) y en cada iteración se restará uno (–) hasta que llegue al valor mínimo, 1.

    En el interior del bucle sucede lo mismo que en el caso anterior, pero almacenando los botones que si se mostrarán como pulsados. Las clases esta vez se añadirán al elemento.

Después de cada **for** se almacenarán en la posición del instrumento del array global, mediante su slug, los valores obtenidos en estos bucles. Por último, se llama a las dos funciones que ya se mencionaban al inicializar la aplicación: addControl y removeControl, para ejecutar las sentencias que en ellas se incluyen. El resultado de estas operaciones modifica el diseño de los botones con la nueva clase que define el color.

4.3.3. Función de almacenamiento de filtros

Para el filtrado de la señal de audio se ha desarrollado una función encargada de controlar qué filtro y de qué instrumento se encuentra activo en cada momento, cuando ha sido pulsado por el usuario. La estructura que permite almacenar los filtros activos se muestra a continuación:

```
1  var addFilters = new Array();
2
3  function filtros(filter){
4      Slug = document.getElementById("replay").classList;
5
6      if(!addFilters[Slug])
7          addFilters[Slug] = new Array;
8
9      if(!addFilters[Slug][filter]){
10         addFilters[Slug][filter] = filter;
11         addFiltersBySlug(addFilters);
12         return;
13     }
14     addFilters[Slug][filter] = false;
15     addFiltersBySlug(addFilters);
16 }
```

Listing 4.34: Función JS para almacenar los filtros pulsados

Lo primero que se ha necesitado es crear un *array* que se utilizará para almacenar en cada instrumento los filtros activos.

Después, en el interior de la función, se extrae el *slug* del instrumento en el que ha sido pulsado el filtro. A continuación, se comprueba que el *array* en la posición del instrumento actual esté vacío. Si está vacío se encarga de crear un nuevo *array* en esa posición.

Si se sigue avanzado se observa otro condicional, pero esta vez la condición comprueba que el filtro pulsado en ese instrumento no haya sido pulsado anteriormente. Si la condición se cumple almacenará el nombre del filtro y llamará a la función que se encarga de darle estilo a un botón pulsado. Si por el contrario ya ha sido pulsado,

se pondrá a *false* su valor y se volverá a llamar a la función para que actualice el estilo.

En la siguiente subsección se tratará, entre otras, la función que aplica el estilo a los botones de filtrado.

4.3.4. Funciones para Añadir o Eliminar clases

Una vez se han descrito las funciones y Arrays necesarios para cargar estas funciones, se puede comenzar con su análisis.

```
1  /** Aderir **/
2  function addControl(controlIntrumentArray){
3      Slug = document.getElementById("replay").classList;
4      var controls = new Array();
5      controls = controlIntrumentArray[Slug];
6      if(controls){
7          for(var item = 0 ; item < controls.length ; item++){
8              father = document.getElementById(controls[item][0]);
9              father.querySelector(controls[item][1]).classList.add(
                   controls[item][2]);
10         }
11     }
12 }
13 /** Eliminar **/
14 function removeControl(controlIntrumentArray){
15     Slug = document.getElementById("replay").classList;
16     var controls = new Array();
17     controls = controlIntrumentArray[Slug];
18     if(controls){
19         for(var item = 0 ; item < controls.length ; item++) {
20             father = document.getElementById(controls[item][0]);
21             father.querySelector(controls[item][1]).classList.remove(
                   controls[item][2]);
22         }
23     }
24 }
```

Listing 4.35: Funciones JS para añadir o eliminar estilos a la segunda plantilla

Ya que ambas funciones tienen la misma estructura, modificando simplemente el nombre y la adición o eliminación de una clase, se definirá tan solo una de ellas.

Cuando se ejecuta esta función realiza las siguientes operaciones:

- Inicialmente, se extrae el slug relativo al instrumento y se crea un array vacío para almacenar los datos que se introducen para ese instrumento.

- Si existen datos que ejecutar se creará un **for** que recorra los valores del vector y añadirá, o eliminará, a los botones del nivel correspondiente la clase que contiene un tono más claro.

Para el cambio de estilo al pulsar una tecla de filtrado el proceso varía un poco de la forma en qué se tratan el resto de controles.

```
1  // Aderir el estilo a un filtro
2  function addFiltersBySlug(slugFilters){
3      Slug = document.getElementById("replay").classList;
4
5      for (var propiedad in slugFilters[Slug]) {
6          if (slugFilters[Slug].hasOwnProperty(propiedad)) {
7              if(slugFilters[Slug][propiedad]!=false){
8                  document.getElementById(propiedad).classList.add('orange-
                       light');
9              }else{
10                 document.getElementById(propiedad).classList.remove('
                       orange-light');
11             }
12         }
13     }
14 }
```

Listing 4.36: Función JS para añadir filtros a la segunda plantilla

En este caso, la función recorre el *array* de filtros del instrumento y comprueba que el filtro, al que hace referencia propiedad, exista. Si este existe como posición en el *array* asociativo se leerá el condicional que comprueba que esté activo o no el filtro para añadir o quitar la clase que corresponda.

4.3.5. Eliminar todo

A la hora de volver a la pantalla principal, como se observa en VOLVER, se necesita eliminar todo estilo pintado con las funciones AÑADIR/ELIMINAR.

Para este propósito se ha creado una función llamada removeAllControls:

```
/** Eliminar todo **/
function removeAllControls(){
    for(var id = 1 ; id <= 5 ; id++) {
        father = document.getElementById(id);
        var volumen = father.querySelector('.cian-light');
        var panoramica = father.querySelector('.blue-light');
        var reverberacion = father.querySelector('.green-lighter-light');
        var filtros = father.querySelector('.orange-light');
        if(volumen){
            volumen.classList.remove('cian-light');
        }
        if(panoramica){
            panoramica.classList.remove('blue-light');
        }
        if(reverberacion){
            reverberacion.classList.remove('green-lighter-light');
        }
        if(filtros){
            filtros.classList.remove('orange-light');
        }
    }
}
```

Listing 4.37: Función JS para eliminar todo estilo añadido a la segunda plantilla

Inicialmente, se encarga de recorrer los 5 niveles que existen llamando a los identificadores de cada fila.

Después, almacena en tres variables los objetos que devuelve cada nivel, de clase 'cian-light', 'blue-light' y 'green-lighter-light', respectivamente.

Por último, cuando existe el objeto volumen, panorámica o reverberación con las clases mencionadas se eliminan. Esto se realiza para darle el color inicial a la plantilla y así inicializarla con los valores del instrumento que se toque.

4.3.6. Controles de instrumento

Cada instrumento contiene como botones principales un Play, Pause y Stop, que visualmente no varían, pero que al ser pulsados llamarán a estas funciones:

```
function insPlay(){
    Slug = document.getElementById("replay").classList;
    alert('Play');
}
function insPause(){
    Slug = document.getElementById("replay").classList;
    alert('Pause');
}
function insStop(){
    Slug = document.getElementById("replay").classList;
    alert('Stop');
}
```

Listing 4.38: Funciones JS para controlar las acciones básicas

En la interfaz visual se muestra un alert para demostrar su funcionamiento, mientras que cuando se programe el control sonoro realizará esa acción.

4.3.7. Volver

La última función que compone este Launchpad es aquella que permite retroceder de un instrumento a la pantalla principal para seguir escogiendo otros instrumentos y mezclando, hasta conseguir una base instrumental.

```
function insReturn(){
    Slug = document.getElementById("replay").classList;
    removeAllControls();
    document.getElementById("first-part").style.display = 'block';
    document.querySelector(".instrument-"+Slug).style.display = 'none';
    document.querySelector(".instrument").classList.remove("instrument-"+
        Slug);
    document.getElementById("replay").classList.remove(Slug);
}
```

Listing 4.39: Función JS para volver a la primera plantilla

Para esto se ha creado insReturn(), que extrae del botón que está siendo pulsado, id='replay', el *slug* del instrumento. Después, llama a la función que se encarga de eliminar cualquier estilo añadido en la plantilla del instrumento.

Y, por último, para volver a la plantilla inicial se muestra la sección 'first-part', se oculta la otra sección, la que contiene al instrumento, y se elimina la clase que hace

referencia al mismo: “instrument+slug”. Además, también elimina la clase, equivalente al *slug* del instrumento, que contiene el botón de *return*.

CAPÍTULO 5

Conexión - Front-end (HTML5, CSS y JS) con Back-end (Csound6 y JS)

En este capítulo, se tratará la forma en que la programación web y sonora han sido unidas. Para ello, se hablará de CsoundQt.

El compilador de Csound tiene la capacidad de usar HTML para definir las siguientes funcionalidades:

- La interfaz de usuario, facilitando su diseño.
- Controlar Csound y generar Csound scores y orquestas.

Dentro de esta plataforma el código HTML está embebido en el elemento <html> del archivo .csd (*Csound Structured Data*). Mediante este elemento se define una página web que contiene, además, lenguaje Csound.

CsoundQt, también lee el resto de estándares de la web incluyendo CSS y JS, definidos y mantenidos por el comercio generado de WHATWG (*Web Hypertext Application Technology Working Group*) el cual mueve a W3C.

La finalidad principal de esta conexión es conseguir llevar el control de audio al mundo de la web, y viceversa.

Como ya se ha descrito en el apartado de HTML5 (sección 2.1), el rendimiento y la consistencia que ofrece la 5ª versión de este es muy superior a las anteriores. Y, gracias a este compilador, las prestaciones que ofrece este lenguaje se pueden utilizar en Csound.

5.1 Instalación y unión entre lenguajes

5.1.1. Instalación previa a la composición de códigos

Según el sistema operativo que se utilice, el procedimiento de instalación puede variar.

Para usuarios que utilicen un SO Windows, sólo tienen que descargar el instalador de Csound que se encuentra en SourceForge y ejecutarlo para cargar todos los archivos en el ordenador. En este paquete ya va contenido CsoundQt y la lectura de HTML que, además, está activado por defecto en el compilador.

El resto de SO tienen sus propias versiones descargables desde los enlaces que proporciona Csound en `http://csound.github.io/download.html`.

En estos instaladores suele venir incluido un paquete de ejemplos y manuales que facilitan el uso y las consultas que se realicen a través del compilador. Si por algún problema estos paquetes no se instalan correctamente, en Github en el apartado de Csound (`https://github.com/csound/csound`), se pueden observar todos los ejemplos de todas las versiones disponibles.

5.1.2. Uso de la conexión entre lenguajes en CsoundQt

Para conseguir plasmar cómo se consigue la conexión entre HTML5, CSS y JS con Csound, se va a tratar un ejemplo en particular que el mismo programa proporciona.

