ESCUELA SUPERIOR DE INGENIERÍA

Ingeniería Técnica en Informática
de Sistemas

oFlute: reconocimiento de señales aplicado al aprendizaje de la flauta dulce

José Tomás Tocino García

Cádiz, 7 de septiembre de 2011

ESCUELA SUPERIOR DE INGENIERÍA

Ingeniería Técnica en Informática de Sistemas

oFlute: reconocimiento de señales aplicado al aprendizaje de la flauta dulce

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

Cádiz, 7 de septiembre de 2011

Fdo.: José Tomás Tocino García

Agradecimientos

Quiero agradecer y dedicar la presente memoria y, por extensión, el proyecto oFlute al completo:

- A Julian Raschke por crear y mantener Gosu.
- A Manuel Palomo Duarte y Rafael Rodríguez Galván, de la OSLUCA, por confiar en mí como becario de la oficina, apoyarme y darme facilidades para desarrollar el proyecto durante la beca.
- A Antonio García Domínguez, como co-tutor del proyecto, por sus acertados consejos, su visión para descubrir cosas mejorables y su implicación con el software y el conocimiento libre, gracias a la cual esta memoria fue posible.
- A mis compañeros más cercanos de la carrera, que me han apoyado y de los que me he nutrido para desarrollar este proyecto.
- A mi familia y mi pareja, que finalmente no tendrán que recibir como excusa el desarrollo del proyecto a la hora de dedicarles el tiempo que se merecen.
- A Robert Penner por su magnífico conjunto de ecuaciones para animaciones, sin las que oFlute tendría una interfaz mucho más pobre.
- A los que apuestan por el software libre y el conocimiento abierto. Sus herramientas, bibliotecas y documentos han hecho posible que oFlute llegase a buen puerto.

Índice general

Índice de figuras

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 interactivos, 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, en la flauta dulce, un instrumento económico y fácil de aprender que se usa tradicionalmente en la educación musical obligatoria en España.

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 emitidas por una flauta dulce y capturadas mediante

un micrófono en tiempo real.

- Crear una aplicación de usuario que identifique y muestre en pantalla las notas que toca el usuario con la flauta dulce 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.
- Adquirir soltura en la programación de audio bajo 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 3. El conjunto de bibliotecas y módulos utilizados tienen las siguientes licencias:

- A lo largo del proyecto se utilizan diferentes partes de las bibliotecas **Boost** [6], 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** [58], 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** [39], la biblioteca utilizada para hacer el análisis de frecuencias, utiliza una licencia BSD (Berkeley Software Distribution).
- **PugiXML** [48], biblioteca de procesamiento de ficheros XML, se distribuye bajo la licencia MIT.
- **PulseAudio** [29] 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.
- **Fundamentos**, 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.
- **Conclusiones**. Comentamos las conclusiones a las que se han llegado durante el transcurso y al término del proyecto.

Y los siguientes apéndices:

- **Herramientas utilizadas**, donde detallamos qué hemos usado para la elaboración del proyecto.
- **Manual de instalación** del proyecto en sistemas nuevos.
- **Manual de usuario**, donde se explica cómo usar la aplicación.
- **Manual para añadir nuevas canciones** a **oFlute**.
- **Manual para añadir nuevas lecciones** a **oFlute**.
- **Tutorial de traducción de proyectos con GNU Gettext**, utilizado a lo largo del proyecto.

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.

1.4.1. 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

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 [25].

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

Se ha diseñado un diagrama de Gantt para reflejar la distribución de las tareas a lo largo del tiempo (figura 2.1 en la página 25).

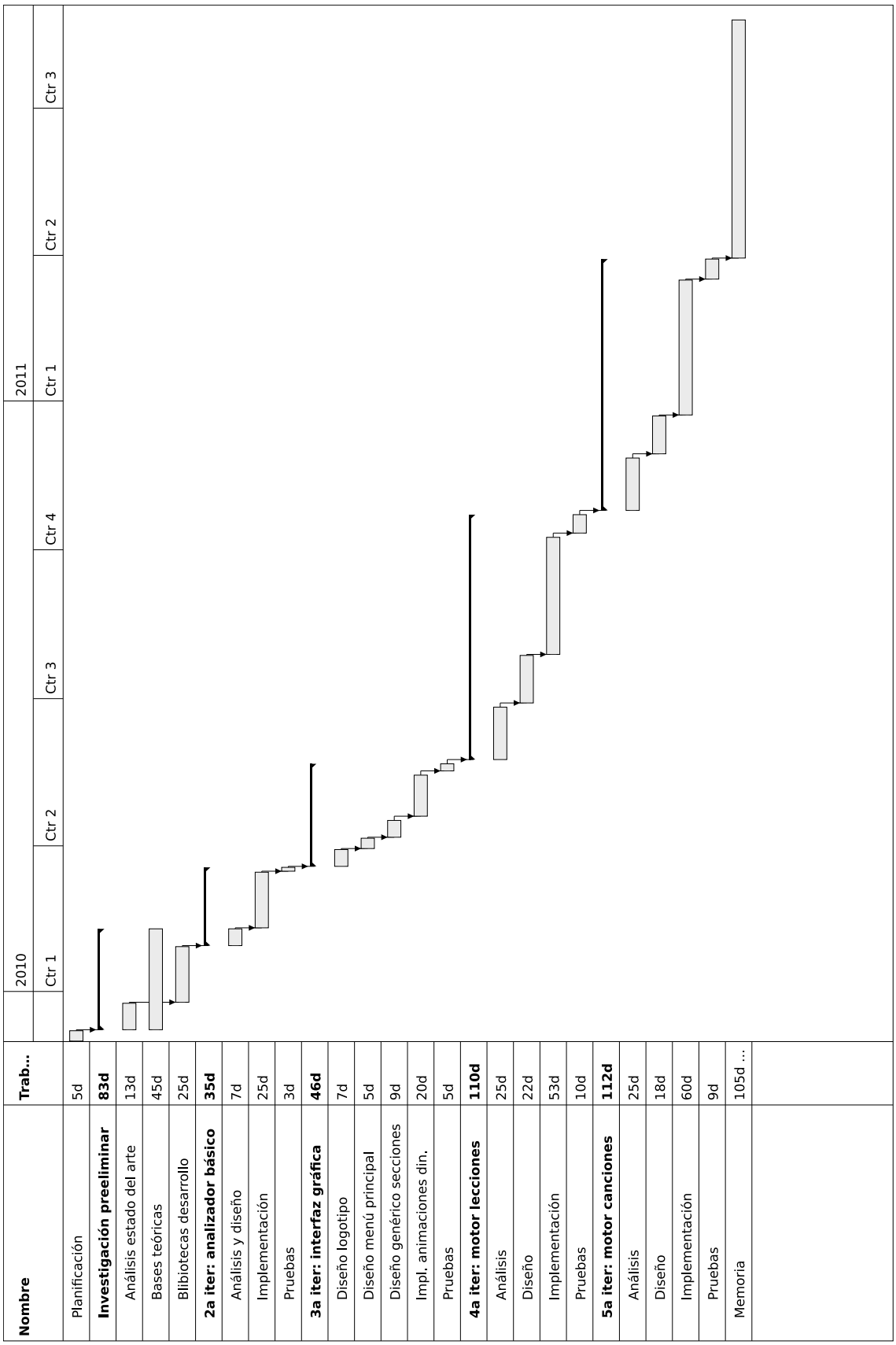


Figura 2.1.: Diagrama Gantt de iteraciones

3. Fundamentos

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. Durante el proceso, fuimos guiándonos por las necesidades que iban surgiendo, haciendo uso de publicaciones sobre la materia, tanto en línea como en papel, así como pidiendo consejo a iniciados y expertos en la materia. En particular, el grupo de noticias `comp.dsp` fue de gran utilidad a la hora de resolver numerosas dudas.

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.

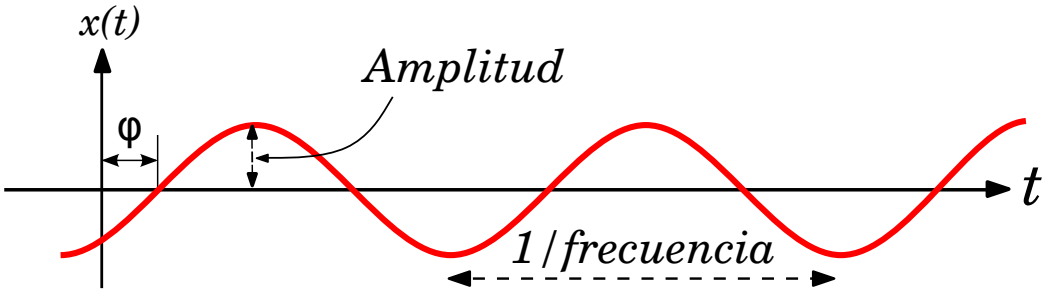


Figura 3.2.: Componentes de una señal senoidal básica

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

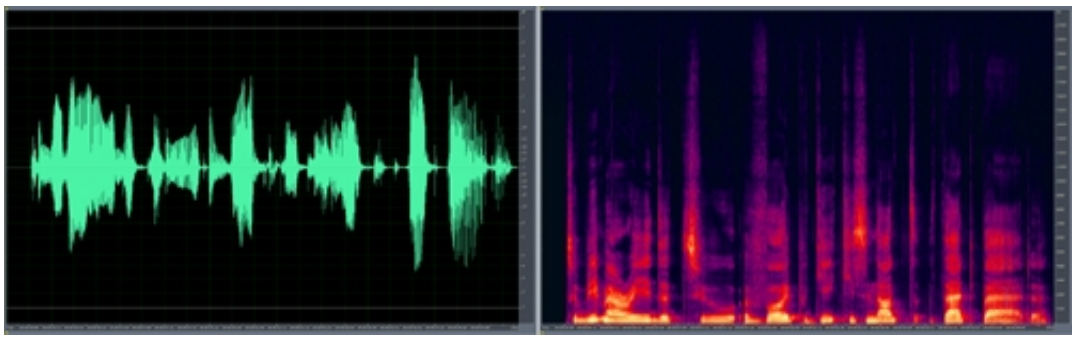


Figura 3.3.: Forma de ondas vs representación espectral

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* [51], en el que el desarrollador de Adobe, Mike Melanson, hace un repaso sobre la *jungla* que supone la programación 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.

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, etcé-

tera. 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 obteniendo así resultados en etapas más tempranas del desarrollo.

En las sucesivas secciones y capítulos se detallará el proceso de análisis y posteriores fases del proyecto en la última iteración del proyecto. Así se consigue una documentación más concisa y cercana al producto final.

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 [58], 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:

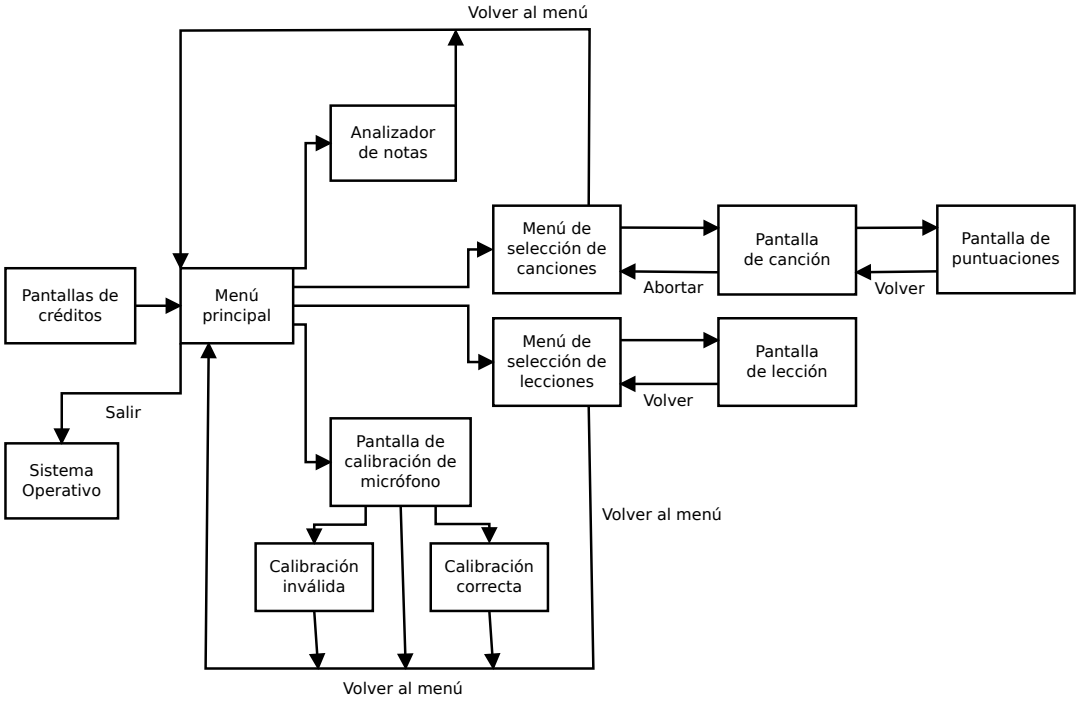


Figura 4.1.: Diagrama de flujo de las pantallas de oFlute

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.

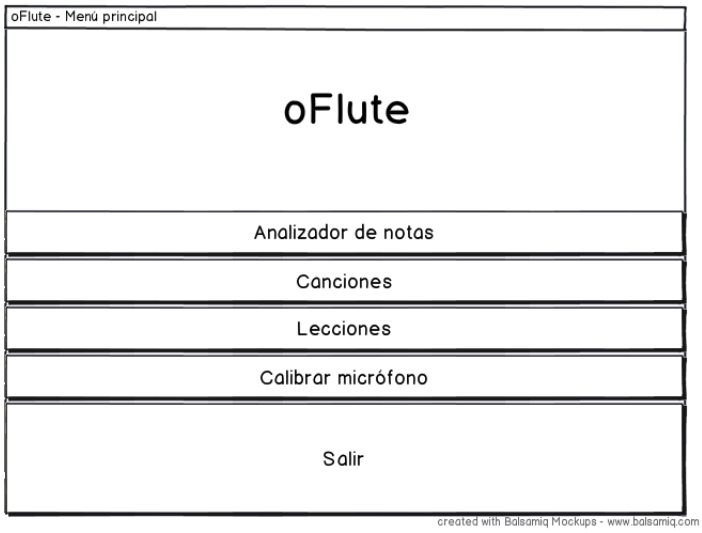


Figura 4.2.: Maqueta del menú principal

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.

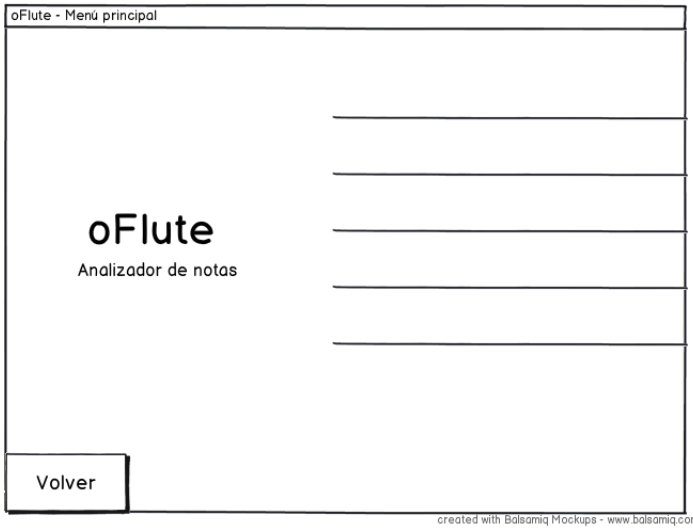


Figura 4.3.: Maqueta de la sección *analizador de notas*

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.

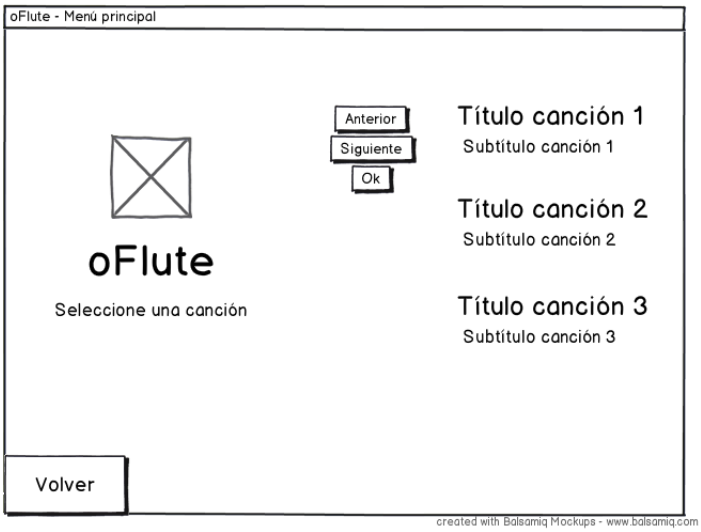


Figura 4.4.: Maqueta del menú de selección de canción

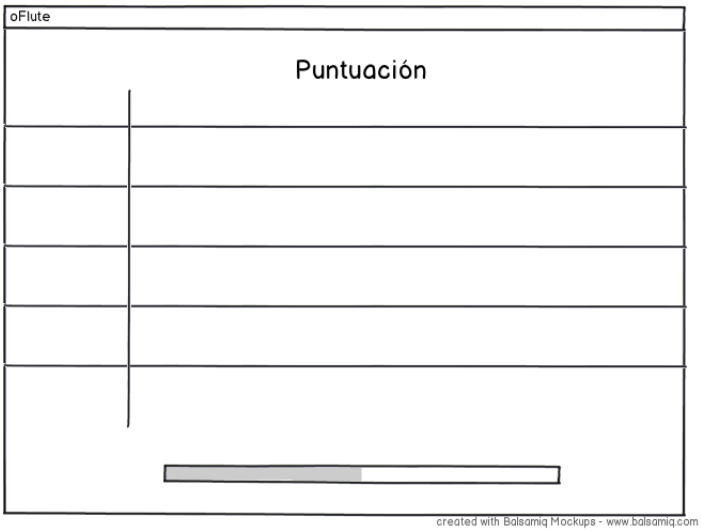


Figura 4.5.: Maqueta de la pantalla de interpretación de canción

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.

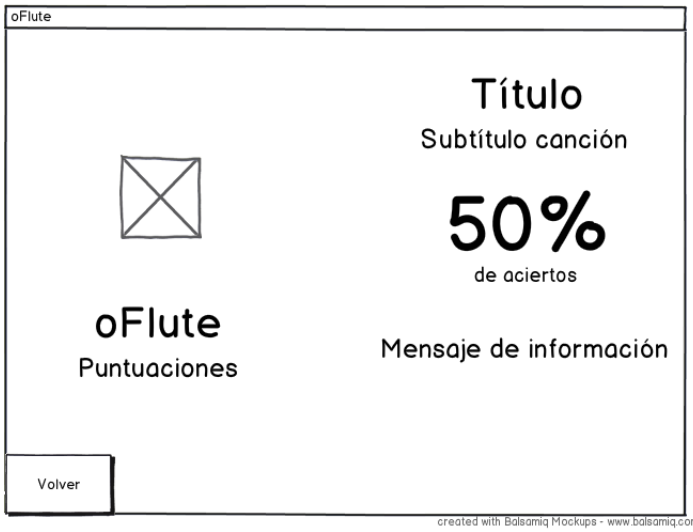


Figura 4.6.: Maqueta de la pantalla de puntuaciones

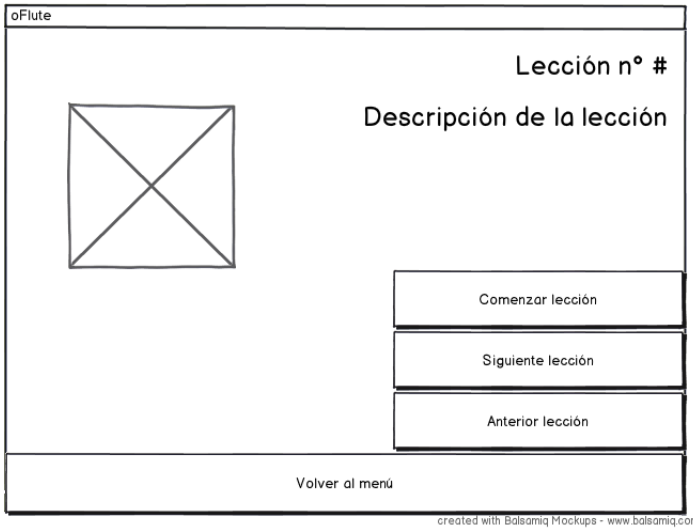


Figura 4.7.: Maqueta del menú de selección de lecciones

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 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

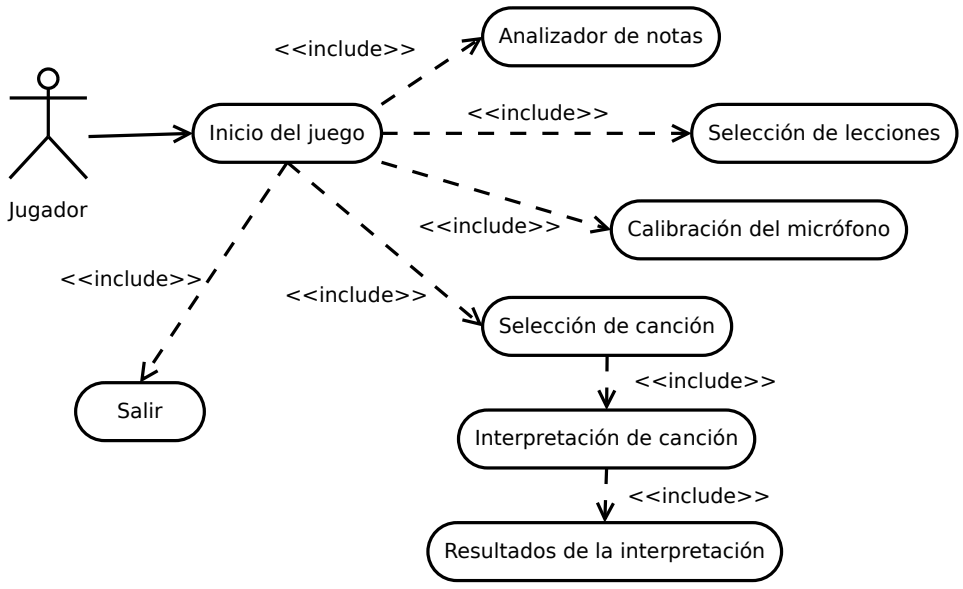


Figura 4.8.: Diagrama de casos de uso

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.

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

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.

EstadoLección Corresponde a la canción elegida desde el menú *EstadoMenúLecciones*.

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.

