Universidad de las Ciencias Informáticas
Facultad 5
Realidad Virtual

Herramienta de creación y edición de mapas de sonidos para aplicaciones de realidad virtual

Trabajo de Diploma para optar por el título de
Ingeniero en Ciencias Informáticas

Autor: Argelio de la Rosa Rodríguez

Tutor: Dagoberto Marrero López

junio de 2009

Datos de Contacto

Nombre y Apellidos: Dagoberto Marrero López.

Edad: 25 años.

Ciudadanía: cubano.

Categoría Instructor recién graduado. Graduado de Ingeniero en Ciencias Informáticas. Líder del proyecto DJKD.

E-mail: dmarrero@uci.cu

Dedicatoria

A la memoria de mi padre.

A mi madre.

Agradecimientos

Quisiera comenzar agradeciendo a todas aquellas personas que de una forma u otra han contribuido a mi formación, y a todas aquellas que me han brindado su ayuda cuando la he necesitado.

Agradecer a todos mis profesores a lo largo de todos estos años de estudios. En especial quisiera agradecer a María, mi profesora de pre-escolar; a la profe Mirian, por enseñarme a leer y escribir; al profe Lalín, por hacer de las clases, clases para toda la vida; al profe Gonzalo por su responsabilidad y entrega; al profesor Arquímedes, por enseñarme que el respeto no se impone, se gana; a la profe Greta, por iniciarme en los estudios de la historia; a las profes de Español de la secundaria, por la cultura aportada; a los profes Toni y Javier, por sus clases de matemáticas; a la profe de Historia de 11no y 12mo, por ayudarme a comprender la importancia de conocer bien la historia de nuestro país y del mundo en general; al profe Jorge, por toda la ayuda brindada durante el "pre"; a la profesora Zaida, por su apoyo en los momentos difíciles.

Agradecer igualmente a todos mis compañeros(as) y amigos(as) de la primaria, secundaria y pre, con los cuales tuve la oportunidad de compartir, crecer y descubrir la vida. En especial quisiera agradecer a Maidelín, por demostrarme que la verdadera amistad es capaz de imponerse a la distancia y la separación física, por ser mi confidente y bastón de apoyo en los días más difíciles, por todas las horas de conversación. Agradecer a mis compañeros(as) durante estos años de universidad, por todos los momentos que hemos vivido juntos, en particular agradecer a Raúl, Israel, Leandro, Prado, Idelvis, Yuliet e Irais, y muy especialmente a Yusnier y Gustavo, por ser más que amigos, hermanos.

Asimismo agradecer a todas las personas que han contribuido al desarrollo de este trabajo. A Yenifer y Alexei, por su constante apoyo, y valiosa ayuda. A Tamara por sus correcciones y consejos. A Yeisnier por las ideas.

Agradecer igualmente a toda mi familia. A mis primos, muy especialmente a Yoel – mi hermano –, Yoenia y Randy. A mis abuelos Pipío – quién lamentablemente no pudo llegar a vivir este momento – y Amea. A tía Maité, por ser mi segunda madre. A tío Fulge, por ser ejemplo y guía. A mis hermanos, en especial a Tata, por toda su ayuda y apoyo; a Pipo, por ser más que un hermano, un padre; a Margara, por estar siempre ahí, por ser mi hermana en todo el sentido de la palabra.

Por último agradecer a mis padres, a los cuales les debo todo lo que soy, y de los cuales siempre será poco lo que pueda decir. A mi padre, por todo lo que hizo por mí, por enseñarme a ser por encima de todo honrado, por enseñarme a mantener y defender mis principios e ideales “contra viento y marea”, por inculcarme su amor por el deporte y su deseo perenne de aprender e informarse, por demostrarme que no importa lo que hayamos hecho siempre se podrá hacer más, por hacerme ver la importancia del trabajo, por tantas, tantas cosas que no soy capaz de describir. Papi, no importa lo que digan, tú sigues aquí, conmigo. A mi madre, por todo su amor y cariño, por su constante preocupación, por inculcarme la importancia del estudio, por su empeño de hacer de mí un profesional y hombre de bien, por demostrarme que es capaz de hacer lo posible y lo prácticamente imposible para lograr la felicidad de todos sus seres queridos, por acompañarme siempre en cada instante de mi vida, por ser la estrella alrededor de la cual gira mi vida. Mami, si pudiese pedir un deseo, desearía que fueras eterna, porque no puedo siquiera imaginar cómo sería la vida sin ti.

