ESCUELA SUPERIOR DE INGENIERÍA

Ingeniería Técnica en Informática de Sistemas

oFlute: análisis de blablablá con un bláblá

Añadir nombre completo

José Tomás Tocino García

Cádiz, 7 de agosto de 2011

ESCUELA SUPERIOR DE INGENIERÍA

Ingeniería Técnica en Informática de Sistemas

oFlute: análisis de blablablá con un bláblá

Añadir nombre completo

DEPARTAMENTO: Lenguajes y Sistemas Informáticos.
DIRECTOR DEL PROYECTO: Manuel Palomo Duarte.
AUTOR DEL PROYECTO: José Tomás Tocino García.

Cádiz, 7 de agosto de 2011

Fdo.: José Tomás Tocino García

Agradecimientos

A Julian Raschke por crear y mantener Gosu.

Índice general

Índice de figuras

Índice de cuadros

1. Introducción

1.1. Contexto y motivación

Las nuevas tecnologías van filtrándose gradualmente en los centros educativos, y las técnicas de enseñanza se están adaptando a las opciones que ofrecen. El reparto de ordenadores portátiles a los alumnos andaluces de 5º y 6º de primaria, dentro del marco de la Escuela TIC 2.0, es buena muestra de ello.

Por otro lado, las nuevas generaciones están en plena simbiosis con las tecnologías de la información, cada vez más acostumbradas al empleo de dispositivos electrónicos, y su uso ya les es prácticamente instintivo. Por tanto, es beneficioso buscar nuevos métodos educativos que hagan uso de las nuevas tecnologías.

En la búsqueda de materias educativas en las que aplicar el uso de las nuevas tecnologías, la música, parte fundamental del programa curricular en la educación primaria, ofrece una gran variedad de aspectos que podrían desarrollarse utilizando tecnologías de la información. Es ahí donde este proyecto hace su aportación.

1.2. Objetivos

A la hora de definir los objetivos de un sistema, podemos agruparlos en dos tipos diferentes: **funcionales** y **transversales**. Los primeros se refieren a *qué* debe hacer la aplicación que vamos a desarrollar, e inciden directamente en la experiencia del usuario y de potenciales desarrolladores.

Por otro lado, los objetivos transversales son aquellos invisibles al usuario final, pero que de forma inherente actúan sobre el resultado final de la aplicación y sobre la experiencia de desarrollo de la misma.

1.2.1. Funcionales

- Crear un módulo de análisis del sonido en el dominio de la frecuencia para poder identificar las notas capturadas por el micrófono en tiempo real.

- Crear una aplicación de usuario que identifique y muestre en pantalla las notas que toca el usuario en cada momento.
- Reutilizar el módulo de análisis en un juego en el que el usuario debe tocar correctamente las notas que aparecen en pantalla siguiendo un pentagrama.
- Incluir un sistema de lecciones multimedia individuales que sirvan al alumno de referencia y fuente de aprendizaje.
- Potenciar el uso de interfaces de usuario amigables, con un sistema avanzado de animaciones que proporcione un aspecto fluido y evite saltos bruscos entre secciones.

1.2.2. Transversales

- Obtener una base teórica sobre cómo se representa y caracteriza digitalmente el sonido.
- Conocer las bases del DSP (Digital Signal Processing), y su uso en aplicaciones de reconocimiento básico de sonidos, tales como sintonizadores y afinadores de instrumentos.
- Introducirme en la programación de audio en sistemas GNU/Linux.
- Entender las bases del análisis de sonidos en el dominio de la frecuencia.
- Utilizar un enfoque de análisis, diseño y codificación orientado a objetos, de una forma lo más clara y modular posible, para permitir ampliaciones y modificaciones sobre la aplicación por terceras personas.
- Hacer uso de herramientas básicas en el desarrollo de software, como son los **Sistemas de Control de Versiones** para llevar un control realista del desarrollo del software, así como hacer de las veces de sistema de copias de seguridad.

1.3. Alcance

oFlute se modela como una herramienta lúdico-educativa para alumnos que comiencen a aprender a usar la flauta dulce, proporcionando un entorno atractivo y ameno para el estudiante. Éstos tendrán la posibilidad de recorrer una serie de pequeñas lecciones sobre música en general, y el uso de la flauta dulce en particular.

Además, el usuario tendrá la posibilidad de comprobar sus conocimientos sobre el uso de la flauta practicando, gracias a las secciones de análisis de notas y de canciones, en las que la aplicación valorará la pericia del estudiante con la flauta.

1.3.1. Limitaciones del proyecto

El proyecto se limita al uso de la flauta dulce y no a otros instrumentos por la enorme variabilidad de timbre entre ellos, lo que supondría un enorme esfuerzo a la hora de generalizar el analizador de frecuencias.

El sistema de lecciones se basa en plantillas XML en las que es posible definir imágenes y texto para formar una pantalla de información. En un futuro se ampliará para incluir otros elementos multimedia así como lecciones con varias pantallas consecutivas.

Los sistemas de audio son una de las áreas en las que menos consenso hay entre plataformas informáticas, por lo que la transportabilidad de las aplicaciones suele ser compleja. El presente proyecto utiliza la API Simple de PulseAudio como subsistema de sonido, que es en teoría compatible con plataformas Win32, pero en la práctica su complejidad hace prácticamente inviable la portabilidad de la aplicación.

1.3.2. Licencia

El proyecto está publicado como software libre bajo la licencia GPL (General Public License) versión 2. El conjunto de bibliotecas y módulos utilizados tienen las siguientes licencias:

- A lo largo del proyecto se utilizan diferentes partes de las bibliotecas **Boost** [1], que utilizan la licencia *Boost Software License* [1]. Se trata de una licencia de software libre, compatible con la GPL, y comparable en permisividad a las licencias BSD y MIT.
- **Gosu** [6], la biblioteca de desarrollo de videojuegos que ha proporcionado el subsistema gráfico, utiliza la licencia MIT (Massachusetts Institute of Technology). Cuando se compila en sistemas Windows, utiliza la biblioteca FMOD que es gratuita pero de código cerrado; en sistemas GNU/Linux, utiliza SDL_mixer, que utiliza la licencia LGPL (Lesser General Public License).
- **Kiss FFT** [4], la biblioteca utilizada para hacer el análisis de frecuencias, utiliza una licencia BSD (Berkeley Software Distribution).
- **PugiXML** [5], biblioteca de procesamiento de ficheros XML, se distribuye bajo al licencia MIT.
- **PulseAudio** [3] utiliza una licencia LGPL 2.1.

[1] http://www.boost.org/LICENSE_1_0.txt

1.4. Estructura del documento

El presente documento se rige según la siguiente estructura:

- **Introducción**. Se exponen las motivaciones y objetivos detrás del proyecto **oFlute**, así como información sobre las licencias de sus componentes, glosario y estructura del documento.
- **Desarrollo del calendario**, donde se explica la planificación del proyecto, la división de sus etapas, la extensión de las etapas a lo largo del tiempo y los porcentajes de esfuerzo.
- **Investigación preliminar**, que explica las labores de documentación y experimentación previas al desarrollo, que han servido para labrar una base de conocimientos que nos diera las suficientes garantías para afrontar el proyecto.
- **Análisis**. Se detalla la fase de análisis del sistema, explicando los requisitos funcionales del sistema, los diferentes casos de uso, así como las principales operaciones con sus diagramas de secuencia y contratos.
- **Diseño**. Seguido del análisis, se expone en detalle la etapa de diseño del sistema, con los diagramas de clases.
- **Implementación**. Una vez analizado el sistema y definido su diseño, en esta parte se detallan las decisiones de implementación más relevantes que tuvieron lugar durante el desarrollo del proyecto.
- **Pruebas**. Listamos y describimos las pruebas que se han llevado a cabo sobre el proyecto para garantizar su fiabilidad y consistencia.

RELLENAR

Tras una revisión del calendario seguido, detallaremos a lo largo del resto de la memoria el proceso de análisis, diseño, codificación y pruebas que se siguió al realizar el proyecto.

Los manuales de usuario y de instalación se incluyen tras un resumen de los aspectos más destacables de proyecto y las conclusiones. En dicho manual, se hallan dos apartados dirigidos a la ampliación de la aplicación mediante la creación de nuevas lecciones y de nuevas canciones, respectivamente.

1.5. Glosario

1.5.1. Acrónimos

BSD Berkeley Software Distribution

DSP Digital Signal Processing

FDL Free Documentation License

FFT Fast Fourier Transform

FSF Free Software Foundation

GPL General Public License

LGPL Lesser General Public License

MIT Massachusetts Institute of Technology

1.5.2. Definiciones

Timbre Cualidad de un sonido que permite distinguir la misma nota producida por dos instrumentos musicales u orígenes diferentes.

2. Desarrollo del calendario

El proyecto no se ha desarrollado siguiendo un calendario estricto, dado que era imposible cuantificar el tiempo que tomaría el adquirir las bases teóricas necesarias para poder afrontarlo con garantías. Su desarrollo se ha compaginado con los estudios del último curso de Ingeniería Técnica en Informática de Sistemas y las labores como becario en la Oficina de Software Libre y Conocimiento Abierto de la Universidad de Cádiz [2].

2.1. Iteraciones

Para la realización del proyecto se ha utilizado un modelo de desarrollo iterativo incremental. En la redacción del presente documento se presentarán la fase de investigación preliminar y las etapas de análisis, diseño, implementación y pruebas del proyecto en su estado final. A continuación se detallan cada una de las iteraciones por las que ha ido pasando el proyecto.