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

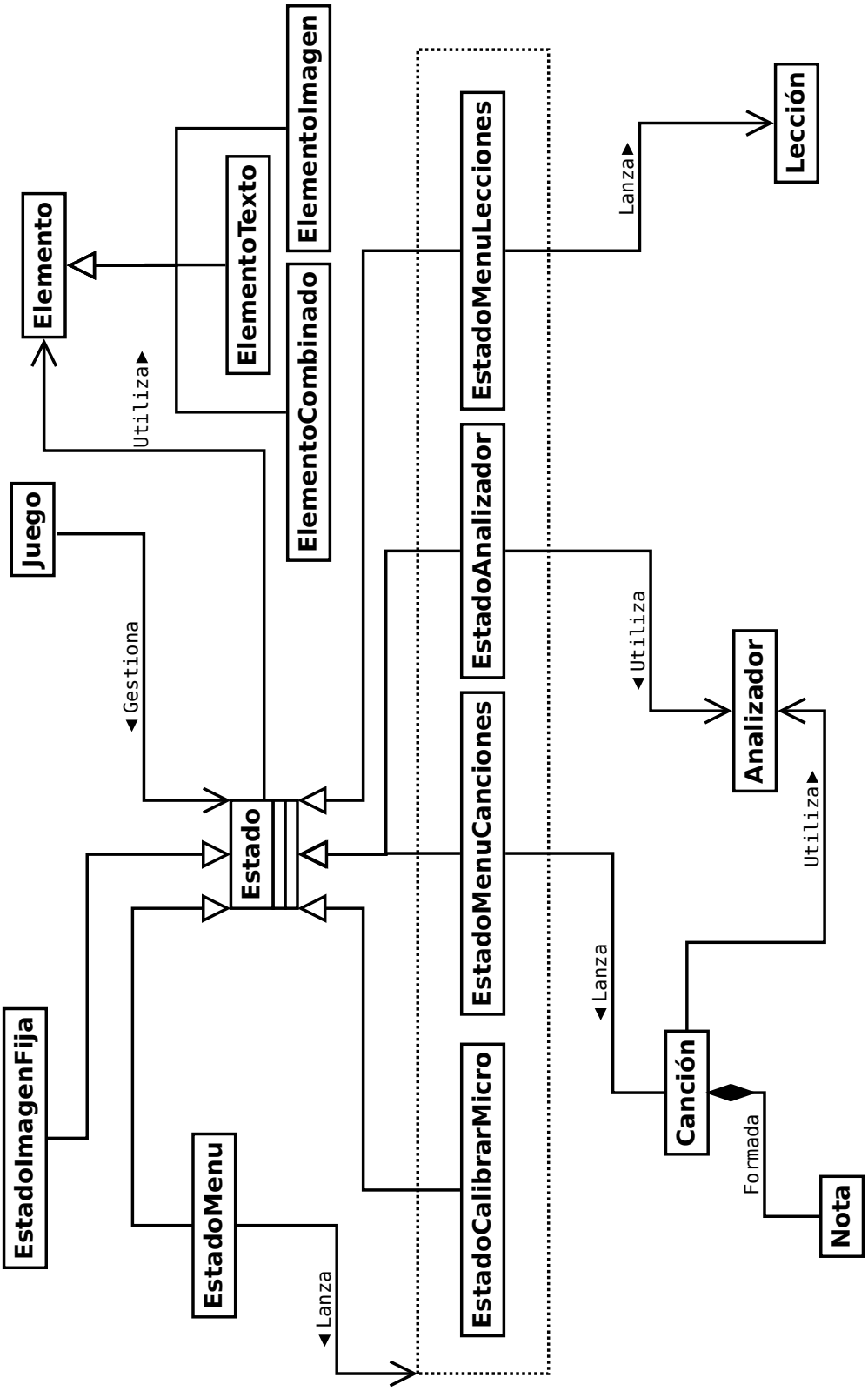


Figura 4.9.: Diagrama de clases conceptuales

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.

NOTA: No se han reflejado, por triviales, los escenarios alternativos en los que el usuario cierra la ventana, correspondientes a los flujos **a* definidos en la sección anterior.

4.5.1. Caso de uso: inicio del juego

Escenario principal

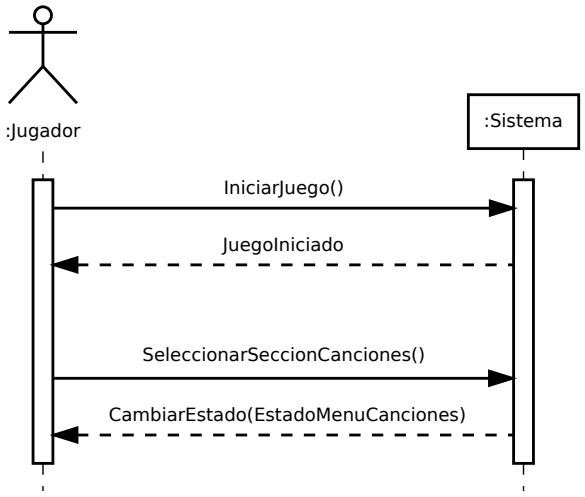


Figura 4.10.: Diagrama de secuencia, incio del juego, escenario principal

Operación IniciarJuego()

Actores *Jugador Sistema*

Responsabilidades Cargar y lanzar la aplicación, mostrar los títulos de crédito y el menú principal.

Precondiciones Ninguna.

Postcondiciones

- Se crea una instancia de la clase *Juego*, que gestiona la creación y destrucción de los estados.
- Se crean y posteriormente destruyen dos estados *EstadoImagenFija* para mostrar los títulos de crédito.
- Se crea y permanece un estado *EstadoMenú*, que representa el menú principal de la aplicación.

Operación SeleccionarSeccionCanciones()

Actores *Jugador Sistema*

Responsabilidades Esconder el menú principal y cargar el menú de selección de canciones.

Precondiciones

- El estado actual es una instancia de *EstadoMenú*.

Postcondiciones Se destruye el estado *EstadoMenú* y se carga *EstadoMenúCanciones*.

Escenario alternativo 4a

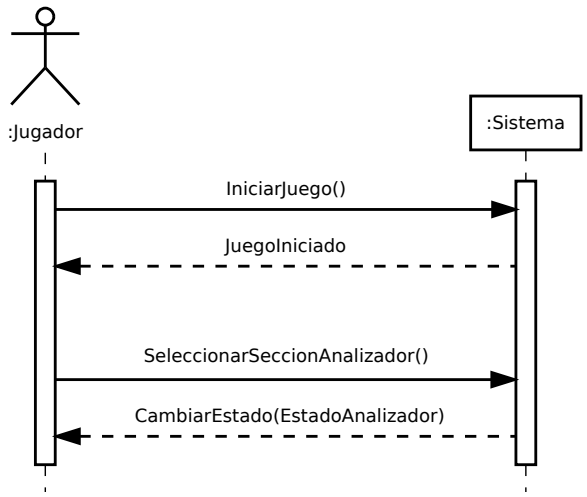


Figura 4.11.: Diagrama de secuencia, incio del juego, escenario alternativo 4a

Operación SeleccionarSeccionAnalizador()

Actores *Jugador Sistema*

Responsabilidades Esconder el menú principal y cargar la sección de análisis de notas.

Precondiciones

- El estado actual es una instancia de *EstadoMenú*.

Postcondiciones Se destruye el estado *EstadoMenú* y se carga *EstadoAnalizador*.

Escenario alternativo 4b

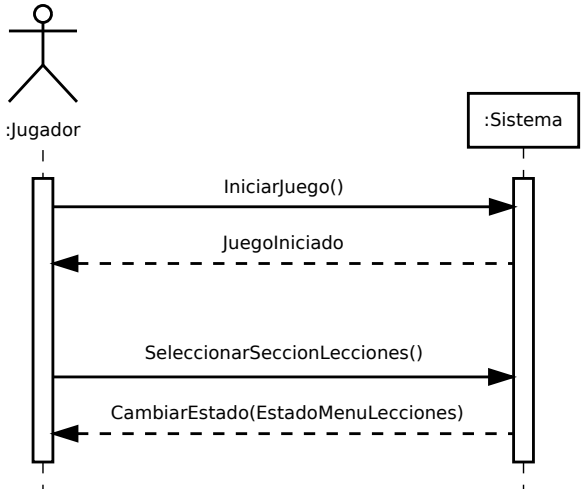


Figura 4.12.: Diagrama de secuencia, incio del juego, escenario alternativo 4b

Operación SeleccionarSeccionLecciones()

Actores *Jugador Sistema*

Responsabilidades Esconder el menú principal y cargar la menú de selección de lecciones.

Precondiciones

- El estado actual es una instancia de *EstadoMenú*.

Postcondiciones Se destruye el estado *EstadoMenú* y se carga *EstadoMenúLecciones*.

Escenario alternativo 4c

Operación SeleccionarSeccionCalibracion()

Actores *Jugador Sistema*

Responsabilidades Esconder el menú principal y cargar la sección de calibración de micrófono.

Precondiciones

- El estado actual es una instancia de *EstadoMenú*.

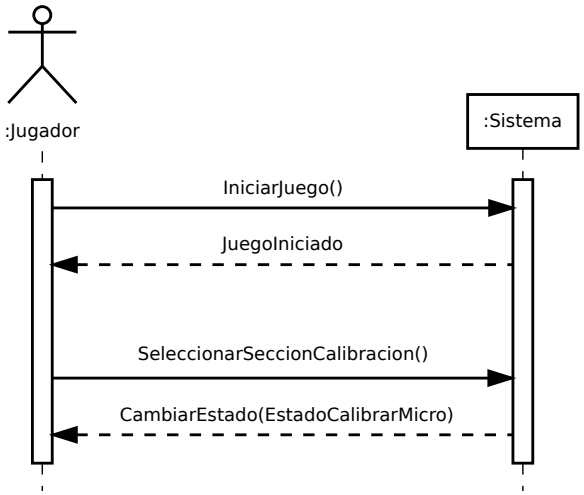


Figura 4.13.: Diagrama de secuencia, incio del juego, escenario alternativo 4c

Postcondiciones Se destruye el estado *EstadoMenú* y se carga *EstadoCalibrarMicro*.

Escenario alternativo 4d

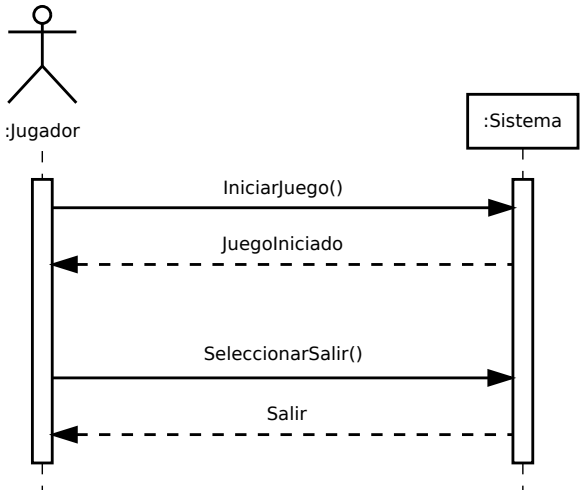


Figura 4.14.: Diagrama de secuencia, incio del juego, escenario alternativo 4d

Operación SeleccionarSalir()

Actores *Jugador Sistema*

Responsabilidades Esconder el menú principal, descargar los recursos y cerrar la aplicación.

Precondiciones

- El estado actual es una instancia de *EstadoMenú*.

Postcondiciones Se destruye el estado *EstadoMenú*, se destruye la instancia de la clase *Juego* y termina la ejecución de la aplicación.

4.5.2. Selección de canción

Escenario principal

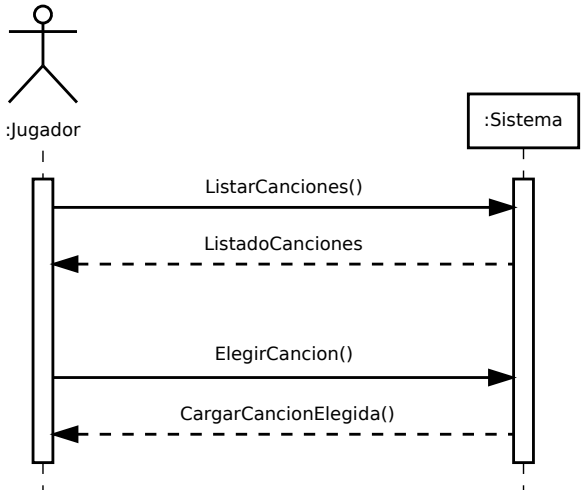


Figura 4.15.: Diagrama de secuencia, selección de canción, escenario principal

Operación ListarCanciones()

Actores *Jugador Sistema*

Responsabilidades Cargar y mostrar la lista de canciones cargadas en el sistema.

Precondiciones Se ordenó la carga del estado *EstadoMenuCanción*

Postcondiciones

- El estado actual es una instancia de *EstadoMenuCanción*.
- Se ha cargado la lista de canciones y se muestra en pantalla.

Operación ElegirCanción()

Actores *Jugador Sistema*

Responsabilidades Cargar la canción que el usuario ha elegido para interpretar.

Precondiciones Existe una lista de canciones cargada de entre las que el usuario ha elegido una.

Postcondiciones

- Se carga la canción indicada.
- Se oculta la lista de canciones.
- Se pasa a un sub-estado de interpretación de canción.

Escenario alternativo 3a

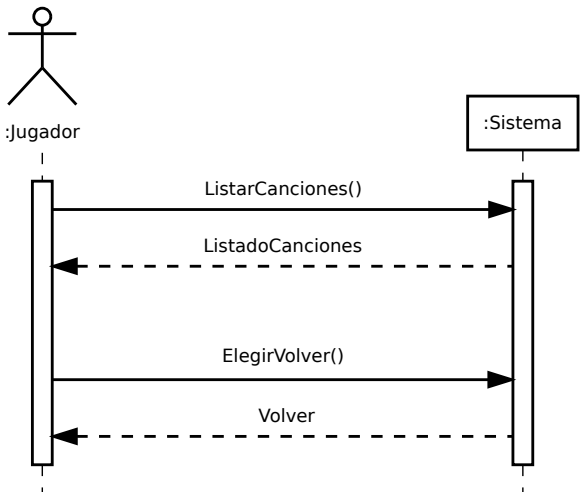


Figura 4.16.: Diagrama de secuencia, selección de canción, escenario alternativo 3a

Operación ElegirVolver()

Actores *Jugador Sistema*

Responsabilidades Descargar la sección actual y volver al menú anterior.

Precondiciones El estado actual es una instancia de *EstadoMenuCanción*.

Postcondiciones

- El estado instancia de *EstadoMenuCanción* queda descargado.
- Se carga y se muestra *EstadoMenú*.

4.5.3. Interpretación de canción

NOTA: No se reflejan los escenarios alternativos al estar englobados en la operación *InteractuarConFlauta*.

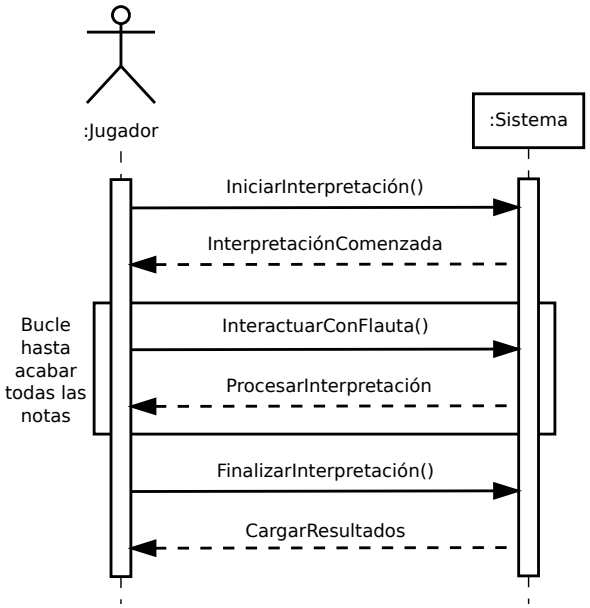


Figura 4.17.: Diagrama de secuencia, interpretación de canción, escenario principal

Escenario principal

Operación IniciarInterpretación()

Actores *Jugador Sistema*

Responsabilidades Parsear el fichero de canción, cargar la interfaz y comenzar la interpretación.

Precondiciones El usuario ha elegido una canción en el estado anterior.

Postcondiciones

- Se muestra la interfaz de interpretación de canción.
- El fichero de canción queda cargado e interpretado, instanciando los elementos de la clase *Nota* que sean necesarios.
- Comienza la interpretación

Operación InteractuarConFlauta()

Actores *Jugador Sistema*

Responsabilidades El *Jugador* interactúa con el sistema mediante la flauta a través del micrófono, y el *Sistema* analiza los datos y muestra una respuesta en pantalla.

Precondiciones

- La interpretación ha comenzado.

- El micrófono está correctamente configurado.

Postcondiciones

- El sistema captura y analiza los datos de audio.
- Según el análisis, el sistema responde de una forma u otra (según los escenarios alternativos 3a, 4a y 5a del caso de uso *interpretación de canción*).

Operación FinalizarInterpretación()

Actores *Jugador Sistema*

Responsabilidades Descargar la pantalla de interpretación, descargar la canción y finalizar la interpretación.

Precondiciones Todas las notas se han interpretado.

Postcondiciones

- Se descarga la *Canción* actual.
- Se ocultan los elementos de la interfaz de interpretación.
- Se lanza la sección de puntuación.

4.5.4. Resultados de interpretación

Escenario principal

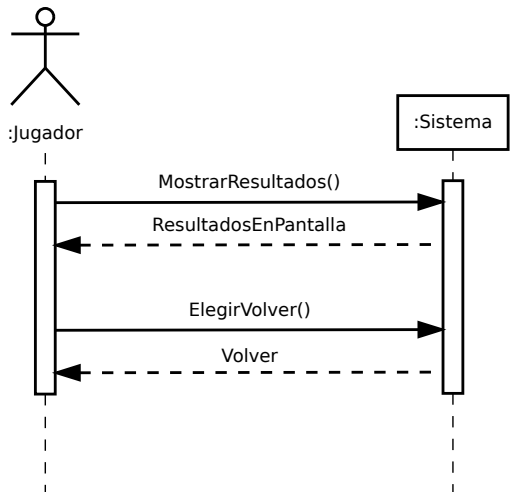


Figura 4.18.: Diagrama de secuencia, resultados de interpretación, escenario principal

Operación MostrarResultados()

Actores *Jugador Sistema*

Responsabilidades Interpretar los resultados de la interpretación y mostrar los resultados en pantalla.

Precondiciones El *Jugador* ha concluído satisfactoriamente una interpretación completa de una canción, obteniendo una suma de puntos X.

Postcondiciones Mostrar en pantalla los resultados en forma de porcentaje de aciertos, y un mensaje según aquél.

Operación ElegirVolver()

Actores *Jugador Sistema*

Responsabilidades Descargar la sección actual y volver al menú anterior.

Precondiciones

- La aplicación se encuentra en la pantalla de muestra de resultados.
- El usuario ha pulsado la tecla escape o el botón *volver*.

Postcondiciones

- Se descargan todos los datos referentes a la canción actual.
- Se carga y se muestra *EstadoMenúCanciones*.

4.5.5. Analizador de notas

NOTA: No se reflejan los escenarios alternativos al estar englobados en la operación *InteractuarConFlauta*.

Escenario principal

Operación IniciarAnálisis

Actores *Jugador Sistema*

Responsabilidades Cargar la interfaz e iniciar el análisis de notas.

Precondiciones El usuario eligió la sección *Analizador de notas* en el menú principal.

Postcondiciones

- Aparece la interfaz del analizador de notas.
- Se inicia el análisis de notas

Operación InteractuarConFlauta

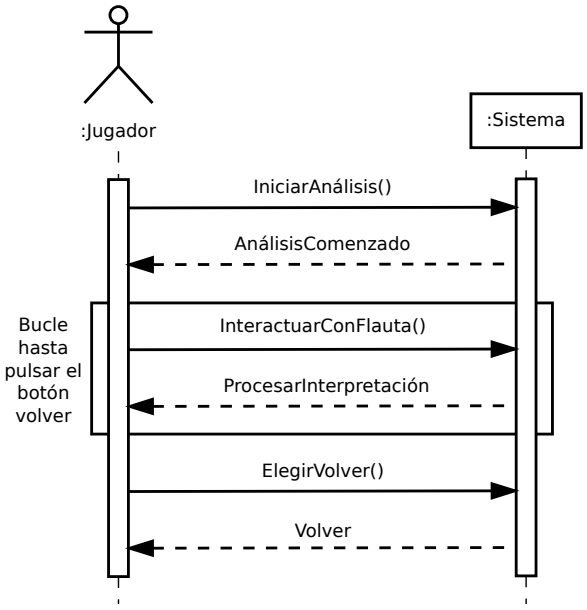


Figura 4.19.: Diagrama de secuencia, interpretación de canción, escenario principal

Actores *Jugador Sistema*

Responsabilidades El *Jugador* toca notas en la flauta y el *Sistema* captura y reconoce el audio, indicando la nota tocada en pantalla.

Precondiciones Se ha iniciado el análisis.

Postcondiciones

- El *Sistema* recoge y analiza el sonido que emite la flauta del *Jugador*.
- El *Sistema* representa en pantalla la nota identificada, o no muestra nada en caso de identificación defectuosa.

4.5.6. Calibración de micrófono

Escenario principal

Operación CargarCalibración()

Actores *Jugador Sistema*

Responsabilidades Cargar la sección y preparar el sistema para comenzar la calibración del micrófono.

Precondiciones El usuario ha elegido en el menú principal la opción *Calibrar micrófono*.

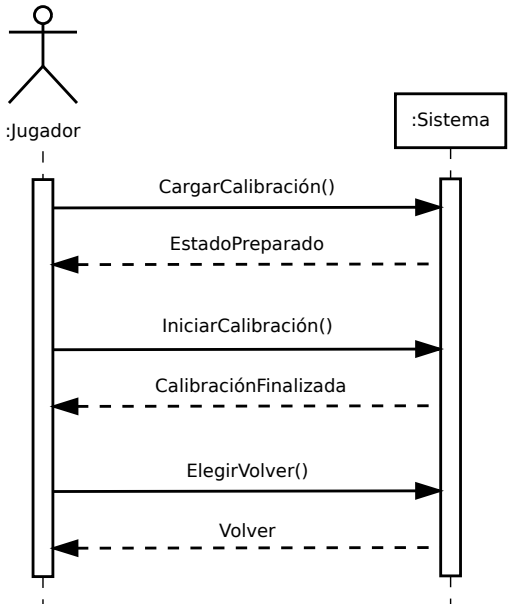


Figura 4.20.: Diagrama de secuencia, calibración de micrófono, escenario principal

Postcondiciones La sección está cargada y la calibración lista para iniciarse.