Resumen

A medida que la informática ha ido desarrollándose se ha hecho posible el desarrollo de los Sistemas de Realidad Virtual (SRV). En la actualidad estos sistemas son utilizados en disímiles áreas que van desde la medicina, hasta la industria del entretenimiento.

Dentro de los SRV los sonidos son un elemento de gran importancia, ya que sirven para darle realismo a una simulación pues estos están presentes en todos y cada uno de los sucesos que ocurren a diario a nuestro alrededor.

La configuración de los sonidos en estos sistemas resulta generalmente un proceso engorroso, tedioso y conlleva a una considerable pérdida de tiempo para el equipo de desarrollo por lo que es necesaria la creación de una herramienta que mediante una interfaz visual permita la inclusión de sonidos a una escena y la configuración de estos.

Este trabajo propone una herramienta que permite la creación y edición de mapas de sonidos para aplicaciones de realidad virtual.

Palabras clave

Interfaz visual, sonido, mapas de sonido, realidad virtual.

ÍNDICE

Introducción

Desde sus inicios la Realidad Virtual (RV) intentó simular procesos y entornos que se acercaran con mayor exactitud a la realidad, pero no importa cuán exacto sea un entorno o cuán realista sean los efectos visuales de este, que si carece de sonido sin duda no tendrá el realismo esperado por el usuario.

Con el surgimiento y desarrollo de las tarjetas de sonidos se ha dado la posibilidad de lograr en los sistemas de realidad virtual (SRV) un ambiente acústico capaz de abstraer al usuario del mundo que lo rodea, dando lugar al surgimiento de aplicaciones que logran una mejor simulación de la realidad.

A través de los sonidos las personas reciben constantemente información del medio que está a su alrededor, y esta información se hace más relevante cuando existe una pobre visualización del entorno. Los efectos de sonido con frecuencia reavivan a los efectos visuales. En las aplicaciones de realidad virtual generalmente se hace un uso intensivo de los sonidos ya que a través de ellos se logra enriquecer la información que el usuario está recibiendo por la pantalla.

La facultad 5 de la Universidad de las Ciencias Informáticas (UCI) se dedica al desarrollo de aplicaciones de realidad virtual, entre los que se destacan los juegos con fines educativos, médicos y de entretenimiento. Para el trabajo con los sonidos en estas aplicaciones se cuenta con la biblioteca SoundToolkit.

En la actualidad la biblioteca SoundToolkit logra abstraer al programador del trabajo a bajo nivel con los sonidos. Sin embargo, el proceso de añadir sonidos a los objetos estáticos de una escena, agregar los sonidos ambientales que tendrá esta, así como configurar dichos sonidos, se convierte mayoritariamente en un proceso tedioso y que requiere un considerable gasto de tiempo. Esto es provocado debido a que el proceso de añadir estos sonidos a la escena es realizado por los programadores, los cuales mediante instrucciones de código se encargan de cargar el sonido, posicionarlo en el lugar deseado – en el caso de los sonidos aplicados a los objetos estáticos de la escena – y configurar las propiedades de este; y posteriormente compilar y ejecutar la aplicación para poder escuchar el efecto de los cambios realizados. En un gran número de ocasiones los resultados obtenidos no son los deseados por lo que se hace necesario repetir el proceso nuevamente, y así para cada uno de los sonidos que se le desee aplicar a los

objetos estáticos de la escena o para cada sonido ambiental que se vaya a tener en la aplicación. Por lo que surge el **problema científico** de ¿Cómo facilitar la creación y edición de mapas de sonidos para aplicaciones de realidad virtual?

El **objeto de estudio** de este trabajo es el proceso de configuración de sonidos en sistemas de realidad virtual, de ahí se deriva como **campo de acción** las herramientas para la configuración de sonidos en sistemas de realidad virtual. Planteando como **objetivo general** diseñar e implementar una herramienta que permita la creación y edición de mapas de sonidos estáticos para aplicaciones de realidad virtual.