2.1.1. Primera iteración: conocimientos preliminares

Antes de poder comenzar con el análisis y diseño del propio proyecto, era esencial adquirir una serie de conocimientos para poder afrontar su desarrollo con todas las garantías. Durante esta iteración, se llevaron a cabo labores de documentación y aprendizaje autodidacta con las que se asentaron los conocimientos necesarios.

Además, durante este periodo también se barajaron las diferentes posibilidades de implementación del sistema, así como las posibles herramientas y bibliotecas de terceros que pudieran ser de ayuda.

2.1.2. Segunda iteración: analizador básico

Una vez adquiridos los conocimientos teóricos necesarios, y decididas las técnicas y herramientas para llevar aquellos a la práctica, fue obvia la necesidad de empezar por diseñar un analizador de notas básico, que sería el corazón del programa. Del éxito del desarrollo temprano del módulo que se encargaría del análisis de sonidos dependería la viabilidad completa del proyecto.

2.1.3. Tercera iteración: interfaz gráfica de usuario

Con el módulo de análisis desarrollado, *sólo* restaba desarrollar el resto de la aplicación alrededor del mismo. En esta tercera iteración se propusieron numerosos diseños para la interfaz gráfica de usuario y, una vez decantados por uno de ellos, comenzó el desarrollo de los elementos de la interfaz, haciendo énfasis en conseguir un aspecto dinámico y jovial.

2.1.4. Cuarta iteración: motor de lecciones

Uno de los subproductos de la aplicación es el motor de lecciones, que presenta una serie de unidades didácticas en formato multimedia, compuestas de imágenes y textos, con conceptos sobre música. En esta iteración se hizo un análisis de las posibilidades de este motor, concluyendo con el diseño y desarrollo de un mecanismo muy sencillo de ampliar y utilizar.

2.1.5. Quinta iteración: motor de canciones

La parte de mayor interactividad de la aplicación es el motor de canciones, en el que el usuario tiene la posibilidad de tocar una canción que aparece en pantalla, usando la flauta, mientras la aplicación valora en tiempo real su interpretación. Durante la quinta iteración se elaboró este sistema, encargado de listar y cargar las diferentes canciones, y puntuar al usuario según cómo lo haga.

2.2. Diagrama de Gantt

por hacer

2.3. Porcentajes de esfuerzo

por hacer

3. Investigación preliminar

3.1. Adquisición de conocimientos

Para poder enfrentarnos con garantías al desarrollo del proyecto fue necesario adquirir una **base de conocimientos** que nos permitiera entender los conceptos que se iban a usar y las herramientas para trabajar con ellos. Así, fuimos guiándonos por la intuición y, sobre todo, por las necesidades que iban surgiendo.

Los conceptos que se presentan a continuación son básicos para comprender cómo funciona el módulo de análisis del proyecto.

3.1.1. El sonido

Un **sonido** es una vibración que se propaga por un medio elástico en forma de onda. Estas vibraciones se transmiten de forma longitudinal, esto es, en la misma dirección en la que se propaga la onda. El medio más común para la transmisión del sonido es el **aire**.

El sonido, en su forma más simple, se compone de una sola onda sinusoidal básica, con las características tradicionales: amplitud, frecuencia y fase. Una **onda sinusoidal** es aquella cuyos valores se calculan utilizando funciones seno.

Frecuencia y tono

La **frecuencia** mide el número de oscilaciones de la onda por unidad de tiempo. Por regla general, se utiliza el **hercio** como unidad de medida de frecuencia, que indica la cantidad de repeticiones por segundo. La frecuencia determinará la **altura** del sonido, es decir, cómo de grave o agudo es. Los sonidos graves tienen una frecuencia baja, mientras que los sonidos agudos tienen una frecuencia alta.

A lo largo de los años se ha establecido un estándar de referencia que establece que la nota *la* que se encuentra encima del *do* central del piano debe sonar a 440 hercios de frecuencia. Esta medida se utiliza a la hora de afinar los instrumentos, de modo que si al tocar la nota *la* se detecta un tono con una frecuencia de 440 hercios, entonces el instrumento estará bien afinado.

El espectro audible por las personas lo conforman las **audiofrecuencias**, esto es, el conjunto de frecuencias que pueden ser percibidas por el oído humano.

Figura 3.1.: Rango de frecuencias de sonido

Un oído sano y joven es capaz de detectar sonidos a partir de los 20 hercios. Los sonidos por debajo de esa frecuencia se conocen como **infrasonidos**. Por otro lado, el límite auditivo en frecuencias altas varía mucho con la edad: un adolescente puede oir sonidos con frecuencias hasta los 18kHz, mientras que un adulto de edad media solo suele llegar a captar sonidos de hasta 13kHz. El límite genérico superior se establece en 20kHz, por encima de los cuales los sonidos se denominan **ultrasonidos**.

Amplitud

La **amplitud** representa la energía que transporta la onda. Cuando un instrumento u otro objeto genera una vibración, la amplitud es la cantidad de movimiento que esa vibración genera. Podría equipararse (de forma no estricta) a la intensidad del sonido: cuanto mayor sea la amplitud, más fuerte se oirá el sonido.

Fase

Por último, la **fase** (φ) indica el desplazamiento horizontal de la onda respecto del origen. Si la fase de una onda no es cero, entonces parecerá que está *desplazada* hacia la derecha, si la fase es positiva, y hacia la izquierda si la fase es negativa.

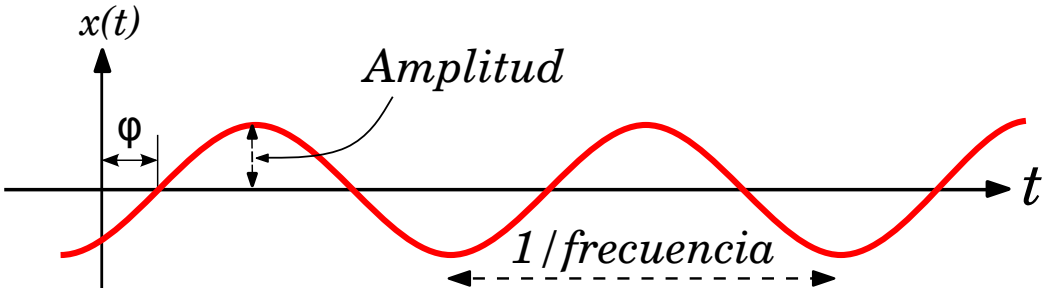

Figura 3.2.: Componentes de una señal senoidal básica

3.1.2. Descomposición de sonidos

Para desarrollar oFlute nos interesa conocer la altura de la nota que está tocando la flauta en un instante concreto. Para un tono puro, podríamos conocer la altura fijándonos en su frecuencia. El problema es que, en la naturaleza, **no existen** los tonos puros, sino que los sonidos se componen de multitud de tonos de diferentes amplitudes, frecuencias y fases.

Afortunadamente, la teoría dicta que cualquier tono complejo puede descomponerse como suma de tonos puros de distintas amplitudes, fases y frecuencias, llamados **parciales**. La menor de todas las frecuencias de los parciales se conoce como **frecuencia fundamental**, y es la que que dicta la altura general del sonido – *general*, ya que aunque el resto de frecuencias puede corresponder a otras notas, es la altura de la frecuencia fundamental la que mayor relevancia tiene en el sonido.

Un subconjunto de esos parciales, conocidos como **armónicos**, tienen frecuencias múltiplos de la frecuencia fundamental. Estos armónicos sirven para enriquecer el sonido y, sobre todo, determinar el **timbre musical** del origen del sonido: dos instrumentos (o personas) pueden estar tocando la misma nota y emitir la misma frecuencia fundamental, pero será el conjunto total de armónicos el que nos ayude a distinguir qué instrumento está emitiendo el sonido.

Así pues, el objetivo es encontrar una forma de descomponer una señal (el sonido) en sus componentes y analizar sus frecuencias, buscando la frecuencia fundamental, que nos informará de la nota que se está tocando.

Representación gráfica de sonidos

Las representación habitual de las señales se hace en el **dominio del tiempo**, es decir, podemos observar cómo la señal cambia a lo largo del tiempo, viendo el valor de su **amplitud** en cada instante. Por otro lado, la representación en el **dominio de la frecuencia** nos permite analizar una señal respecto a las frecuencias que la componen, dividiendo la señal en sus componentes.

En la figura 3.3 podemos comparar la representación de un sonido en el dominio del tiempo, en **forma de ondas**, tal y como aparecería en un osciloscopio, frente a su representación en **forma espectral**, en la que el eje vertical indica la frecuencia, y la intensidad del color indica la intensidad de esa componente frecuencial en el sonido.

Herramientas de descomposición de señales

La herramienta fundamental a la hora de descomponer una señal periódica, como puede ser un sonido, en sus parciales o armónicos es el **análisis armónico** o **análisis de Fourier**. Esta rama del análisis matemático estudia la representación de funciones o

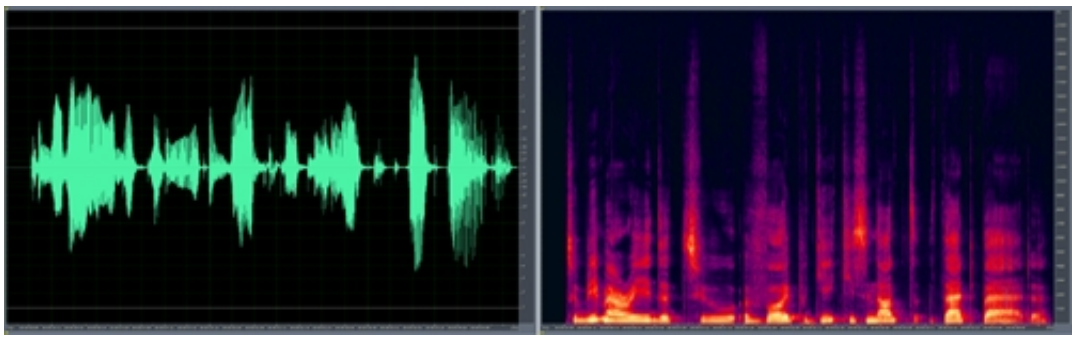

Figura 3.3.: Forma de ondas vs representación espectral

señales como superposición de ondas básicas, y hoy en día se aplica en innumerables campos de la ciencia, desde el procesamiento de señales para el reconocimiento de patrones, como es nuestro caso, a la neurociencia.