En este código, se observarán parte de las cosas que se pueden hacer con HTML en Csound. Además, se podrá ver el resultado visual a la izquierda del código.

En la figura superior el código HTML realiza las acciones que se explican a continuación:

1. **Definición de widgets**

   Para este ejemplo se propone la creación de *widgets* con forma de *slider* para controlar Csound.

   Para conseguir conectar HTML con Csound se cuenta con el API de canal, usando el HTML como elemento de entrada y captando los eventos mediante JS[1].

2. **Estilo visual**

   Para dar estilo a los *sliders*, y a la tabla que contiene a estos, se utiliza el lenguaje de diseño CSS.

[1] La captación de eventos sonoros por JS se desarrolla en el apartado siguiente.

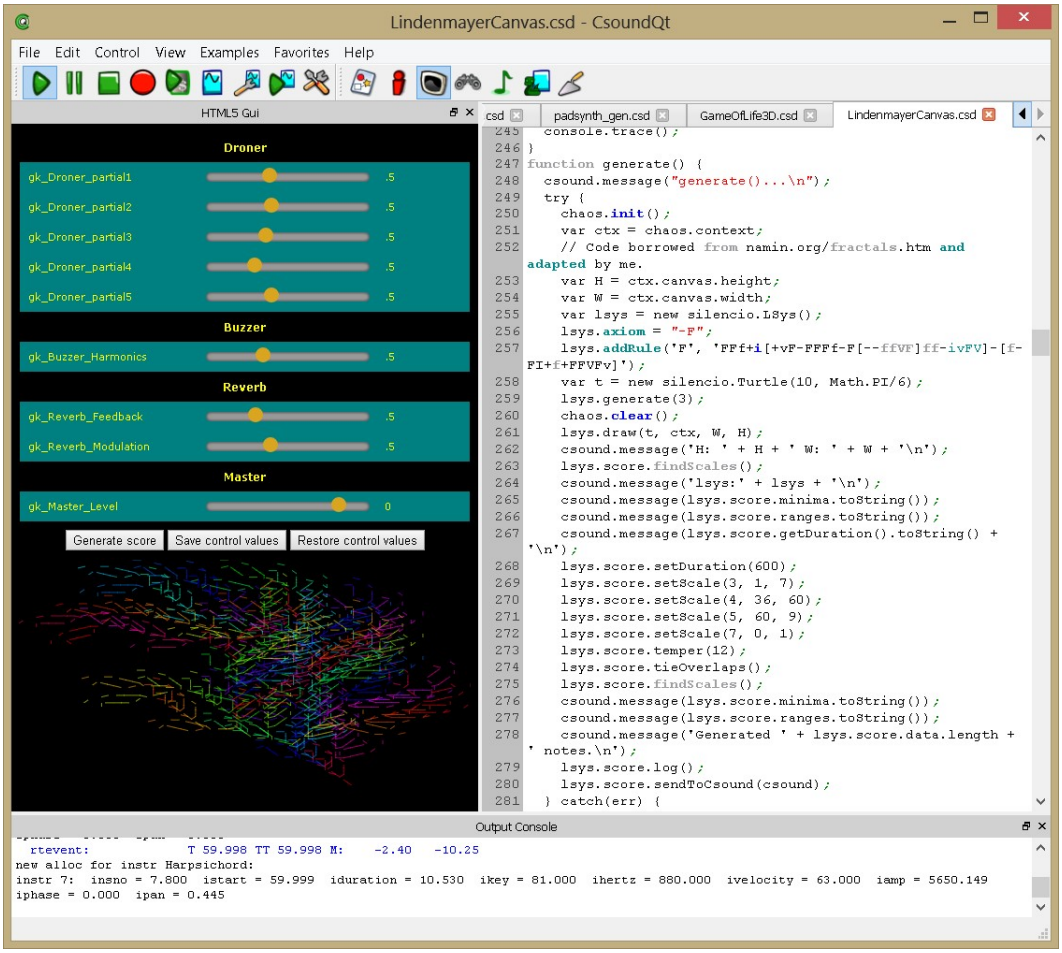

Figura 5.1: Ejemplo de un código HTML insertado en Csound

3. **Almacena la posición de los elementos HTML**

   Guarda la posición de los *widgets* cuando se pulsa el botón 'Guardar los valores de control'.

4. **Vuelve a un estado inicial los elementos**

   Se encarga de devolver a su posición inicial a los sliders cuando se pulsa el botón 'Restaurar los valores de control', usando la característica que tiene el lenguaje HTML5 para el almacenamiento local.

5. **Utilidad de JavaScript**

   Usa JS para definir un sistema Lindenmayer y generar, tanto la imagen del árbol dibujada en HTML, como un score de Csound que representa el árbol cuando se pulsa 'Generar score'.

Este ejemplo, junto con otros, se encuentra en el compilador si se realiza la instalación explicada en el punto anterior.

Csound escogió los lenguajes de programación Web porque reúnen las funcionalidades que se pretendían encontrar, uniendo usabilidad y agilidad con rapidez de

procesado. Estos lenguajes desarrollan, en una medida menor, varias características que otros tratan en profundidad.

Un ejemplo de ello, es que C o C++ son lenguajes más rápidos de compilar que JS, o que el almacenamiento de HTML5 no contiene las mismas prestaciones que las bases de datos de MySQL. Pero ninguno de los ejemplos mencionados contiene esas funcionalidades conjuntas, cosa que CSS, HTML5 y JS, sí.

Por lo tanto, HTML hace un gran front end para Csound. Además, teniendo en cuenta que el mundo Web está constantemente renovándose, los estándares que se utilizan son mantenidos de forma estable por la comunidad de desarrolladores que lo conservan. Y, también, tanto Csound como HTML son compatibles hacia atrás (*backwardly*).

5.1.3. Inclusión del código HTML, CSS y JS en CsoundQt

CsoundQt cuenta con un navegador Web embebido llamado Chromium Embedded Framework.

El proceso que sigue el compilador para leer el código HTML introducido es el siguiente:

- En primer lugar, analiza el elemento <html> del archivo .csd donde se incluye.
- Después, lo guarda como un archivo HTML aparte, equivalente a una página web.
- Y, por último, el navegador embebido carga la página resultante, la compila y ejecuta cualquier tipo de script que contenga, la muestra, y permite que el usuario interactué con la interfaz resultante.

Para ejemplificar el método que sigue, si se programara en un archivo .csd llamado 'MyCsoundPiece' el archivo resultante de este análisis sería MyCsoundPiece.csd.html. Esta página resultante será vista en el navegador incluido.

La programación que se realiza en el interior de estas etiquetas es equivalente al 100 % con el HTML que se utiliza en la Web 3.0 actualmente.

A continuación se muestra el código Csound que contendrá el HTML, explicado en el capítulo 4, el cual se pasará de PhpStorm a CsoundQt:

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>
</CsInstruments>
<html>
<!-- Entre estas etiquetas se coloca el HTML y se incluyen las llamadas a
    los archivos CSS y JS-->
</html>
<CsScore>
</CsScore>
</CsoundSynthesizer>
```

Listing 5.1: Código de los lenguaje estándares de la web incluidos en CsoundQt

Como se observa, para incluir el código se ha colocado entre el fin de la orquesta, delimitada por <CsInstruments> y </CsInstruments>, y el inicio de la partitura, entre <CsScore> y </CsScore>.

Una vez añadidas las etiquetas <html> en esta ubicación de la estructura Csound se puede representar cualquier estructura válida del lenguaje HTML.

5.2 Funciones JS en Csound

Mediante JS los lenguajes estándar de la Web mantienen una relación directa. Es decir, gracias a JS, HTML puede controlar eventos y darle vida a su código. Además, JS contiene una funcionalidad que facilita la creación de documentos HTML creando objetos DOM (*Document Object Model*).

Como ya se ha descrito en la sección 2.3, JavaScript es el ingenio y lenguaje mayoritario en las 3W. Y, más en concreto, es el código que ejecutan los navegadores.

JS es un lenguaje de programación funcional, ya que trabaja con métodos constantemente, no tan diferente del Scheme, lenguaje funcional académico. Por otro lado, JS no sólo crea funciones si no que también permite la definición de clases mediante prototipos.

En este proyecto, la utilidad principal del JS está relacionada con las funciones. Realizando la conexión HTML y Csound por medio de ellas.

Antes de tratar en profundidad las acciones llevadas a cabo en Csound desde JS, es necesario conocer las funciones que proporciona este lenguaje. Estos métodos vienen contenidos en un objeto 'csound' JS para conectar la creación y reproducción de instrumentos con la interfaz.

A continuación, se muestra una tabla con la nomenclatura utilizada y el valor resultante de cada llamada.

Métodos	**Definición**
getVersion	devuelve un número
compileOrc{orchestra_code} evalCode{orchestra_code}	devuelve el resultado numérico de la evaluación
readScore{score_lines}	
setControlChannel{channel_name, number}	
getControlChannel{channel_name}	devuelve un número representando el valor del canal
message{text}	
getSr	devuelve un número
getKsmps	devuelve un número
getNchnls	devuelve un número
isPlaying	devuelve 1 si Csound está en marcha, 0 si no

Tabla 5.1: Métodos JS de control Csound. Define y nombra los métodos que proporciona Csound para su uso en la Web.

CsoundQt se encarga de reenviar llamadas de JS desde la página web a funciones nativas del compilador que, a su vez, transformará en lenguaje Csound. Para llevar a cabo este reenvío, cuenta con un mecanismo que trabaja con código C++ que el usuario o programador no observarán.

El código JS que relaciona los lenguajes trabaja con IPC (*Inter-Process Communication*) asíncrona. Esto quiere decir, que nada más finalizar el envío al receptor, llegue o no, JS continúa su ejecución, trabajando en paralelo. Sucede porque el framework embebido se bifurca en varios hilos o procesos para implementar el navegador Web, y de esta manera agilizar la ejecución obteniendo menor tiempo de carga.

HTML y JS pueden llevar casi al completo las funcionalidades que proporcionan el resto de Front ends. Pero siempre existe algún defecto y, en este caso, es que en comparación con el resto de front ends disponibles, este tiene mayor retardo. Esta latencia se observa también en cualquier caso basado en Chrome, por las comunicaciones entre procesos.

Además de este campo, los estándares Web experimentan crecimiento día a día, llegando a realizar otras tareas como videojuegos, 3D, aplicaciones, entre otras.

CAPÍTULO 6

Back-end - Csound6: Funcionalidad sonora

Una vez realizada la plantilla visual de este prototipo, se ha desarrollado la funcionalidad sonora que la da uso a la misma. Se han llevado a cabo varios algoritmos programados en Csound, unidos a las funciones creadas mediante JS. Esta plataforma está pensada para que en un futuro se pueda ampliar o modificar su contenido por personas especializadas en el lenguaje.

De esta manera, en el capitulo actual, se va a exponer el proceso y la estructura seguida en esta parte del proyecto. Antes de nada, se ha de diferenciar entre la parte de composición de instrumento (orc) y la de partitura (sco) del archivo .csd, como se explicaba anteriormente en la sección 2.4.5. Mientras que en la partitura, únicamente, se hará referencia al inicio y velocidad de lectura de cada instrumento, en el apartado de instrumentos se encuentra todo el desarrollo, tanto de efectos como de variables modificables por el usuario.

6.1 Estructura instrumental

Toda la estructura instrumental que contiene este prototipo está definida entre las etiquetas de <CsInstruments></CsInstruments>. A continuación, se define la estructura que seguirá el código, explicándose el contenido en profundidad en los siguientes apartados.

CsInstruments comienza declarando los parámetros iniciales que serán utilizados por todos los instrumentos declarados:

```
1 sr = 44100
2 ksmps = 20
3 nchnls = 2 ;dos canales - stereo
```

Listing 6.1: Parámetros iniciales

Las variables que se definen son:

- **sr**

  Con este parámetro se configura la frecuencia de muestro del audio creado. En este caso se utiliza la misma que para un CD: 44100.

- **ksmps**

  Define el número de muestras por periodo. Este parámetro por defecto vale 10, mientras que se le da el valor 20 para este ejecutable. Por ello, se reduce a la mitad el valor de kr, que se trata de la relación entre la frecuencia de muestreo y el número de muestras, equivalente al control de la velocidad.

- **nchnls**

  Establece el número de canales de salida de audio.

Después de la definición de estos parámetros se puede comenzar a concretar qué instrumentos y qué controles se reproducirán en este programa.

Por otro lado, para el funcionamiento de la relación entre JS y Csound se debe definir un instrumento que esté siempre leyéndose, sin necesidad de realizar una llamada por partitura a este. Para ello, se ha utilizado el opcode (*operation code*) que se muestra a continuación:

```
alwayson 'Controls'
```

Siendo *Controls* el instrumento que mantendrá la conexión entre las variables globales y el JS, que envía el valor según los elementos pulsados por el usuario.

Lo siguiente que se observa es la definición de los instrumentos que propone la aplicación en la pantalla inicial. Cada uno de ellos consta de tres fases:

- En la primera se inicializan las variables globales que tienen que ver con ese instrumento.

- A continuación, de la misma forma, se inicializan las variables que definen los efectos activos para ese instrumento.

- Por último, se declara el instrumento que estará definido entre '*instr*' y '*endin*'. En el siguiente algoritmo se puede observar un ejemplo básico de su funcionamiento:

```
instr 'nombre-instrumento'

; Carga del archivo
aL, aR soundin "sounds/'nombre-instrumento'.wav"

; Efectos sonoros aplicados a la entrada
aoutleft = 'efectos' * aL
aoutright = 'efectos' * aR

; Salida con todos los efectos aplicados
outs aoutleft, aoutright

endin
```

Listing 6.2: Declaración de instrumentos básica

Como se puede comprobar, una vez declarado el instrumento con el nombre escogido, ya que no es necesario utilizar números para nombrarlo, se comienza a desarrollar el código.

Primero realiza la carga del archivo de audio, seguido del tratamiento que se llevará a cabo según los valores escogidos por el usuario y la posterior reproducción del resultado obtenido.

Una vez realizadas todas las acciones, se cierra el instrumento quedando en su interior todo el tratamiento.

Por último, como ya se ha mencionado, para la interacción del usuario se necesita un instrumento que conecte con las funciones desarrolladas en JS y los valores de las variables globales definidas anteriormente. En el siguiente ejemplo se muestra la forma de extraer la información requerida desde Csound.

```
instr Controls

gk_'instrumento-variableControl'_ chnget "gk_'instrumento-variableControl'"
if gk_'instrumento-variableControl'_ != 0 then
gk_'instrumento-variableControl' = gk_'instrumento-variableControl'_
endif

endin
```

Listing 6.3: Declaración del instrumento de control

El opcode '*chnget*' es el que se encarga de recoger el valor que se envía desde JS mediante la función csound.setControlChannel('variable','valor');, siendo 'variable' el nombre que se coloca al lado de *chnget* entre dobles comillas.

Además, se define un condicional que controla que el valor recibido no sea nulo, para así darle valor a la variable en Csound.

6.2 Controles

Antes de nada, se debe conocer la activación de los instrumentos controlada mediante sentencias definidas en la partitura. Para ello, se ha utilizado la función JS definida para el *Play* del instrumento, la cual se expone a continuación:

```
/* - Variable global */
var instrControl = new Array;

function insPlay(){
    Slug = document.getElementById("replay").classList;

    /* Inicializaciones */
    if(!instrControlVol[Slug]){
        instrControlVol[Slug] = 0.2;
    }
    csound.setControlChannel('gk_'+Slug+'MasterLevel', instrControlVol[
        Slug]);
    if(!instrControl[Slug]){
        instrControl[Slug] = true;
    }else{
        return;
    }

    /* Activar instrumento */
    for(var count=0;count<120; count=count+0.5){
        note = 'i "'+Slug+'" '+count+' 0.5';
        csound.readScore(note);
    }
}
```

Listing 6.4: Función Play JS con funcionalidad

Cuando pulsamos un instrumento o el botón de Play se ejecuta la función que se observa. El proceso que sigue consta de dos pasos diferenciados:

- **Inicializaciones**

  Para inicializar los parámetros que el instrumento tuviera almacenados de una interacción anterior o dar valor por primera vez a estos, se utilizan las variables globales de volumen y de activación.

Inicialmente, se comprueba que la posición del array del instrumento en concreto esté vacía, obteniendo el nombre del instrumento mediante el *Slug*.

Una vez comprobado su valor se realizan dos acciones diferentes para ambos controles.

Por una parte, para el caso de inicialización del volumen, si la posición del array contiene *NULL* se almacenará el valor 0.2 y después se aplicará a la variable global de Csound. En caso contrario, si existe ya un valor definido se aplicará directamente. Para el paso de valor de esta variable se utiliza la función, del objeto *csound*, `setControlChannel('variable', 'valor');`.

Por otro lado, para controlar si el instrumento ya ha sido activado o no, se almacena el valor *true* en la primera interacción. Esto sucede cuando el valor en el array es *NULL*. En caso contrario, se ejecuta un *return* que le permite salir de la función sin realizar ninguna sentencia más, lo cual optimiza el código desarrollado.

- **Activación**

  No obstante, como indica el nombre de la función, el instrumento debe ser activado. Para ello, se ha programado un **for** que crea sentencias que son enviadas a la partitura de Csound y leídas mediante `readScore('instrumento y tempos');`.

En la segunda plantilla de la interfaz, como se explicaba en la sección 4.1.2, existen tres columnas que varían el nivel de un efecto.

En este apartado se pretende describir el proceso que siguen esos tres efectos con respecto a la señal de entrada, así como la conexión que ha sido necesaria con los otros efectos y JS.

6.2.1. Control de variables comunes

En este apartado se pretende conocer la forma en que el sistema genérico, no en cada instrumento, activa o para todos aquellos instrumentos que hayan sido activados hasta el momento.

En el código siguiente se muestran las tres funciones JS que conectan con la variable máster, que ha sido creada en Csound para el control del volumen, y controlan su valor para los casos de activación o parada de todos los instrumentos.

```
1  // Funciones generales
2  function commonPlay() {
3      for (var slug in instrControl) {
4          if (instrControl.hasOwnProperty(slug)) {
5              csound.setControlChannel('gk_'+slug+'MasterLevel',
                   instrControlVol[slug]);
6          }
7      }
8  }
9  function commonPause() {
10     for (var slug in instrControl) {
11         if (instrControl.hasOwnProperty(slug)) {
12             csound.setControlChannel('gk_'+slug+'MasterLevel', 0.00001);
13         }
14     }
15 }
16 function commonStop() {
17     for (var slug in instrControl) {
18         if (instrControl.hasOwnProperty(slug)) {
19             csound.setControlChannel('gk_'+slug+'MasterLevel', 0.00001);
20         }
21     }
22 }
```

Listing 6.5: Función Play JS con funcionalidad

Para realizar el cambio de variable para los casos de instrumentos activos se ha utilizado un tipo de **for** especializado. Colocando una variable recién creada, *slug*, delante del operador **in** que a su vez va seguido del array asociativo instrControl, se consigue recorrer el array asociativo. Mientras se recorre el array *slug* almacena el nombre del *ítem* en el que se encuentra y colocando el nombre en el array de la forma *instrControl[slug]* se obtiene el valor en esa posición.

Cada iteración del **for** se comprueba que el slug obtenido exista antes de realizar cualquier acción. Una vez se sabe que esta posición existe se envía el valor correspondiente a la variable global dependiendo del caso del que se trate. Para la activación se reutiliza el valor ya existente y para la parada se pone a 0.

6.2.2. Volumen

Inicialmente, se desarrolló el control del nivel de un instrumento en concreto, de esta manera cada uno tiene su propio volumen asociado y no uno genérico para todos

Para poder describir de manera exacta el funcionamiento de este control básico, en Csound, se va utilizar un ejemplo con el instrumento '*bell*' (campana).