Operación CalibrarMicrófono()

Actores *Jugador Sistema*

Responsabilidades Llevar a cabo la calibración correcta del micrófono.

Precondiciones El usuario ha lanzado la calibración del micrófono.

Postcondiciones

- Se cierra el sistema de sonido.
- Se obtiene un valor umbral de ruido ambiente, fruto de una calibración exitosa.

Operación ElegirVolver()

Actores *Jugador Sistema*

Responsabilidades Descargar la sección actual y volver al menú anterior.

Precondiciones

- La calibración ha concluído exitosamente.
- El usuario ha pulsado la tecla escape o el botón *volver*.

Postcondiciones

- Se descarga la sección actual.

- Se carga y se muestra *EstadoMenúCanciones*.

Escenario alternativo 5a

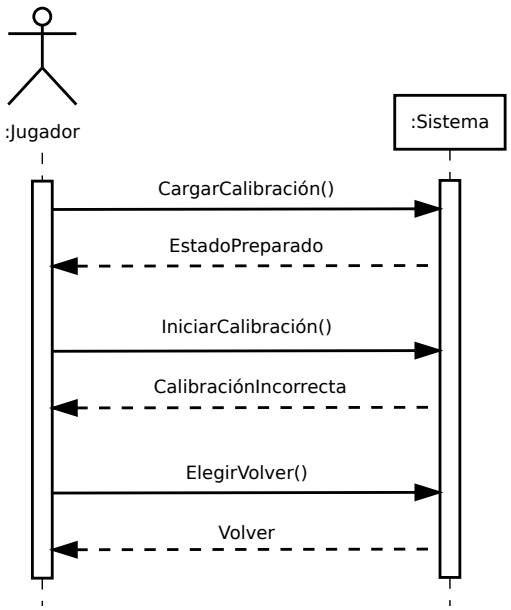


Figura 4.21.: Diagrama de secuencia, calibración de micrófono, escenario principal

Operación CalibrarMicrófono()

Actores *Jugador Sistema*

Responsabilidades Llevar a cabo la calibración correcta del micrófono.

Precondiciones El usuario ha lanzado la calibración del micrófono.

Postcondiciones

- Se cierra el sistema de sonido.
- La calibración ha fallado.
- No se obtiene valor umbral de ruido ambiental.

4.5.7. Selección de lecciones

Escenario principal

Operación ListarLecciones()

Actores *Jugador Sistema*

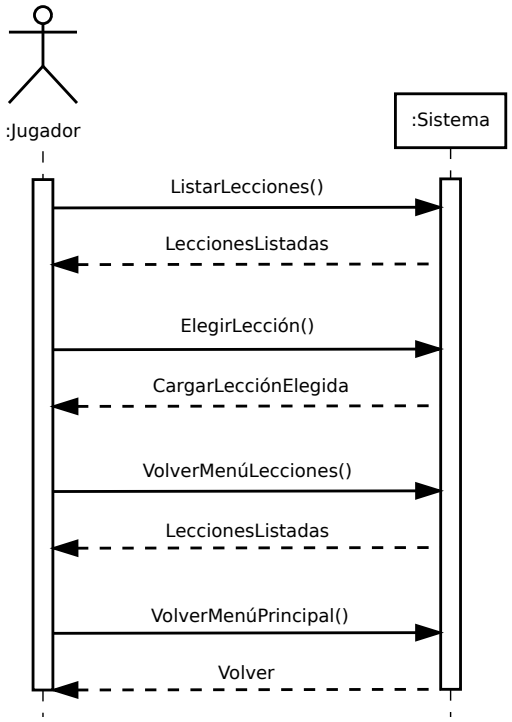


Figura 4.22.: Diagrama de secuencia, selección delecciones, escenario principal

Responsabilidades Mostrar el menú de selección de lecciones y listar todas las lecciones disponibles.

Precondiciones El usuario eligió la opción *Lecciones* en el menú principal.

Postcondiciones

- Se muestra la interfaz del menú de selección de lecciones.
- Se listan las lecciones cargadas en el sistema

Operación ElegirLección()

Actores *Jugador Sistema*

Responsabilidades Cargar y mostrar la lección elegida por el usuario.

Precondiciones El menú de selección de lecciones está cargado y el usuario ha elegido una de las lecciones.

Postcondiciones

- Se oculta el menú de selección de lecciones.
- Se interpreta el fichero de lección elegido.
- Se muestran los elementos multimedia pertenecientes a la lección elegida.

Operación VolverMenuLecciones()

Actores *Jugador Sistema*

Responsabilidades Descargar la lección actual y volver al menú anterior.

Precondiciones

- El usuario ha terminado de ver la lección elegida.
- El usuario ha pulsado la tecla escape o el botón *volver*.

Postcondiciones

- Se descarga la lección actual.
- Se muestra de nuevo el menú de selección de lecciones.

Operación VolverMenuPrincipal()

Actores *Jugador Sistema*

Responsabilidades Descargar el menú de selección de lecciones y volver al menú principal.

Precondiciones El usuario ha pulsado la tecla escape o el botón *volver*.

Postcondiciones

- Se descarga el menú de selección de lecciones.
- Se carga y muestra el menú principal

Escenario alternativo

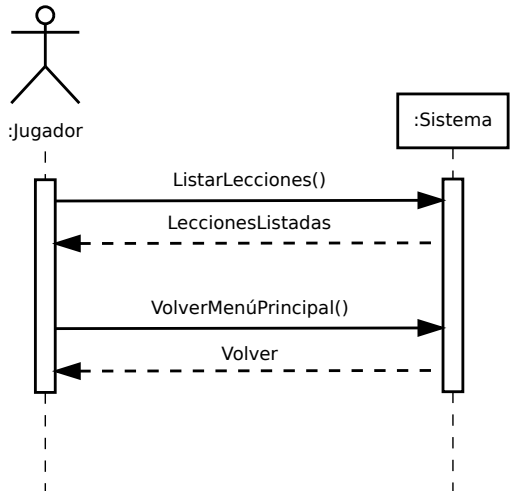


Figura 4.23.: Diagrama de secuencia, selección de lecciones, escenario alternativo

Operación ListarLecciones()

Actores *Jugador Sistema*

Responsabilidades Mostrar el menú de selección de lecciones y listar todas las lecciones disponibles.

Precondiciones El usuario eligió la opción *Lecciones* en el menú principal.

Postcondiciones

- Se muestra la interfaz del menú de selección de lecciones.
- Se listan las lecciones cargadas en el sistema

Operación VolverMenuPrincipal()

Actores *Jugador Sistema*

Responsabilidades Descargar el menú de selección de lecciones y volver al menú principal.

Precondiciones El usuario ha pulsado la tecla escape o el botón *volver*.

Postcondiciones

- Se descarga el menú de selección de lecciones.
- Se carga y muestra el menú principal

5. Diseño

En este capítulo se presentarán los detalles de diseño del proyecto, basándonos en el análisis mostrado en el capítulo anterior. El modelo de clases de diseño representado aquí es más fiel a la implementación final que el diagrama de clases conceptuales, pero aún así hay detalles que no se han contemplado por ser demasiado cercanos a los detalles de implementación.

También en el presente capítulo se detallarán las decisiones de diseño en relación al aspecto visual de la aplicación, extendiéndonos en el proceso de creación del logotipo del juego así como de la interfaz gráfica de usuario.

5.1. Diagrama de clases de diseño

Tras la fase de diseño, componían el sistema más de 40 clases, por lo que hemos tenido que dividir los diagramas en varias partes, intentando seguir cierto criterio a la hora de elegir qué clases formarán parte de cada división.

En todos los diagramas aparecen, en aras de mantener el contexto, las clases básicas de la aplicación: *Juego* y *Estado*. Además, hay algunas otras clases que también se repetirán entre diagramas por conveniencia. Para mantener la legibilidad, se han ocultado los miembros privados y protegidos.

- En el primer diagrama (figura 5.1) aparecen las clases relacionadas con el *menú principal*, clases de utilidades (logging y animación), y clases para representar elementos gráficos.
- En el segundo diagrama (figura 5.2) aparecen las clases relacionadas con el subsistema de análisis del audio, así como las secciones *Analizador de Notas* y *Calibrar micrófono*.
- En el tercer diagrama (figura 5.3) aparecen todas las clases relacionadas con la sección de *Canciones*.
- En el cuarto y último diagrama (figura 5.4) aparecen las clases relacionadas con el motor de *lecciones*.

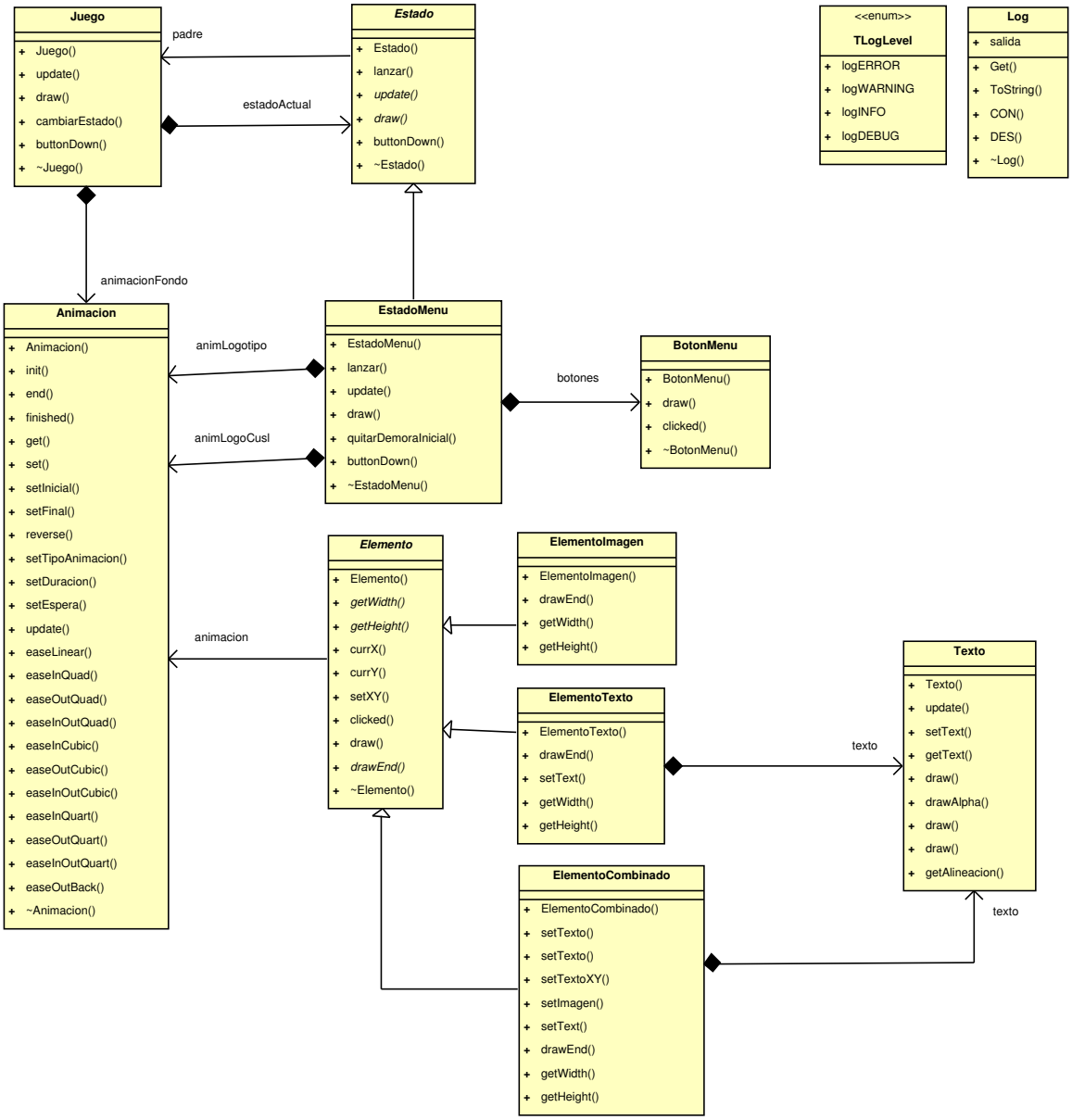


Figura 5.1.: Diagrama de clases de diseño, parte I

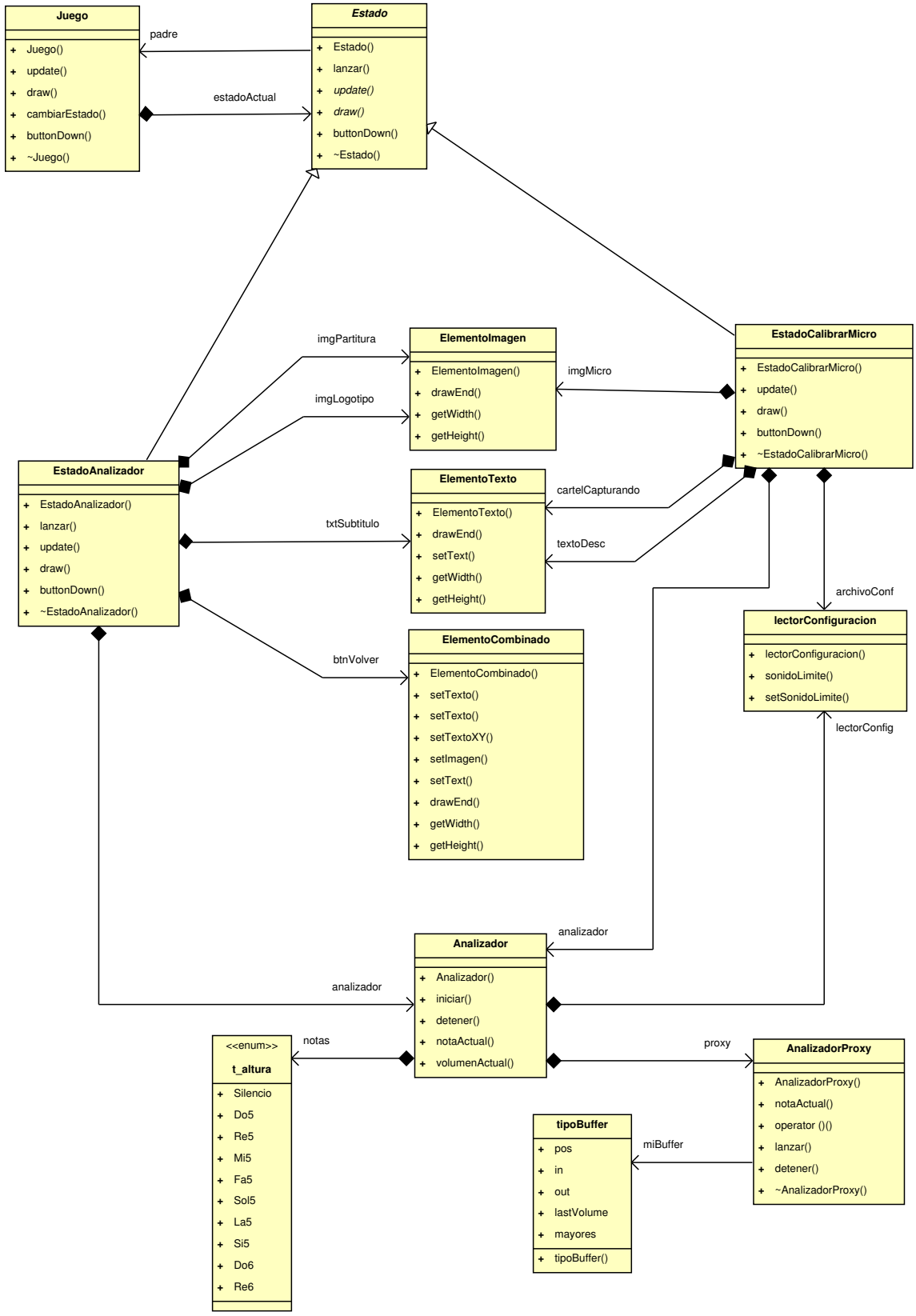


Figura 5.2.: Diagrama de clases de diseño, parte II

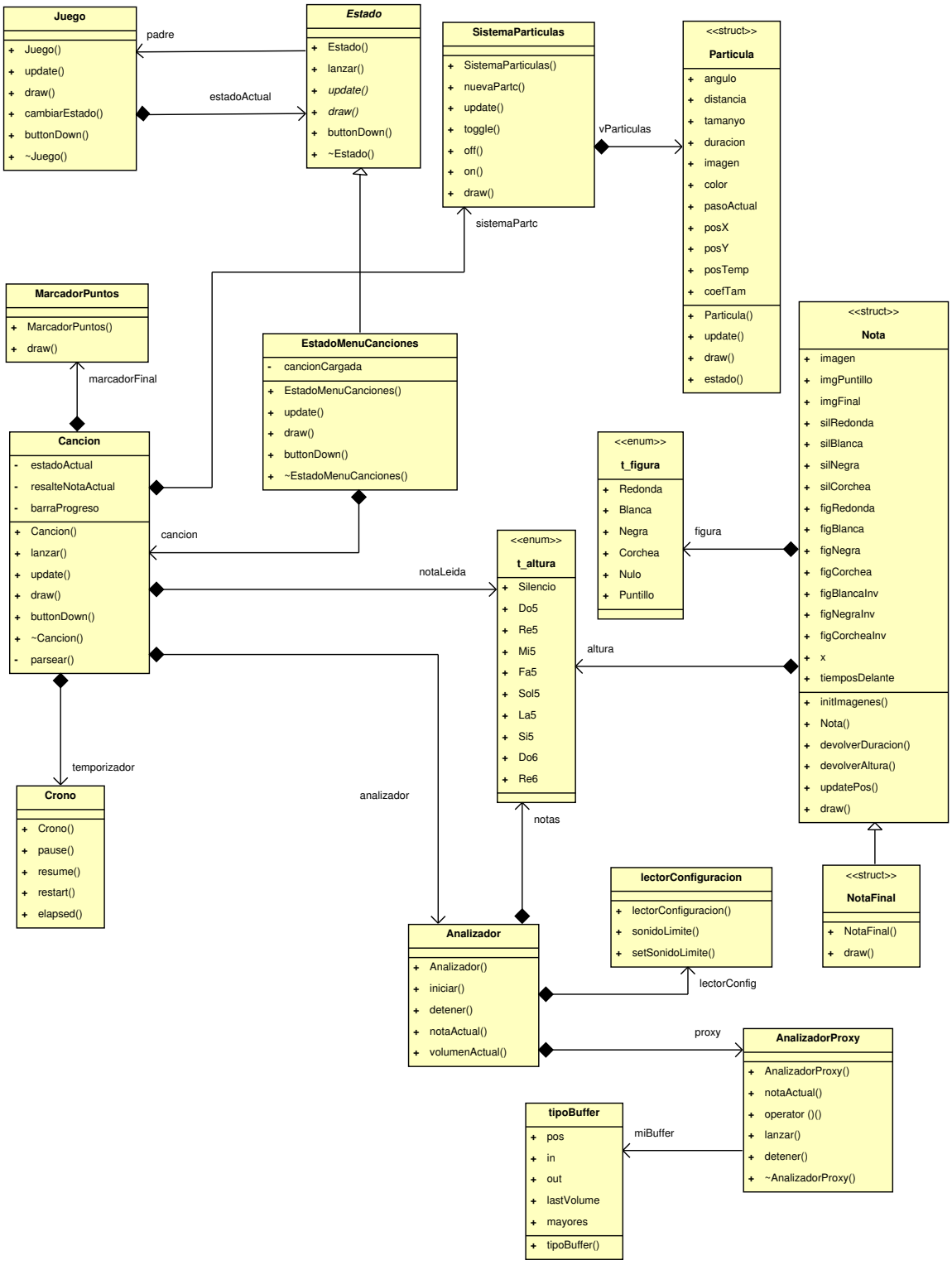


Figura 5.3.: Diagrama de clases de diseño, parte III

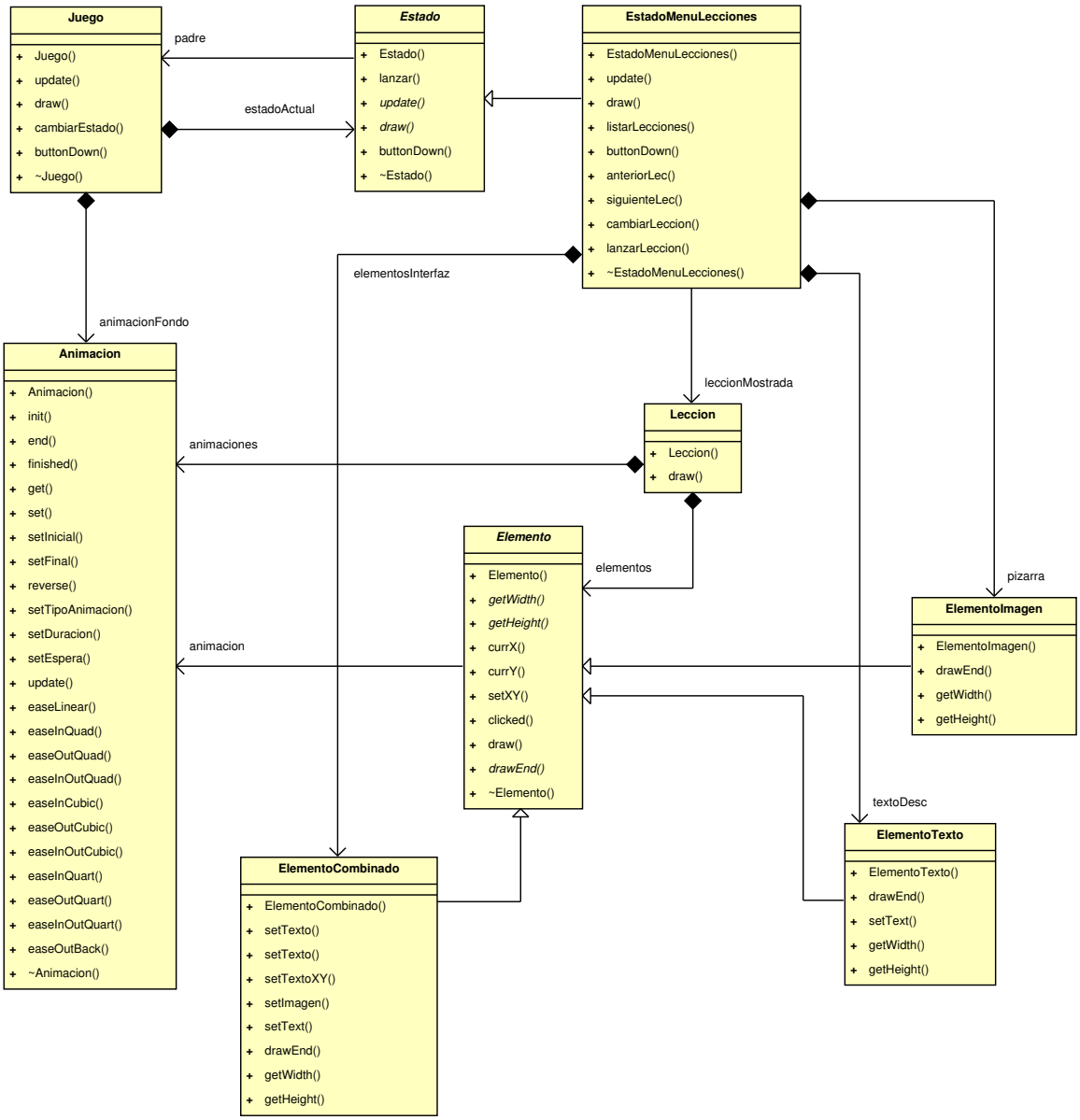


Figura 5.4.: Diagrama de clases de diseño, parte IV

5.2. Diseño visual

Teniendo muy presente que el público objetivo de **oFlute** es joven y dinámico, el diseño visual del proyecto intentó adaptarse a lo que la audiencia podría considerar más atractivo. Desde el inicio se fijaron ciertas premisas o *guías de branding* para oFlute, que se siguieron a rajatabla a la hora de diseñar cada uno de los elementos.