Para dar cumplimiento al objetivo propuesto se trazan las siguientes **tareas investigativas**:

- Análisis de las técnicas para el desarrollo de las interfaces gráficas de usuario con el objetivo de determinar los requisitos y principios a tener en cuenta a la hora de diseñar y construir una interfaz gráfica de usuario.
- Estudio de las bibliotecas de desarrollo de interfaces gráficas de usuario con el propósito de determinar la más adecuada para la construcción de interfaces gráficas de usuario de editores.
- Estudio de los motores gráficos libres de mejores prestaciones existentes en la actualidad para elegir el más apropiado para visualizar escenas que puedan contener una gran cantidad de polígonos.
- Análisis de las principales tendencias en cuanto al uso de aplicaciones para la configuración de sonidos en SRV.
- Estudio de los diferentes formatos de fichero con el objetivo de seleccionar el más adecuado para almacenar la información persistente.
- Diseño e implementación de una herramienta que permita la creación y edición de mapas de sonidos estáticos.

Con este sistema se pretende crear una aplicación que logre el incremento de la productividad de los desarrolladores, disminuyendo drásticamente el tiempo necesario para configurar los sonidos ambientales y sonidos aplicados a los objetos estáticos de la aplicación de realidad virtual que se esté desarrollando.

Además de permitir delegar esta responsabilidad en personas que no tengan conocimientos de programación, permitiendo al equipo de desarrollo concentrarse en otras tareas.

El presente trabajo de diploma está compuesto por 5 capítulos organizados de la siguiente manera:

- Capítulo 1: Fundamentación teórica.
- Capítulo 2: Soluciones técnicas.
- Capítulo 3: Descripción de la solución propuesta.
- Capítulo 4: Diseño del sistema.
- Capítulo 5: Implementación.

Capítulo 1: Fundamentación teórica

Introducción

En este capítulo se analizan en detalle las interfaces gráficas de usuario. Para ello se determinan las características humanas y los principios a tener en cuenta para desarrollar una interfaz gráfica de usuario; exponiéndose finalmente los pasos necesarios para el desarrollo de una interfaz gráfica de usuario que cumpla con las expectativas de los usuarios que harán uso de ella.

A continuación se hace un estudio de las principales bibliotecas de desarrollo de interfaces gráficas de usuario, con el objetivo de determinar la que más se adecue a las necesidades de desarrollo.

Además se realiza un estudio de los principales motores gráficos libres existentes en la actualidad, resaltándose las potencialidades y características de cada uno, para así poder determinar el que mejor cumpla con los requisitos necesarios para el desarrollo del sistema.

Por último, se presentan las principales características de los sonidos digitales y su uso en las aplicaciones de realidad virtual; y se hace una presentación del formato XML, mostrándose las principales ventajas que trae el empleo de este como formato para almacenar información.

1.1. Interfaces gráficas de usuario

1.1.1. Características humanas a tener en cuenta para desarrollar una interfaz gráfica de usuario

Cuando se vaya a diseñar interfaces de usuario se debe tomar en cuenta las habilidades cognitivas y de percepción de las personas que la vayan a usar, y adaptar el programa a estas personas.

Un factor fundamental en el que una interfaz puede influir es reducir la dependencia de las personas de su propia memoria, que no tenga que memorizar cosas innecesariamente o repetir operaciones ya realizadas.

Algunas de los aspectos generales que se deben tomar en consideración para realizar una interfaz gráfica son (1):

- **Velocidad de Aprendizaje**: Está referido al tiempo que demora una persona en aprender a usar el sistema. Aquí es importante que el usuario se familiarice con el sistema lo más rápido posible.
- **Velocidad de Respuesta**: Es el tiempo que se necesita para realizar una operación en el sistema.
- **Tasa de errores**: No es más que el porcentaje de errores que comete el usuario.
- **Retención**: Se refiere a qué cantidad de información es capaz de recordar un usuario transcurrido un tiempo determinado.
- **Satisfacción**: En qué medida está el usuario satisfecho con el sistema.

1.1.2. Principios básicos para el diseño de una interfaz gráfica de usuario.

- **Anticipación**: Las aplicaciones deberían intentar anticiparse a las necesidades del usuario y no esperar a que el usuario tenga que buscar la información, recopilarla o invocar las herramientas que va a utilizar (2).

- **Autonomía**: La computadora, la interfaz gráfica de usuario y el entorno de trabajo deben estar a disposición del usuario. Se debe dar al usuario el ambiente flexible para que pueda aprender rápidamente a usar la aplicación, aunque el entorno de trabajo debe de ser explorable pero no azaroso. No puede existir autonomía en ausencia de control, y el control no puede ser ejercido sin información suficiente. Se debe mantener la información del estado del sistema en ubicaciones fáciles de visualizar (2).