```
; Valor inicial variable global de volumen
gk_bellMasterLevel init 0.00001

; Instrumento: campana
instr bell

; Carga del archivo
aL, aR soundin "sounds/bell.wav"

; Aplicar el nivel del master (volumen)
aoutleft = gk_bellMasterLevel * aL
aoutright = gk_bellMasterLevel * aR

; Salida con el volumen aplicado
outs aoutleft, aoutright
endin
```

Listing 6.6: Control del volumen en un instrumento

En este código lo primero que se observa es la declaración de una variable global modificable llamada *gk_bellMasterLevel*, donde **g** representa que es de tipo global y **k** que será dinámica. Su valor inicial es 0.00001 y se ha escogido *init* para su inicialización, puesto que permite igualar una variable de tipo estática (**i**) con otra de cualquier tipo.

Después de la inicialización, comienza la declaración del instrumento que contendrá su funcionalidad entre las etiquetas *instr bell* y *endin*. Una vez la partitura ha activado el instrumento se realizan las siguientes acciones:

- **Lectura de las muestras del archivo de audio**

  Los sonidos de los instrumentos desarrollados son archivos de tipo .wav o .aiff, por lo que para su modificación o reproducción se debe leer su contenido por muestras.

  El *opcode* que se encarga de realizar esta acción es **soundin**.

  Este operador pide como entradas un archivo de audio y parámetros relacionados con su lectura, pero en este caso solo será necesario un archivo.

  Los valores que devuelve son un array de muestras sonoras para un caso mono o dos para el *stereo*.

  Una solución que reúne las salidas en ambos casos es poner dos variables de salida, puesto que para el caso mono serán la misma repetida y para el *stereo* diferentes. De esta forma, se tratará cualquier señal como *stereo*, pudiendo llegar a jugar con este efecto en el apartado 6.2.3.

- **Aplicación del nivel del Master**

  Las variables obtenidas del operador **soundin** vienen con un volumen predeterminado. Para modificar este parámetro no se necesita cambiar el valor inicial sino multiplicarlo por factores que lo reduzcan o aumenten.

  Para este acometido, se ha creado la variable *gk_bellMasterLevel* que trabaja entre 0 y 1, según la elección realizada por el usuario. Una vez se ha escogido el valor y la función de JS ha modificado esta variable, se aplica a la señal obtenida multiplicando su valor y almacenándolo en otra variable de tipo audio.

  Para no perder el *stereo* comentado en el paso anterior esta operación se realiza tanto para el lado izquierdo como el derecho, guardando los resultados en variables separadas *Left* y *Right*.

- **Salida del audio por altavoces en tiempo real**

  Una vez aplicados todos los efectos el *opcode* **outs** se encarga de escribir los datos de audio *stereo*, por eso la 's' final, como se indique en CsOptions.

  En este caso, la opción escogida es en tiempo real y esto se ha expresado mediante la sentencia: -odac.

Como se ha ido comentando a lo largo de la descripción del instrumento en Csound, la variable global obtiene su valor por la interacción de la plataforma con el usuario desde JS.

A continuación se muestra la función que ha llevado a cabo este proceso:

```
1  /* - Variable global */
2  var instrControlVol = new Array;
3
4  /* - Variables para estilos CSS */
5  function volumen(id){
6
7      /* - Sentencias para aplicar los estilos CSS */
8
9      instrControlVol[Slug] = id/5;
10     csound.setControlChannel('gk_'+Slug+'MasterLevel', instrControlVol[
           Slug]);
11 }
```

Listing 6.7: Variación del volumen por pulsación del usuario

Observando la función de control de volumen sin la aplicación de estilos a la interfaz se puede distinguir claramente el proceso seguido. Si el usuario pulsa una tecla que llame a la función volumen, esta tecla dará un valor mediante el *id* de su

fila. Sabiendo que el valor máximo que puede contener es 5 y el valor mínimo 1, se realiza una división para normalizar ese valor entre 0.2 y 1.

El valor obtenido de la normalización se guarda en la posición correspondiente al instrumento en el array de control de volumen. Una vez se ha realizado esta acción, se modifica el valor del master en Csound mediante la función mencionada anteriormente en la sección 6.2, `setControlChannel('variable', 'valor');`.

Estas acciones se realizan mientras el usuario interactúe y la partitura no finalice.

Antes de finalizar este apartado es necesario comentar el uso del volumen para la simulación del *Pause* del instrumento. Se trata simplemente del envío del valor 0.00001 por `setControlChannel('variable', 'valor');` a la variable de Csound que controla el máster, quedando la sentencia:

```
1 if(instrControl[Slug]){
2     csound.setControlChannel('gk_'+Slug+'MasterLevel', 0.00001);
3 }
```

eso sí, comprueba que el instrumento esté activo en ese momento.

Al igual que en el caso de *Pause*, para *Stop* se realiza la misma acción para conseguir el efecto de parada.

6.2.3. Panoramización

En base a los resultados obtenidos con el control de volumen, se planteó un control de panorámica aplicado a este.

Para entender mejor este control y observar su relación con el volumen se explicarán en profundidad los cambios en el instrumento Csound que logran este propósito.

En el código siguiente se observan las acciones ya explicadas en el apartado 6.2.2 y las introducidas para este control en concreto.

```
; Valor inicial panoramica (iguales)
gk_bellPanoramicaLeft init 1
gk_bellPanoramicaRight init 1

; Instrumento: campana
instr bell

; Carga del archivo
aL, aR soundin "sounds/bell.wav"

; Aplicar el nivel del master (volumen) y panoramica, izquierda y derecha
aoutleft = (gk_bellPanoramicaLeft) * gk_bellMasterLevel * aL
aoutright = (gk_bellPanoramicaRight) * gk_bellMasterLevel * aR

; Salida con todos los efectos aplicados
outs aoutleft, aoutright
endin
```

Listing 6.8: Control de la panorámica en un instrumento

Para este caso ya no se necesita una sola variable global a modificar, puesto que se recibirán valores distintos para cada extremo, teniendo el lado izquierdo su propia variable y el derecho ídem.

Al igual que la variable *gk_bellMasterLevel*, *gk_bellPanoramicaLeft* y *gk_bellPanoramicaRight* se inicializan a través del operador **init**, pero con el valor 1 que equivale al estado original del audio reproducido.

Por último, los valores que se irán obteniendo mediante las operaciones realizadas en JS, que se explicarán a continuación, variarán el valor de las variables panorámicas y con ellas la salida interpretada por **outs**.

```
/* - Variables globales */
var instrControlPanL = new Array;
var instrControlPanR = new Array;

/* - Variables para los estilos CSS */
function panoramica(id){

    /* - Sentencias para aplicar los estilos CSS */

    instrControlPanL[Slug] = 1;
    instrControlPanR[Slug] = 1;
    if(id != 3){
        instrControlPanL[Slug] = (id/5);
        instrControlPanR[Slug] = (1.2 - (id/5));
    }

    csound.setControlChannel('gk_'+Slug+'PanoramicaLeft',
        instrControlPanL[Slug]);
    csound.setControlChannel('gk_'+Slug+'PanoramicaRight',
        instrControlPanR[Slug]);
}
```

Listing 6.9: Variación de la panorámica mediante pulsación

En este caso, al igual que en el lenguaje Csound se necesitaban dos variables en JS también.

Realizando un proceso similar al del volumen, cuando el usuario pulsa una tecla de este control, la tecla envía el nivel en el que se encuentra. Independientemente del nivel obtenido se igualará a 1 la posición del array para que ambos lados estén por igual en el caso de pulsar el nivel intermedio (nivel 3).

Por otro lado, si se trata de un nivel distinto del intermedio se realizarán dos operaciones distintas: una para el lado izquierdo normalizando el valor obtenido, como pasaba en el control de volumen, y otra para el derecho que obtiene justo el valor opuesto pero controlando que nunca de 0.

Por último, se envían los valores calculados a las variables de Csound para reproducir el resultado en el instrumento.

6.2.4. Reverberación

Después de aplicar los valores asociados al volumen y la panorámica existe un tercer control que genera el efecto de reverberación.

Para ello, se han definido dos variables globales. La primera define la relación del volumen de la señal añadida con el de la original, que en este caso será de la

mitad. Mientras que la otra variable representa el retardo escogido por el usuario al pulsar los botones de la columna de este control.

Estas mismas variables se inicializan antes de comenzar con el desarrollo del instrumento, al igual que en los casos de volumen y panorámica.

```
; Valor inicial rever
gk_ReverWet init .5
gk_bellReverDelay init .00001

; Instrumento: campana
instr bell

; Carga del archivo
aL, aR soundin "sounds/bell.wav"

; Aplica el nivel del master (volumen) y panoramica, izquierda y derecha
aoutleft = (gk_bellPanoramicaLeft) * gk_bellMasterLevel * aL
aoutright = (gk_bellPanoramicaRight) * gk_bellMasterLevel * aR

; Rever a la salida anterior
kdry = 1.0 - gk_ReverWet
awetleft, awetright reverbsc aoutleft, aoutright, gk_bellReverDelay,
    18000.0
aoutleftRev = aoutleft * kdry + awetleft * gk_ReverWet
aoutrightRev = aoutright * kdry + awetright * gk_ReverWet

; Salida con todos los efectos aplicados
outs aoutleftRev, aoutrightRev
endin
```

Listing 6.10: Control de la reverberación en un instrumento

Una vez se han obtenido los valores de salida, después del procesado de volumen y panorama, comienza el cálculo de la reverberación que se aplicará seguidamente.

Primero, se restará al valor máximo 1.0 el valor del volumen de la reverberación añadida *gk_ReverWet*, lo cual se almacenará en una variable para su posterior uso.

Inmediatamente después, gracias al operador **reverbsc**, se han obtenido las señales que irán añadidas a la señal con un retardo generado. Este *opcode* admite como entradas las variables de audio de los canales izquierdo y derecho, el retardo escogido en este caso por el usuario y un valor que hace referencia a la frecuencia de corte de filtros de paso simple de primer orden en la señal retardada de valor 18 kHz. De estos valores de entrada resultarán *awetleft y awetright*, que son las señales retardadas que se añadirán en un porcentaje a las originales.

Por lo tanto, las salidas de audio que serán interpretadas por **outs** vendrán calculadas por la suma entra la señal original y la señal retardada ambas ponderadas por 0.5, siendo modificable la ponderación cambiando el valor de *gk_ReverWet*. Si quisiéramos dar menor importancia a la señal retardad el valor de la variable global *wet* será inferior a 0.5, de esa forma *kdry* será mayor.

Ahora se presentará la función de JS que realiza el cálculo del retardo y se envía a su variable correspondiente en Csound, del mismo modo en que se realiza el control de volumen.

```
1  /* - Variable global */
2  var instrControlRever = new Array;
3
4  /* - Variables para los estilos CSS */
5  function reverberacion(id){
6
7      /* - Sentencias para aplicar los estilos CSS */
8
9      instrControlRever[Slug] = id/5;
10     csound.setControlChannel('gk_'+Slug+'ReverDelay', instrControlRever[
           Slug]);
11 }
```

Listing 6.11: Variación de la reverberación de un instrumento

6.3 Filtros

En este apartado se trata el uso de efectos sobre los instrumentos. Utilizados para darle mayor funcionalidad al *launchpad virtual*.

La forma de ejecutar cada filtrado tiene la misma estructura. El proceso seguido es:

1. **Inicialización de la variable de activación**

   Para detectar el filtro que ha sido escogido, y en qué instrumento, se ha creado una variable global para cada caso. Esta variable se forma mediante el *slug* del instrumento y el nombre del filtro. Para entender el concepto se mostrará un ejemplo con el instrumento *bell* y un filtro de distorsión:

   ```
   gk_bellFilterDistor init 0
   ```

   Esta variable inicialmente está a 0 para que el filtro no se ejecute, este solo se activará si su valor es 1, como se explicará en el paso siguiente.

2. **Aplicación del filtro sobre el audio**

Una vez el audio ha sido ponderado por los factores de volumen, panorámica y reverberación, se comprueba que los filtros estén o no activados:

```
if(gk_bellFilterDistor == 1) then
; operaciones del filtro
endif
```

Listing 6.12: Activación de filtrados – Ejemplo distorsión

Si está activo, su valor es 1, se ejecutarán las sentencias que contenga ese filtro. Por el contrario, si su valor es 0 saltará ese filtro y comprobará los siguientes. Si en ningún caso se cumple la condición *outs* leerá el audio de salida que tenga en ese momento.

3. **Cambio de valor a la variable de activación**

   Como todos los pasos de parámetros realizados en esta plataforma, cuando el usuario escoge una tecla de filtros, el instrumento *Controls* recibe la información y guarda el valor en la variable global de Csound.