Interfaces limpias y minimalistas. Era imprescindible que las interfaces gráficas de cada una de las secciones tuviera un diseño limpio y cuidado, manteniendo en el mínimo la cantidad de elementos a mostrar en pantalla sin descuidar el diseño.

Para conseguirlo, se optó por utilizar un fondo blanco con un sutil patrón gris claro, que se mantiene a lo largo de todas las secciones. Un gran ejemplo de esto es la imagen de los títulos de crédito del juego.

Figura 5.5.: Imagen de títulos de crédito de oFlute

Paleta de colores pastel. Para acompañar al blanco limpio de los fondos de la interfaz se ideó una paleta de colores amplia, basado en tonos brillantes, cercanos a pastel. Esta paleta está presente tanto en el logotipo de oFlute, como en el resto de secciones. En especial, el menú principal se benefició enormemente del uso de esta paleta de colores en los botones de las secciones.

Figura 5.6.: Detalle de la paleta de colores de oFlute

Logotipo. A la hora de diseñar el logtipo, se hizo un *brainstorming* inicial en busca de conceptos relacionados con el proyecto: notas musicales, flautas, pentagramas, claves de sol, etcétera.

Finalmente, decidimos utilizar el concepto de la flauta a través del espacio negativo que genera dibujar solo los orificios de la misma, manteniendo el tamaño original de los mismos.

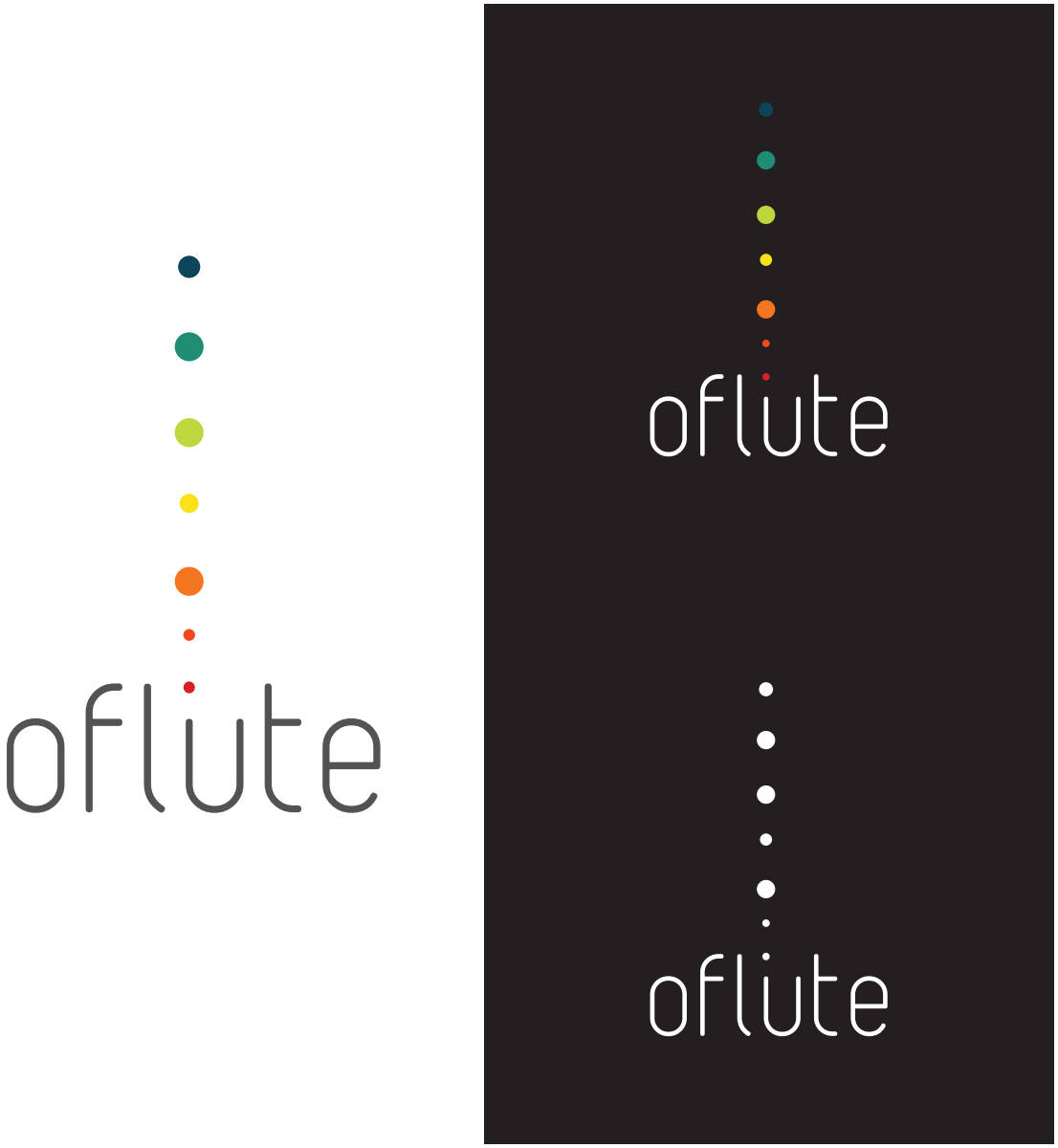


Figura 5.7.: Logotipo de oFlute original sobre blanco, sobre negro y en blanco y negro

Tipografía. Para adecuarse al diseño minimalista propuesto, se buscaron tipografías simples y limpias. Para el logotipo se usó la fuente 232 MKSD Round [1], y para los textos de la aplicación se usó la fuente Miso [24].

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 aplicación se mantuvo actualizada una bitácora, accesible en línea [45], 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 [46]. En él se encuentra todo el código fuente original, así como la documentación en formato *Doxygen* [11].

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 de un sistema a otro.

Una de estas desigualdades se presentaba a la hora de **cargar fuentes** para mostrar textos. En su versión para GNU/Linux, el módulo para el pintado de textos de Gosu (que comprende la clase `Gosu::Font` y las funciones `Gosu::drawText` y `Gosu::createText`) solo permitía utilizar fuentes que estuvieran ya instaladas en el sistema de forma global, de modo que era imposible adjuntar un fichero con una fuente en formato `TrueType` [35] para su carga en el juego.

La instalación de fuentes en sistemas GNU/Linux siempre ha sido una operación engorrosa que ha requerido permisos de administrador. Esto, unido a que tanto en Windows como en Mac OS la opción de cargar fuentes desde un fichero sí estaba disponible, supuso un grave problema en el planteamiento del proyecto.

Inicialmente se investigaron las razones de esta limitación. Al ser un proyecto libre y abierto, revisamos el código de Gosu, concretamente la implementación de la

clase `Gosu::Font` previamente citada, de la que dependían todas las otras funciones relacionadas con fuentes.

Las conclusiones que se sacaron fueron que Gosu, bajo GNU/Linux, implementaba el renderizado de fuentes mediante una biblioteca llamada Pango [26], de bastante bajo nivel, y que por diseño está limitada al uso de fuentes de sistema, ya que se orienta a herramientas del sistema operativo, no a aplicativos de terceros.

Así pues, era necesario buscar una alternativa. En numerosos proyectos previos [50] se había utilizado la biblioteca **SDL** [33], un framework multimedia muy utilizado en aplicaciones de audio, vídeo y juegos. Uno de los subsistemas que proporciona SDL es **SDL_ttf** [34], una biblioteca para el renderizado de fuentes basadas en ficheros `TrueType`. Esto, unido a que la propia SDL ya era una dependencia de Gosu, propició que se iniciara la implementación de una solución basada en este medio.

6.1.1. Funcionamiento de `SDL_ttf`

Para comprender cómo se implementó la solución al problema citado, el primer paso es entender cómo funciona la biblioteca `SDL_ttf` para poder ver cómo se corresponde luego con Gosu.

Tipos de datos utilizados

En común con el resto de módulos de SDL, SDL_ttf se basa en un tipo de datos conocido como *superficies* (`SDL_Surface`). Estas superficies representan mapas de bits cargados en memoria, y se utilizan para guardar las imágenes que se cargan de los ficheros, así como para representar otros destinos gráficos intermedios, como la propia pantalla.

Para representar el color del texto que vamos a pintar, SDL_ttf utiliza la estructura `SDL_Color`, que guardará los valores de color de 8 bits para los tres canales de color primario (rojo, verde y azul).

Además, SDL_ttf define un tipo de datos propio, de nombre `TTF_Font`, que representa una fuente cargada a partir de un fichero, con un tamaño determinado.

Funciones utilizadas

SDL_ttf precisa de una **inicialización** previa, que se lleva a cabo mediante la llamada a `TTF_Init()`. Igualmente, liberaremos los recursos ocupados mediante `TTF_Quit()`.

Para **cargar la fuente** a partir de los ficheros TrueType (con extensión `.ttf`), se utiliza la función `TTF_OpenFont(const char * fichero, int tamaño)`. Esta función

recibe una cadena con el nombre del fichero y el tamaño de letra a utilizar, y devuelve un puntero al tipo `TTF_Font`.

```
TTF_Font * fuente = TTF_OpenFont("fichero_fuente.ttf", 25);
```

Una vez cargada la fuente de nuestro interés, tendremos que **generar una superficie** con el texto que queramos. Para ello, SDL_ttf ofrece una gran variedad de funciones, según el suavizado del texto y el tipo de caracteres que estemos utilizando. En nuestro caso, queríamos que la superficie tuviera fondo transparente, así como usar caracteres en UTF-8 [59], por lo que la función que se eligió fue

```
SDL_Surface * TTF_RenderUTF8_Blended (TTF_Font * fuente,
                                      const char * texto,
                                      SDL_Color color);
```

Esta función devolverá un puntero a una superficie que se integrará (*blends*) sobre cualquier imagen, al tener el fondo transparente.

6.1.2. Representación de imágenes en Gosu

Una vez conocidos los tipos de datos y funciones que ofrece SDL_ttf, es necesario conocer cuáles son los tipos de datos equivalentes en Gosu, de forma que pueda haber una *conversión* de un tipo a otro.

Mapas de bits de bajo nivel: Gosu::Bitmap

La clase `Gosu::Bitmap` representa, en Gosu, un mapa de bits de bajo nivel. Éste no contará con aceleración por hardware ni podrá mostrarse en pantalla, pero podremos acceder directamente a los datos de los píxeles y trabajar con ellos.

`Gosu::Bitmap` nos servirá de paso intermedio entre la superficie de SDL y las imágenes habituales de Gosu.

Imágenes de uso común: Gosu::Image

En Gosu las imágenes y assets gráficos se manejan habitualmente con la clase `Gosu::Image`, que representa un contenedor de más alto nivel que `Gosu::Bitmap`, además de estar optimizado y poder pintarse en pantalla.

Podremos crear un objeto de esta clase a partir de un `Gosu::Bitmap`, consiguiendo finalmente un objeto *pintable*.

6.1.3. Implementación final

Conocidas las representaciones internas de ambas bibliotecas, se implementó una clase con una interfaz similar a `Gosu::Font`, de modo que la transición pudiera ser lo más transparente posible, de nombre `CustomFont`.

En el constructor se inicializa (de forma única, mediante una variable estática) el subsistema `SDL_ttf` utilizando la llamada `TTF_Init()`. Después, se carga la fuente indicada por los parámetros del constructor. En ambos casos, se comprueba que el procedimiento ha sido correcto, lanzando una excepción en caso contrario.

```
static int initResult = TTF_Init();
if (initResult < 0)
   throw std::runtime_error("Could not initialize SDL_TTF");

font = TTF_OpenFont(fontName, fontHeight);
if (!font)
   throw std::runtime_error("Could not open TTF file " + fontName);
```

Seguidamente, en el método de pintado (`draw`) se genera la superficie con el texto que se indique, pasando el contenido de los píxeles a un contenedor `Gosu::Bitmap` y finalmente generando un objeto `Gosu::Image` a partir del mismo.

```
SDL_Surface * surf = TTF_RenderUTF8_Blended(font, text, color);
if (!surface)
   throw std::runtime_error("Could not generate the surface");

Gosu::Bitmap temp;
temp.resize(surf.width(), surf.height());
std::memcpy(temp.data(), surf.data(), temp.width() * temp.height() * 4);

Gosu::Image image (graphic_target, temp);
image.draw(x, y, z);
```

6.1.4. Mejora de rendimiento

Con la implementación anterior, en cada fase de dibujado se tenía que repetir el proceso completo, aún cuando el texto no cambiase. Esto suponía una pérdida de rendimiento considerable, sobre todo a la hora de hacer el *trasvase* de píxeles de la superficie de `SDL` al mapa de bits de Gosu.

La solución que se utilizó se basó en las implementaciones previas de la clase `Gosu::Font`. La idea principal es utilizar un búffer de imágenes para cada uno de los glifos de la fuente, que se iría rellenando a medida que fuera necesario pintar cada caracter. Posteriormente, se utilizarían las imágenes en ese búffer para pintar el texto indicado.

Así, por ejemplo, si en primer lugar se pide pintar el texto *"Score"*, se renderizarían los caracteres *'S'*, *'c'*, *'o'*, *'r'* y *'e'*, y se añadirían al búffer mencionado. Si más tarde se necesita pintar el texto *"Scoreboard"*, solo sería necesario renderizar los caracteres *'b'*, *'a'* y *'d'*, ya que de los otros ya tenemos la representación gráfica.

Con este proceso se reducen al mínimo las llamadas a las funciones de fuentes de `SDL_ttf`, que solo se ejecutarán una vez por caracter y tipografía.

6.1.5. Recepción

Una vez contamos con una implementación funcional de la clase, se presentó el parche en el foro oficial de Gosu [37]. En poco tiempo, el desarrollador principal verificó el código propuesto e hizo una adaptación al propio código original de la clase `Gosu::Font` para añadir la propuesta.

Por ende, desde la versión 0.7.20, el *parche* forma parte oficial de Gosu, y así se hace indicar en el fichero `TextUnix.cpp` de la distribución oficial:

```
// [...]

// Used for custom TTF files
// Adapted from customFont class by Jose Tomas Tocino Garcia (TheOm3ga)
class SDLTTFRenderer : boost::noncopyable

// [...]
```

Así pues, dado que la solución propuesta se integró en la distribución oficial, nos deshicimos de la clase `customFont` a favor de la actualización de `Gosu::Font`, dado que al fin y al cabo el funcionamiento interno es el mismo.

6.2. Animaciones dinámicas

Una de las decisiones iniciales de diseño fue la de hacer la interfaz gráfica de usuario lo más atractiva posible, intentando utilizar gráficos amigables y, en la medida de lo posible, animaciones y efectos dinámicos.

Con esto, se tornaba necesario crear un sistema de animaciones lo más versátil posible, de forma que dotar a los elementos de la interfaz de movimiento fuera un proceso sencillo.

6.2.1. Antecedentes: animaciones en Flash

Para evitar reinventar la rueda y tener una base estable con la que empezar, se investigaron sistemas de animaciones ya existentes, evaluando diferentes enfoques y aproximaciones, sobre todo en sistemas altamente relacionados con las animaciones. En particular, nos centramos en Adobe Flash (*anteriormente Macromedia Flash*) y la gran cantidad de código disponible relacionado con la generación de animaciones dinámicas.

Especialmente interesante es el trabajo de Robert Penner, un programador americano muy interesado en la programación matemática, que ideó un sistema de animaciones dinámicas para ActionScript [2] 1.0 y 2.0. Penner, en su libro *Programming Macromedia Flash MX* [53] incluyó un capítulo llamado *Motion, Tweening and Easing* (que dada su popularidad acabó ofreciendo de forma gratuita [52]), en el que por primera vez presentó y explicó con detalle su sistema de animaciones.

En el citado capítulo se desgranan las animaciones como ecuaciones dependientes del tiempo y de las posiciones inicial y final, de forma que fuera fácil generar una serie de funciones para determinar la posición de un objeto en cada instante. Una vez presentados los conceptos iniciales, Penner desvela una serie de ecuaciones de movimiento (que acabaron siendo bautizadas y mundialmente conocidas como las *Penner's Easing Equations*). Estas ecuaciones modelan un gran número de movimientos, en función de cómo varía la posición respecto del tiempo (y perceptualmente la aceleración/deceleración del objeto animado).

Además, por cada tipo de ecuación, Robert Penner generó tres tipos de movimientos: de aceleración (*ease in*), de deceleración o frenada (*ease out*), y de aceleración-deceleración (*ease in-out*). Con esto, quedaban cubiertas la práctica totalidad de los tipos de animaciones posibles.

Así, un ejemplo de ecuación podrían ser las cúbicas, que hacen que la posición del movimiento se rija por una función cúbica del tiempo. Si graficamos la relación del tiempo por la posición, ambos de 0 a 1, el resultado sería la gráfica de la figura 6.1.

Tras estudiar bien el código original, se decidió hacer una adaptación en C++, de forma que se adaptara a las necesidades del proyecto.

6.2.2. Adaptación en C++

El primer paso fue portar las ecuaciones propiamente dichas – esto es, las funciones que generaban las posiciones intermedias. Dado que el lenguaje original, ActionScript, es un derivado de EcmaScript [47], la sintaxis es prácticamente idéntica a C++ en cuanto a asignaciones y operadores se refiere, por lo que este paso fue prácticamente transparente.

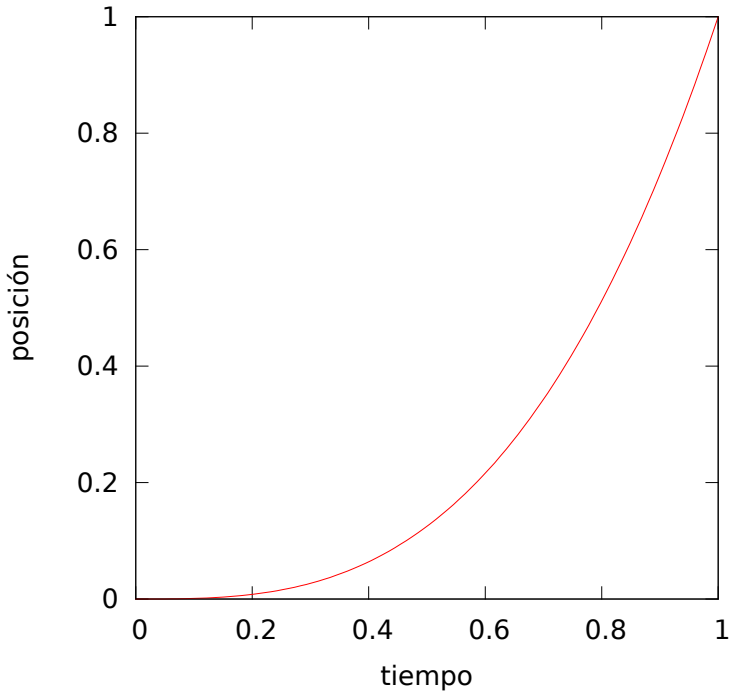


Figura 6.1.: Relación cúbica entre la posición y el tiempo

Seguidamente, se ideó un *wrapper* para estas ecuaciones, de forma que fuera un objeto independiente el que se encargara de calcular las posiciones intermedias de la animación, en lugar de tener que hacerlo los propios objetos animados. Para ello, se creó la clase `Animación`, con las siguientes características:

- Para cada instancia, permite animar un número arbitrario de valores, de forma que con un solo objeto *Animacion* podamos animar la posición horizontal y vertical de un objeto, por ejemplo.
- Permite elegir entre tres tipos de movimiento (*ease in*, *ease out* y *ease in-out*) para tres ecuaciones distintas (cuadrática, cúbica y cuártica), además de una ecuación especial con movimiento de ida y vuelta, y, lógicamente, movimiento uniforme. En total, 11 posibles movimientos.
- Capacidad para atrasar las animaciones, de forma que sea sencillo generar animaciones escalonadas de varios elementos sin tener que recurrir a *callbacks*.

6.2.3. Ejemplo de uso

Supongamos que tenemos una pelota que queremos mover diagonalmente con un movimiento de aceleración, desde la posición 0,0 hasta la posición 150, 300, en 30 pasos. Además, tenemos un cuadrado que habrá de moverse de la posición 0, 150

hasta la posición 150,150 cuando la pelota vaya por la mitad de su camino, y que llegue a la vez que aquella.

Para cumplir este objetivo, crearemos dos objetos `Animación`, uno para la pelota y otro para el cuadrado. En el caso de la pelota estaremos animando dos parámetros, la posición horizontal y la vertical, y la duración será de 30 pasos. El tipo de movimiento será de aceleración, y la ecuación que elegiremos será la cuadrática. Al no indicarse nada, no habrá espera inicial.

```
// Creamos la instancia
Animacion animPelota (2, 30, Animacion::tEaseInQuad, 0);

// Asignamos la pos inicial y final de la coordenada horizontal
animPelota.set(0, 0, 150);

// Coordenada vertical
animPelota.set(1, 0, 300);
```

En el caso del cuadrado el movimiento sólo será horizontal, por lo que animaremos un parámetro. Además, el movimiento comenzará cuando la anterior animación vaya por la mitad (esto es, en el paso 15), y deberá llegar a la vez, por lo que la duración será de 15 pasos.

```
// Creamos la instancia de la clase
Animacion animCuadrado (1, 15, Animacion::tEaseInQuad, 15);

// Asignamos las posiciones inicial y final
animCuadrado.set(0, 0, 150);
```

Una vez inicializados los objetos `Animación`, tendremos que hacer que, en cada iteración del bucle de juego, las animaciones se actualicen. Para ello, la clase `Animación` dispone de un método `update`. Además, haremos uso del método `get` para obtener las posiciones intermedias y así actualizar el objeto

```
// ...
// En la fase update del bucle de juego
animPelota.update();
animCuadrado.update();

imagenPelota -> draw(animPelota.get(0), animPelota.get(1));
imagenCuadrado -> draw(animCuadrado.get(0), 150);
```

Con esto, ya estará lista la animación de ambos objetos. De cualquier modo, la clase `Animación` provee otros métodos que pueden resultar de interés, como el método `finished`, que comprueba si las animaciones han concluído.