Una de las herramientas más conocidas de este área es la **transformada de Fourier**. Se trata de una aplicación matemática que descompone una función en su espectro de frecuencias a lo largo del dominio. Al aplicarla sobre una función f, se define de la siguiente manera:

$$g(\xi) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} f(x) e^{-i\xi x} dx$$

De cualquier modo, al estar tratando con un sistema digital como es una computadora, no es viable aplicar esta definición de la transformada de Fourier, ya que se basa en funciones continuas y derivables, y en nuestro caso dispondremos de datos discretos.

De ahí, aparece la **transformada discreta de Fourier** o **DFT**, que tiene el mismo uso que la transformada tradicional pero requiere que la función de entrada sea una secuencia discreta y de duración finita.

Existe un gran número de aproximaciones al cálculo de la transformada de Fourier, pero claramente el algoritmo más utilizado y eficiente es el **FFT**, **Fast Fourier Transform**. A pesar de imponer algunas limitaciones para mantener la eficiencia, el algoritmo FFT es la implementación que más habitualmente se encuentra en los chips DSP. Por regla general, computar la transformada de Fourier para N puntos usando FFT tardaría un tiempo $O(N \cdot log(N))$, mientras que hacerlo utilizando la definición estándar de la DFT llevaría un tiempo $O(N^2)$.

A pesar de que fue el **DFT** el algoritmo que se utilizó finalmente en el proyecto, se estudiaron otras posibles herramientas para la detección de la frecuencia fundamental, como por ejemplo la **función de autocorelación**, que también suele utilizarse en análisis de señales para encontrar patrones repetitivos, como señales enmascaradas por ruido. A pesar de ello, dada la poca bibliografía encontrada sobre estas técnicas secundarias y la conocida eficiencia de la transformada de Fourier, se decidió optar por la técnica más conocida.

3.1.3. Digitalización de sonidos

Antes de poder aplicar ninguna técnica sobre los sonidos, es necesario transformarlos de forma que el ordenador pueda trabajar con ellos.

Captación de sonidos

Lo más habitual a la hora de digitalizar un sonido es, primeramente, utilizar algún dispositivo que transforme las ondas sonoras en algo que pueda transmitirse al computador en forma de ondas eléctricas. Este dispositivo es el **micrófono**, en nuestro caso de tipo **electret**, que es el más utilizado en ordenadores personales, teléfonos móviles y demás dispositivos de consumo con requisitos de audio de media o baja fidelidad.

Estos micrófonos constan de una membrana que vibra libremente cuando capta cualquier onda acústica o de sonido, ya sea voz, música o ruidos, convirtiéndola en una señal eléctrica de baja frecuencia y de muy poca tensión o voltaje, semejante a la del sonido captado. Una vez que esta señal eléctrica llega a la tarjeta de sonido, comienza la siguiente parte del proceso.

Curiosamente, el proceso es el inverso del que ocurre en un altavoz. Es por eso que en el caso de algunos auriculares intrauditivos, como los que habitualmente acompañan a los reproductores MP3 de bolsillo, es posible utilizarlos como micrófonos de baja fidelidad. También es posible, aunque bastante más difícil, utilizar ciertos micrófonos de escritorio como altavoces improvisados, limitados a la reproducción de altas frecuencias.

Muestreo de la señal

El siguiente paso es el **muestreo** (o *sampling*) de la señal. El proceso consiste en medir la amplitud de la señal analógica en diferentes puntos, uniformemente espaciados, a lo largo del tiempo. El número de veces que se muestrea la señal por unidad de tiempo es conocido como **frecuencia de muestreo**, e influye directamente en la calidad de la digitalización del sonido.

La elección de la frecuencia de muestreo no suele ser trivial y tiene un impacto importante en el rendimiento y calidad del sistema, ya que el número de elementos a procesar es directamente proporcional a la frecuencia.

Otro factor importante es la clase de sonidos que vamos a digitalizar. Por regla general, los sonidos que se captan son los audibles por el oído humano. Tal y como se comentó en la sección anterior, estos sonido son aquellos cuyas frecuencias se encuentran por debajo de los 20 kHz. Existe un teorema dictado por el ingeniero sueco **Harry Nyquist** que defiende que *"la frecuencia de muestreo mínima requerida para muestrear una señal debe ser igual al doble de la máxima frecuencia contenida en la señal"*. En

nuestro caso, como la máxima frecuencia audible es de 20 kHz, lo normal será utiliar una frecuencia de muestreo de 40 kHz. El estándar de CD, que normalmente se utiliza como base de muestreo en la mayoría de tarjetas de sonido, amplía la tasa un 10 % con objeto de contemplar el uso de filtros no ideales, quedando la frecuencia de muestreo en 44,1 kHz.

Cuantificación de las muestras

Una vez decidida la frecuencia de muestreo de la señal, será necesario acordar qué utilizaremos para representar sus niveles de amplitud de forma digital. Este proceso se conoce como **cuantificación**, y de él se desprenderá el número de bits de cada muestra. Cabe notar que tanto el muestreo como la cuantificación son procesos con **pérdidas**, ya que es imposible discretizar con total fidelidad un rango continuo de tensiones.

Existen diferentes métodos para decidir los niveles a los que se ajustarán las muestras. El más utilizado es el **PCM - modulación por impulsos codificados** en su variante *uniforme*, que utiliza una escala uniforme para digitalizar los valores de amplitud, a diferencia de la versión *no uniforme*, que utiliza escalas como la logarítmica.

La **resolución de cuantificación** (o *resolución digital*) más habitual es de 16 bits, que es la utilizada en los CDs de audio. Esto nos permite tener $2^{16} = 65536$ niveles distintos con los que cuantizar cada muestra.

Codificación de las muestras

El último paso antes de tener los datos listos para el procesamiento es la **codificación** en forma de bits. Aunque podría parecer un proceso trivial – convertir los valores digitales de las muestras en binario – existen multitud de parámetros que influyen a la hora de representar estas señales:

- **Tamaño de la muestra**: decidido en la cuantificación, la muestra puede tener tamaños desde los 8 a los 64 bits.
- **Orden de los bytes**: para muestras de más de un byte, es importante decidir el orden de los mismos – ***endianness***. Popularmente, la mayoría de computadoras basadas en procesadores Intel utilizan *little endian* – esto es, se almacena primero los bytes de menos relevancia.
- **Signo**: cualquier señal en forma de ondas pasa constantemente por el origen, de forma que la amplitud toma valores positivos y negativos a cada momento. Puede parecer intuitivo usar un entero con signo para la representación, pero también es posible utilizar uno sin signo, de forma que el origen se represente como la mitad del rango, ahorrándonos así posibles complicaciones en la representación de números negativos.

- **Canales**: la mayoría de micrófonos de baja calidad producen sonido monoaural, de forma que solo es necesario utilizar un canal para su reproducción, a diferencia de los micrófonos estereofónicos que utilizan dos o más canales. En este aspecto, el flujo que se genera durante la digitalización de una señal estéreo es más complejo de procesar en tanto en cuanto los datos de cada canal vienen entrelazados en el flujo y, a veces, es difícil distinguirlos.

3.2. Estudio del software disponible

Existen algunas soluciones de software, juegos en la amplia mayoría de los casos, que explotan la idea del análisis de sonido en tiempo real como principal modo de interactuar con el usuario. En esta sección vamos a conocer algunas de estas soluciones y las ideas que adquirimos de su estudio.

3.2.1. Aplicaciones comerciales

SingStar

SingStar fue el primer videojuego en explotar el uso de un micrófono para que el usuario cantase y la aplicación reconociese el sonido. Apareció por primera vez en mayo de 2004 para sistemas PlayStation 2, y desde entonces han aparecido nada menos que 23 ediciones para este sistema y otras 6 para PlayStation 3.

La ventaja de SingStar es la que da ser el primero en explotar una idea atractiva, que rápidamente consiguió adeptos, principalmente entre el público más joven. Este éxito se vio fortalecido por la firma de una gran cantidad de contratos con discográficas a lo largo del tiempo, que permitió el lanzamiento de ediciones regulares con los *singles* más populares.

Las últimas ediciones de SingStar incluyen algoritmos avanzados que permiten, entre otras opciones, añadir efectos a las voces de los usuarios ó automáticamente limpiar las pista vocales de las canciones que los jugadores carguen mediante almacenamiento externo.

Lips

Lips fue la respuesta de Microsoft a SingStar para sus sistemas **Xbox 360**. El planteamiento es similar al de la versión de PlayStation, aunque incluye una serie de mejoras bastante atractivas.

Los micrófonos utilizados en Lips son inalámbricos e incluyen un sistema de detección de movimientos, de forma que es posible utilizarlos en secciones sin pista vocal

pero con percusión, siguiendo el ritmo a modo de maracas, o imitando movimientos que aparecen en pantalla.

Desde el principio, Lips ha permitido utilizar canciones de terceros mediante la conexión de un reproductor MP3. Esta opción solo estuvo disponible en sus competidores después de la aparición de Lips.