```
gk_bellFilterDistor_ chnget "gk_bellFilterDistor"
if gk_bellFilterDistor_ != 0 then
gk_bellFilterDistor = gk_bellFilterDistor_
endif
```

Listing 6.13: Código para el paso de valor de la variable global de filtrado

Cuando la forma de llevar a cabo la lectura y aplicación de un filtro se han entendido, es necesario conocer como el lenguaje JS controla esta información. Para ello, se va a mostrar la parte de código en la función de filtrado encarga de conectar con Csound, omitiendo las sentencias que se explicaban en la sección 4.3.3.

```
function filtros(filter){
    Slug = document.getElementById("replay").classList;

    if(!instrControlFilter[Slug+'-'+filter]){
        instrControlFilter[Slug+'-'+filter] = 1;
        csound.setControlChannel('gk_'+Slug+'Filter'+filter, 1);
        return;
    }
    csound.setControlChannel('gk_'+Slug+'Filter'+filter, -1);
    instrControlFilter[Slug+'-'+filter] = 0;
}
```

Listing 6.14: Código para la activación del filtro mediante JS

Las sentencias introducidas se encargan de comprobar si el filtro ya está activo. Esto se realiza mediante un *array* asociativo que nombra las posiciones con el *slug* del instrumento y el nombre del filtro. Existen dos posibles casos:

1. **El botón del filtro no ha sido pulsado o está inactivo**

   Si esto sucede, la condición del *if* se cumple y se ejecutan las sentencias que hay programadas. Primero se pone a 1 el instrumento en el *array* asociativo para controlar su estado y luego se pone al mismo valor la variable global de Csound para ese filtro en concreto.

2. **El botón está activado**

   Si ya ha sido pulsado anteriormente se ejecutarán las acciones que siguen al condicional, que se encargan de realizar la acción opuesta. Primero desactiva en Csound el filtro cambiando el valor de activación por un número negativo y, después, pone el *array* de JS a 0 en esa posición.

Esto se repite con cada interacción del usuario con los filtros de la plataforma.

Por último, para probar el efecto de filtrados sobre la señal de audio se han realizado cuatro tipos de ejemplo, dejando el resto para creación futura y centrándonos en la relación interfaz – programación interna.

Los filtros escogidos han sido:

- **Filtro de distorsión aleatoria**

  El filtro que se muestra a continuación realiza una distorsión distinta en cada extremo (izquierda y derecha), ya que el operador **distort** tiene como entrada dos líneas de duraciones distintas por estar calculadas con valores aleatorios, a su vez, diferentes.

```
1  gifn ftgen 0,0, 257, 9, .5,1,270 ; define un 'sigmoid' fuera del
       instrumento
2
3  if(gk_bellFilterDistor == 1) then
4  iresL random 0, 0.25
5  iresR random 0, 0.5
6  kdistL line 0, iresL, 2
7  kdistR line 0, iresR, 2
8  ; aumenta gradualmente la distorsion
9  aoutleftRev distort aoutleftRev, kdistL, gifn
10 aoutrightRev distort aoutrightRev, kdistR, gifn
11 endif
```

Listing 6.15: Filtro de distorsión aleatoria en Csound

- **Filtro de excitación**

  Para añadir brillo a la señal se utiliza el operador **exciter**, que realiza este efecto creando una señal de armónicos con la misma señal de audio a la que irá sumada.

  Este operador requiere de dos frecuencias, una para el armónico más bajo y otra para el más alto permitidos, la cantidad de armónicos, en el rango de 0.1 a 10, y la mezcla de estos entre el 2º y 3er orden, entre -10 y 10.

  En este caso se ha utilizado el máximo posible de la configuración, todos los parámetros de entrada tienen el valor más elevado posible.

```
1 if(gk_bellFilterExci == 1) then
2 aexciL exciter aoutleftRev, 3000, 20000, 10, 10
3 aexciR exciter aoutrightRev, 3000, 20000, 10, 10
4 aoutleftRev = aexciL + aoutleftRev
5 aoutrightRev = aexciR + aoutrightRev
6 endif
```

Listing 6.16: Filtro de excitación en Csound

- **Filtro de armónicos**

  Para este filtro se ha escogido el operador **harmon** que se encarga de analizar una entrada de audio y generar voces de armonización en sincronía con la actual.

  Para realizar este efecto necesita una serie de parámetros comenzando por la señal con la que pretende estar en sincronización, seguido de la frecuencia que se estima para la entrada y la máxima variación esperada. Además, también cuenta con la primera y segunda frecuencia generadas ponderando la estimada de la entrada, y el modo de interpretación de las mismas.

  Por último, a la hora de añadir los armónicos calculados, se suman a la señal pero se disminuye su nivel un 40 % para evitar saturaciones en el resultado.

  El código utilizado hace referencia a uno de los ejemplos de Csound que se puede consultar aquí.

```
if(gk_bellFilterHarm == 1) then
kestfrq = 440
kmaxvar = 0.1
imode = 1
iminfrq = 100
iprd = 0.02
asigL harmon aoutleftRev, kestfrq, kmaxvar, kestfrq*.5, kestfrq*4,
    imode, iminfrq, iprd
asigR harmon aoutrightRev, kestfrq, kmaxvar, kestfrq*.5, kestfrq*4,
    imode, iminfrq, iprd

aoutleftRev = (asigL + aoutleftRev)*0.6
aoutrightRev = (asigR + aoutrightRev)*0.6
endif
```

Listing 6.17: Filtro de armónicos en Csound

- **Filtro paso bajo**

  Este último filtro de ejemplo realiza un filtro de paso bajo a través del operador **tone** que necesita las muestras de audio de entrada y una línea descendente de 0.5s de duración. Dando como resultado una señal de salida suavizada y más nítida.

```
if(gk_bellFilterLowP == 1) then
kton line 10000, 0.5, 0
aoutrightRev tone aoutrightRev, kton
aoutleftRev tone aoutleftRev, kton
endif

endin
```

Listing 6.18: Filtro paso bajo en Csound

Aunque se hayan utilizado estos filtros cualquier otro puede realizarse siguiendo la estructura explicada al principio de esta sección.

CAPÍTULO 7
Conclusiones

Como se ha observado a lo largo del trabajo, se trata de un proyecto en el que se ha desarrollado un prototipo, pero a través de un largo periodo de investigación. Gracias a la indagación realizada sobre la conexión entre HTML y Csound se ha logrado aumentar la capacidad de diseño de las interfaces visuales de este lenguaje sonoro. La posibilidad de introducir HTML en Csound existe desde hace varios años, pero no se ha llegado a experimentar en profundidad en este campo hasta el punto de darle menor importancia en las versiones superiores a la 6.09. Es necesario retomar esta línea de investigación ya que el lenguaje de maquetación web está en constante crecimiento. En este proyecto se ha conseguido superar las limitaciones que suponían las interfaces gráficas como los widgets, pudiendo crear cualquier tipo de forma y efecto desde el mismo código.

A pesar de la gran dificultad que conlleva realizar un prototipo sin tener conocimientos previos de la conexión entre los lenguajes a utilizar, y de no tener el tiempo suficiente para realizar una búsqueda exhaustiva de información al respecto, se ha logrado desarrollar una plataforma que reúne la funcionalidad básica sonora con una interfaz pensada para la experiencia del usuario (UX). Este trabajo profundiza en la creación de interfaces más interactivas dejando una puerta abierta a Csound para introducirse en la creación de aplicaciones.

Aun habiendo obtenido los resultados esperados, es cierto que no se ha podido optimizar suficientemente el código debido a la falta de experiencia en los lenguajes utilizados. No obstante, en un futuro sería una buena línea de trabajo plantear la introducción de lenguajes funcionales de programación como PHP para evitar la repetición de estructuras idénticas.

Uno de los grandes retos que ha planteado este proyecto es el control de la partitura y CsoundQt. Como se ha podido observar en las funciones de JS, tanto *Pause* como *Stop* no realizan la funcionalidad deseada, sino que ocultan la señal que está siendo reproducida pasándole un valor nulo al control del *Master*. Esto se debe a que después de la realización de varias pruebas para la activación de instrumentos

se decidió reproducirlos enviando órdenes desde la partitura. De este modo, una vez creada la partitura no se puede eliminar o finalizar el contenido en el momento escogido, a no ser que se envíe una orden de cierre de partitura (**e**), lo que no permite retomar la activación de ningún instrumento controlado por la misma. Es cierto que existe una solución a este problema, lo que conlleva no utilizar la partitura como método de activación, pero debido a la falta de tiempo no se ha podido reorganizar el código para que lo interprete de la forma correcta.

Por otro lado, la idea principal que supone una innovación en el mundo de las plataformas de control sonoro actuales, es el uso que se puede hacer de ella. Este prototipo puede ser utilizado tanto por su interfaz gráfica como programado por el mismo usuario, no es necesario dejar una puerta cerrada a las mejoras del código.

Por último, uno de los grandes objetivos de este proyecto era el desarrollo de un software libre y modificable lo que se ha conseguido fácilmente al utilizar dos lenguajes que tienen estas cualidades.

Bibliografía

[1] Teclado controlador MIDI – Teclado maestro EL EQUIPO DIGITAL – CAPÍTULO 8 Consultado en https://musicalecer.wordpress.com/el-equipo-digital/dispositivos-controladores/teclado-controlador-midi-teclado-maestro/.

[2] Frecuencia de las notas musicales, escrita el 19 de Agosto 2016 Consultado en http://latecladeescape.com/h/2015/08/frecuencia-de-las-notas-musicales.

[3] Foro para dudas sobre las notas musicales y MIDI Consultado en http://www.analfatecnicos.net/.

[4] Teclados MIDI – ¿Qué son y para qué sirven?, publicado el 19 de Agosto 2016 Consultado en http://tuhomestudio.com/teclados-midi-que-son-y-para-que-sirven/.

[5] Fundamentos básicos Online sobre la producción musical, Controladores MIDI Consultado en http://www.unimusica-peru.com/produccion_musical_controlador_midi.htm.

[6] Online Launchpad virtual, plataforma Consultado en https://agile-spire-1086.herokuapp.com/.

[7] Tutorial sobre la plataforma Online Launchpad virtual Consultado en https://www.youtube.com/watch?v=tAjIG_cl7ss.

[8] Plataforma Launchpad Intro Consultado en http://intro.novationmusic.com/harry-coade.

[9] Plataforma de dubstep virtual Consultado en http://createyourmusiconline.blogspot.com.es/p/dubstep-tool_31.html.

[10] Documento oficial de HTML5 Consultado en https://www.w3.org/TR/html5/.

[11] Foro sobre dudas HTML Consultado en http://www.lawebdelprogramador.com/foros/HTML.

[12] Libro guía sobre CSS Consultado en http://librosweb.es/libro/css.

[13] Tutorial genérico para el lenguaje CSS Consultado en http://es.html.net/tutorials/css/.

[14] Descripción de Hoja de estilos en cascada, de manera técnica Consultado en https://es.wikipedia.org/wiki/Hoja_de_estilos_en_cascada.

[15] Libro básico y en profundidad para el lenguaje Javascript Consultado en http://librosweb.es/libro/javascript/capitulo_1.html.

[16] Ayuda técnica y para aplicación del lenguaje Javascript, ejemplos de uso. Consultado en https://www.w3schools.com/jsref/dom_obj_document.asp.

[17] Microsoft 'jubila' el Paint, artículo publicado el 24 de julio de 2017 Consultado en http://www.eitb.eus/es/noticias/tecnologia/detalle/4986449/microsoft-eliminara-paint-su-nueva-actualizacion-windows-10-2017/.

[18] Página original de descarga de iconos Flaticon Consultado en https://www.flaticon.com/.

[19] Página oficial de descarga de sonidos FreeSound Consultado en http://freesound.org/.

[20] Página oficial de licencias de divulgación de contenidos Consultado en https://creativecommons.org/.

[21] Datos sobre la base de datos de Freesound Consultado en https://en.wikipedia.org/wiki/Freesound.

[22] Información y descarga de PhpStorm Consultado en https://www.jetbrains.com/phpstorm/.

[23] Posibilidades de uso del HTML5 en la web Consultado en https://html5test.com/.

APÉNDICE A
Algoritmos

A continuación se muestran los algoritmos realizados.

A.1 Algoritmo que contiene el lenguaje Csound

En este algoritmo sólo se expone el instrumento campana, ya que el resto de instrumentos son idénticos reemplazando la palabra *bell* en todas las variables y en el nombre del propio instrumento.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>
;Variables iniciales
sr = 44100
ksmps = 20
nchnls = 2 ;stereo

gifn ftgen 0,0, 257, 9, .5,1,270 ; define un 'sigmoid'

; Lectura constante de los instrumentos nombrados
alwayson "Controls"

; Valor inicial volumen
gk_bellMasterLevel init 0.00001
; Valor inicial panoramica (iguales)
gk_bellPanoramicaLeft init 1
gk_bellPanoramicaRight init 1
; Valor inicial reverberacion
gk_ReverWet init .5
gk_bellReverDelay init .00001