Además, las ecuaciones de los movimientos han sido implementadas en forma de funciones estáticas, por lo que es posible acceder a las mismas para hacer cálculos puntuales si fuera necesario. En tal caso, hay que tener en cuenta que todas las ecuaciones reciben cuatro parámetros, estos son, en orden:

- Tiempo transcurrido.
- Valor inicial del atributo a animar.
- *Delta* del atributo en la animación (final - inicial).
- Duración de la animación en pasos.

Con esto, será posible controlar las animaciones de forma independiente, sin necesidad de crear una instancia de la clase.

6.3. Implementación del analizador básico

Como se comentó en la planificación (ver *Desarrollo del calendario*), la viabilidad del proyecto dependía de conseguir una implementación temprana y correcta del analizador de notas, pues es el núcleo de la aplicación y sin él, ésta no tendría ningún sentido.

La implementación del analizador se dividió en dos partes. Por un lado, había que iniciar la captura de audio, lanzando el subsistema de sonido y empezando a capturar datos. Por otro lado, se tenía que hacer el análisis de los datos leídos para determinar qué nota se estaba tocando.

6.3.1. Lanzando el subsistema de audio

A lo largo del proyecto se han utilizado distintas bibliotecas de sonido, aunque en general la filosofía de todas era la misma, así que nos centraremos en los detalles de la que finalmente se dio por definitiva, PulseAudio. Dentro de PulseAudio, nos decantamos por la *Simple API* [30], que cubría todas las necesidades del proyecto sin añadir excesiva complejidad.

El principal concepto en una API de sonido de bajo o medio nivel es el **flujo de sonido** o *stream*. Los flujos pueden ser de entrada, de salida o dúplex, y tienen una serie de características según las necesidades de la aplicación. En PulseAudio, al tratarse de un servidor de sonido y no de una simple biblioteca, exiten algunos parámetros necesarios más. Se utiliza la función `pa_simple_new`, con bastantes parámetros, entre los que se pueden encontrar:

Servidor Normalmente será el servidor de sonido por defecto.

Nombre del cliente y de flujo Identifica la aplicación (*oFlute*) y el flujo (*record*) frente al servidor de audio. Este nombre podrá verse, por ejemplo, en la aplicación de control de volumen (`pavucontrol`).

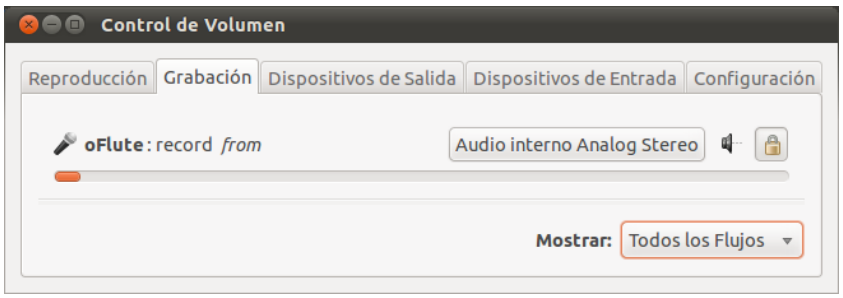


Figura 6.2.: Control de volumen, mostrando el cliente *oFlute*

Tipo de flujo En nuestro caso será de entrada, por lo que se utilizó la macro `kPA_STREAM_RECORD`.

Tipo de *sample* Definirá el tipo de datos que se usará para representar los sonidos.

Opciones de búffering Según busquemos rendimiento, fiabilidad, poca latencia... tendremos que variar las opciones de búffering.

Una vez hayamos creado un flujo, que correrá en segundo plano, tendremos que leer los datos que arroja. Para ello, la API simple de PulseAudio provee la función `pa_simple_read`, que nos permitirá volcar en un búffer temporal un fragmento de audio. La elección del **tamaño del búffer** es otro de los valores que repercutirá enormemente en el rendimiento del sistema.

Finalmente, una vez hayamos terminado de leer datos, podremos liberar el flujo utilizando `pa_simple_free`. Con esto, ya podremos abrir, leer y cerrar el flujo de sonido del micrófono.

La parte más compleja a la hora de tratar con flujos es la de elegir los parámetros apropiados. Son muchísimos los valores que hay que configurar: frecuencia de muestreo, tamaño del búffer, número de canales, longitud del búffer intermedio, latencia deseada... Si no sabemos exactamente en qué influye cada uno, acabamos en un proceso de prueba y error que puede resultar muy tedioso.

6.3.2. Análisis del audio capturado

La segunda parte del proceso consiste en procesar y analizar el audio capturado en cada momento, de forma que podamos conocer qué nota se está tocando con la flauta a través del micrófono.

Como se ha comentado previamente, el alma del procesamiento en este caso es la aplicación de la **transformada rápida de Fourier** (3.1.2). Esta herramienta será

capaz de descomponer los fragmentos de sonido en sus componentes de frecuencia. La biblioteca usada en este caso es KissFFT [39], que da unos resultados muy buenos en cuanto a rendimiento y precisión.

En el caso que nos concierne, un procesamiento básico de la señal será suficiente para poder discernir la nota, por lo que podremos trabajar con transformadas de fourier reales (no complejas). Para ello, KissFFT ofrece variantes de sus funciones, muy optimizadas.

El primer paso es inicializar la biblioteca. KissFFT es capaz de utilizar el mismo búffer entre distintas iteraciones de un cálculo, por lo que ahorramos el tiempo de guardar y liberar memoria. Esta inicialización se guarda en un objeto de configuración.

```
kiss_fftr_cfg configFFT = kiss_fftr_alloc (BUFSIZE, 0, NULL, NULL);
```

El siguiente paso será realizar el cálculo sobre el búffer de sonido. La forma en que el algoritmo FFT almacena sus resultados depende de dos parámetros principalmente: la frecuencia de muestreo y el tamaño del búffer de entrada. En el caso de la FFT real, KissFFT devuelve el espectro positivo, que contendrá tantos valores (en forma de números complejos) como el búffer de entrada más uno. Cada uno de estos elementos representará la intensidad (y fase) de una componente de frecuencia en la muestra inicial.

Pero, ¿cómo asociamos cada componente a la frecuencia apropiada? Ahí es donde entra la frecuencia de muestreo. En el vector que arroja el cálculo del FFT, haremos una simple regla de tres: el primer elemento corresponderá a la menor frecuencia (0Hz), y el mayor elemento a la mayor frecuencia audible (en nuestro caso será 22050Hz, ya que la frecuencia de muestreo es de 44.1kHz, por el teorema de Nyquist – sección 3.1.3).

$$\begin{array}{rcl} \text{Pos. final} & \longrightarrow & 22050Hz \\ \text{Pos. X} & \longrightarrow & \text{Frecuencia Y} \end{array}$$

Una vez identificadas las frecuencias en el resultado de la operación, es necesario computar la **magnitud** de cada una de ellas, hallando el módulo del número complejo. Tras esto, la **frecuencia fundamental** (que representará la nota que está tocando la flauta) *debería* ser la que mayor magnitud tenga.

6.3.3. Acotando el resultado

El siguiente paso fue analizar las frecuencias que proveía la flauta dulce para las nueve notas que puede emitir utilizando técnicas básicas – esto es, sin contar las notas que se pueden emitir utilizando técnicas avanzadas, como tapar a medias ciertos orificios. De este estudio se extrajeron los siguientes resultados (tabla 6.1).

Frecuencia aproximada	Nota
523.25 Hz	Do, octava 5
592.163 Hz	Re, octava 5
656.763 Hz	Mi, octava 5
699.829 Hz	Fa, octava 5
785.692 Hz	Sol, octava 5
893.628 Hz	La, octava 5
1001.29 Hz	Si, octava 5
1076.66 Hz	Do, octava 6
1195.09 Hz	Re, octava 6

Cuadro 6.1.: Frecuencias aproximadas de las notas de la flauta dulce

Redondeamos el rango de frecuencias a $[450, 1500]$, cualquier sonido fuera de ese rango no se considera una nota válida. En el resto de casos, se compara la frecuencia fundamental obtenida con los valores en la tabla, y se devuelve la nota apropiada.

6.3.4. Ejecución concurrente

Uno de los problemas que se encontraron fue que, inicialmente, el proceso de análisis se incluyó dentro de la fase de cálculo de la lógica dentro del *game loop*. El resultado fue bastante poco satisfactorio, ya que el cálculo de la FFT ralentizaba mucho el hilo, que tenía que esperar a que concluyera para seguir renderizando el siguiente fotograma.

Así, se decidió, como suele ser habitual en el desarrollo de videojuegos, mover el control y análisis del audio a un hilo aparte. Para ello, se hizo uso de la biblioteca de hilos de Boost, que resulta mucho más amigable y sencilla de usar que otras alternativas como los *POSIX Threads*.

A nivel de implementación, el uso de hilos en Boost es muy sencillo. Es necesario tener una clase con el operador de función – esto es, `operator()` – sobrecargado. Creamos un objeto de esa clase y se lo pasamos al constructor de `boost::thread`, que llamará al anteriormente citado operador en una copia del objeto pasado.

```
// Instancia de la clase
ClaseCallable objeto;

// Hilo, crea una copia de objeto y llama al operador ()
boost::thread hilo(objeto);

// [...]
```

```
// Esperamos a que termine el hilo
hilo.join();
```

Uno de los problemas que rápidamente se encontraron fue que, al lanzar el hilo, se hace una copia del objeto referenciado. Esto resultaba un problema, ya que el analizador hace cierta inicialización previa, y al duplicarse el objeto ocurrían problemas. El resultado fue utilizar `boost::ref`, que permite pasar referencias por copia.

```
// Hilo sobre el propio objeto, no se hace copia
boost::thread hilo(boost::ref(objeto));
```

6.4. Gestión de estados

La gestión de estados es uno de los retos principales a la hora de desarrollar un videojuego. Es necesario contar un sistema de gestión de estados robusto, que permita pasar de un estado a otro de forma sencilla, sin errores, y manteniendo información entre transiciones si fuera necesario.

En el caso de **oFlute**, el gestor de estados no necesitaba funciones avanzadas; las secciones son consecutivas, por lo que no hay necesidad de incluir concurrencia. Así pues, se editó el gestor de estados en base a la clase `Gosu::Window`. Esta clase modela, mediante métodos, las diferentes fases del *game loop*, a saber:

1. Gestión de la entrada del usuario, mediante los métodos `buttonDown` y `buttonUp`.
2. Procesamiento de la lógica, en el método `update`.
3. Dibujado de los gráficos, en el método `draw`.

Así, se creó una clase `Estado` con los mismos métodos, y una clase `Juego` (heredera de `Gosu::Window`), que sirviese de controlador de estados. Inicialmente, el gestor tenía un método `cambiarEstado`, que descargaba de memoria el estado actual y cargaba el siguiente. Durante un tiempo funcionó bien, pero llegó un momento en que empezaron a aparecer fallos de memoria difíciles de acotar.

Tras bastantes días de depuración, finalmente el problema se encontró en el *momento* en el que se hacía la carga y descarga de estados. El conflicto surgía porque se hacían llamadas para cambiar de estado *en cualquier momento*, ya fuera en la etapa de lógica, de dibujado, o de gestión de la entrada. En el instante de la llamada, se descargaba el estado actual y se cargaba el siguiente. Esto originaba que las tareas que se habían lanzado, como por ejemplo las imágenes pendientes de dibujar, se ejecutasen sobre objetos que habían sido reemplazados en memoria. A menudo las imágenes se encontraban en búferes de Gosu y no había problema, pero una vez se integró el analizador de notas, el sistema empezó a fallar.

La solución fue bastante trivial. Se creó un sistema con una pequeña cola de un elemento en el gestor de estados (la clase `Juego`). En el momento en que alguien

quisiese cambiar de estado, se añadía a la cola el nuevo estado a cargar, y se activaba una bandera. Tras completar esa iteración del *game loop*, al principio de la siguiente se hacía la carga y descarga de estados.

Este sistema de gestión de estados se extendió para incluir estados concurrentes (por ejemplo, para mostrar menús de pausa), y aunque no se incluyó en oFlute, está prevista su integración en la próxima versión de Freegemas [44].

6.5. Representación de canciones y lecciones

El sistema de lecciones y el de canciones son los elementos que más dinamicidad aportan a oFlute. El diseño e implementación de ambos dieron lugar a variadas cuestiones de diseño. Una de las más importantes fue la forma de representar lecciones y canciones en el sistema de ficheros.

Desde un principio se decidió que la representación de los ficheros de lecciones y canciones debería ser sencilla y tener un formato cercano al usuario, para que se pudiesen crear y modificar mediante el uso de un editor de texto plano.

El primer paso fue descartar por completo el uso de una estructura o notación propia, ya que dificultaría el proceso de parseo, imposibilitando usar bibliotecas de terceros para procesar los ficheros de entrada.

El siguiente paso fue comprobar las tecnología de representación más habituales y utilizadas en el mercado. Se barajaron bastantes posibilidades, y finalmente redujimos las posibilidades a dos opciones.

JSON

JSON (*Javascript Object Notation* [19]) es un formato de intercambio de datos muy sencillo y liviano, basado en un subconjunto de la sintaxis de JavaScript. Hoy en día es el formato más utilizado en las aplicaciones web, ya que el texto que representa la estructura es mínimo.

Se estructura en torno a dos elementos principales:

Pares Formados por una clave o nombre, y un valor. Es la forma principal de representar un objeto o registro:

```
objeto = { nombre : "Pedro" };
```

Conjuntos Que representan arreglos o vectores tradicionales. Lo habitual es que formen parte de un *objeto*.

```
objeto = {coordenadas : ["norte", "sur", "este", "oeste"]};
```

Las ventajas de usar JSON son varias. Por un lado, el marcado necesario es mínimo, por lo que el tamaño de los ficheros es muy reducido. Por otro, la sintaxis es sencilla y se limita a las dos reglas anteriormente citadas.

XML

XML [40] son las siglas, en inglés, de *eXtensible Markup Language* – lenguaje de marcado extensible, un especificación para la creación de documentos con marcas definidas por el usuario.

Utiliza una serie de *etiquetas*, organizadas de forma jerárquica, que siguen la forma `<nombre>`, donde *nombre* es la identificación del elemento que se está representando. Las etiquetas deben cerrarse utilizando la sintaxis `</nombre>`. Entre las etiquetas de apertura y cierre pueden anidarse otras etiquetas, así como información en formato texto.

```
<?xml version="1.0" ?>
<Documento>
  <Nombre>Nombre del documento</Nombre>
  <Autor>Autor del documento</Nombre>
  <Secciones>
    <Seccion>Lorem ipsum dillum</Seccion>
    <Seccion color_texto = "rojo">Sit amet, consectetuer</Seccion>
  </Secciones>
</Documento>
```

XML ofrece varias ventajas, desde el punto de vista de este proyecto, sobre JSON:

- Aunque la cantidad de texto necesario para marcar un documento es mayor que en el caso de JSON, las etiquetas de cierre aportan mayor legibilidad al documento, siendo más sencillo saber dónde termina cada bloque.
- JSON solo permite introducir contenido como propiedades de objetos, usando el segundo elemento de los pares *clave/valor*. En XML, es posible introducir contenido tanto dentro de las etiquetas como en forma de atributos de las mismas, de forma que con un mismo elemento podamos tener varias unidades de información.
- XML es un lenguaje con más tiempo y uso que JSON, y las herramientas y el soporte existente es mucho más amplio. La inmensa mayoría de editores ofrecen resaltado de sintaxis para esta clase de ficheros, existen herramientas de validación XML, etcétera.

Por estas y otras razones, finalmente, se acabó eligiendo XML como medio de representación de las lecciones y las canciones en oFlute.

7. Pruebas

En este capítulo se detallan las pruebas a las que se ha sometido **oFlute**. Habitualmente, probar un videojuego o aplicación con gran interactividad suele ser un proceso complejo. Dada la gran cantidad de situaciones distintas que pueden tener lugar, es difícil recrear y probar todos los escenarios posibles. Tanto es así que, en los grandes equipos de desarrollo, hay empleados que se dedican al completo a probar los productos.

En oFlute, las pruebas se han organizado al final de cada iteración, comprobando el correcto funcionamiento de cada nueva funcionalidad, por separado, y luego integrando el conjunto. Además, ciertos módulos no asociados a iteraciones (como el sistema de animaciones, sección 6.2) también fueron probados de manera independiente.

7.1. Pruebas unitarias

Las **pruebas unitarias** se encargan de comprobar el correcto funcionamiento de módulos y fragmentos por separado. En nuestro caso, todos los módulos fueron probados de forma independiente, utilizando a tal efecto ramas de desarrollo alternativas (*branches*), antes de integrarlos en la rama de desarrollo principal.

Mención especial merecen las pruebas unitarias relacionadas con el analizador básico de notas, en la segunda iteración del proyecto. Tal y como se comentó en la sección 6.3, al no tener experiencia previa se tuvieron que hacer bastantes pruebas con el código relacionado con el control del sonido para que el rendimiento fuera correcto y no apareciesen problemas. Las pruebas unitarias en este módulo dieron lugar a decisiones de cambios de bibliotecas en este módulo. Además, al ejecutarse en un hilo separado, el código del analizador era propenso a generar fugas de memoria si no se controlaban bien la vida y visibilidad de las variables utilizadas.

7.2. Pruebas de integración

Según se comprobaban los módulos de manera individual, se iban integrando en la rama de desarrollo principal, pasando a formar parte de la aplicación. Esta integración también ha estado sujeta a pruebas, por un lado comprobando que los módulos no

cambiaban su comportamiento al ser integrados con el resto, y por otro verificando que el funcionamiento general del sistema siguiera siendo correcto.

Como se comentó en la sección 6.4, uno de los errores que más dificultad mostró, asociado a la gestión de estados, fue también el más difícil de encontrar y reproducir. Gracias a las pruebas de integración y al uso de herramientas de depuración, su resolución fue un éxito y sentó las bases para un nuevo gestor de estados (sección 6.4).

7.3. Pruebas de jugabilidad

Las **pruebas de jugabilidad** permitieron encontrar y rellenar lagunas en el proyecto, haciendo que su uso fuera más sencillo para el usuario. En esta etapa del proyecto se contó con la ayuda de usuarios reales inicialmente ajenos al proyecto.

Entre las mejoras que surgieron a raíz de las pruebas de jugabilidad se encuentra la implementación de la **barra de resaltado** (ver *Manual de usuario*), que aparece durante la interpretación de las canciones para ayudar al usuario a conocer qué notas está tocando en cada momento. Esto mejoró en gran medida las puntuaciones de los jugadores en las partidas.

7.4. Pruebas de interfaz

Por último, una vez ajustada la jugabilidad y verificado que los módulos funcionan bien de forma independiente y en integración, se dedicó un tiempo a comprobar que la interfaz gráfica de usuario cumplía los requisitios mínimos que se esperaban.

En particular, se hizo mucho hincapié en la duración de cada una de las animaciones de cada sección. Animaciones largas podían llegar a ser tediosas, mientras que unas animaciones cortas darían una desagradable sensación de premura que no interesaba. Una vez ajustadas, se implementó la opción de poder omitir las animaciones en pantalla, en cualquier sección y momento, mediante la pulsación de la tecla escape. Esto permitiría que jugadores avanzados pudieran llegar rápidamente a la sección de su interés de forma rápida.

Otra de las adiciones que surgieron tras las pruebas de interfaz fueron los atajos de teclado. Se añadieron atajos para las opciones del menú principal, así como el menú de selección de canciones y el de selección de lecciones. También se habilitó la tecla escape para volver atrás en cualquier momento.

8. Conclusiones

Durante el transcurso del desarrollo de oFlute, y sobre todo al término del mismo, se han obtenido unas conclusiones y unos resultados, tanto de forma personal como para con la comunidad, que intentaremos reflejar en este capítulo.

8.1. Objetivos cumplidos

Al término del desarrollo del proyecto, el proyecto ha completado todos los objetivos a cumplir, presentes en el planteamiento inicial y detallados en la sección *1.2*. En particular:

- Se llevó a cabo la creación del módulo de análisis de sonido, consiguiendo una efectividad en la detección de las notas muy alta.
- Se hizo uso del módulo en una sección de análisis de notas y en el propio juego de interpretación de canciones, con resultados muy satisfactorios y funcionamiento correcto.
- Además, el sistema de lecciones planteado se llevó a cabo correctamente, consiguiendo un motor dinámico bastante potente y fácilmente ampliable, con opción a añadir nuevas lecciones al sistema (apéndice E).
- Finalmente, todos los objetivos se llevaron a cabo respetando la premisa de mantener una interfaz de usuario amigable y fluida, agradable a la vista y sencilla de usar.

8.2. Conclusiones personales

oFlute ha sido el proyecto más longevo al que me he enfrentado hasta ahora. A pesar de tener conciencia de la envergadura del mismo desde el principio, el tiempo para completarlo ha superado todas mis expectativas, sobre todo en lo que a documentación se refiere.

La causa de esto ha sido, en primer lugar, una planificación inicial algo ineficiente y, por otro lado, mi recelo sobre los procedimientos actuales sobre ingeniería del softwa-

re. De cualquier modo, estoy bastante satisfecho con lo obtenido, y esto me ha servido para aprender a regirme por una planificación de forma más estricta.

8.2.1. Conocimiento adquirido

Gracias a oFlute he aprendido las técnicas básicas de la programación de audio, sobre todo en la parte técnica más que teórica. Es esta parte teórica, sobre todo la de análisis, la que me gustaría reforzar en un futuro, ahora que cuento con las bases para conseguirlo. Como se comentó en el capítulo *Fundamentos*, los videojuegos relacionados con el audio son un nicho aún poco explorado y que puede dar muchas satisfacciones, sobre todo cuando el producto se orienta a un público joven.

Por otro lado, oFlute también me ha ayudado a aprender a usar bastantes tecnologías auxiliares, en algunos casos meras herramientas que han facilitado el trabajo y, en otros casos, elementos que resultaron ser de inestimable ayuda e imprescindible uso al término del proyecto.