Además, Lips introdujo un sistema de juego colaborativo en forma de duetos, y competitivo, en el que los jugadores cantaban secciones consecutivas de una canción en busca de conseguir la mejor interpretación.

Apariciones menores en otros títulos

Aunque Lips y SingStar han sido los dos principales juegos del género, muchos otros juegos musicales han incluido pequeñas pruebas y minijuegos que han hecho uso de micrófonos. Por ejemplo, **DJ Hero**, **Guitar Hero** , **Band Hero** y **Def Jam Rapstar** permiten utilizar el micrófono para añadir acompañamiento vocal al juego. La ventaja es que en la mayoría de los casos, es posible utilizar los micrófonos de Lips y SingStar con estos juegos de terceros, evitando tener que adquirir más dispositivos.

3.2.2. Aplicaciones libres

UltraStar

UltraStar fue el primer clon libre de SingStar. Fue desarrollado por Patryk Cebula en 2007 y ha servido como base para diferentes forks posteriores. El juego permite a varias personas jugar a la vez mediante la conexión de varios micrófonos a una tarjeta de sonido, así como la adición de nuevas canciones de forma sencilla mediante ficheros de configuración en formato texto.

Aunque las versiones iniciales se liberaron bajo una licencia GNU GPL, desgraciadamente en la actualidad UltraStar se encuentra con licencia *freeware*, utilizando como excusa inválida que así *"[...] se protegen los datos privados de los usuarios al ser enviados al servidor mediante SSL"*. Realmente no existe razón para no utilizar software de código abierto con SSL.

UltraStar Deluxe

UltraStar Deluxe nació como una modificación básica de UltraStar, pero consiguió atraer la atención de muchos usuarios y desarrolladores, y finalmente se constituyó como un producto indepèndiente. Los desarrolladores de UltraStar Deluxe decidieron trabajar

en varios aspectos que vieron mejorables respecto al UltraStar original. Primero, mejorar la fiabilidad del programa, arreglando numerosos bugs y aumentando el rendimiento. Segundo, trabajar la apariencia visual, basándose en gran medida en los efectos del SingStar de PlayStation 3. Finalmente, facilitar la expansibilidad del sistema, permitiendo un gran número de formatos para los ficheros de vídeo y audio, y creando un sistema de scripting basado en Lua para los modos de juego colaborativos.

Performous

Performous es uno de los juegos musicales open source más populares. Nació como una reescritura del UltraStar original, aunque posteriormente lo superó con creces. La fortaleza de Performous reside en su capacidad de reconocimento de voces, basado en la *transformada rápida de Fourier (FFT)* y en una serie de algoritmos de post-procesamiento.

Performous ha evolucionado con el tiempo, naciendo como un juego de cante pero añadiendo características colectivas como Guitar Hero o Rock Band, permitiendo el uso de controladores adicionales, como guitarras o baterías electrónicas. Además, en las últimas versiones Performous incluye un modo de baile, muy similar a los clásicos **DDR** o **StepMania**.

3.3. Desarrollo con audio en GNU/Linux

El desarrollo de aplicaciones que realicen tareas de sonido en sistemas GNU/Linux es uno de los casos en los que más **dificultades** se encuentran. Tradicionalmente, el soporte del hardware de sonido en estos sistemas siempre ha sido de lo más básico, incluso limitándose, en ciertas ocasiones, a la reproducción de sonido, ignorando por completo la grabación. Afortunadamente, con el paso de los años el soporte ha ido mejorando gracias a la colaboración de los fabricantes y a la proliferación del sonido integrado en placa base.

A nivel de software, existen bastantes componentes diferentes, algunos alternativos y otros complementarios entre sí, que pueden conducir a confusiones. En otros sistemas operativos, como Windows o Mac OS, el programador cuenta con una interfaz de sonido común, que se encarga de la mezcla y de la comunicación de bajo nivel con la tarjeta de sonido. En GNU/Linux, dada su naturaleza modular, esa misma tarea se descompone en diferentes sistemas, por lo que una misma tarea puede realizarse de muchas formas distintas.

Buena muestra de ello es el artículo *Welcome To The Jungle* [1], en el que el desarrollador de Adobe, Mike Melanson, hace un repaso sobre la *jungla* que supone la pro-

[1] http://blogs.adobe.com/penguinswf/2007/05/welcome_to_the_jungle.html

gramación de audio en Linux. El artículo es antiguo y las cosas han mejorado desde entonces, pero aún así es muy fácil que los no iniciados se sientan abrumados por la cantidad de opciones disponibles.

PONER DIAGRAMA

La organización de las capas de software se asemeja al siguiente diagrama.

3.3.1. Interfaces de bajo nivel, OSS y ALSA

El elemento de menor nivel en esta *escala* de componentes son las interfaces de hardware, que podrían equipararse al *driver de audio* de Windows. En ambos casos se encuentran como módulos del kernel de Linux.

OSS

Open Sound System (OSS) fue durante muchos años la interfaz de audio por defecto en todos los sistemas GNU/Linux. Está basada en el stándard UNIX para la comunicación con dispositivos mediante las funciones POSIX habituales (`open`, `read`, etc), lo que la hace relativamente sencilla de utilizar.

Antiguamente, la mayor parte de los ordenadores personales con capacidades multimedia utilizaban tarjetas de sonido basadas en la Creative Sound Blaster 16. De hecho, las tarjetas de la competencia incluían modos de emulación de esta tarjeta. Su popularidad hizo que todos los esfuerzos en el desarrollo de audio en Linux se concentraran en dar soporte a esta tarjeta, surgiendo unos drivers de buena calidad. Finalmente, a la API generada se le dió el nombre de *Linux Sound API* y posteriormente, junto a los controladores de otras tarjetas, se empaquetó en lo que hoy es conoce por OSS.

Por desgracia, los desarrolladores de OSS decidieron privatizar el código. Aunque finalmente, en 2008, se volvió a liberar todo el código, para entonces su mayor rival, ALSA, ya había tomado su lugar como API predeterminada en el kernel de Linux.

ALSA

Como alternativa a OSS surgió **Advanced Linux Sound Architecture** (ALSA), que acabó colocándose como la alternativa por defecto en todos los sistemas GNU/Linux a partir de la versión 2.6 del kernel.

Entre sus características, ALSA permite la síntesis de sonidos MIDI mediante hardware, soporte multiprocesador, configuración automática de tarjetas de sonido, etcétera. En gran parte, los objetivos de ALSA fueron las deficiencias de OSS en aquella época.

ALSA está estructurada en tres componente. La primera parte son los controladores en el kernel. La segunda parte es una API para los desarrolladores. Esta API es de muy bajo nivel, y es utilizada principalmente por middlewares y frameworks en lugar de

por aplicaciones de usuario. Por último, el tercer componente es un mezclador que permite el multiplexado del sonido.

Curiosamente, tanto ALSA como OSS han incluído una capa de emulación del otro módulo, por lo que en un sistema con ALSA, los programas basados en OSS pueden funcionar, aunque la calidad varíe enormemente de un caso a otro.

3.3.2. Servidores de sonido

Como se comentó previamente, uno de los problemas principales de OSS es que no tenía mezclador, por lo que era imposible que varias aplicaciones emitieran sonido a la vez. Para arreglar este problema surgieron los **servidores de sonido**. La principal tarea de estos servidores es la de gestionar el acceso a los subsistemas de sonido, mezclando los flujos de sonido de las diferentes aplicaciones en uno solo, de forma que sea posible escuchar el sonido de varios programas al mismo tiempo.

Por otro lado, los servidores de sonido ofrecen una API más amigable, siendo más sencillo programar a través de ellos. Aunque existen multitud de servidores de sonido, los más conocidos son JACK y PulseAudio.

Aunque actualmente tanto ALSA como OSSv4 ofrece mezcla de audio, los servidores de sonido siguen usándose por sus otras características, aunque varios autores han declarado que en la mayoría de los casos son inútiles y solo empeoran el rendimiento en general y la latencia en particular.

JACK

JACK es un servidor de sonido de uso profesional que proporciona servicios de audio en tiempo real, consiguiendo latencias muy pequeñas.

Su nombre (*enchufe* en inglés) se debe a su arquitectura en forma de conexiones, subtitulándose a menudo *"Connection kit"*. Así, es posible hacer conexiones de flujo de audio entre aplicaciones y la interfaz de audio de igual forma que entre dos clientes, o con servidores de streaming online, etcétera.

Hay gran cantidad de aplicaciones que ofrecen JACK como forma de comunicación de audio, aunque su uso no está lo suficientemente extendido como para formar parte por defecto de ninguna distribución mayoritaria.

PulseAudio

PulseAudio es otro servidor de sonido, multi-plataforma, que ha ganado mucha popularidad en los últimos tiempos. Ofrece funcionalidades avanzadas, como audio

por red, control de volumen independiente por aplicación, ecualización del sonido a nivel global, etcétera.

Se utiliza de forma oficial en muchas distribuciones, como Ubuntu o Fedora, e incluso en dispositivos móviles de Nokia. Es de uso muy sencillo y se integra fácilmente con muchos back-ends, tanto ALSA y OSS, ya comentados previamente, como túneles RTP para la emisión por red.

PulseAudio permite también servir como reemplazo transparente de OSS, emulando el acceso directo a los dispositivos (como `/dev/dsp`) mediante la utilidad `padsp`. Además, existen muchas otras utilidades de línea de comandos para controlar PulseAudio. Por ejemplo, `pacmd` nos permite enviar comandos al demonio para

PulseAudio fue la opción que finalmente se ha utilizado en este proyecto, comentaremos los detalles más adelante.