```

```
25 ; Valores iniciales filtros
26 gk_bellFilterDistor init 0
27 gk_bellFilterExci init 0
28 gk_bellFilterHarm init 0
29 gk_bellFilterLowP init 0
30
31 ; Instrumento: campana
32 instr bell
33 ; Carga del archivo
34 aL, aR soundin "sounds/bell.wav"
35 ; Aplicacion del nivel del master (volumen) y panoramica, izquierda y
       derecha
36 aoutleft = (gk_bellPanoramicaLeft) * gk_bellMasterLevel * aL
37 aoutright = (gk_bellPanoramicaRight) * gk_bellMasterLevel * aR
38 ; Reverberacion a la salida anterior
39 kdry = 1.0 - gk_ReverWet
40 awetleft, awetright reverbsc aoutleft, aoutright, gk_bellReverDelay,
       18000.0
41 aoutleftRev = aoutleft * kdry + awetleft * gk_ReverWet
42 aoutrightRev = aoutright * kdry + awetright * gk_ReverWet
43
44 if(gk_bellFilterDistor == 1) then
45 iresL random 0, 0.25
46 iresR random 0, 0.5
47 kdistL line 0, iresL, 2
48 kdistR line 0, iresR, 2
49 ; aumenta gradualmente la distorsion
50 aoutleftRev distort aoutleftRev, kdistL, gifn
51 aoutrightRev distort aoutrightRev, kdistR, gifn
52 endif
53 if(gk_bellFilterExci == 1) then
54 aexciL exciter aoutleftRev, 3000, 20000, 10, 10
55 aexciR exciter aoutrightRev, 3000, 20000, 10, 10
56 aoutleftRev = aexciL + aoutleftRev
57 aoutrightRev = aexciR + aoutrightRev
58 endif
59 if(gk_bellFilterHarm == 1) then
60 kestfrq random 100,600
61 kmaxvar = 0.1
62 imode = 1
63 iminfrq = 100
64 iprd = 0.02
65 asigL harmon aoutleftRev, kestfrq, kmaxvar, kestfrq*.5, kestfrq*4, imode,
       iminfrq, iprd
66 asigR harmon aoutrightRev, kestfrq, kmaxvar, kestfrq*.5, kestfrq*4, imode
       , iminfrq, iprd
67
68
```

```
69  aoutleftRev = (asigL + aoutleftRev)*0.6
70  aoutrightRev = (asigR + aoutrightRev)*0.6
71  endif
72  if(gk_bellFilterLowP == 1) then
73  kton line 10000, 0.5, 0
74  aoutrightRev tone aoutrightRev, kton
75  aoutleftRev tone aoutleftRev, kton
76  endif
77
78  ; Salida con todos los efectos aplicados
79  outs aoutleftRev, aoutrightRev
80  endin
81
82  ; Intrumento de controles
83  instr Controls
84
85  ; Controles de campana
86  gk_bellMasterLevel_ chnget "gk_bellMasterLevel"
87  if gk_bellMasterLevel_ != 0 then
88  gk_bellMasterLevel = gk_bellMasterLevel_
89  endif
90  gk_bellPanoramicaLeft_ chnget "gk_bellPanoramicaLeft"
91  if gk_bellPanoramicaLeft_ != 0 then
92  gk_bellPanoramicaLeft = gk_bellPanoramicaLeft_
93  endif
94  gk_bellPanoramicaRight_ chnget "gk_bellPanoramicaRight"
95  if gk_bellPanoramicaRight_ != 0 then
96  gk_bellPanoramicaRight = gk_bellPanoramicaRight_
97  endif
98  gk_bellReverDelay_ chnget "gk_bellReverDelay"
99  if gk_bellReverDelay_ != 0 then
100 gk_bellReverDelay = gk_bellReverDelay_
101 endif
102 gk_bellFilterDistor_ chnget "gk_bellFilterDistor"
103 if gk_bellFilterDistor_ != 0 then
104 gk_bellFilterDistor = gk_bellFilterDistor_
105 endif
106 gk_bellFilterExci_ chnget "gk_bellFilterExci"
107 if gk_bellFilterExci_ != 0 then
108 gk_bellFilterExci = gk_bellFilterExci_
109 endif
110 gk_bellFilterHarm_ chnget "gk_bellFilterHarm"
111 if gk_bellFilterHarm_ != 0 then
112 gk_bellFilterHarm = gk_bellFilterHarm_
113 endif
114 gk_bellFilterLowP_ chnget "gk_bellFilterLowP"
115 if gk_bellFilterLowP_ != 0 then
116 gk_bellFilterLowP = gk_bellFilterLowP_
```

```
117 endif
118
119 endin
120 </CsInstruments>
121 <html>
122 // Algoritmo siguiente
123 </html>
124 <CsScore>
125 </CsScore>
126 </CsoundSynthesizer>
```