- **Percepción del color**: Aunque se utilicen convenciones de color en la interfaz gráfica de usuario, se deberían usar otros mecanismos secundarios para proveer la información a aquellos usuarios con problemas en la visualización de colores (2).

- **Valores por defecto**: En una interfaz gráfica de usuario no se debe utilizar la palabra "Defecto", esta puede ser reemplazada por "Estándar" o "Definida por el Usuario", "Restaurar Valores Iníciales" o algún otro término especifico que describa lo que está sucediendo. Los valores por defecto deberían ser opciones inteligentes, sensatas y fáciles de modificar (2).

- **Consistencia**: Para lograr este principio se requiere profundizar en diferentes aspectos, los cuales se mencionan a continuación (2):
    - **Interpretación del comportamiento del usuario**: La interfaz gráfica de usuario debe comprender el significado que le atribuye un usuario a cada requerimiento.
    - **Estructuras invisibles**: Se requiere una definición clara de las mismas, pues de no ser así el usuario nunca podría llegar a descubrir su uso.
    - **Pequeñas estructuras visibles**: Conjunto de objetos visibles capaces de ser controlados por el usuario, que permitan ahorrar tiempo en la ejecución de tareas específicas.
    - **Una sola aplicación o servicio**: La interfaz gráfica de usuario permite visualizar a la aplicación utilizada como un componente único.
    - **Un conjunto de aplicaciones o servicios**: La interfaz gráfica de usuario visualiza a la aplicación o servicio utilizado como un conjunto de componentes.
    - **Consistencia del ambiente**: La interfaz gráfica de usuario se mantiene en concordancia con el ambiente de trabajo.

  - **Consistencia de la plataforma**: La interfaz gráfica de usuario es concordante con la plataforma.

- **Eficiencia del usuario**: Es importante considerar la productividad del usuario antes que la productividad de la máquina. Si el usuario debe esperar la respuesta del sistema por un período prolongado, estas pérdidas de tiempo se pueden convertir en pérdidas económicas para la organización. Los mensajes de ayuda deben ser sencillos y proveer respuestas a los problemas. Los menús y etiquetas de botones deberían tener las palabras claves del proceso (2).

- **Ley de Fitt**: El tiempo para alcanzar un objetivo es una función de la distancia y tamaño del objetivo, por lo que es conveniente usar objetos grandes para las funciones importantes (2).

- **Interfaces explorables**: Se debe permitir que el usuario pueda salir ágilmente de la interfaz gráfica de usuario, dejando una marca del estado de avance de su trabajo, para que pueda continuarlo en otra oportunidad. La interfaz gráfica de usuario debe poder realizar la inversa de cualquier acción que pueda llegar a ser de riesgo, de esta forma se apoya al usuario a explorar el sistema sin temores (2).

- **Objetos de interfaz humana**: Los objetos de interfaz humana no son necesariamente los objetos que se encuentran en los sistemas orientados a objetos, estos pueden ser vistos, escuchados, tocados o percibidos de alguna forma. Estos objetos deberían ser entendibles, consistentes y estables (2).

- **Uso de metáfora**: Las buenas metáforas crean figuras mentales fáciles de recordar. La interfaz gráfica de usuario puede contener objetos asociados al modelo conceptual en forma visual, con sonido u otra característica perceptible por el usuario que ayude a simplificar el uso del sistema (3).

- **Curva de aprendizaje**: El aprendizaje de un producto y su usabilidad no son mutuamente excluyentes. Lo ideal es que la curva de aprendizaje sea nula, y que el usuario principiante pueda alcanzar el dominio total de la aplicación sin esfuerzo (2).

- **Reducción de latencia**: el uso de tramas permite colocar la latencia en segundo plano. Las técnicas de trabajo multitarea posibilitan el trabajo ininterrumpido del usuario, realizando las tareas de transmisión y computación de datos en segundo plano (2).

- **Protección del trabajo**: Se debe asegurar que el usuario nunca pierda su trabajo, ya sea por error de su parte, problemas de transmisión de datos, de energía, o alguna otra razón inevitable (2).

- **Auditoría del sistema**: Para cualquier aplicación es conveniente conocer un conjunto de características tales como: hora de acceso al sistema, ubicación del usuario en el sistema y lugares a los que ha accedido, entre otros. Además, el usuario debería poder salir del sistema y al volver a ingresar continuar trabajando en el lugar dónde se había quedado (2).