3.3.3. Otras APIs

A los anteriores elementos, que en la mayoría de los casos son suficientes, hay que sumarles una serie de APIs y frameworks independientes que, en mayor o menor medida, han ido estableciéndose en distintos ámbitos:

- **GStreamer** es un framework basado en GObject muy ligado al proyecto GNOME. Su funcionamiento se basa en complementos cuyas salidas y entradas es posible conectar, de modo que podemos comenzar con un módulo de lectura de ficheros, pasar a un decodificador, luego a un módulo de efectos y finalmente a la salida (o *sink*).

  La herramienta `gst-launch` permite probar el funcionamiento de estos módulos desde la línea de comandos. Por ejemplo, podemos hacer un *loopback* (esto es, escuchar por los altavoces la entrada de audio, como el micrófono) mediante el siguiente comando:

  ```
  gst-launch-0.10 alsasource ! alsasink
  ```

- **SDL Mixer** es una biblioteca que forma parte de SDL (*Simple DirectMedia Layer*, un framework multimedia muy popular). Provee una API básica de reproducción de sonidos, y es muy utilizada en videojuegos por su facilidad de uso. El principal inconveniente es, precisamente, que su facilidad de uso se basa en un muy limitado rango de funciones. SDL Mixer no presenta ninguna capacidad de grabación o lectura de flujos de entrada, por lo que es imposible trabajar con micrófonos.

- **RtAudio, PortAudio**. Ambas bibliotecas de entrada y salida de audio, escritas en C/C++, multi-plataforma pero con algunos fallos. Inicialmente, el proyecto se basó en RtAudio, pero se encontraron bastantes problemas con la biblioteca. A

partir de ahí, se empezó a utilizar PortAudio, que funcionó bastante bien en las pruebas iniciales. Desafortunadamente, al comienzo del trabajo en el resto del proyecto se descubrieron problemas de estabilidad y finalmente se desechó.

4. Análisis

4.1. Metodología

oFlute ha seguido una metodología de desarrollo ágil en la que, mediante fases de desarrollo rápidas y ligeras, se intenta evitar los formales caminos de las metodologías tradicionales, enfocándose en las personas y los resultados.

4.2. Especificación de requisitos del sistema

4.2.1. Requisitos de interfaces externas

En esta sección describiremos los requisitos que deben cumplir las interfaces con el hardware, el software y el usuario.

En cuanto a la comunicación con el subsistema gráfico y de E/S, utilizaremos la biblioteca Gosu [6], un proyecto de software libre que proporciona un framework de desarrollo de videojuegos 2D, multiplataforma y muy sencillo de usar. Para el acceso al subsistema de audio, tal y como se ha comentado en la sección anterior, optamos por utilizar la API simple de PulseAudio.

oFlute dispondrá de una resolución fija de 800 por 600 píxeles, requisito fácilmente alcanzable en cualquier ordenador actual. Al tratar con un público objetivo joven, los gráficos y la interactividad deberán ser sencillos y fáciles de interpretar. Así, se ha trabajado en limitar la interacción del usuario con la aplicación al uso del ratón y, obviamente, del instrumento musical, en este caso la flauta dulce. La navegación resultante de este planteamiento queda reflejada en el siguiente diagrama:

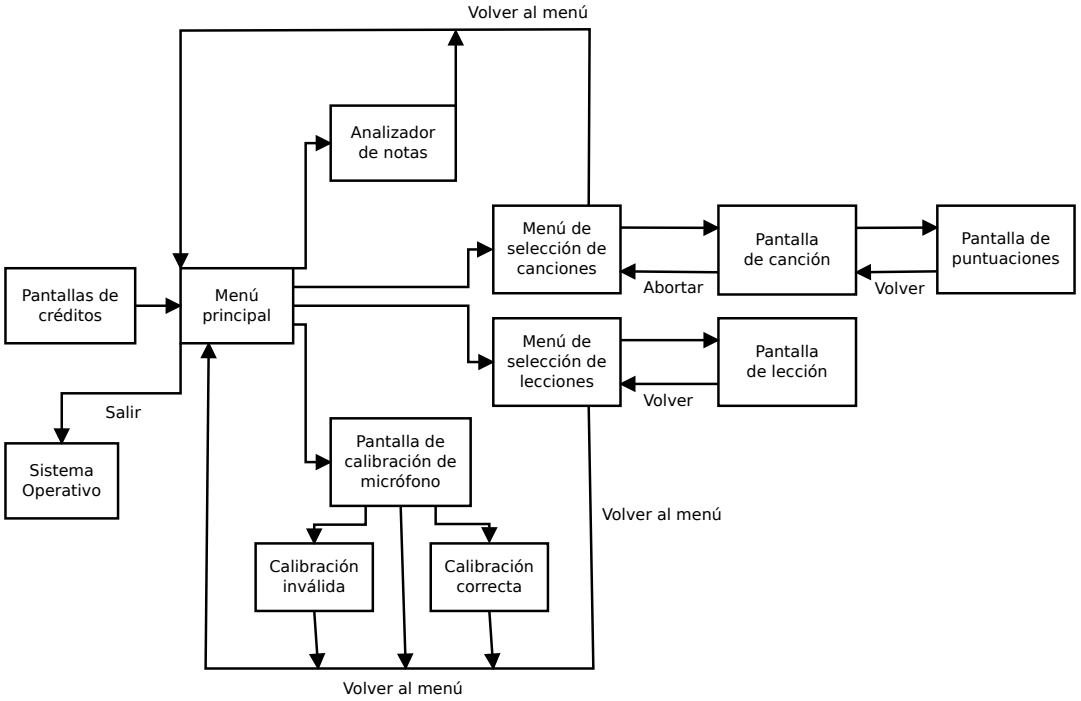

Figura 4.1.: Diagrama de flujo de las pantallas de oFlute

Inicialmente, deberán aparecer unas pantallas de crédito con información sobre el desarrollador y sobre el propio videojuego. Tras las mismas, que deberá ser posible omitir, habrá de aparecer el **menú principal**, con las cinco opciones posibles.

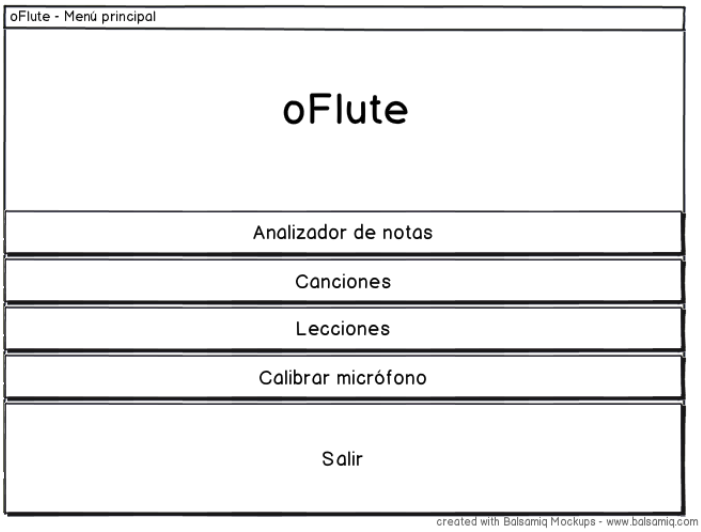

Figura 4.2.: Maqueta del menú principal

Las opciones que se incluirán son:

- **Analizador de notas**: comprobar las notas que tocamos sobre un pentagrama.
- **Canciones**: sección principal del juego, en el que aparecerán las canciones a tocar.
- **Lecciones**: sección de lecciones de aprendizaje.
- **Calibrar micrófono**, para ajustarse al nivel de ruido ambiental.
- **Salir** al sistema operativo.

La siguiente pantalla a modelar será el **analizador de notas**. Simplemente mostrará el logotipo del videojuego a un lado, y un pentagrama al otro, que se actualizará con la nota detectada por el micrófono. También contendrá un botón *volver* para ir al menú principal.

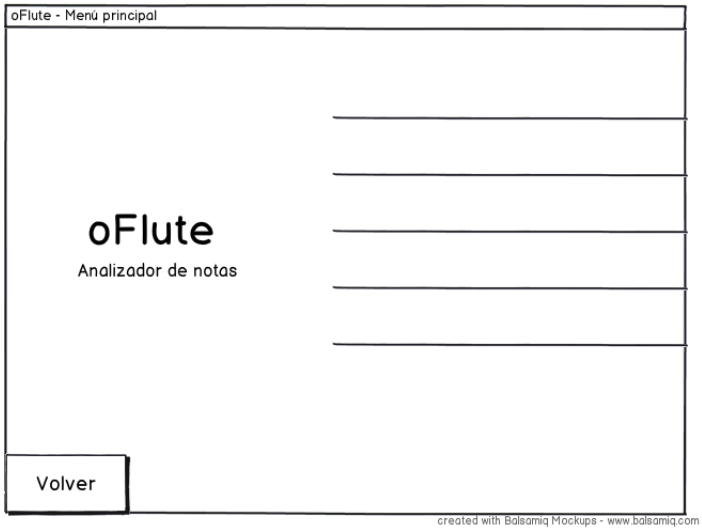

Figura 4.3.: Maqueta de la sección *analizador de notas*

La segunda sección a la que se podrá ir desde el menú principal será la de **canciones**. Inicialmente, la primera pantalla será la de **selección de canción**, que contendrá el logotipo del juego, un botón para volver al menú principal, y un menú dinámico de canciones que nos permitirá elegir el tema a interpretar.