Listing A.1: Algoritmo en Csound - Instrumento: campana

A.2 Algoritmo que contiene el lenguaje HTML

```
<html>
  <!-- Cabeceras -->
  <head>
    <!-- Sentencia para cargar el estilo -->
    <link rel=StyleSheet href="css/launchpad.css" type="text/css">
    <!-- Sentencia para cargar el javascript -->
    <script type="text/javascript" src="js/launchpad.js"></script>
  </head>
  <!-- Cuerpo de la estructura HTML5 -->
  <body>
    <!-- Primera seccion - Pantalla incial -->
    <section id="first-part">
      <div class="container">
        <div class="row-5 col-5">
          <!-- Primera fila -->
          <div class="row">
            <!-- Estructura del primer instrumento -->
            <div class="celda style-button violet" onclick="instrument('
                bell')">
              <figure class="style-figure">
                <img src="img/percusion/bell.png" width="48" height="48">
              </figure>
            </div>
            <!-- Apartado sin uso -->
            <div class="celda style-button violet">
              <figure class="style-figure hide-figure">
                <img width="48" height="48">
              </figure>
            </div>
            <!-- Estructura del segundo instrumento -->
            <div class="celda style-button violet" onclick="instrument('
                pellet')">
              <figure class="style-figure">
                <img src="img/percusion/pellet.png" width="48" height="48
                    ">
              </figure>
            </div>
            <!-- Apartado sin uso -->
            <div class="celda style-button violet">
              <figure class="style-figure hide-figure">
                <img width="48" height="48">
              </figure>
            </div>
            <!-- Estructura del tercer instrumento -->
```

```
42              <div class="celda style-button violet" onclick="instrument('
                    pailas')">
43                <figure class="style-figure">
44                  <img src="img/percusion/pailas.png" width="48" height="48
                        ">
45                </figure>
46              </div>
47            </div>
48            <!-- Segunda fila -->
49            <div class="row">
50              <div class="celda style-button violet">
51                <figure class="style-figure hide-figure">
52                  <img width="48" height="48">
53                </figure>
54              </div>
55              <div class="celda style-button violet" onclick="instrument('
                    cowbell')">
56                <figure class="style-figure">
57                  <img src="img/percusion/cowbell.png" width="48" height="
                        48">
58                </figure>
59              </div>
60              <div class="celda style-button violet">
61                <figure class="style-figure hide-figure">
62                  <img width="48" height="48">
63                </figure>
64              </div>
65              <div class="celda style-button violet" onclick="instrument('
                    gong')">
66                <figure class="style-figure">
67                  <img src="img/percusion/gong.png" width="48" height="48">
68                </figure>
69              </div>
70              <div class="celda style-button violet">
71                <figure class="style-figure hide-figure">
72                  <img width="48" height="48">
73                </figure>
74              </div>
75            </div>
76            <!-- Tercera fila -->
77            <div class="row">
78              <div class="celda style-button violet" onclick="instrument('
                    cymball')">
79                <figure class="style-figure">
80                  <img src="img/percusion/cymbal.png" width="48" height="48
                        ">
81                </figure>
82              </div>
```

```
83              <div class="celda style-button violet">
84                <figure class="style-figure hide-figure">
85                  <img width="48" height="48">
86                </figure>
87              </div>
88              <div class="celda style-button violet" onclick="instrument('
                    triangle')">
89                <figure class="style-figure">
90                  <img src="img/percusion/triangle.png" width="48" height="
                        48">
91                </figure>
92              </div>
93              <div class="celda style-button violet">
94                <figure class="style-figure hide-figure">
95                  <img width="48" height="48">
96                </figure>
97              </div>
98              <div class="celda style-button violet" onclick="instrument('
                    tambourin')">
99                <figure class="style-figure">
100                 <img src="img/percusion/tambourine.png" width="48" height
                        ="48">
101               </figure>
102             </div>
103           </div>
104           <!-- Cuarta fila -->
105           <div class="row">
106             <div class="celda style-button violet">
107               <figure class="style-figure hide-figure">
108                 <img width="48" height="48">
109               </figure>
110             </div>
111             <div class="celda style-button violet" onclick="instrument('
                    maracas')">
112               <figure class="style-figure">
113                 <img src="img/percusion/maracas.png" width="48" height="
                        48">
114               </figure>
115             </div>
116             <div class="celda style-button violet">
117               <figure class="style-figure hide-figure">
118                 <img width="48" height="48">
119               </figure>
120             </div>
121             <div class="celda style-button violet" onclick="instrument('
                    metallophone')">
122               <figure class="style-figure">
```

```
123                 <img src="img/percusion/metallophone.png" width="48"
                        height="48">
124               </figure>
125             </div>
126             <div class="celda style-button violet">
127               <figure class="style-figure hide-figure">
128                 <img width="48" height="48">
129               </figure>
130             </div>
131           </div>
132           <!-- Quinta fila -->
133           <div class="row">
134             <div class="celda style-button violet" onclick="instrument('
                    bigdrum')">
135               <figure class="style-figure">
136                 <img src="img/percusion/bigdrum.png" width="48" height="
                        48">
137               </figure>
138             </div>
139             <div class="celda style-button violet">
140               <figure class="style-figure hide-figure">
141                 <img width="48" height="48">
142               </figure>
143             </div>
144             <div class="celda style-button violet" onclick="instrument('
                    clapping')">
145               <figure class="style-figure">
146                 <img src="img/percusion/clapping.png" width="48" height="
                        48">
147               </figure>
148             </div>
149             <div class="celda style-button violet">
150               <figure class="style-figure hide-figure">
151                 <img width="48" height="48">
152               </figure>
153             </div>
154             <div class="celda style-button violet" onclick="instrument('
                    drum')">
155               <figure class="style-figure">
156                 <img src="img/percusion/drum.png" width="48" height="48">
157               </figure>
158             </div>
159           </div>
160         </div>
161         <!-- Controles columna derecha -->
162         <div class="row-5 col-1 colocate">
163           <div class="row">
164             <div class="celda style-button green"></div>
```

```
165           </div>
166           <div class="row">
167             <div class="celda style-button green" onclick="commonPlay()">
168               <figure class="style-figure">
169                 <img src="img/play.png" width="48" height="48">
170               </figure>
171             </div>
172           </div>
173           <div class="row">
174             <div class="celda style-button green" onclick="commonPause()"
                  >
175               <figure class="style-figure">
176                 <img src="img/pause.png" width="48" height="48">
177               </figure>
178             </div>
179           </div>
180           <div class="row">
181             <div class="celda style-button green" onclick="commonStop()">
182               <figure class="style-figure">
183                 <img src="img/stop.png" width="48" height="48">
184               </figure>
185             </div>
186           </div>
187           <div class="row">
188             <div class="celda style-button green"></div>
189           </div>
190         </div>
191         <div class="row-1 col-6">
192           <div class="celda style-button green"></div>
193           <div class="celda style-button green"></div>
194           <div class="celda style-button green"></div>
195           <div class="celda style-button green"></div>
196           <div class="celda style-button green"></div>
197           <div class="celda style-button spaced-first green" onclick="
                reset()">
198             <figure class="style-figure">
199               <img src="img/back.png" width="48" height="48">
200             </figure>
201           </div>
202         </div>
203       </div>
204     </section>
205
206     <!-- Segunda seccion - Pantalla interna del instrumento -->
207     <section class="instrument" style="display: none;">
208       <div class="container">
209         <div class="row-5 col-5">
210           <div id="5" class="row">
```

```
211             <div class="celda style-button cian" onclick="volumen(5)"></
                    div>
212             <div id="Distor" class="celda style-button orange" onclick="
                    filtros('Distor')"></div>
213             <div id="Exci" class="celda style-button orange" onclick="
                    filtros('Exci')"></div>
214             <div class="celda style-button blue" onclick="panoramica(5)">
                    </div>
215             <div class="celda style-button green-lighter" onclick="
                    reverberacion(5)"></div>
216           </div>
217           <div id="4" class="row">
218             <div class="celda style-button cian" onclick="volumen(4)"></
                    div>
219             <div id="Harm" class="celda style-button orange" onclick="
                    filtros('Harm')"></div>
220             <div id="LowP" class="celda style-button orange" onclick="
                    filtros('LowP')"></div>
221             <div class="celda style-button blue" onclick="panoramica(4)">
                    </div>
222             <div class="celda style-button green-lighter" onclick="
                    reverberacion(4)"></div>
223           </div>
224           <div id="3" class="row">
225             <div class="celda style-button cian" onclick="volumen(3)"></
                    div>
226             <div class="celda style-button orange"></div>
227             <div class="celda style-button orange"></div>
228             <div class="celda style-button blue" onclick="panoramica(3)">
                    </div>
229             <div class="celda style-button green-lighter" onclick="
                    reverberacion(3)"></div>
230           </div>
231           <div id="2" class="row">
232             <div class="celda style-button cian" onclick="volumen(2)"></
                    div>
233             <div class="celda style-button orange"></div>
234             <div class="celda style-button orange"></div>
235             <div class="celda style-button blue" onclick="panoramica(2)">
                    </div>
236             <div class="celda style-button green-lighter" onclick="
                    reverberacion(2)"></div>
237           </div>
238           <div id="1" class="row">
239             <div class="celda style-button cian" onclick="volumen(1)"></
                    div>
240             <div class="celda style-button orange"></div>
241             <div class="celda style-button orange"></div>
```

```
242             <div class="celda style-button blue" onclick="panoramica(1)">
                    </div>
243             <div class="celda style-button green-lighter" onclick="
                    reverberacion(1)"></div>
244           </div>
245         </div>
246         <div class="row-5 col-1 colocate">
247           <div class="row">
248             <div class="celda style-button green"></div>
249           </div>
250           <div class="row">
251             <div class="celda style-button green" onclick="insPlay()">
252               <figure class="style-figure">
253                 <img src="img/play.png" width="48" height="48">
254               </figure>
255             </div>
256           </div>
257           <div class="row">
258             <div class="celda style-button green" onclick="insPause()">
259               <figure class="style-figure">
260                 <img src="img/pause.png" width="48" height="48">
261               </figure>
262             </div>
263           </div>
264           <div class="row">
265             <div class="celda style-button green" onclick="insStop()">
266               <figure class="style-figure">
267                 <img src="img/stop.png" width="48" height="48">
268               </figure>
269             </div>
270           </div>
271           <div class="row">
272             <div class="celda style-button green"></div>
273           </div>
274         </div>
275         <div class="row-1 col-6">
276           <div class="celda style-button green">
277             <figure class="style-figure">
278               <img src="img/volume.png" width="48" height="48">
279             </figure>
280           </div>
281           <div class="celda style-button green">
282             <figure class="style-figure">
283               <img src="img/filters/003-sound.png" width="48" height="48"
                      >
284             </figure>
285           </div>
286           <div class="celda style-button green">
```

```
287             <figure class="style-figure">
288               <img src="img/filters/003-sound.png" width="48" height="48"
                      >
289             </figure>
290           </div>
291           <div class="celda style-button green">
292             <figure class="style-figure">
293               <img src="img/filters/002-pulse.png" width="48" height="48"
                      >
294             </figure>
295           </div>
296           <div class="celda style-button green">
297             <figure class="style-figure">
298               <img src="img/filters/001-stopwatch.png" width="48" height=
                      "48">
299             </figure>
300           </div>
301           <div class="celda style-button spaced green" onclick="insreturn
                  ()">
302             <figure class="style-figure">
303               <img id="replay" src="img/back.png" width="48" height="48">
304             </figure>
305           </div>
306         </div>
307       </div>
308     </section>
309   </body>
310 </html>
```

Listing A.2: Algoritmo en HTML5

A.3 Algoritmo que contiene el lenguaje CSS

```
/* ----------------------
 Estructuras
 ---------------------- */
.container{
    width: 600px;
    height: 600px;
    display: block;
}
/* Filas */
.row{
    height: 100px;
}
.row-5{
    height: 500px;
}
.row-1{
    height: 100px;
    margin-top: 7.5px!important;
    display: inline-block;
}
/* Columnas */
.col-6{
    width: 600px;
}
.col-5{
    width: 500px;
    float: left;
}
.col-1{
    width: 100px;
    float: right;
}
/* Celdas */
.celda{
    width: 80px;
    display: inline-block;
    height: 80px;
}
/* ----------------------
 Espaciado y Ajustes
 ---------------------- */
.spaced{
    margin-left: 13px!important;
}
.colocate{
```

```
46      margin-left: -7.5px;
47  }
48  /* Celdas */
49  .style-button{
50      margin: 7.5px;
51      border-radius: 7px;
52  }
53  /* ----------------------
54   Figuras
55   ---------------------- */
56  .style-figure{
57      padding: 15px;
58      margin: 0px;
59  }
60  .style-figure:hover{
61      cursor: pointer;
62  }
63  .spaced-first .style-figure{
64      float: right;
65  }
66  /* Oculta */
67  .hide-figure{
68      visibility: hidden;
69  }
70  /* ----------------------
71   Colores
72   ---------------------- */
73  /* Estaticos */
74  .violet{
75      background-color: #9e54bd;
76      box-shadow: 0 5px #823aa0;
77  }
78  .green{
79      background-color: #17aa56;
80      box-shadow: 0 5px #119e4d;
81  }
82  .orange {
83      background-color: #f97508;
84      box-shadow: 0 5px #e06907;
85  }
86  /* Para efectos */
87  .orange-light {
88      background-color: #fcc79c;
89      box-shadow: none;
90      transform: translateY(2.5px);
91  }
92  .green-lighter{
93      background-color: #8bd200;
```

```
94      box-shadow: 0 5px #7cbb00;
95  }
96  .green-lighter-light {
97      background-color: #afee32;
98  }
99  .cian{
100     background-color: #00cccc;
101     box-shadow: 0 5px #008B8B;
102 }
103 .cian-light{
104     background-color: #00ffff;
105 }
106 .blue {
107     background-color: #7069c3;
108     box-shadow: 0 5px #5e56bb;
109 }
110 .blue-light {
111     background-color: #a6a1d9;
112 }
```