Una de estas tecnologías es **Boost** [6], un conjunto de bibliotecas para C++ que amplían en gran medida la biblioteca estándar del lenguaje. Un amplio número de componentes de Boost formarán parte del nuevo estándar C++0x, por lo que me ha servido para ponerme al día en las novedades que están por llegar.

Figura 8.1.: Logotipo de Boost

Utilizando como material de apoyo el libro *"Beyond the C++ Standard Library"* [49], pude conocer una gran parte de las bibliotecas de Boost, incluyendo punteros inteligentes, programación funcional, bibliotecas para hilos, y utilizarlas en el proyecto.

Al tratarse de la principal biblioteca utilizada durante el desarrollo, oFlute me ha provisto de un profundo conocimiento de **Gosu** [58], permitiéndome implementar videojuegos con mucha más fluidez y labrándome un pequeño *framework* personal que utilizar de base en próximos proyectos.

Figura 8.2.: Logotipo de Gosu

Otra de las tecnologías que he aprendido ha sido **GNU Gettext**, que ha servido para internacionalizar el proyecto. Aunque inicialmente se pensó en utilizar una solución propia para la internacionalización el proyecto, posteriormente se decidió usar esta tecnología, ampliamente conocida y utilizada.

8.3. Difusión y derivados

A raíz del desarrollo de oFlute se abrieron varias vías de trabajo, en forma de actividades de difusión de conocimiento, proyectos informáticos y participación en comunidad.

8.3.1. Conocimiento generado

Además de servir como recurso para el correcto transcurso del proyecto, los conocimientos adquiridos me permitieron, en numerosos casos, generar documentación adicional e impartir talleres relacionados las tecnologías previamente mencionadas.

En cuanto a Boost, se llevó a cabo un taller durante los *Cursos de Verano de la OSLUCA* [8], en el que se explicaron las partes más importantes de esta colección de bibliotecas, con numerosos ejemplos de muchas de ellas. Toda la documentación es libre [54].

Por otro lado, impartí un taller [56] sobre la biblioteca Gosu en colaboración con la ADVUCA [5], cuya afluencia superó las 50 personas. Los materiales pueden descargarse libremente [55].

Por último, tras la aplicación de GNU Gettext en oFlute, escribí una guía concisa sobre traducción de proyectos con esta herramienta, que se puede encontrar en el apéndice *Tutorial de traducción de proyectos con GNU Gettext*.

8.3.2. Freegemas

Uno de estos proyectos *"hijos"* de oFlute ha sido **Freegemas** [44], un clon libre del popular juego tipo puzzle *Bejeweled*. Freegemas es multiplataforma, funciona tanto en GNU/Linux como en Windows, y además forma parte oficial de Guadalinex [14], por lo que es posible encontrarlo en los repositorios oficiales.

Además, Freegemas sirvió como base para una serie de tres artículos que publiqué, junto al director del presente PFC, en la revista Linux Magazine [22] sobre desarrollo de videojuegos en C++. Es posible encontrar estos artículos en el archivo de la revista [41] [42] [43] bajo una licencia Creative Commons.

Figura 8.3.: Logotipo de Freegemas

8.3.3. IV Concurso Universitario de Software Libre

Por otro lado, gracias a oFlute tuve la oportunidad de participar en el IV Concurso Universitario de Software Libre [18]. En el transcurso del concurso formé parte de una comunidad muy unida, en la que reinó el apoyo y la ayuda entre los concursantes. La final del concurso se celebró en la Escuela Superior de Ingeniería de Cádiz, en la que oFlute obtuvo una mención especial [38].

Figura 8.4.: IV Concurso Universitario de Software Libre

Previa a la final nacional tuvo lugar la fase local del concurso, en el que el proyecto también recibió un accésit al mejor proyecto de innovación [28].

A. Herramientas utilizadas

En este primer apéndice se hará un repaso por las herramientas que han hecho posible el desarrollo de oFlute, indicando qué tareas se han llevado a cabo con cada una de ellas.

Dado que en la actualidad, la tendencia es la de mover la potencia de cálculo a la *nube*, también se mencionarán las herramientas *online* que han aportado al proyecto.

A.1. Lenguajes y bibliotecas de programación

A.1.1. C++

El lenguaje elegido para elaborar el proyecto fue **C++**. C++ es un lenguaje de programación que extendió el lenguaje original C hacia el paradigma de la orientación a objetos, mediante el uso de clases, y el paradigma de la programación genérica, mediante plantillas, además de incluir muchas otras novedades y facilidades.

Se decidió utilizar C++ por la familiaridad que teníamos con el mismo, así como por la velocidad y optimización que se obtienen en comparación con otros lenguajes, como Java.

A.1.2. Gosu

Gosu [58] es una biblioteca libre para el desarrollo de videojuegos 2D para Ruby y C++. Entre sus características más atractivas se encuentran la aceleración gráfica por hardware, la sencillez de su API, completamente orientada a objetos, y su compatibilidad con sistemas GNU/Linux, Windows y Mac OS.

Esta biblioteca, que está cerca de lanzar su versión 0.8, cuenta con una base de usuarios cada vez mayor, en parte gracias a su relativa popularidad entre los desarrolladores indie y en los eventos de programación rápida de videojuegos.

A.1.3. PulseAudio

PulseAudio [29] (mencionado previamente en la sección 3.3.2) es un servidor de sonido multiplataforma, muy popular en distribuciones GNU/Linux actuales. Se decidió su uso por lo intuitiva y sencilla de utilizar que resultaba su API, así como su estabilidad, en comparaciones con otras opciones previamente probadas.

A.1.4. GNU Gettext

GNU Gettext [13] es un conjunto de herramientas de internacionalización de proyectos muy sencilla y popular. Se utilizó porque es la solución más eficaz y aceptada a la hora de traducir un proyecto. Se puede encontrar un manual introductorio en el apéndice *Tutorial de traducción de proyectos con GNU Gettext*.

A.1.5. KissFFT

KissFFT [39] es una biblioteca que implementa el algoritmo FFT para el cálculo de transformadas rápidas de Fourier. Es una biblioteca pequeña, razonablemente eficiente y portable, con capacidad para realizar operaciones en distintos formatos.

Dado que el módulo de análisis de sonidos se basa en la transformada de Fourier, era necesario encontrar una biblioteca que pudiera realizar el cálculo de forma rápida y precisa. KissFFT resultó ser una candidata ideal para esta tarea.

A.1.6. PugiXML

PugiXML [48] es una ligera biblioteca para el parseo y procesamiento de ficheros XML. Tiene soporte completo Unicode, un parseador muy veloz y capacidad para usar consultas XPath. Además, su implementación apenas ocupa cuatro ficheros de trivial compilación, por lo que resulta muy sencillo incluirla en cualquier proyecto.

Esta biblioteca se utilizó para el procesamiento de los ficheros de canciones y lecciones, que utilizan XML para representar los distintos elementos.

A.1.7. Boost

Boost [6] es un conjunto de bibliotecas para C++ para un gran cantidad de situaciones distintas. Entre sus componentes se incluyen bibliotecas para el procesamiento de textos, herramientas de gestión de memoria, expresiones regulares, parseo de opciones de línea de comandos, adaptadores para programación funcional y un largo etcétera.

Entre sus desarrolladores se encuentran muchos de los mejores programadores de C++, y tal es su calidad y popularidad, que 10 de las bibliotecas que componen Boost formarán parte del nuevo estándar de C++.

En oFlute se han utilizado bastantes partes de Boost: punteros inteligentes, soporte para expresiones regulares, acceso al sistema de ficheros, y operaciones con cadenas. Esto ha simplificado mucho el código, consiguiendo un estilo más limpio y elegante, sin sacrificar funcionalidad alguna.

A.2. Gestión de código

A.2.1. Subversion

Subversion [4], popularmente conocido como **SVN**, es un sistema de control de versiones muy popular y utilizado. Hasta hace unos años era el indiscutible ganador entre los sistemas de control de versiones de código, y en la actualidad sigue siendo la opción principal en muchas de las forjas de código más importantes de internet.

Se basa en un repositorio central, al que se envían los cambios de los ficheros versionados. Además, a diferencia de otros sistemas como CVS, lleva un control de revisiones a nivel global, no a nivel de fichero.

Subversion ha servido, en oFlute, sobre todo como backup online, además de historial de cambios. En más de una ocasión ha sido necesario revertir los cambios, y Subversion ha facilitado mucho el proceso, que en otras circunstancias podría no haber sido posible.

A.2.2. Forjas: RedIris y Google Code

Para llevar un control del código del proyecto y servir como punto de encuentro con otros usuarios y desarrolladores se abrió una forja para oFlute, inicialmente dentro de la **Forja de Conocimiento Libre de la Comunidad RedIRIS**[1], gestionada por la Junta de Andalucía. Esta forja ofrecía, además de un sistema de control de versiones del código, una serie de herramientas para la gestión de bugs, emisión de noticias, listas de correos, etcétera.

Desafortunadamente la forja de RedIris carecía de un mantenimiento adecuado, lo que llevó al traspaso del proyecto a **Google Code** [46], un repositorio de código ofrecido por Google mucho más actualizado y con un mantenimiento adecuado. Una de los atractivos de Google Code es su sistema de páginas wiki, que permite gestionar documentación online utilizando una sintaxis sencilla y potente.

[1] http://forja.rediris.es

A.2.3. Doxygen

Doxygen [11] es una herramienta de generación automática de documentación de código fuente. Basándose en unos comentarios con una sintaxis especial, Doxygen genera documentación en un gran número de formatos: HTML, PDF, RTF, LaTeX...

Esta herramienta se utilizó para generar toda la documentación del código del proyecto en formato HTML.

A.3. Herramientas de diseño y diagramas

A.3.1. Adobe Photoshop

Adobe Photoshop [3] es la herramienta de edición gráfica de mapa de bits más popular y utilizada en el mundo. Su longeva historia y profuso desarrollo ha hecho que se convierta en la herramienta estándar a la hora de trabajar con imágenes no vectoriales.

Aunque no se trata de una herramienta libre, se utilizó esta aplicación para la creación de todo el apartado visual de oFlute. Todas las secciones fueron diseñadas con esta herramienta, obteniendo un apartado gráfico atractivo y minimalista.

A.3.2. Inkscape

Inkscape [17] es una herramienta libre de edición de gráficos vectoriales. A diferencia de Adobe Photoshop, Inkscape trabaja con formas definidas matemáticamente, que es posible escalar y distorsionar sin causar pérdida de datos. Esto hace que sea la herramienta perfecta para trabajar con logotipos y diagramas.

Esta aplicación ayudó al diseño del logotipo oficial de oFlute, así como al ajuste de ciertos diagramas pertenecientes a la documentación.

A.3.3. Dia

Dia [10] es un editor de diagramas libre, parte de la suite ofimática de GNOME. Tiene un diseño modular que le permite abordar una gran variedad de diagramas distintos, desde diagramas de flujo hasta modelos entidad-relación, UML, etcétera.

Este software se empleó para la generación de la mayor parte de los diagramas presentes en esta memoria.

A.3.4. BoUML

BOUML [7] es una herramienta libre para diseñar diagramas siguiendo la notación UML. Cuenta con utilidades de generación automática de código a partir de diagramas en lenguajes C++, Java, PHP, Python e IDL. Es multiplataforma y permite la generación de varios tipos de diagramas: diagramas de clases, de interacción, de secuencia, de casos de uso, etc

A.3.5. ImageMagick

ImageMagick [16] es un conjunto de utilidades de línea de comandos para mostrar, editar y convertir ficheros de imagen. Resultan especialmente útil a la hora de procesar un gran número de imágenes de forma automática, como por ejemplo cuando es necesario convertir varias imágenes de un formato a otro, o cambiar el tamaño de numerosos ficheros.

A.4. Edición de textos

A.4.1. GNU Emacs

GNU Emacs [12] es un potente editor multiplataforma. Su propio manual lo describe como *"un editor extensible, personalizable, auto-documentado y de tiempo real"*. Emacs es tan potente que a menudo los usuarios experimentados no tienen que salir de su interfaz para realizar cualquier operación, ya que desde el propio editor se puede, desde navegar por la red, hasta revisar el correo electrónico.

Emacs ha sido el editor utilizado para escribir todo el código fuente del proyecto, así como toda la documentación en LaTeX. Sus diversos *modos* se ajustan a cada tipo de documento, proporcionando funciones que faciliten y aceleren la edición de los textos.

A.4.2. LaTeX

LaTeX [20] es un sistema de composición de textos, orientado especialmente a la creación de libros, documentos científicos y técnicos que contengan fórmulas matemáticas. Es muy utilizado para la composición de artículos académicos, tesis y libros técnicos, dado que la calidad tipográfica de los documentos realizados con LaTeX es comparable a la de una editorial científica de primera línea.

La presente memoria ha sido escrita y maquetada con LaTeX.

B. Manual de instalación

A continuación se presentan las instrucciones de instalación de **oFlute** en sistemas GNU/Linux basados en paquetería Debian [9], por ser los que más auge están teniendo en la actualidad. Tomaremos como referencia la distribución Ubuntu 10.10 Maverick [36].

B.1. Instalación de dependencias

El primer paso a la hora de instalar oFlute es hacernos con las dependencias del proyecto. Para ello, utilizaremos el gestor `apt-get`, pero para otros sistemas el procedimiento será similar.

Primero, instalaremos las dependencias de la biblioteca Gosu:

```
sudo apt-get install g++ libgl1-mesa-dev libpango1.0-dev \
libopenal-dev libsndfile-dev libxdamage-dev libsdl-ttf2.0-dev \
libboost-dev libfreeimage3 libfreeimage-dev
```

Seguidamente, instalaremos el resto de dependencias:

```
sudo apt-get install libboost-regex-dev libboost-filesystem-dev \
libboost-thread-dev libpulse-dev
```

B.2. Descarga del código fuente

El siguiente paso será hacernos con una copia local del repositorio, que contiene el código fuente. Al tratarse de un proyecto libre, el repositorio es de acceso público [46].

Haciendo uso del comando `svn`, haremos una copia local del repositorio Subversion [4] en nuestro sistema con la siguiente orden:

```
svn checkout http://oflute.googlecode.com/svn/trunk oflute
```

Con ello, se creará una carpeta llamada `oflute` en la que se encontrarán los ficheros de la rama principal (`trunk`) del proyecto.

B.3. Compilación

Como se ha comentado previamente, será necesario compilar la biblioteca Gosu previamente. Para ello, se ha facilitado un objetivo en el script de compilación `Makefile` [23] que automatiza el proceso.

Así, el primer paso será, estando en la carpeta recién creada, utilizar el siguiente comando:

```
make libgosu
```

Una vez concluida la compilación de Gosu, pasaremos a compilar el propio proyecto mediante la siguiente orden:

```
make
```

B.4. Configuración del micrófono

Antes de poder lanzar el juego, será necesario configurar las opciones de audio del sistema operativo. Por defecto, la entrada de audio suele estar mutada, por lo que será imposible que la aplicación tenga acceso al sonido proveniente del micrófono.

Siguiendo con la referencia de Ubuntu, iremos al menú *Sistema*, luego a *Preferencias* y, finalmente, elegiremos *Sonido*. Una vez ahí, pasaremos a la pestaña *Entrada* y, en caso de que estuviera marcada, desmarcaremos la opción *Silenciar*, tal y como muestra la imagen.

Figura B.1.: Panel de configuración del micrófono

B.5. Ejecución

Al término de las anteriores etapas, se habrá generado un ejecutable de nombre `oflute`, por lo que será posible lanzar la aplicación a través del siguiente comando:

```
./oflute
```

C. Manual de usuario

En este capítulo se explicará el funcionamiento de la aplicación desde el punto de vista del usuario final, detallando las opciones de cada una de las secciones.

Para poder acceder a la aplicación, tal y como se explicó en el apéndice *Manual de instalación*, será necesario utilizar el comando:

```
./oflute
```

C.1. Ejecución inicial

Al lanzar la aplicación, se abrirá una ventana y aparecerán, consecutivamente, la pantalla de créditos del autor y la pantalla de presentación del juego. Pasados unos momentos éstas desaparecerán, aunque tiene la opción de omitir cada una de las pantallas pulsando la tecla escape.

Figura C.1.: Pantalla de créditos del autor

Figura C.2.: Pantalla de presentación del juego

Una vez desaparezcan ambas pantallas, se iniciará una animación para mostrar el menú principal. Puede cancelar la animación y mostrar rápidamente el menú principal pulsando la tecla escape.

Figura C.3.: Pantalla del menú principal

Cuando las animaciones concluyan, el menú principal ya será completamente funcional. Podremos acceder a las diferentes opciones, ya sea haciendo click con el ratón en los botones de pantalla, o pulsando alguna de las teclas de acceso directo. Las secciones y teclas asignadas son las siguientes:

- **Analizador de notas** (*tecla 1*) Abre el analizador de notas.
- **Canciones** (*tecla 2*) Pasa al menú de selección de canciones.
- **Lecciones** (*tecla 3*) Abre el menú de elección de lecciones.
- **Calibrar micrófono** (*tecla 4*) Lanza el sistema de calibración del micrófono.
- **Salir** (*tecla 5*) Cierra la aplicación

Lo recomendado, en la primera ejecución de **oFlute**, es lanzar la sección *Calibrar micrófono*, ya sea pulsando el botón con el ratón o mediante la tecla 4. Con esto nos cercioramos de que el programa es capaz de distinguir el sonido del micrófono del ruido de fondo. Una vez hecho, ya estará todo listo para acceder al resto de secciones.

C.2. Sección – calibrar micrófono

Pulsando el botón correspondiente, o mediante la tecla 4, pasaremos a la sección de calibración del micrófono. En primer lugar, aparecerá el siguiente mensaje:

Figura C.4.: Pantalla inicial de calibración del micrófono

Como se indica en el mensaje, para comenzar la calilbración el usuario deberá pulsar la tecla `espacio` y guardar silencio en el micrófono durante dos segundos. En ese momento, aparecerá el siguiente mensaje:

Figura C.5.: Mensaje al inicio del calibrado

Una vez pasen los dos segundos y concluya la calibración, se mostrará el siguiente mensaje:

Figura C.6.: Mensaje al final del calibrado

Cuando concluya el proceso, la aplicación habrá guardado el valor umbral de ruido del micrófono, lo que facilitará el análisis del sonido de la flauta.

Hay casos en los que la calibración será defectuosa, y un mensaje informará de ello. En tal caso, no habrá más que comprobar la configuración del micrófono y repetir el proceso de calibrado.

C.3. Sección – analizador de notas

Al pulsar el botón correspondiente a la sección del analizador de notas, mediante una animación aparecerán los elementos de esta parte de la aplicación (figura C.7 en página 115).

Una vez terminen de mostrarse todos los elementos, el usuario podrá empezar a utilizar la funcionalidad de la sección. En concreto, el analizador de notas nos permite ver reflejada en el pentagrama la nota que toquemos con la flauta.

Su uso, pues, es muy sencillo. Simplemente tendremos que tocar nuestra flauta de forma que el micrófono sea capaz de captar su sonido. Si lo hacemos correctamente, el analizador mostrará en cada momento la nota que está tocando sobre el pentagrama.

Una vez hayamos acabado de utilizar esta sección de la aplicación, podremos pulsar en el botón *Volver*, o en la tecla `escape` del teclado, lo que nos llevará de vuelta al menú principal.

Figura C.7.: Pantalla del analizador de notas

C.4. Sección – lecciones

Si seleccionamos la opción *Lecciones* en el menú principal, o alternativamente presionamos la tecla 3 del teclado, la aplicación mostrará una animación para ocultar el menú principal y nos mostrará el menú de selección de lecciones.

Figura C.8.: Pantalla del menú de selección de lecciones

En cualquier momento podemos hacer click en el botón *Volver al menú*, o pulsar la tecla escape del teclado, para ir a la sección anterior.

En el menú de selección de lecciones podremos encontrar los siguientes elementos:

- **Título** de la lección
- **Descripción** de la lección
- Botón **Comenzar lección**.
- Botón **Anterior lección**.
- Botón **Siguiente lección**.
- Botón **Volver al menú**.

Mediante los botones de *anterior* y *siguiente lección* podremos navegar entre las diferentes lecciones cargadas en el sistema. Haciendo uso de los paneles de *título* y *descripción* nos informaremos sobre la lección elegida. Una vez que tengamos claro la lección que queremos tomar, pulsaremos en el botón *comenzar lección*.

Una vez que decidamos comenzar la lección, el sistema ocultará los elementos del menú de selección de lecciones mediante animaciones y, posteriormente, cargará y mostrará los elementos de la lección elegida.

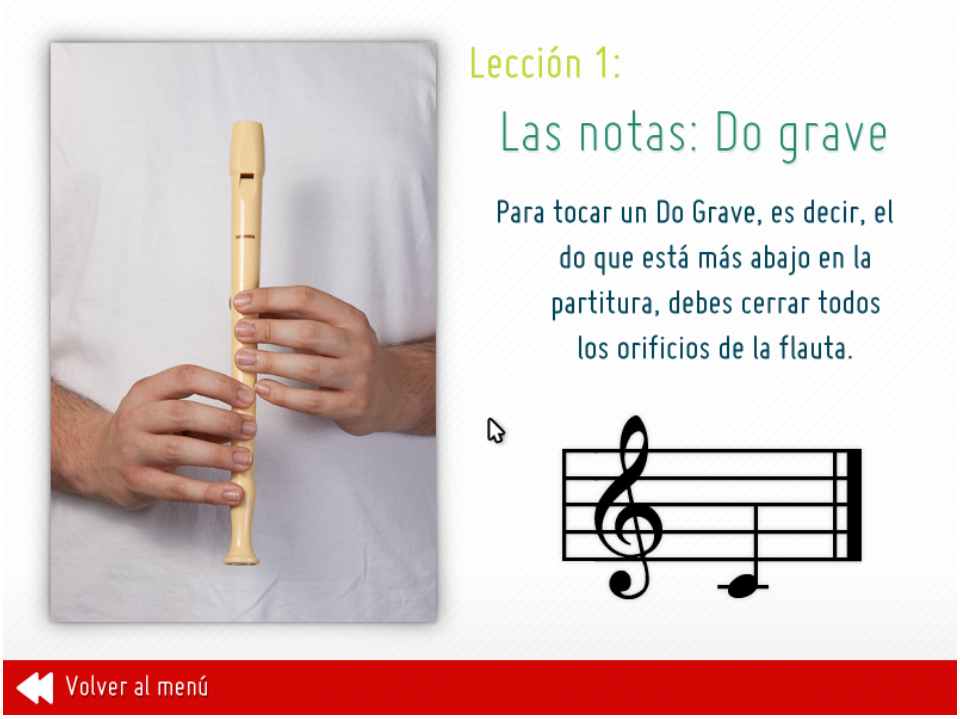


Figura C.9.: Pantalla de lección de ejemplo

Las lecciones se muestran en forma de conjunto de elementos multimedia – imágenes y texto – de fácil comprensión, que el usuario podrá leer y estudiar de forma

independiente. En futuras versiones de la aplicación se podrán utilizar lecciones de varias etapas, así como integrar sonidos y otro tipo de multimedia.