Una vez seleccionada la canción, pasaremos a la zona de **interpretación de canción**. Contendrá un pentagrama que ocupará todo el ancho de la pantalla, con una línea que indicará la zona donde empezar a tocar las notas que aparezcan. Además, en la parte superior habrá un indicador con la puntuación obtenida y, abajo, una barra de progreso que nos indicará cuánto queda de canción.

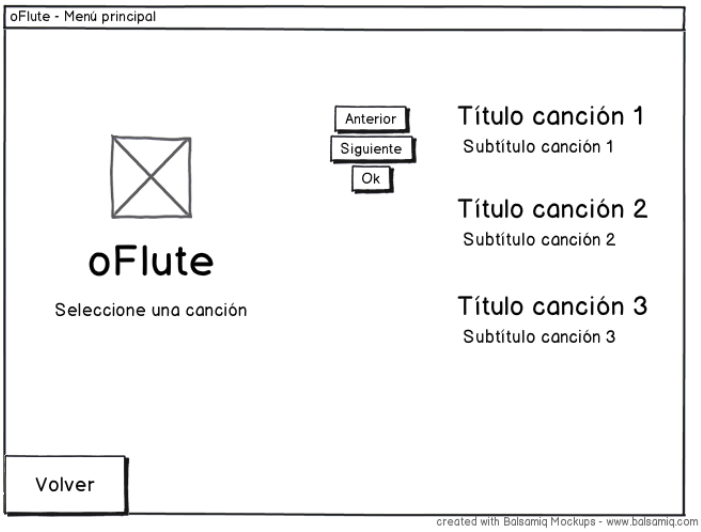

Figura 4.4.: Maqueta del menú de selección de canción

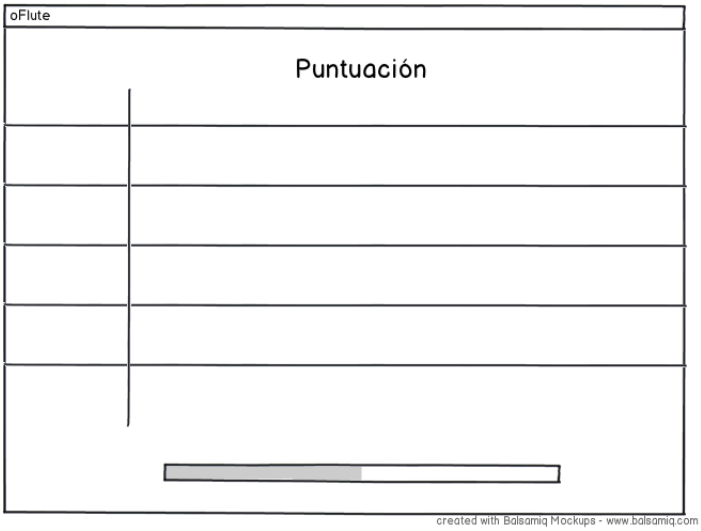

Figura 4.5.: Maqueta de la pantalla de interpretación de canción

Al completar la interpretación de la canción, aparecerá la **sección de resultados**. Contendrá el logotipo del juego, el título y subtítulo de la canción, y un cuadro con información sobre nuestra interpretación, representada en forma de porcentaje de aciertos. Además, en la zona inferior aparecerá un mensaje de ánimo dependiendo del resultado obtenido.

La pantalla de **selección de lecciones**, a la que se llega desde el menú principal, contendrá el título, una imagen decorativa, y varios botones para navegar entre las lecciones cargadas en el sistema. Se mostrará el título y la descripción de cada lección, así como un botón para comenzar.

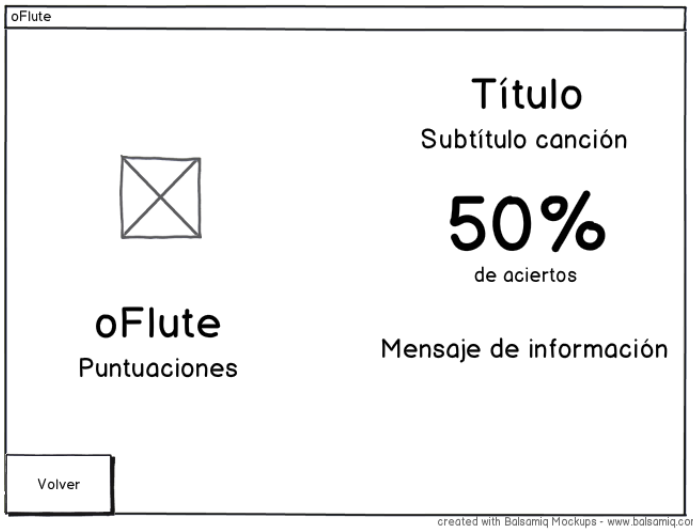

Figura 4.6.: Maqueta de la pantalla de puntuaciones

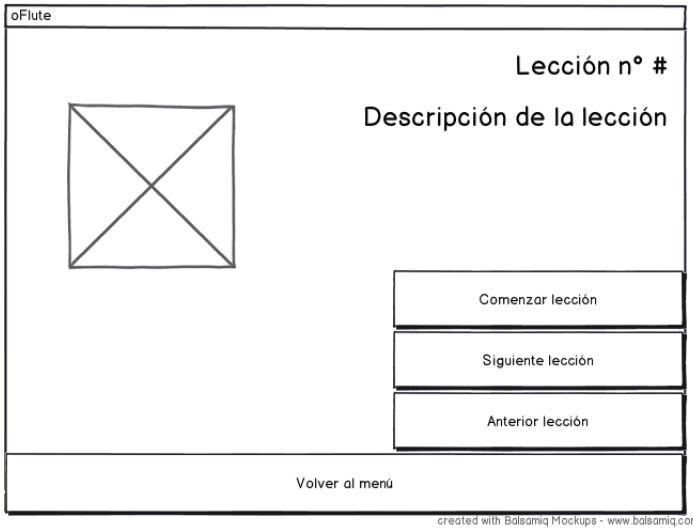

Figura 4.7.: Maqueta del menú de selección de lecciones

Una vez elegida una lección, pasaremos a la pantalla de reproducción de lecciones. Dada la naturaleza **dinámica** de esta sección, cada lección podrá tener una apariencia y elementos distintos. El único elemento común entre todas las lecciones será el botón de **volver al menú**.

4.2.2. Requisitos funcionales

oFlute se basa en los siguientes requisitos funcionales:

- Poder terminar la aplicación pulsando el botón de cierre en cualquier instante.
- Comprobar la correcta interpretación de notas individuales mediante el analizador de notas.
- Calibrar el micrófono de forma que el sistema se pueda adaptar al ruido ambiental del entorno.
- Navegar por toda la aplicación de forma sencilla utilizando solo el ratón.
- Elegir entre varias canciones a interpretar, cada una con su título y subtítulo informativos.
- Interpretar las canciones mediante el uso de la flauta, siguiendo el pentagrama en pantalla.
- Elegir entre bastantes lecciones informativas, poder ejecutarlas y seguirlas.
- Capacidad de añadir nuevas lecciones y canciones de forma sencilla.

4.2.3. Requisitos de rendimiento

La aplicación **oFlute** precisa de unos requisitos bastante básicos, que en su mayor parte se reducen a cuatro puntos principales:

- Posesión de una tarjeta de sonido o subsistema de audio similar con un micrófono, para poder captar el sonido de la flauta.
- Pantalla con una resolución de, al menos, 800 por 600 píxeles.
- Sistema gráfico compatible con OpenGL.
- Dispositivo apuntador, como un ratón.

La práctica totalidad de los ordenadores personales de la actualidad cumplen los citados requisitos.

4.2.4. Requisitos de diseño

¿?¿?¿?

4.2.5. Requisitos del sistema software

El sistema de software deberá cumpliar los requisitos siguientes:

- Deberá funcionar en cualquier sistema **GNU/Linux** con los requisitos anteriormente indicados.
- Deberá limitarse el número de dependencias, así como facilitar al máximo la instalación de las que resultasen imprescindibles.

- El uso del teclado quedará en segundo plano, haciendo posible utilizar la aplicación completamente con el ratón.
- Al tratarse de un público objetivo juvenil, la aplicación deberá ser dinámica, intuitiva y fácil de usar, y la apariencia debe ser agradable.
- Se evitará el uso de constantes y recursos dentro del código de la aplicación, utilizando como alternativa ficheros para representar las lecciones y las canciones.

4.3. Modelo de casos de uso

A la hora de modelar los casos de uso del sistema, hemos optado por utilizar notación *UML*, siguiendo los siguientes pasos:

- Identificación de los usuarios del sistema y sus roles.
- Para cada rol, determinar las formas de interactuar con el sistema.
- Creación de casos de uso para los objetivos que debe cumplir la aplicación.
- Modularización de los casos de usos mediante la implementación de relaciones de inclusión o extensión.