- **Legibilidad**: Para favorecer la usabilidad del sistema, la información que se exhiba en ella debe ser fácil de ubicar y leer. Para lograr obtener este resultado se deben tener en cuenta, los siguientes aspectos (2):
    - El texto que aparezca en la interfaz debe tener un alto contraste.
    - El tamaño de las fuentes tiene que ser lo suficientemente grande como para poder ser leído en monitores estándar.
    - Es importante hacer clara la presentación visual.

1.1.3. Pasos para el diseño y construcción de una interfaz gráfica de usuario

1. **Reunir y analizar la información del usuario**: Se debe determinar qué tipo de usuarios van a utilizar el programa, qué tareas van a realizar los usuarios y cómo las van a realizar, qué exigen los usuarios del programa, en qué entorno se desenvuelven los usuarios (físico, social, cultural) (3).

2. **Diseñar la interfaz de usuario**: Es importante dedicar tiempo y recursos a esta fase, antes de entrar en la codificación. En este paso se definen los objetivos de usabilidad del programa, las tareas del usuario, los objetos y acciones de la interfaz, los iconos, vistas y representaciones visuales de los objetos, los menús de los objetos y ventanas. Los elementos visuales se pueden diseñar a mano y luego refinar con las herramientas adecuadas (3).

3. **Construir la interfaz de usuario**: Se debe realizar un prototipo previo, una primera versión del programa que se realice rápidamente y permita visualizar el producto para poderlo probar antes de codificarlo definitivamente (3).

4. **Validar la interfaz de usuario**: Se deben realizar pruebas de usabilidad del producto, es recomendable que estas pruebas se lleven a cabo con los propios usuarios finales. Es importante realizar un diseño que parta del usuario, y no del sistema (3).

1.2. Bibliotecas de desarrollo de interfaces gráficas de usuarios

1.2.1. GTK+

GTK+ fue desarrolla inicialmente como un conjunto de herramientas para el programa de manejo de imágenes GIMP. GTK+ es una biblioteca construida sobre GDK (el conjunto de herramientas de dibujo GIMP), que a su vez utiliza las funciones de Xlib. Actualmente su uso se ha expandido, es muy usado en el desarrollo de muchos programas, principalmente en los programas para sistemas GNU/Linux.

GTK+ es un API orientado a objetos. Aunque está escrita completamente en C, soporta la idea de clases y funciones de respuesta (es decir punteros a funciones).

GTK+ se basa en varias bibliotecas del equipo de GTK+ y de GNOME:

- **GLib**: Biblioteca de bajo nivel, es la estructura básica de GTK+ y GNOME.

- **GTK**: Biblioteca que contiene los objetos y funciones para crear la interfaz.

- **GDK**: Biblioteca que actúa como intermediario entre gráficos de bajo nivel y gráficos de alto nivel.

- **ATK**: Biblioteca para crear interfaces con características de una gran accesibilidad muy importante para personas discapacitadas o minusválidas. Pueden usarse utilerías como lupas de aumento, lectores de pantalla, o entradas de datos alternativas al clásico teclado o mouse.

- **Pango**: Biblioteca para el diseño y visualización de texto, hace hincapié especialmente en la internacionalización. Es el núcleo para manejar las fuentes y el texto de GTK+2.

- **Cairo**: Biblioteca de visualización avanzada de controles de aplicación.

1.2.1.1. Características

- **Multiplataforma**: Originalmente GTK+ fue desarrollado para X Windows, pero con el transcurrir de los años ha incluido soporte para otros bien conocidos sistemas de ventanas. En la actualidad GTK+ se puede usar en GNU / Linux, Windows y Mac OSX (4).

- **Multi-lenguaje**: GTK+ está disponible para muchos otros lenguajes además de C, esto es gracias a los *bindings* que se han creados con este propósito, la última versión hasta el momento de GTK+ – GTK+ 2.12 – soporta los siguientes lenguajes: C++, C#, Java, Python, Perl, R, Lua y D; y es parcialmente soportado por Guile, Ruby y PHP (4).

- **Otras** (4)
  - Aspecto nativo.
  - Enfoque orientado a objetos.
  - Internacionalización.
  - Localización.
  - Accesibilidad.
  - Soporte de texto bidireccional (LTR / RTL, de izquierda a derecha / de derecha a izquierda).
  - Soporte UTF8.
  - *Parser* de XML.