C.5. Sección – canciones

El usuario podrá acceder a esta sección desde el menú principal, pulsando en la botón *Canciones*, o con la tecla 2 del teclado.

Al acceder, desaparecerá el menú principal y se mostrará el menú de selección de canciones.

Figura C.10.: Pantalla del menú de selección de canciones

Esta pantalla consta de los siguientes botones:

- **Anterior canción**, representado con una flecha hacia arriba.
- **Siguiente canción**, representado con una flecha hacia abajo.
- **Comenzar canción**, simbolizado con la palabra *OK*.
- **Volver**.

Mediante los botones *anterior* y *siguiente canción*, el usuario podrá elegir el tema a interpretar con la flauta. Una vez esté resaltada la canción correcta, el usuario deberá pulsar el botón *Comenzar canción – OK* para que dé comienzo la interpretación.

También es posible pulsar el botón *volver* para ir de nuevo al menú principal.

C.5.1. Interpretación de la canción

Al lanzar la canción, aparecerá la pantalla de interpretación de canción, que contiene numerosos elementos importantes para el jugador.

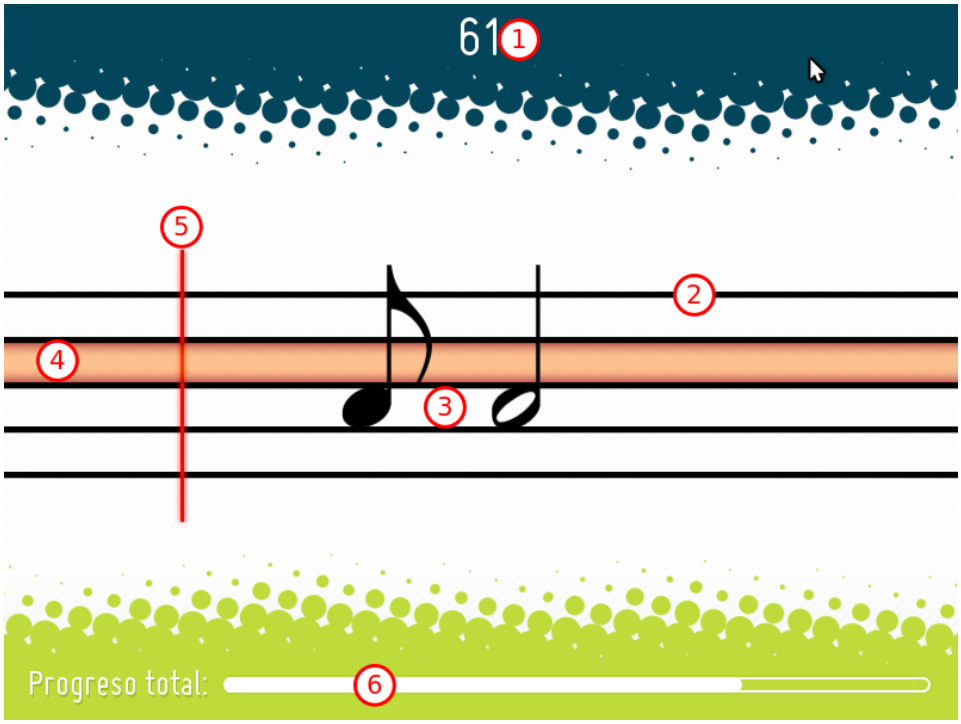


Figura C.11.: Pantalla de interpretación de canción

1. **Marcador** de la puntuación del usuario.
2. **Pentagrama**. Sobre él aparecerán las notas.
3. **Notas**. Éstas irán apareciendo por el lado derecho del pentagrama, y moviéndose hacia la izquierda de forma ordenada y rítmica.
4. **Resaltado de nota**. Resalta la posición en el pentagrama de la nota que está tocando el usuario con la flauta en cada momento. Esto ayuda visualmente a saber si estamos interpretando la nota correcta o no.
5. **Barra de interpretación**. Esta barra indica cuándo debe el usuario empezar a tocar la nota.
6. **Barra de progreso**. Indica el progreso de la interpretación de la canción. Cuando la barra esté completa, la canción concluirá.

La forma de juego es sencilla. Como se ha indicado, las notas aparecerán sobre el pentagrama por el lado derecho, e irán desplazándose hacia la izquierda. Una vez que una nota llegue a la *barra de interpretación*, el jugador deberá tocar con la flauta la nota correcta, de forma constante hasta que llegue el turno de la siguiente nota.

Para ayudar al jugador, la barra de *resaltado de nota* indicará en cada momento qué nota se está tocandoo. Así, si en un instante es necesario tocar un *Sol* pero la barra de resaltado está por encima, el usuario sabrá que debe cerrar más orificios de la flauta.

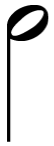
Las posibles **notas** que pueden aparecer son *Do*, *Re*, *Mi*, *Fa*, *Sol*, *La*, *Si*, *Do grave* y *Re grave*. Las posibles figuras que podrán aparecer son las siguientes:

Redonda – Dura cuatro tiempos.

Figura C.12.: Figura musical – Redonda

Blanca – Dura dos tiempos.

(a) Posición normal

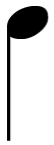


(b) Posición invertida

Figura C.13.: Figura musical – Blanca

Negra – Dura un tiempo.

(a) Posición normal



(b) Posición invertida

Figura C.14.: Figura musical – Negra

Corchea – Dura la mitad que una negra.

(a) Posición normal

(b) Posición invertida

Figura C.15.: Figura musical – Corchea

De manera excepcional, es posible alargar el tiempo de una figura mediante el uso del **puntillo**, que hará que su duración aumente la mitad de su tiempo original. El puntillo se representa mediante un pequeño círculo a la derecha de la base de la nota.

Por otro lado, las siguientes son las figuras que representan los **silencios**. Funcionan igual que las notas normales, solo que el jugador, en lugar de tocar una nota, deberá mantenerse en silencio desde que el silencio toque la barra de interpretación hasta que llegue la siguiente figura.

Silencio de redonda – Dura cuatro tiempos, y se coloca unido, por debajo, a la cuarta línea del pentagrama.

Figura C.16.: Silencio de redonda

Silencio de blanca – Dura dos tiempos, y se coloca posado sobre la tercera línea del pentagrama.

Figura C.17.: Silencio de blanca

Silencio de negra – Dura lo mismo que una negra.

Figura C.18.: Silencio de negra

Silencio de corchea – Dura la mitad que una negra.

Figura C.19.: Silencio de corchea

El final de la interpretación vendrá indicado por la doble barra final. Una vez que ésta llegue a la zona de interpretación, se dará por concluida la canción.

Figura C.20.: Doble barra de final de pentagrama

C.5.2. Resultados de la interpretación

Tras la interpretación de la pieza, aparecerá la zona de puntuaciones.

Figura C.21.: Pantalla de resultados tras la interpretación

En esta pantalla, además del **título** y **subtítulo** de la canción, se mostrará el **porcentaje de aciertos** que ha tenido el jugador en su interpretación. Además, aparecerá una frase de apoyo que variará según el éxito de la partida.

Una vez llegados a este punto, el jugador puede volver a la pantalla anterior pulsando la tecla escape.

D. Manual para añadir nuevas canciones

En **oFlute** existe la posibilidad de añadir nuevas canciones al juego, de forma que los usuarios puedan ampliar el juego y que éste se convierta en una herramienta lúdica prácticamente ilímitada.

Así, en un futuro podrán existir, por ejemplo, packs temáticos de canciones descargables desde internet, con los que seguir practicando nuestra pericia con la flauta dulce.

D.1. Ficheros necesarios

Cada una de las canciones presentes en el juego necesitará un fichero donde se definirán las características y notas de la canción. El nombre de estos ficheros sigue la estructura `songN.xml`, donde *N* es el número de canción a añadir.

Los ficheros de las canciones se encuentran en la carpeta `canciones`, ubicada en el directorio raíz de la instalación de oFlute.

D.2. Estructura del fichero de canciones

D.2.1. Lenguaje XML

Cada fichero de canción es un archivo *XML* [40]. Como se comentó en la sección 6.5, XML es un lenguaje de marcado extensible. La tecnología XML busca dar solución al problema de expresar información estructurada de la manera más abstracta y reutilizable posible.

Todo documento XML tiene una etiqueta *raíz*, de la cual cuelgan todas las demás. En nuestro caso, los ficheros que representan las canciones tendrán como raíz el elemento `Song`. Este es un ejemplo de un fichero completo:

```
<?xml version="1.0" ?>
<Song>
  <Title>Los pajaritos</Title>
  <Desc>Van por el aire</Desc>
```

```
  <BPM>60</BPM>
  <Notes>do5n mi5c fa5c sol5n sol5c sol5c la5n la5c la5c sol5n
   sol5n fa5n fa5n mi5n mi5n re5c mi5c fa5c re5c do5n do5n</Notes>
</Song>
```

D.2.2. Campos iniciales

Título de la canción

El título de la canción se indicará mediante la etiqueta `<Title>`, por ejemplo:

```
<Title>Éste es el título</Title>
```

Descripción de la canción

La descripción de la canción sirve como subtítulo de la misma. Se definirá con la etiqueta `<Desc>`, por ejemplo:

```
<Desc>Ésta es la descripción</Desc>
```

Ritmo

El ritmo de la canción se define como el número de pulsos por minuto – en inglés, **bpm**, *beats per minute*. Podemos definir este atributo con la etiqueta `<BPM>`, por ejemplo:

```
<BPM>60</BPM>
```

D.2.3. Definición de las notas

Ésta es la parte más importante del fichero de definición de canciones. Aquí se indicarán las notas que queremos que aparezcan en el pentagrama y, consecuentemente, que el jugador interprete.

Las notas se escribirán dentro de la etiqueta `<Notes>`, en orden consecutivo y separadas por un espacio.

```
<Notes>do5n mi5c fa5c sol5n sol5c sol5c la5n la5c la5c sol5n sol5n fa5n
fa5n mi5n mi5n re5c mi5c fa5c re5c do5n do5n</Notes>
```

Cada una de las notas deberá representarse según la siguiente expresión regular:

```
(do|re|mi|fa|sol|la|si|xx)(5|6|0)(r|b|n|c)(p)?
```

La descomposición de esta sintaxis es la que sigue:

Altura Se indicará con el texto de la nota (*do*, *re*, *mi*...) o, en el caso de querer incluir un silencio, con el texto *xx*.

Octava La flauta dulce habitualmente solo cubre desde el Do en la quinta octava hasta el Re en la sexta octava. Ésta se indicará con un número, 5 ó 6, ó un 0 si estamos incluyendo un silencio.

Duración Las posibles duraciones son la redonda (4 tiempos), la blanca (2 tiempos), la negra (1 tiempo) y la corchea (medio tiempo). Para indicar el mismo, se usarán las iniciales de los nombres, esto es, *"r"*, *"b"*, *"n"* o *"c"*.

Puntillo Es posible añadir puntillo a las notas, incrementando su duración en la mitad de su tiempo original. Para ello, incluiremos la letra *"p"* al final.

Una vez escritas las notas, ya tendremos el fichero de nuestra canción listo para ser incluido en oFlute. Un ejemplo de fichero de canción completo podría ser el siguiente:

```
<?xml version="1.0" ?>
<Song>
  <Title>Los pajaritos</Title>
  <Desc>Van por el aire</Desc>
  <BPM>60</BPM>
  <Notes>do5n mi5c fa5c sol5n sol5c sol5c la5n la5c
    la5c sol5n sol5n fa5n fa5n mi5n mi5n re5c mi5c
    fa5c re5c do5n do5n</Notes>
</Song>
```

E. Manual para añadir nuevas lecciones

oFlute fue diseñado de forma que se pudiera expandir de manera sencilla. Además de poder añadir canciones nuevas tal y como se explicó en el apéndice *Manual para añadir nuevas canciones*, también es posible añadir nuevas lecciones a la aplicación.

E.1. Ficheros necesarios

Las lecciones necesitan de más ficheros que las canciones, ya que se basan en elementos multimedia para mejorar la experiencia de usuario. Así, una lección dispondrá de un fichero base en formato XML y de una serie de archivos auxiliares de recursos. Por ahora, **oFlute** acepta recursos en formato imagen PNG [27] y fuentes TTF [35], pero en un futuro será posible añadir otros tipos de elementos.

E.1.1. Fichero de lección

El fichero base deberá alojarse en el directorio `lecciones`, dentro de la carpeta raíz de la instalación de oFlute. Además, su nombre debe seguir la estructura `lecN.xml`, donde *N* es el número de la lección a añadir.

E.1.2. Ficheros de recursos

Los ficheros de recursos podrán guardarse en cualquier lugar dentro de la carpeta de instalación de oFlute. De cualquier modo, se recomienda guardar las imágenes en la carpeta `media/lecciones` y las fuentes en la carpeta `media`, ya que es ahí donde están los recursos de las otras lecciones, facilitando la reutilización.

E.2. Estructura del fichero de lección

Como se comentó en la sección D.2.1, todos los documentos XML necesitan un elemento raíz. En este caso, usaremos la etiqueta `<Lec>` como raíz del fichero de lección.

E.2.1. Campos iniciales

Índice

Es posible añadir el índice de la lección, de forma que podamos decidir fácilmente el orden en que aparecerá dentro de la lista de lecciones del sistema.

Para ello, utilizaremos la etiqueta `<index>`, tal que así:

```
<index>1</index>
```

Nombre

El nombre de la lección lo añadiremos utilizando la etiqueta `<nombre>`. Éste se utilizará en la pantalla de selección de lecciones.

```
<nombre>Éste es el nombre de la lección</nombre>
```

Descripción

La descripción de la lección nos permitirá conocer brevemente su contenido sin tener que acceder directamente a la misma. Se empleará la etiqueta `<descrip>` para identificar la descripción.

```
<descrip>Ésta es la descripción,
que puede ser un texto largo.</descrip>
```

E.2.2. Elementos multimedia

Tras los campos iniciales, se indicarán los elementos multimedia. Podremos encontrar dos tipos: imágenes, cargadas a partir de los ficheros de recursos previamente comentados, y textos, que se dibujan de forma dinámica. Para ello, marcaremos una sección con la etiqueta `<elementos>`.

```
<elementos>
...
</elementos>
```

Cabe notar que todos los elementos multimedia aparecerán en pantalla mediante una animación de desvanecimiento o *fade-in*. Para indicar el orden en que éstos aparecerán, las etiquetas de los elementos multimedia tendrán un atributo `wait`, que indica el retraso en el inicio de la animación.

Imágenes

Las imágenes se indicarán mediante la etiqueta simple `<img/>` – *simple*, porque se cerrará a sí misma, desapareciendo la etiqueta de cierre posterior.

Esta etiqueta tendrá los siguientes atributos:

src Indica la ruta al fichero de imagen, relativa al directorio raíz de oFlute.

x, y, z Posición horizontal, vertical y profundidad del elemento en pantalla, en píxeles.

wait Como se ha comentado, es el retraso en la animación de aparición.

Un ejemplo de definición de elemento multimedia en formato imagen podría ser el siguiente:

```
<img src="media/lecciones/sombraFoto.png" x="15" y="13" z="2" wait="30"/>
```

Textos

Los textos utilizan la etiqueta `<texto>`. Dentro de la etiqueta, esto es, entre la etiqueta de apertura `<texto>` y la de cierre `</texto>`, se habrá de escribir el texto que queremos que se muestre. Hay que tener en cuenta que los saltos de línea que insertemos aparecerán tal cual posteriormente en pantalla.

Además, la etiqueta consta con numerosos atributos obligatorios:

tam Indica el tamaño de la fuente en pantalla.

ca, cr, cg, cb Para definir el color del texto utilizamos su definición RGBA [32] (*Red, Green, Blue, Alpha*), esto es, los valores individuales de los canales rojo, verde, azul y alfa (opacidad). Así, cada uno de los atributos indica el valor de un canal:

- **ca** Canal alfa.
- **cr** Canal rojo.
- **cg** Canal verde.
- **cb** Canal azul.

x, y, z Definen la posición horizontal, vertical y profundidad del texto en pantalla.

align Indica la alineación del bloque de texto. Valdrá 1 para alineación a la izquierda, 2 para alineación centrada y 3 para alineación a la derecha.

sombra Indica si el texto tendrá sombra o no, con un valor 0 ó 1. En el caso de que valga 1, habrá que indicar un atributo adicional, **opacSombra**, que indicará la opacidad de la sombra.

wait Retraso en la animación de aparición.

Como ejemplo de bloque de texto, podemos poner el siguiente fragmento:

```
<texto x="580" y="80" z="2" tam="60"
  ca="255" cr="34" cg="139" cb="114"
  align="2" sombra="1" opacSombra="23"
  fuente="media/fLight.ttf" wait="45">Éste es el texto</texto>
```

Así, un ejemplo de fichero de lección con varias imágenes y texto podría ser el siguiente:

```
<?xml version="1.0" ?>
<Lec>
  <index>1</index>
  <nombre>Las notas: Do grave</nombre>
  <descrip>Aprende a tocar la
    nota Do grave.</descrip>
  <elementos>
    <img src="media/lecciones/sombraFoto.png" x="15" y="13"
         z="2" wait="30"/>
    <img src="media/lecciones/1 dograve.png" x="40" y="37"
         z="3" wait="15" />
    <img src="media/lecciones/pentaDo.png" x="465" y="345"
         z="2" wait="55"/>

    <texto tam="45" ca="255" cr="191" cg="219" cb="59"
               align="1" sombra="1" x="390" y="30" z="2"
               opacSombra="0" wait="40">Lección 1:</texto>

    <texto x="580" y="80" z="2" tam="60"
               ca="255" cr="34" cg="139" cb="114"
               align="2" sombra="1" opacSombra="23"
               fuente="media/fLight.ttf"
           wait="45">Las notas: Do grave</texto>

    <texto x="580" y="161" z="2" tam="33"
               ca="255" cr="3" cg="69" cb="90"
               sombra="0" align="2"
           wait="50">Para tocar un Do Grave, es decir, el
      do que está más abajo en la
      partitura, debes cerrar todos
      los orificios de la flauta.</texto>
  </elementos>
</Lec>
```

F. Tutorial de traducción de proyectos con GNU Gettext

F.1. Antecedentes

Durante el desarrolllo del proyecto surgió la necesidad de preparar el proyecto para su traducción. Inicialmente se optó por utilizar un sistema propio de traducción, basado en un script en Python y en una pequeña clase que generaba un diccionario según el idioma elegido. Sin embargo, esta opción resultó ineficiente y, sobre todo, alejada del resto de soluciones estándar a la hora de traducción de proyectos.

Por ello, se decidió investigar sobre las tecnologías de traducción más utilizadas en el panorama del software libre, y finalmente se optó por **GNU Gettext** [13]. Para afianzar los conocimientos y facilitar el aprendizaje a otros desarrolladores, se editó el presente documento.

La primera edición del manual se presentó en el I Hackathón [15] organizado por la Oficina de Software Libre y Conocimiento Abierto de la Universidad de Cádiz, acompañada de un pequeño taller con una demostración en vivo. Se encuentra disponible para su descarga en el repositorio Rodin de la UCA [57] y se puede difundir bajo los términos de la licencia *Creative Commons - Reconocimiento, Compartir igual 3.0* [21]

F.2. Introducción

GNU Gettext es un conjunto de herramientas libres de internacionalización que permite traducir nuestros proyectos de una manera sencilla. Es el sistema de i18n[1] más utilizado, pudiendo encontrarse en una gran cantidad de proyectos.

Las ventajas de usar un sistema como *Gettext* en lugar del típico menú de elección de idiomas dentro de la aplicación son muchas. Por ejemplo, con Gettext el ajuste del idioma es transparente al usuario. Además, ofrece todas las ventajas de usar un sistema establecido y prácticamente estándar.

[1] *"i18n"* es una alternativa a escribir *internacionalización*, el 18 simboliza los 18 caracteres entre la *i* inicial y la *n* final

F.2.1. Internacionalización vs localización

La *internacionalización* de un proyecto consiste en prepararlo de forma que sea capaz de trabajar y presentarse en una multitud de idiomas y configuraciones regionales. Por otro lado, la *localización* trata de coger un programa internacionalizado y darle la suficiente información para que se adapte al idioma y configuración del usuario actual.

F.2.2. El paquete de herramientas de GNU Gettext

GNU Gettext está compuesto de un conjunto de elementos que forman un *framework* de trabajo común. En particular, esto incluye:

- Una serie de directivas y convenciones a la hora de escribir el código fuente de nuestro proyecto.
- Un esquema estándar de organización de ficheros y directorios para guardar los ficheros relacionados con la traducción.
- Una biblioteca que provee funciones relacionadas con las cadenas traducidas.
- Algunas utilidades para la creación y edición de los ficheros con las cadenas de traducción.
- Un modo de edición para **Emacs**.

F.3. Pasos en el proceso de traducción

A la hora de adaptar un proyecto para internacionalizarlo hay que seguir una serie de pasos bastante mecánicos que se repiten cada vez que añadamos o modifiquemos las cadenas de nuestro proyecto.

F.3.1. Adaptación del código fuente

Primero, necesitamos adaptar el código de nuestro programa, marcando de alguna forma las cadenas que queremos que queden traducidas. Podría pensarse que este paso podría ser automático, pero hay cadenas que a buen seguro no queremos que sean traducidas, como por ejemplo parámetros de funciones o mensajes con campos variables, como los que se utilizan con la función `printf`.

Para ello, añadiremos las siguientes líneas al inicio de nuestro fichero fuente:

```
#include <libintl.h>
#include <locale.h>
#define _(x) gettext(x)
```

La biblioteca `libintl` será la encargada de la internacionalización de nuestro proyecto, siendo de especial interés su función `char * gettext (const char *)`, a la que le pasaremos las cadenas originales y nos devolverá la cadena traducida dependiendo del *locale* del sistema. Para hacer menos evidente su uso, utilizamos la macro definida arriba.

> **NOTA:** Habitualmente, es una buena práctica en el desarrollo de un proyecto libre el usar el inglés como idioma por defecto para los mensajes de interacción con el usuario. Gettext supone que se usará el inglés como lenguaje principal, por lo que nos acogerémos a esta práctica.

El siguiente paso será cambiar todas las cadenas que queramos traducir en nuestro proyecto, de forma que en realidad sean siempre llamadas a la función `gettext`, o preferiblemente a su macro `_()`.

```
string mensaje = "Hello world";
```

Pasará a ser:

```
string mensaje = _("Hello world");
```

Algo a tener en cuenta es que es necesario elegir un nombre para el *dominio* de la internacionalización, que por regla general coincidirá con el proyecto. En nuestro ejemplo, este nombre será **oflute**.