Listing A.3: Algoritmo en CSS

A.4 Algoritmo que contiene el lenguaje JavaScript

```
var instrControl = new Array;
var instrControlVol = new Array;
var instrControlPanL = new Array;
var instrControlPanR = new Array;
var instrControlRever = new Array;
var instrControlFilter = new Array;

// Funcion Inicial Instrumento
function instrument(elementSlug) {

    document.getElementById("first-part").style.display = 'none';
    document.querySelector(".instrument").classList.add("instrument-"+
        elementSlug);
    document.getElementById("replay").classList.add(elementSlug);
    document.querySelector(".instrument-"+elementSlug).style.display = '
        block';

    /* Restaurar */
    /* - Volumen */
    addControl(addVolumen);
    removeControl(removeVolumen);
    /* - Panoramica */
    addControl(addPanoramica);
    removeControl(removePanoramica);
    /* - Reverberacion */
    addControl(addReverberacion);
    removeControl(removeReverberacion);
    /* - Filtros */
    addFiltersBySlug(addFilters);

    insPlay();
}

// Controles Basicos
/** Volumen **/
/* - Variables */
var addVolumen = new Array();
var removeVolumen = new Array();
/* - Funcion */
function volumen(id){

    var addVolumenInstrument = new Array();
    var removeVolumenInstrument = new Array();

    Slug = document.getElementById("replay").classList;
```

```
44
45      var indice = 0;
46      for(contador=id+1;contador<=5;contador++){
47          removeVolumenInstrument[indice]= [contador,'.cian','cian-light'];
48          indice++;
49      }
50      removeVolumen[Slug] = removeVolumenInstrument;
51      removeControl(removeVolumen);
52
53      indice = 0;
54      for(contador=id;contador>=1;contador--){
55          addVolumenInstrument[indice]= [contador,'.cian','cian-light'];
56          indice++;
57      }
58      addVolumen[Slug] = addVolumenInstrument;
59      addControl(addVolumen);
60
61      instrControlVol[Slug] = id/5;
62      csound.setControlChannel('gk_'+Slug+'MasterLevel', instrControlVol[
            Slug]);
63  }
64  /** Panoramica **/
65  /* - Variables */
66  var addPanoramica = new Array();
67  var removePanoramica = new Array();
68  /* - Funcion */
69  function panoramica(id){
70      var addPanoramicaInstrument = new Array();
71      var removePanoramicaInstrument = new Array();
72
73      Slug = document.getElementById("replay").classList;
74
75      var indice = 0;
76      for(contador=id+1;contador<=5;contador++){
77          removePanoramicaInstrument[indice]= [contador,'.blue','blue-light
                '];
78          indice++;
79      }
80      removePanoramica[Slug] = removePanoramicaInstrument;
81      removeControl(removePanoramica);
82
83      indice = 0;
84      for(contador=id;contador>=1;contador--){
85          addPanoramicaInstrument[indice]= [contador,'.blue','blue-light'];
86          indice++;
87      }
88      addPanoramica[Slug] = addPanoramicaInstrument;
89      addControl(addPanoramica);
```

```
90
91      instrControlPanL[Slug] = 1;
92      instrControlPanR[Slug] = 1;
93      if(id != 3){
94          instrControlPanL[Slug] = (id/5);
95          instrControlPanR[Slug] = (1.2 - (id/5));
96      }
97
98      csound.setControlChannel('gk_'+Slug+'PanoramicaLeft',
            instrControlPanL[Slug]);
99      csound.setControlChannel('gk_'+Slug+'PanoramicaRight',
            instrControlPanR[Slug]);
100 }
101 /** Reverberacion **/
102 /* - Variables */
103 var addReverberacion = new Array();
104 var removeReverberacion = new Array();
105 /* - Funcion */
106 function reverberacion(id){
107     var addReverberacionInstrument = new Array();
108     var removeReverberacionInstrument = new Array();
109     Slug = document.getElementById("replay").classList;
110
111     var indice = 0;
112     for(contador=id+1;contador<=5;contador++){
113         removeReverberacionInstrument[indice]= [contador,'.green-lighter
                ','green-lighter-light'];
114         indice++;
115     }
116     removeReverberacion[Slug] = removeReverberacionInstrument;
117     removeControl(removeReverberacion);
118
119     indice = 0;
120     for(contador=id;contador>=1;contador--){
121         addReverberacionInstrument[indice]= [contador,'.green-lighter','
                green-lighter-light'];
122         indice++;
123     }
124     addReverberacion[Slug] = addReverberacionInstrument;
125     addControl(addReverberacion);
126
127     instrControlRever[Slug] = id/5;
128     csound.setControlChannel('gk_'+Slug+'ReverDelay', instrControlRever[
            Slug]);
129 }
130
131 // Filtrados
132 /** Filtros **/
```

```
133 /* - Variables */
134 var addFilters = new Array();
135 /* - Funcion */
136 function filtros(filter){
137     Slug = document.getElementById("replay").classList;
138 
139     if(!addFilters[Slug])
140         addFilters[Slug] = new Array;
141 
142     if(!instrControlFilter[Slug+'-'+filter]){
143         addFilters[Slug][filter] = filter;
144         addFiltersBySlug(addFilters);
145         instrControlFilter[Slug+'-'+filter] = 1;
146         csound.setControlChannel('gk_'+Slug+'Filter'+filter, 1);
147         return;
148     }
149     addFilters[Slug][filter] = false;
150     addFiltersBySlug(addFilters);
151     csound.setControlChannel('gk_'+Slug+'Filter'+filter, -1);
152     instrControlFilter[Slug+'-'+filter] = 0;
153 }
154 // Add el estilo a un filtro
155 function addFiltersBySlug(slugFilters){
156     Slug = document.getElementById("replay").classList;
157 
158     for (var propiedad in slugFilters[Slug]) {
159         if (slugFilters[Slug].hasOwnProperty(propiedad)) {
160             if(slugFilters[Slug][propiedad]!=false){
161                 document.getElementById(propiedad).classList.add('orange-
                        light');
162             }else{
163                 document.getElementById(propiedad).classList.remove('
                        orange-light');
164             }
165         }
166     }
167 }
168 // Funciones para Add o Eliminar
169 /** Add **/
170 function addControl(controlIntrumentArray){
171     Slug = document.getElementById("replay").classList;
172     var controls = new Array();
173     controls = controlIntrumentArray[Slug];
174     if(controls){
175         for(var item = 0 ; item < controls.length ; item++){
176             father = document.getElementById(controls[item][0]);
177             father.querySelector(controls[item][1]).classList.add(
                    controls[item][2]);
```

```
178         }
179     }
180 }
181 /** Eliminar **/
182 function removeControl(controlIntrumentArray){
183     Slug = document.getElementById("replay").classList;
184     var controls = new Array();
185     controls = controlIntrumentArray[Slug];
186     if(controls){
187         for(var item = 0 ; item < controls.length ; item++) {
188             father = document.getElementById(controls[item][0]);
189             father.querySelector(controls[item][1]).classList.remove(
                    controls[item][2]);
190         }
191     }
192 }
193 /** Eliminar todo **/
194 function removeAllControls(){
195     for(var id = 1 ; id <= 5 ; id++) {
196         father = document.getElementById(id);
197         var volumen = father.querySelector('.cian-light');
198         var panoramica = father.querySelector('.blue-light');
199         var reverberacion = father.querySelector('.green-lighter-light');
200         var filtros = father.querySelector('.orange-light');
201         if(volumen){
202             volumen.classList.remove('cian-light');
203         }
204         if(panoramica){
205             panoramica.classList.remove('blue-light');
206         }
207         if(reverberacion){
208             reverberacion.classList.remove('green-lighter-light');
209         }
210         if(filtros){
211             filtros.classList.remove('orange-light');
212         }
213     }
214 }
215
216 // Funciones principales
217 function insPlay(){
218     Slug = document.getElementById("replay").classList;
219     /* Inicializacion */
220     if(!instrControlVol[Slug]){
221         instrControlVol[Slug] = 0.2;
222     }
223     csound.setControlChannel('gk_'+Slug+'MasterLevel', instrControlVol[
        Slug]);
```

```
224     if(!instrControl[Slug]){
225         instrControl[Slug] = true;
226     }else{
227         return;
228     }
229     for(var count=0;count<120; count=count+0.5){
230         note = 'i "'+Slug+'" '+count+' 0.5';
231         csound.readScore(note);
232     }
233 }
234 function insPause(){
235     Slug = document.getElementById("replay").classList;
236     if(instrControl[Slug]){
237         csound.setControlChannel('gk_'+Slug+'MasterLevel', 0.00001);
238     }
239 }
240 function insStop(){
241     Slug = document.getElementById("replay").classList;
242     if(instrControl[Slug]){
243         csound.setControlChannel('gk_'+Slug+'MasterLevel', 0.00001);
244     }
245 }
246 // Funciones generales
247 /* - Funcion */
248 function commonPlay() {
249     for (var slug in instrControl) {
250         if (instrControl.hasOwnProperty(slug)) {
251             csound.setControlChannel('gk_'+slug+'MasterLevel',
                    instrControlVol[slug]);
252         }
253     }
254 }
255 /* - Funcion */
256 function commonPause() {
257     for (var slug in instrControl) {
258         if (instrControl.hasOwnProperty(slug)) {
259             csound.setControlChannel('gk_'+slug+'MasterLevel', 0.00001);
260         }
261     }
262 }
263 /* - Funcion */
264 function commonStop() {
265     for (var slug in instrControl) {
266         if (instrControl.hasOwnProperty(slug)) {
267             csound.setControlChannel('gk_'+slug+'MasterLevel', 0.00001);
268         }
269     }
270 }
```

```
271 // Volver
272 /* - Funcion */
273 function insreturn(){
274     Slug = document.getElementById("replay").classList;
275     removeAllControls();
276     document.getElementById("first-part").style.display = 'block';
277     document.querySelector(".instrument-"+Slug).style.display = 'none';
278     document.querySelector(".instrument").classList.remove("instrument-"+
          Slug);
279     document.getElementById("replay").classList.remove(Slug);
280 }
```

Listing A.4: Algoritmo en JS