1.2.2. Qt

Es una biblioteca multiplataforma para desarrollar interfaces gráficas de usuario. Utiliza C++ de forma nativa, aunque además permite programar en otros lenguajes. Corre sobre las plataformas más usadas mundialmente. Además de posibilitar el desarrollo de interfaces gráficas de usuario, incluye otras series de características como son el acceso a base de datos SQL, *parser* de XML, y una API unificada multiplataforma para el manejo de ficheros.

A partir de la versión 4.5.0 la licencia de Qt es LGPL, lo que ha aumentado aún más el uso de esta biblioteca, ya que anterior a esta versión la licencia para los sistemas operativos Windows era comercial. La última versión hasta la fecha de Qt puesta a disposición por QtSoftware – la compañía que desarrolla Qt – es la 4.5.1.

1.2.2.1. Características

- **Características generales**: El diseño modular de la biblioteca Qt proporciona un rico conjunto de aplicaciones que ofrecen todas las funcionalidades necesarias para la construcción de potentes interfaces gráficas de usuario. Qt es una biblioteca fácil de aprender y usar. Permite crear código altamente legible y fácil de mantener (5).

- **Entorno de desarrollo**: Qt cuenta con entorno desarrollo integrado (IDE, por sus siglas en inglés) el cual posee un editor avanzado de código C++ y permite el diseño de las ventanas a través de una GUI, entre otras funcionalidades (5).

- **Multi-plataforma**: Qt está disponible para las plataformas Mac OSX, Windows, Linux/X11, Windows CE y S60 (5).

- **Lenguajes de programación**: Qt proporciona una intuitiva interfaz de clases para el desarrollo de aplicaciones en C++. Incluso va un poco más allá del lenguaje C++ en lo referido a la comunicación entre objetos y la flexibilidad para el desarrollo de interfaces gráficas de usuario agregando un poderoso mecanismo de comunicación entre objetos (*signals/slots*). Además

existen *bindings* para C#, Python, Ada, Pascal, Perl, PHP, Ruby y Java, aunque el desarrollo del *binding* (Qt Jambi) para este último ha sido interrumpido y se le seguirá dando mantenimiento hasta mayo del 2010 (5).

1.3. Motores gráficos

1.3.1. *Object Oriented Graphics Rendering Engine*

***O**bject Oriented **G**raphics **R**endering **E**ngine* (OGRE) es un motor gráfico escrito en C++, el cual está diseñado para hacer más fácil e intuitivo el trabajo de los desarrolladores de aplicaciones gráficas aceleradas por hardware. OGRE abstrae todos los detalles referidos a la utilización de las bibliotecas gráficas OpenGL y Direct3D, proporcionando una interfaz basada en objetos globales y otras clases intuitivas (6).

OGRE no es un motor de juego propiamente dicho – aunque ha sido usado para el desarrollo de juegos – sino que está diseñado con propósitos más generales, por lo que no cuenta con módulos de red, inteligencia artificial, física, etc. La razón de que esto sea así es que no todo el que utilice un motor gráfico lo hará con el objetivo de desarrollar un juego y además los requisitos necesarios para desarrollar un juego u otro, pueden variar considerablemente por lo que incluir todas estas características al unísono conllevaría a que se tuviesen incluidas muchas funcionalidades que no se fueran a usar, siendo esto un mal diseño. De esta forma posibilita que sea usado para el desarrollo de disímiles tipo de aplicaciones 3D (juegos, simuladores, aplicaciones de negocio, etc.), dándole al usuario la posibilidad de agregar el resto de las bibliotecas que necesite para el desarrollo de su aplicación.

1.3.1.1. Características

- **Características generales** (6)
  - Interfaz orienta a objeto fácil de usar, diseñada para minimizar el esfuerzo requerido para visualizar escenas 3D. y para ser independiente de la implementación 3D (Direct3D o OpenGL).
  - Arquitectura de *plugins* que permite extender las funcionalidades del motor gráfico.

- Buena y precisa documentación referente a todas las clases que lo componen, contando además con una activa comunidad en Internet.
- Soporta zip/pk3 para compactar.
- Disponible para Linux, Mac OSX y todas las versiones de Windows.