4.3.1. Diagrama de casos de uso

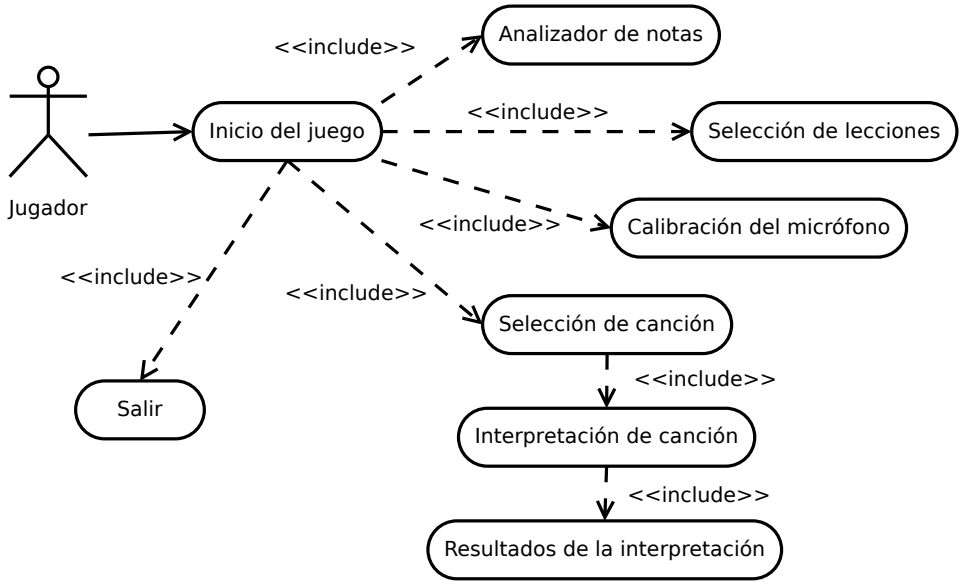

Figura 4.8.: Diagrama de casos de uso

4.3.2. Descripción de los casos de uso

Caso de uso: inicio del juego

Descripción Se muestran los créditos del juego, la pantalla de presentación, y finalmente el menú principal, desde donde se accederá al resto de secciones del juego.

Actores *Jugador*.

Precondiciones Ninguna.

Postcondiciones Ninguna.

Escenario principal

1. El *Jugador* inicia la aplicación.
2. El *Sistema* inicializa el subsistema gráfico.
3. El *Sistema* muestra la pantalla de créditos y la pantalla de presentación de la aplicación.
4. El *Sistema* muestra el menú principal en la pantalla.
5. El *Jugador* selecciona la opción *Canciones*.
6. El *Sistema* accede a la pantalla de *Selección de canción*.

Extensiones — flujo alternativo

***a** El *Jugador* cierra la ventana.

1. El *Sistema* libera los recursos y sale de la aplicación.

4a El *Jugador* selecciona la opción *Analizador de Notas*.

1. El *Sistema* accede a la pantalla del analizador de notas.

4b El *Jugador* selecciona la opción *Lecciones*.

1. El *Sistema* accede a la pantalla de *Selección de lecciones*.

4c El *Jugador* selecciona la opción *Calibrar micrófono*.

1. El *Sistema* accede a la pantalla de calibración de micrófono.

4d El *Jugador* selecciona la opción Salir.

1. El *Sistema* libera los recursos y sale de la aplicación.

Caso de uso: selección de canción

Descripción Al *Jugador* se le muestra una lista de las canciones detectadas, y éste debe elegir entre ellas la que desea interpretar, o volver al menú princial.

Actores *Jugador*.

Precondiciones Ninguna.

Postcondiciones Una canción queda seleccionada.

Escenario principal

1. El *Jugador* accede, desde el menú principal, al panel de selección de canciones.
2. El *Sistema* busca las canciones dadas de alta en el juego y muestra un menú con las mismas.
3. El *Jugador* navega entre las canciones listadas y selecciona una de ellas, pulsando finalmente el botón *Ok*.
4. El *Sistema* carga la canción y pasa a la pantalla de interpretación de canciones.

Extensiones — flujo alternativo

***a** El *Jugador* cierra la ventana.

1. El *Sistema* libera los recursos y sale de la aplicación.

3a El *Jugador* selecciona la opción *Volver*.

1. El *Sistema* muestra la animación de cierre y vuelve al menú principal.

Caso de uso: interpretación de canción

Descripción Tras haber elegido la canción a interpretar, se muestra una partitura con las notas que el *Jugador* deberá tocar para conseguir la puntuación deseada.

Actores *Jugador*.

Precondiciones Se ha elegido una canción.

Postcondiciones Se completa la interpretación de la canción, obteniendo una calificación

Escenario principal

1. El *Sistema* carga la canción, leyendo las notas, y muestra en pantalla, mediante animaciones, el marcador de puntos y el pentagrama.

2. El *Sistema* comienza a mostrar notas en el pentagrama, que van deslizándose hacia el lado izquierdo, en el que se encuentra la aguja de reproducción, e inicia el análisis del sonido.
3. El *Jugador* al llegar la nota a la aguja de reproducción, toca la flauta con la altura y la duración correcta, de forma que el micrófono sea capaz de captar el sonido.
4. El *Sistema* analiza el sonido que captura el micrófono y detecta la nota que toca el usuario.
5. El *Sistema* determina que la nota es la correcta y suma los puntos correspondientes.
6. Mientras existan más notas, se vuelve al punto 2.
7. El *Sistema* determina que no hay más notas que mostrar, e inicia las animaciones para ocultar los elementos en pantalla.
8. El *Sistema* pasa a la sección de *Resultados de la interpretación*.

Extensiones — flujo alternativo

***a** El *Jugador* cierra la ventana.

1. El *Sistema* libera los recursos y sale de la aplicación.

***b** El *Jugador* pulsa la tecla `escape`.

1. El *Sistema* vuelve a la pantalla de *selección de canción*.

3a El *Jugador* toca el instrumento con intensidad insuficiente o nula y el sonido no llega al sistema.

1. El *Sistema* representa esta inconsistencia como un silencio.

4a El *Sistema* no es capaz de determinar fehacientemente la nota que toca el usuario.

1. El *Sistema* representa esta inconsistencia como un silencio.

5a El *Sistema* determina que la nota tocada por el usuario no es la que corresponde a la partitura.

1. El *Sistema* ignora esta situación y no suma los puntos al marcador.

Caso de uso: resultados de la interpretación

Descripción Después de interpretar las notas de la partitura, se muestran los datos obtenidos del análisis de las notas tocadas por el *Jugador*.

Actores *Jugador*.

Precondiciones Se ha elegido e interpretado una canción.

Postcondiciones Se completa la partida actual.

Escenario principal

1. El *Sistema* compara la puntuación conseguida con la máxima puntuación obtenible, y genera un porcentaje de aciertos.
2. El *Sistema* muestra, mediante animaciones, un mensaje con información sobre la canción y sobre la interpretación del *Jugador* representada mediante un porcentaje de aciertos.
3. El *Sistema* muestra un mensaje variable en función del número de aciertos conseguido.
4. El *Jugador* revisa su puntuación y pulsa el botón *volver* para ir de vuelta al menú de *selección de canción*.

Extensiones — flujo alternativo

***a** El *Jugador* cierra la ventana.

1. El *Sistema* libera los recursos y sale de la aplicación.

4a El *Jugador* pulsa la tecla `escape`.

1. El *Sistema* vuelve a la pantalla de *selección de canción*.

Caso de uso: analizador de notas

Descripción El *Jugador* elige la opción *analizador de notas* en el menú principal y es llevado a esta sección, en la que el sistema representará gráficamente la nota que esté tocando con la flauta en cada instante, sin otra interacción

Actores *Jugador*.

Precondiciones Ninguna.

Postcondiciones Ninguna

Escenario principal

1. El *Jugador* accede, desde el menú principal, al panel del analizador de notas.
2. El *Sistema* muestra, mediante animaciones, la pantalla de la sección, representada mediante una fracción de partitura en la que se representará la nota que esté tocando el *Jugador* en cada momento.
3. El *Sistema* inicia el análisis del sonido.
4. El *Jugador* toca la nota que desee con su flauta, de forma que el micrófono sea capaz de captar el sonido.

5. El *Sistema* analiza el sonido que captura el micrófono y detecta la nota que toca el usuario.
6. El *Sistema* muestra en pantalla la nota, sobre la partitura, correspondiente a lo que ha tocado el usuario.
7. Se repite el flujo desde el punto 4, mientras el *Jugador* no pulse en el botón volver.
8. El *Jugador* pulsa en el botón *volver*.
9. El *Sistema* inicia las animaciones para ocultar los elementos en pantalla.
10. El *Sistema* vuelve al menú principal.

Extensiones — flujo alternativo

***a** El *Jugador* cierra la ventana.

1. El *Sistema* libera los recursos y sale de la aplicación.

***b** El *Jugador* pulsa la tecla `escape`.

1. El *Sistema* vuelve al menú principal.

4a El *Jugador* toca el instrumento con intensidad insuficiente o nula y el sonido no llega al sistema.

1. El *Sistema* representa esta inconsistencia como un silencio.

5a El *Sistema* no es capaz de determinar fehacientemente la nota que toca el usuario.

1. El *Sistema* representa esta inconsistencia como un silencio.

Caso de uso: calibración de micrófono

Descripción El *Jugador* elige la opción *calibrar micrófono* en el menú principal y es llevado a esta sección, en la que el *Sistema* calibrará el micrófono de forma que sea posible aislar el sonido de la flauta del ruido ambiental.

Actores *Jugador*.

Precondiciones Ninguna.

Postcondiciones El *Sistema* obtiene un valor umbral con el que discernir entre el sonido del instrumento y el ruido ambiente.

Escenario principal

1. El *Jugador* accede, desde el menú principal, al panel de calibración del micrófono.

2. El *Sistema* muestra la sección, indicando con un mensaje que el usuario debe pulsar la tecla `escape` para iniciar la calibración.
3. El *Jugador* pulsa la tecla `escape` y se mantiene en silencio.
4. El *Sistema* inicia el análisis del sonido, guardando durante dos segundos los valores de ruido que lee del micrófono.
5. El *Sistema* calcula, a partir de los valores leídos, el umbral de ruido, y muestra un mensaje informando del final del proceso.
6. El *Jugador* pulsa la tecla `escape` y el *Sistema* vuelve al menú principal.