En las primeras líneas de nuestro proyecto añadiremos las siguientes instrucciones de inicialización:

```
bind_textdomain_codeset ("oflute", "UTF-8");
setlocale(LC_ALL, "");
bindtextdomain("oflute", "lang" );
textdomain("oflute");
```

Eso le indicará a la biblioteca de i18n cuál es el dominio de la traducción, así como el directorio de las traducciones (`lang`), y pondrá el locale por defecto.

Así pues, un posible fichero `main.cpp` de ejemplo podría ser:

```
#include <iostream>
#include <libintl.h>
#include <locale.h>

#define _(x) gettext(x)

using namespace std;

int main(int argc, char *argv[])
```

```
{
    bind_textdomain_codeset ("oflute", "UTF-8");
    setlocale(LC_ALL, "");
    bindtextdomain("oflute", "lang" );
    textdomain("oflute");

    cout << _("Hello world") << endl;
    return 0;
}
```

F.3.2. Generando los ficheros de traducción

Una vez que tengamos todas las cadenas a traducir apropiadamente adaptadas como se ha comentado antes, pasaremos a crear los ficheros con los que realizaremos las traducciones. Hay varios tipos de estos ficheros:

.POT *(Portable Object Template)* Es el primer fichero que se genera, y contiene todas las cadenas extraídas del código fuente, que servirá luego como plantilla para los ficheros `.po`.

.PO *(Portable Object)* Son los ficheros principales de traducción, los que se editan con las cadenas traducidas. Hay uno por cada *locale* que queramos incluir.

.MO *(Machine Object)* Son la versión binaria de los ficheros `.po`, los que nuestra aplicación usará para leer las cadenas traducidas.

La forma de organizar los diferentes ficheros está bastante estandarizada, de forma que lo más recomendado es seguirla – de no hacerlo podemos tener problemas a la hora de que el programa encuentre los ficheros de traducción.

- En la raíz de nuestro proyecto tendremos una carpeta **po** que albergará el fichero de plantilla `.pot`, en nuestro caso `oflute.pot`, así como los ficheros `.po` de cada *locale*: `en.po`, `es.po`, etc.
- Además, también en la raíz tendremos otra carpeta llamada **lang**. Dentro de ella habrá una carpeta por cada fichero `.po` en la carpeta antes mencionada, y dentro de cada una de ellas, una carpeta `LC_MESSAGES`, que albergará el fichero `.mo` correspondiente, todos con el nombre del dominio, en nuestro caso `oflute.mo`

La estructura de directorios que obtendremos será algo así:

```
lang
|-- en
|   `-- LC_MESSAGES
|       `-- oflute.mo
```

```
‘-- es
    ‘-- LC_MESSAGES
        ‘-- oflute.mo
po
|-- en.po
|-- es.po
‘-- oflute.pot
```

Creando la plantilla .pot

Así pues, el primer paso será generar el fichero `.pot`. Para ello utilizaremos la utilidad `xgettext`. Así pues, creamos los directorios antes comentados y usamos la siguiente expresión:

```
xgettext \
   --package-name oflute \
   --package-version 0.1 \
   --default-domain oflute \
   --output po/oflute.pot \
   --from-code=utf-8 \
   --copyright-holder="Tu nombre" \
   --msgid-bugs-address="tu@mail.com" \
   -s -k_ -C main.cpp
```

La mayoría de las opciones son autoexplicativas, pero es interesante conocer el significado de las que no lo son:

-s Salida ordenada, ordena las cadenas en el fichero de plantilla, útil cuando tenemos muchos ficheros fuente y queremos tener las cadenas organizadas.

-k_ Indica que también busque cadenas marcadas con `_(cadena)` además de `gettext(cadena)`.

-C Indica que el lenguaje es C++.

Con esto tendremos el fichero en `po/oflute.pot`. Es necesario editarlo y cambiar el valor de `CHARSET` (en la línea `Content-Type: ...`) por `UTF-8`, `xgettext` aún no ofrece ninguna opción para autorrellenar este campo.

Creando los ficheros de traducción .po

El siguiente paso será el de crear un fichero `.po` para cada uno de los idiomas a los que queramos traducir nuestro proyecto. Para ello utilizaremos la utilidad `msginit` de la siguiente manera, suponiendo los idiomas inglés y español:

```
msginit -l es -o po/es.po -i po/oflute.pot
msginit -l en -o po/en.po -i po/oflute.pot
```

Al ejecutar los comandos nos pedirán nuestro email, de forma que sea posible recibir feedback sobre la traducción que hagamos.

Si le echamos un vistazo al fichero `po/es.po`, después de todas las cabeceras iniciales, nos encontraremos con las cadenas de nuestro programa de la siguiente manera:

```
#: main.cpp:15
msgid "Hello world"
msgstr ""
```

El formato es muy sencillo: la primera línea indica la situación de la cadena en el código fuente, la segunda es la cadena original, y la tercera es la cadena traducida. Para el fichero en español, quedaría:

```
#: main.cpp:15
msgid "Hello world"
msgstr "Hola mundo"
```

Cabe notar que, como el lenguaje original es el inglés, podemos dejar el fichero `en.po` intacto, ya que por defecto utilizará las cadenas originales.

Creando los binarios .mo

Una vez terminado el apartado anterior, nos queda el último paso, que es generar los ficheros binarios con las traducciones. Para ello, utilizaremos la utilidad `msgfmt`, que convertirá los `.po` en `.mo`, de la siguiente manera:

```
mkdir lang/{es,en}/LC_MESSAGES
msgfmt -c -v -o lang/es/LC_MESSAGES/oflute.mo po/es.po
msgfmt -c -v -o lang/en/LC_MESSAGES/oflute.mo po/en.po
```

La opción **-c** indica que se hagan chequeos ante errores, y la opción **-v** muestra una salida extendida (*verbose*). Con esto, ya tendremos todos los ficheros necesarios.

F.3.3. Compilación y ejecución

Para compilar nuestro proyecto, si utlizamos GCC no será necesario enlazar a ninguna librería especial, puesto que `libintl` ya viene en la biblioteca estándar. Así pues, solo tendremos que compilar de la manera habitual.

```
g++ -o oflute main.cpp
```

Tras esto, podremos probar nuestro programa:

```
jose@jose-desktop:~$ ./oflute
Hola mundo
jose@jose-desktop:~$ LANG=en_UK.utf8 ./oflute
Hello world
```

F.3.4. Mantenimiento

Supongamos ahora que añadimos una línea más a nuestro código, en la que se utiliza una cadena nueva. Si seguimos el proceso anterior perderemos todas las traducciones que ya teníamos, ya que los ficheros se crearían de cero. Para evitar esto, GNU Gettext ofrece una utilidad llamada `msgmerge` que nos permitirá actualizar los ficheros `.po` con las nuevas cadenas manteniendo las traducciones ya realizadas.

Así pues, supongamos que añadimos esta línea al final fichero:

```
cout << _("Bye bye, dear user") << endl;
```

Generamos el fichero de plantilla `.pot` igual que lo hicimos antes, pero a la hora de generar los ficheros `.po` utilizaremos el nuevo comando.

```
xgettext --package-name oflute --package-version 0.1 \
--default-domain oflute --output po/oflute.pot --from-code=utf-8 \
--copyright-holder="Tu nombre" --msgid-bugs-address="tu@mail.com" \
-s -k_ -C main.cpp

msgmerge -s -U po/es.po po/oflute.pot
msgmerge -s -U po/en.po po/oflute.pot
```

La opción **-s** genera una salida ordenada, y **-U** indica que la operación es de actualización (*update*). Con esto, ya tendremos el fichero `.po` con las nuevas cadenas añadidas y las cadenas antiguas sin cambios, podremos proceder a añadir las traducciones y generar los ficheros `.mo` tal y como se ha explicado previamente.

```
msgfmt -c -v -o lang/es/LC_MESSAGES/oflute.mo po/es.po
msgfmt -c -v -o lang/en/LC_MESSAGES/oflute.mo po/en.po

./oflute
Hola mundo
Nos vemos, querido usuario

LANG=en_UK ./oflute
Hello world
Bye bye, dear user
```

F.4. Addendum

F.4.1. PO-mode en Emacs

Como se comentó al principio, existe un modo de Emacs [12] para la edición eficiente de archivos `.po`. Puede instalarse manualmente de la manera habitual o en sistemas basados en paquetería Debian [9] con el siguiente comando:

```
sudo apt-get install gettext-el
```

Una vez hecho esto, al abrir un fichero `.po` en Emacs se activará el *modo PO* (podemos forzarlo con `M-x po-mode`). Hay gran cantidad de comandos para editar los ficheros PO, pero los más útiles son los siguientes:

- Con **n** y **p** iremos al siguiente o anterior mensaje de traducción.
- Para saltar entre los mensajes traducidos usaremos **t** y **T**. Para los no traducidos, **u** y **U**.
- Para editar la traducción, pulsamos **Intro**, que abrirá un marco con el mensaje a editar. Tras modificarlo, podemos confirmar los cambios con **C-c C-c** o cancelarlos con **C-c C-k**
- Podemos acceder a la ayuda en cualquier momento pulsando **h**.

Una vez acostumbrados, la edición de estos ficheros se hará mucho más liviana y rápida. Existen, de cualquier modo, editores íntegramente dedicados a la edición de ficheros `.po`.

F.4.2. Referencia

Para ampliar conocimientos sobre GNU Gettext, lo mejor es dirigirse a la referencia oficial [31] que, aunque bastante extensa, resulta muy interesante y amena de leer, explicando toda clase de casos especiales de traducción, como aquellos en los que aparecen cadenas de formato relacionadas con sentencias al estilo de `printf` y otros casos particulares.

G. GNU Free Documentation License

Version 1.3, 3 November 2008

http://fsf.org/

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below,

refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF

produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "**publisher**" means any person or entity that distributes copies of the Document to the public.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on

the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you

follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of

such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with . . . Texts.” line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Bibliografía y referencias

[1] *232 MKSD Round*. http://www2.wind.ne.jp/maniackers/232mksd.html.

Tipografía utilizada en el logtipo de oFlute

[2] *Actionscript*. http://www.adobe.com/devnet/actionscript.html.

ActionScript es un lenguaje de programación, orientado a objetos, utilizado en la plataforma Adobe (*anteriormente Macromedia*) Flash. Basado en EcmaScript, inicialmente se diseñó para añadir algo de interactividad a las animaciones Flash, pero con el tiempo ha evolucionado hacia un lenguaje muy robusto y preparado para la creación de RIAs (*Rich Internet Apps*).

[3] *Adobe Photoshop*. http://www.adobe.com/es/products/photoshop.html.

Adobe Photoshop es una popular herramienta de edición de imágenes de mapas de bits.

[4] *Apache Subversion*. http://subversion.apache.org/.

Apache Subversion es un sistema de control de versiones muy popular basado en un repositorio central y en revisiones del conjunto completo de ficheros (a diferencia de, por ejemplo, CVS, donde los archivos tienen números de revisión independientes.

[5] *Asociación de Diseño de Videojuegos de la Universidad de Cádiz*. http://www.advuca.com.

La Asociación de Diseño de Videojuegos de la UCA promueve el uso y desarrollo de videojuegos dentro de la Universidad, organizando talleres y conferencias.

[6] *Boost C++ Libraries*. http://www.boost.org.

Boost es un conjunto de bibliotecas para C++ que ofrecen herramientas para una gran diversidad de situaciones. Entre sus desarrolladores se encuentran muchos de los mejores programadores de C++. Dada la calidad de Boost, una gran número de sus componentes parte del nuevo estándar C++0x.

[7] *BoUML*.

BoUML es un editor de diagramas UML de código abierto.

[8] *Cursos de Verano de la OSLUCA*. http://osl.uca.es/node/1132.

Del 28 de junio al 2 de julio de 2010 se celebraron, dentro del marco de la final del IV CUSL, unis Cursos de Verano organizados por la Oficina de Software Libre y Conocimiento Abierto de la UCA.

[9] *Debian GNU/Linux*. http://www.debian.org/index.es.html.

Debian GNU/Linux es un sistema operativo libre, creado por la comunidad Debian, con más de 18 años de edad, y que goza de una base de usuarios muy estable, además de servir como distribución de partida para muchas otras, como Ubuntu.

[10] *Dia*. http://live.gnome.org/Dia.

Dia es un editor de diagramas de código abierto, perteneciente a la familia GNOME

[11] *Doxygen*. http://www.stack.nl/~dimitri/doxygen/.

Doxygen es un generador automático de documentación para C++ y muchos otros lenguajes.

[12] *GNU Emacs*.

GNU Emacs es, según su propio manual, *un editor extensible, personalizable, auto-documentado y de tiempo real*. Inicialmente desarrollado por Richard Stallman, es uno de los editores más populares en los sistemas GNU/Linux junto a Vi.

[13] *GNU Gettext*. http://www.gnu.org/s/gettext/.

GNU Gettext es un conjunto de herramientas libres de internacionalización de proyectos.

[14] *Guadalinex*. http://www.guadalinex.org.

Guadalinex es una distribución Linux promovida por la Junta de Andalucía para fomentar el uso del software libre en su comunidad autónoma.

[15] *I Hackathon UCA de Software Libre*. http://wikis.uca.es/wikiosluca/doku.php?id=hackathon.

El I Hackathon UCA de Software Libre fue un encuentro de programación en el que se realizaron diversas ponencias y los asistentes tuvieron la oportunidad de trabajar en proyectos libres, ampliando funcionalidades, o generando proyectos nuevos.

[16] *ImageMagick*. http://www.imagemagick.org.

ImageMagick es una suite de herramientas de línea de comandos para la edición de imágenes.

[17] *Inkscape*. http://inkscape.org/?lang=es.

Inkscape es un editor de gráficos vectoriales de código abierto.

[18] *IV Concurso Universitario de Software Libre*. http://concursosoftwarelibre.org/0910/.

Cuarta edición del Concurso Universitario de Software Libre

[19] *JSON - Javascript Object Notation*. http://www.json.org.

JSON es un formato de intercambio de datos, basado en un subconjunto de la sintaxis de JavaScript.

[20] *LaTeX– A document preparation system*.

LaTeXes un sistema de preparación de documentos, especialmente orientado a textos científicos y técnicos.

[21] *Licencia Creative Commons By-Sa*. http://creativecommons.org/licenses/by-sa/3.0/es/.

Términos de la licencia Creative Commons Reconocimiento, Compartir Igual, versión 3.0 para España. Básicamente, la licencia permite la distribución del bien siempre y cuando se reconozca la autoría original y los derivados del documento se compartan con una licencia equivalente.

[22] *Linux Magazine, edición en español*. http://linux-magazine.es.

Edición en español de la popular revista Linux Magazine.

[23] *Make*. http://www.uca.es/softwarelibre/publicaciones/make.pdf.

Make es un programa para la gestión y el control de la recompilación de proyectos de cierta envergadura. Se basa en ficheros conocidos como *makefiles*, que guardan información sobre los *objetivos* a generar y los ficheros fuente a utilizar.

[24] *Miso*. http://www.fontsquirrel.com/fonts/Miso.

Tipografía utilizada en los textos de oFlute

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

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

[26] *Pango*. http://www.pango.org.

Pango es una biblioteca de código abierto para el diseño y dibujo de texto como parte del conjunto GTK+ y por lo tanto del entorno gráfico GNOME para sistemas operativos linux.

[27] *PNG – Portable Network Graphics*. http://www.iso.org/iso/catalogue_detail.htm?csnumber=29581.

PNG es un formato estándar de representación de mapas de bits. Se trata de un formato con compresión sin pérdida, con soporte de transparencia de 8 bits, que surgió como alternativa al formato GIF.

[28] *Premios de la fase local del IV CUSL*. http://softwarelibre.uca.es/node/1120.

Noticia de la fase local del IV CUSL en el que se detallan los premios otorgados, entre los que oFlute recibió un accésit al mejor proyecto de innovación.

[29] *PulseAudio*. 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.

[30] *PulseAudio Simple API Reference*. http://www.freedesktop.org/software/pulseaudio/doxygen/simple.html.

Referencia de la API simple de PulseAudio.

[31] *Referencia Gettext*. http://www.gnu.org/software/gettext/manual/gettext.html.

Manual oficial de GNU Gettext.

[32] *RGBA*. http://en.wikipedia.org/wiki/RGBA.

Modelo de color basado en el RGB que, además de almacenar la información de los canales rojo, verde y azul, también guarda datos sobre la opacidad (*canal alfa*).

[33] *SDL – Simple Directmedia Layer*. http://www.libsdl.org.

Conjunto de bibliotecas desarrolladas en el lenguaje de programación C que proporcionan funciones básicas para realizar operaciones de dibujo en dos dimensiones, gestión de efectos de sonido y música, y carga y gestión de imágenes.

[34] *SDL_ttf*. http://www.libsdl.org/projects/SDL_ttf/.

Biblioteca para SDL que provee soporte para la carga y pintado de fuentes TrueType.

[35] *TrueType*.

TrueType es un formato estándar para la representación de tipografías desarrollado por Apple. Es el formato más utilizado para representar fuentes, aunque carece de algunas opciones avanzadas, sí presentes en otros formatos como OpenType y Type1.

[36] *Ubuntu GNU/Linux*. http://www.ubuntu.org.

Ubuntu es una distribución GNU/Linux basada en Debian, orientada al usuario medio y con un fuerte enfoque en la facilidad de uso. Se estima que Ubuntu tiene más de 12 millones de usuarios.

[37] *Workaround to use Custom Fonts in LINUX*. http://www.libgosu.org/cgi-bin/mwf/topic_show.pl?tid=332.

Hilo del foro oficial de Gosu en el que se presenta el parche para el uso de fuentes TrueType en Gosu bajo sistemas GNU/Linux. Este parche fue integrado en la versión 0.7.20 de la biblioteca.

[38] *Premios del IV Concurso Universitario de Software Libre*. http://concursosoftwarelibre.org/0910/finalistas-iv-cusl, abril 2010.

Noticia del IV CUSL en que se da un listado de los premios finales del CUSL, en los que oFlute tiene el honor de aparecer como Mención Especial.

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

Kiss FFT es una biblioteca para realizar Transformadas Rápidas de Fourier (FFT (Fast Fourier Transform)).

[40] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, y François Yergeau. *Extensible Markup Language (XML) 1.0*. Informe técnico, World Wide Web Consortium, septiembre 2006. http://www.w3.org/TR/xml/.

Especificación oficial por el W3C del Extensible Markup Language.

[41] Manuel Palomo Duarte y José Tomás Tocino García. *Gosu I - Creando un videojuego en C++*. *Linux Magazine, edición en Español*, (66), Diciembre 2010.

[42] Manuel Palomo Duarte y José Tomás Tocino García. *Gosu II - Creando un videojuego en C++*. *Linux Magazine, edición en Español*, (67), Enero 2011.

[43] Manuel Palomo Duarte y José Tomás Tocino García. *Gosu III - Creando un videojuego en C++*. *Linux Magazine, edición en Español*, (68), Febrero 2011.

[44] José Tomás Tocino García. *Freegemas*. http://freegemas.googlecode.com.

Videojuego libre, un clon multiplataforma del clásico juego tipo puzzle *Bejeweled*. Está disponible para sistemas GNU/Linux y Windows, y también se encuentra disponible en los repositorios de Guadalinex.

[45] José Tomás Tocino García. *oFlute, blog de desarrollo oficial*. http://oflute.wordpress.com.

En este blog se reflejó el desarrollo del proyecto, incluyendo artículos sobre diferentes dificultades encontradas y cómo se fueron resolviendo.

[46] José Tomás Tocino García. *oFlute, repositorio oficial*. http://oflute.googlecode.com.

Forja oficial del proyecto oFlute, que contiene información sobre el proyecto, así como acceso libre al repositorio de código *Subversion*.

[47] E. C. M. A. International. *ECMA-262: ECMAScript Language Specification*. ECMA (European Association for Standardizing Information and Communication Systems), Geneva, Switzerland, third edición, diciembre 1999. URL http://www.ecma-international.org/publications/standards/Ecma-327.htm.

Especificación oficial de EcmaScript

[48] Arseny Kapoulkine. *PugiXML*. http://code.google.com/p/pugixml.

Biblioteca ligera para el procesamiento de archivos XML en C++.

[49] Björn Karlsson. *Beyond the C++ Standard Library*. Addison-Wesley Professional, 2005. ISBN 0321133544.

[50] Francisco Javier Santacruz López-Cepero, Daniel Salazar Recio, y José Tomás Tocino García. *Robinson 2.0*. http://robinson.forja.rediris.es/.

Robinson 2.0 es un videojuego colaborativo realizado durante el transcurso de la asignatura *Diseño de Videojuegos*, del curso 2009-10. El juego se ambienta en un futuro dominado por máquinas, en el que un pequeño robot debe enfrentarse a enemigos tecnológicos de todo tipo.

[51] Mike Melanson. *Welcome to the Jungle*. http://blogs.adobe.com/penguinswf/2007/05/welcome_to_the_jungle.html.

En este artículo, Mike Melanson hace un repaso subjetivo sobre lo complejo que resulta elegir entre la cantidad de sistemas y APIs de audio en GNU/Linux.

[52] Robert Penner. *Motion, Tweening and Easing*. En *Programming Macromedia Flash MX*. 2002. URL http://robertpenner.com/easing/.

[53] Robert Penner. *Robert Penner's Programming Macromedia Flash MX*. McGraw-Hill, Inc., New York, NY, USA, 1 edición, 2002. ISBN 0072223561.

[54] José Tomás Tocino García. *Materiales del curso de Boost*. http://josetomastocino.com/varios/taller_boost.tar.gz.

Materiales libres del curso sobre Boost que impartí durante los Cursos de Verano de la OSLUCA en junio de 2010.

[55] José Tomás Tocino García. *Materiales Taller Gosu, marzo 2011*. http://advuca.com/blog/actividades.

Materiales del taller sobre desarrollo de videojuegos con Gosu que impartí en marzo de 2011.

[56] José Tomás Tocino García. *Taller: aprende a programar videojuegos en C++ con Gosu*. http://advuca.com/blog/talleres/talleres-blender-y-gosu-en-marzo/.

Anuncio del taller sobre desarrollo de videojuegos con Gosu que impartí en marzo de 2011.

[57] José Tomás Tocino García. *Traducción de proyectos con GNU gettext en 15 minutos*.

[58] Julian Raschke y otros. *Gosu*. 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.

[59] F. Yergeau. *RFC 3629: UTF-8, a transformation format of ISO 10646*. RFC 3629 (Standard), noviembre 2003. URL http://www.ietf.org/rfc/rfc3629.txt.