- **Gestión de Escena** (7)
  - BSP, Octrees, *Occlusion Culling*, nivel de detalle (LOD).
  - Altamente personalizable.
  - Grafo de escena jerárquico.

- **Renderizado** (7)
  - Soporta multi-textura y *multipass blending*.
  - Los objetos transparentes son gestionados automáticamente.
  - Soporte para múltiple técnicas de materiales.
  - Sistema de fuentes con fuentes *TrueType* y texturas pre-creadas.
  - Material LOD.

- **Iluminación** (7)
  - *Per-vertex*, *Per-pixel*, *Lightmapping*.

- **Sombras** (7)
  - *Shadow mapping*, *shadow volume*.
  - Soporte de técnicas: *modulative stencil*, *additive stencil*, *modulative projective*.
  - Texturas-sombras que se desvanecen a larga distancia.

- **Mallas** (7)
  - Cargado de malla, *Skinning*, *Progressive*.
  - Aceleración de *skinning* por hardware.
  - Exportadores para muchas herramientas de modelado incluidas Milkshape3D, 3D Studio Max, Maya, Blender y Wings3D.

- **Texturizado** (7)
  - Básico, *Multi-texturing*, *Bumpmapping*, *Mipmapping*, *Volumetric*, *Projected*.
  - Soporta PNG, JPEG, TGA, BMP y DDS.

- **Shaders** (7)
  - *Vertex*, *Pixel*, *High Level*.
  - Soporta *vertex shaders* y *pixel shaders* de bajo nivel escritos en ensamblador, y programas de alto nivel escritos en Cg, DirectX9 HLSL, y GLSL.

- ***Scripting*** (7)
  - El lenguaje de script permite mantener los materiales fuera del código.
  - Scripts de renderizado multi-paso.

- **Curvas y superficies** (7)
  - *Splines*.
  - Caminos Bezier bicuadrados para superficies curvas.

- **Efectos especiales** (7)
  - Medio ambiente, *billboarding*, sistema de partículas, cielo, agua, niebla.
  - Soporte para *skyboxes*, *skyplanes* y *skydomes*.

- **Animación** (7)
  - Cinemática inversa, animación esquelética, mezcla de animación.
  - Animación del esqueleto, incluyendo la mezcla de múltiples animaciones y de peso variable de *skinning* de huesos.

- **Física** (6)
  - Básica, detección de colisiones, cuerpos rígidos.
  - Incluye *bindings* para sistema físicos de terceras partes (ODE, Novodex y Tokamak).

1.3.2. *Open Scene Graph*

OpenSceneGraph (OSG) es un motor gráfico de código abierto, multiplataforma, utilizado para el desarrollo de aplicaciones gráficas de alto rendimiento tales como, simuladores de vuelo, juegos, aplicaciones de realidad virtual y visualización científica. Esta herramienta está basada en el concepto de grafos de escena, provee una plataforma orientada a objetos que utiliza OpenGL como API gráfica. OSG está disponible tanto para uso comercial como no comercial. Está escrito completamente en Standard C++ y OpenGL, hace un uso extensivo de la STL y de los patrones de diseño.

1.3.2.1. Características

- **Rendimiento**: Soporta *view-frustum culling*, *occlusion culling*, *small feature culling*, LOD, *vertex arrays*, *vertex buffer objects*, *OpenGL Shader Language* y *display lists*. Todo esto hace de OSG una de los motores gráficos de más alto rendimiento disponible (8).

- **Productividad**: El núcleo de OSG encapsula la mayoría de las funcionalidades de OpenGL incluyendo las últimas extensiones de este, provee optimización de la visualización, y un conjunto de bibliotecas adicionales las cuales hacen posible desarrollar aplicaciones gráficas de alto rendimiento muy rápidamente. Combinando las experiencias de otros motores gráficos con modernos métodos de ingeniería de software como patrones de diseño, se ha logrado el diseño de una biblioteca robusta y extensible (8).

- **Extensiones Soportadas**: Para la lectura y escritura de archivos la biblioteca (osgDB) proporciona una amplia variedad de formatos por medio de un mecanismo extensible de *plugins* dinámicos. Actualmente incluye 55 *plugins* para cargar formatos 3D y formatos de imágenes (8).