Extensiones — flujo alternativo

***a** El *Jugador* cierra la ventana.

1. El *Sistema* libera los recursos y sale de la aplicación.

***b** El *Jugador* pulsa la tecla `escape`.

1. El *Sistema* cancela la calibración y vuelve al menú principal.

5a El *Sistema* encuentra valores inválidos al leer el ruido ambiental.

1. El *Sistema* informa al usuario del fallo del proceso de calibración.
2. El *Jugador* pulsa la tecla `escape` y el *Sistema* vuelve al menú principal.

Caso de uso: selección de lecciones

Descripción El *Jugador* elige la opción *lecciones* en el menú principal y es llevado a esta sección, en la que el *Sistema* mostrará una lista de lecciones cargadas, entre las que el usuario deberá elegir.

Actores *Jugador*.

Precondiciones Ninguna.

Postcondiciones Se ha elegido una lección

Escenario principal

1. El *Jugador* accede, desde el menú principal, al panel de selección de lecciones.
2. El *Sistema* carga la lista de secciones y muestra, mediante animaciones, el panel, preseleccionando por defecto la primera lección.
3. El *Jugador* utiliza los botones de la sección para elegir una de las lecciones, y activarla pulsando *comenzar lección*.
4. El *Sistema* oculta de forma animada el panel de selección de lecciones.

5. El *Sistema* lee el fichero `xml` asociado a la lección indicada, cargando los elementos que la componen y las animaciones que se ejecutarán.
6. El *Sistema* ejecuta las animaciones correspondientes a los elementos multimedia de la lección.

Extensiones — flujo alternativo

***a** El *Jugador* cierra la ventana.

1. El *Sistema* libera los recursos y sale de la aplicación.

2a El *Sistema* detecta que una de las lecciones leídas no está correctamente construída.

1. El *Sistema* informa del error en el *log* del programa y omite la carga de esa lección.

3a El *Jugador* pulsa la tecla `escape`.

1. El *Sistema* vuelve al menú principal.

6a El *Jugador* pulsa la tecla `escape`.

1. El *Sistema* vuelve al menú de selección de lecciones.

4.4. Modelo conceptual de datos

El modelo conceptual de datos representa, de forma esquemática, las clases que modelan el sistema y las relaciones que existen entre ellas, además de una pequeña introducción a su utilidad.

Juego Clase de control general. Gestiona el flujo de ejecución principal, así como de la gestión de estados, que permite pasar de una sección a otra del juego.

Estado Clase base para los diferentes estados del juego. Las clases correspondientes a las secciones se basarán en esta clase para interactuar con el gestor de estados y poder pasar de una parte del juego a otra.

EstadoMenú Representa el estado de juego para el menú principal, desde el que se accede al resto de opciones del juego.

EstadoAnalizador Representa el estado del analizador básico de notas. Contendrá los elementos necesarios para iniciar el análisis del audio, así como los elementos de la interfaz.

EstadoCalibrarMicro Representa el estado en el que se calibra el micrófono. Al igual que la clase *EstadoAnalizador*, deberá ser capaz de acceder al sistema de audio para poder leer el volumen ambiente y así calibrar el micrófono.

EstadoImagenFija Modela una imagen fija a modo de pantalla de créditos, de forma que sea sencillo añadir imágenes al inicio del juego, como firmas de desarrolladores, logotipos de patrocinadores, etcétera.

EstadoMenúCanciones Comprende el menú de selección de canciones, que se encargará de leer los ficheros de canciones disponibles. Además, también se encargará de lanzar las canciones en forma de estados secundarios.

EstadoMenúLecciones Corresponde al menú de elección de lecciones, que leerá y listará los ficheros de lección disponibles, y se encargará de lanzar la lección elegida.

Analizador Controla la gestión del subsistema de audio y el análisis de la entrada. Deberá proporcionar información sobre el volumen de la entrada (para la calibración del micrófono) así como de la nota detectada en cada instante.

Animación Se encargará de facilitar la creación de animaciones en forma de interpolación de valores, válidas para cambios de posición, opacidad, etcétera.

Elemento Esta clase de ayuda facilitará la carga y dibujado de elementos para la interfaz, además de servir de capa de abstracción para las animaciones.

ElementoImagen Especialización de la clase *Elemento* para imágenes.

ElementoTexto Especialización de la clase *Elemento* para textos.

ElementoCombinado Especialización de la clase *Elemento* que combina imagen y texto, a usar en casos como los botones del menú.

SistemaPartículas Representa un sistema de partículas simple, para generar efectos de destellos y fuegos artificiales.

Canción Corresponde a la canción que se va a interpretar, lanzada desde el estado *EstadoMenuCanciones*.

Nota Simboliza cada una de las notas cargadas que componen una canción.

DIAGRAMA de clases conceptuales

4.5. Modelo de comportamiento del sistema

En esta sección vamos a especificar cómo se comporta el sistema en forma de dos elementos fundamentales.

- En primer lugar, los **diagramas de secuencia** mostrarán el flujo de eventos entre los actores que participan en la aplicación.
- En segundo lugar, los **contratos de las operaciones** detallarán las condiciones y efectos que tendrán lugar al ejecutarse las operaciones en el sistema.

5. Diseño

wtf

6. Implementación

Un análisis claro y un diseño conciso no garantizan que, a la hora de implementar el sistema planteado, no se encuentre ninguna dificultad o imprevisto. Así pues, en este capítulo comentaremos los retos y detalles que más relevancia o complejidad han presentado durante la fase de la implementación del proyecto.

De igual modo, durante el desarrollo de la aplicacíon se mantuvo actualizada una bitácora, accesible en línea[1], en la que se fueron detallando, a medida que aparecían, muchas de estas dificultades.

Como complemento a la lectura de este capítulo se recomienda tener una copia local del repositorio del proyecto, disponible para su libre descarga desde la forja oficial[2]. En él se encuentra todo el código fuente original, así como la documentación en formato *Doxygen*.

6.1. Carga y uso de fuentes TrueType en Gosu

Como se ha comentado previamente, **oFlute** hace uso de la biblioteca *Gosu*, que proporciona una API sencilla para el acceso al sistema gráfico, entre otras características. Este framework funciona en sistemas Windows, GNU/Linux y Mac OS, aunque dada la dificultad de conseguir la compatibilidad con todos ellos, la calidad y el rendimiento es bastante desigual.

Una de estas desigualdades se presentaba a la hora de cargar fuentes para mostrar textos. En su versión para GNU/Linux, Gosu **no permitía** utilizar fuentes que no estuviesen instaladas en el sistema, esto es, era imposible adjuntar un fichero con una fuente en formato `TrueType` para su carga en el juego. Sin embargo, tanto en Windows como en Mac OS esta carga sí era posible. Esto supuso un grave problema en el planteamiento del proyecto. REFERENCIA

Inicialmente se investigaron las razones de esta limitación. Las conclusiones que se sacaron fueron que Gosu, bajo GNU/Linux, implementaba el renderizado de fuentes mediante una biblioteca llamada Pango , de bastante bajo nivel, y que por diseño está limitada al uso de fuentes de sistema, ya que su uso se orienta a herramientas del sistema operativo, no a aplicativos de terceros. REFERENCIA

[1] http://oflute.wordpress.com
[2] http://oflute.googlecode.com

Así pues, era necesario buscar una alternativa. Basándonos en la experiencia previa con otras bibliotecas de desarrollo, se pensó en `SDL_ttf`, una de las partes de la conocida biblioteca SDL , ampliamente utilizada en el desarrollo de aplicaciones multimedia en toda clases de sistemas.

REFERENCIA

7. Pruebas

8. Conclusiones

A. Herramientas utilizadas

B. Manual de instalación

C. Manual de usuario

D. Manual para añadir nuevas lecciones

E. Manual para añadir nuevas canciones

Bibliografía

[1] *Boost C++ libraries*. URL http://www.boost.org.

Boost es un conjunto de bibliotecas para C++ que ofrecen soluciones a todo tipo de problemas. Están escritas por los mejores desarrolladores de C++, y diez de estas bibliotecas formarán parte del nuevo estándar C++0x.

[2] *Oficina de Software Libre y Conocimiento Abierto de la Universidad de Cádiz*. URL http://osl.uca.es.

Oficina de Software Libre y Conocimiento Abierto de la Universidad de Cádiz, en la que trabajé como becario durante el desarrollo del proyecto, realizando labores de organización y gestión de eventos, administración de software y asistencia técnica.

[3] *PulseAudio*. URL http://pulseaudio.org.

PulseAudio es un servidor de sonido multiplataforma, compatible con sistemas GNU/Linux y Windows, y utilizado en algunas de las distribuciones más conocidas, como Ubuntu, Fedora, Mandriva, openSuse y Linux Mint.

[4] Mark Borgerding. *Kiss FFT*. URL http://sourceforge.net/projects/kisfft.

Kiss FFT es una biblioteca para realizar Transformadas Rápidas de Fourier (FFT (Fast Fourier Transform)). Es una biblioteca muy pequeña, razonablemente eficiente y portable con capacidad para realizar operaciones en distintos formatos.

[5] Arseny Kapoulkine. *Pugixml*. URL http://code.google.com/p/pugixml.

PugiXML es una biblioteca ligera para el procesamiento de archivos XML en C++. Tiene soporte completo Unicode, un parseador muy veloz y capacidad para usar consultas XPath.

[6] Julian Raschke y otros. *Gosu*. URL http://libgosu.org.

Gosu es una biblioteca de desarrollo de videojuegos 2D para Ruby y C++, con aceleración gráfica por OpenGL y orientación a objetos.

F. GNU Free Documentation License

ACTUALIZAR LA GFDL, está por lo menos la 1.3