  Algunos de los formatos de modelos 3D soportados son COLLADA, LightWave (.lwo), Alias Wavefront (.obj), OpenFlight (.flt), Carbon Graphics GEO (.geo), 3D Studio MAX (.3ds), Peformer (.pfb), Quake Character Models (.md2), Direct X (.x), y Inventor Ascii 2.0 (.iv), VRML 1.0 (.wrl), Designer Workshop (.dw), AC3D (.ac) y el formato nativo .osg. Los formatos de imágenes soportados son rgb, .gif, .jpg, .png, .tiff, .pic, .bmp, .dds, .tga y .quicktime. Además brinda soporte para texto por medio de los *plugins* .freetype y .txf (8).

- **Espacios de Nombres (*namespace*)**: OSG está formado por los siguientes espacios de nombres (9):
  - **osg:** Es el núcleo de la biblioteca OSG, y proporciona las clases básicas del grafo de escena tales como *Nodes*, *Status*, y *Drawables*, así como clases matemáticas y otras.
  - **osgDB:** Proporciona soporte para leer y escribir grafos de escena, proporcionando un *framework* para *plugins* y clases para manejo de ficheros.
  - **osgFX:** Es una extensión del núcleo del grafo de escena para proporcionar un *framework* de efectos especiales.
  - **osgGA:** Proporciona herramientas para ayudar a los desarrolladores para trabajar con distintos sistemas de ventanas.
  - **osgIntrospection:** Proporciona un entorno de programación que permite la consulta en tiempo de ejecución de las propiedades y los métodos relacionados con las bibliotecas de OSG.
  - **osgParticle:** Amplía el núcleo del grafo de escena para soportar efectos de partículas.
  - **osgSim:** Extiende el núcleo del grafo de escena para soportar *Nodes* y *Drawables* que especifiquen la simulación visual.
  - **osgTerrain:** Biblioteca de utilidades que proporciona soporte para la generación de bases de datos de terreno.
  - **osgText:** Extiende el núcleo del grafo de escena para dar soporte a texto de alta calidad.
  - **osgUtil:** Proporciona clases de utilidad de propósito general, tales como recorridos de *update*, *cull*, y/o *Draw*, operadores de grafo de escena como son *optimization*, *tri stripping*, y *tessellation*.
  - **osgUtx:** Entorno de programación para la evaluación de aplicaciones.

- **Portabilidad**: El núcleo de OSG ha sido diseño para tener una dependencia mínima de cualquier plataforma, requiriendo poco más que Standard C++ y OpenGL. Esto ha permitido que OSG fuese rápidamente portado a un amplio rango de plataformas. Originalmente fue desarrollado en IRIX, luego portado a Linux, Windows, FreeBSD, Mac OSX, Solaris, HP-UX, AIX y hasta PlayStation2 (9).

  El núcleo de OSG es independiente del sistema de ventana, lo cual permite al usuario de una

forma fácil adicionar su propia biblioteca de interfaz gráfica de usuario. La biblioteca osgViewer provee el soporte para el sistema de ventana nativo de Windows (Win32), Unix (X11) y OSX (Carbon). La biblioteca osgViewer puede ser fácilmente integrada con otras bibliotecas de desarrollo de interfaces gráficas de usuario tales como Qt, GLUT, FLTK, SDL, WxWidget, Cocoa y MFC (9).

- **Soporte multi-lenguaje**: OSG está disponible además en los lenguajes Java, Lua y Pyhton (9).

1.3.3. Irrlicht

Irrlicht es un motor gráfico libre, escrito en lenguaje C++, multiplataforma, oficialmente está disponible para Windows, Linux y Mac OSX, aunque también ha sido adaptado a otras plataformas como Xbox, PlayStation Portable, etc. Este motor es conocido por su pequeño tamaño y alta compatibilidad con hardware. Es una potente API de alto nivel que posibilita la creación de aplicaciones 2D Y 3D como juegos o visualización científica.

Irrlicht posee una buena documentación y cuenta con todas las características de los motores gráficos modernos como sombras dinámicas, sistema de partículas, animación de personajes, detección de colisiones, etc. Todo esto es accesible a través de una interfaz bien diseñada, lo que hace que sea fácil de usar (10).

1.3.3.1. Características

Características generales (11)

- Independiente de la plataforma. Se ejecuta en Windows 95, 98, NT, 2000, XP, Linux, Mac OSX, Solaris, entre otros.
- Trae integrado un *parser* de XML.
- Soporte para archivos ZIP.
- Soporte para 64-bit.
- Potente interfaz de usuario 2D con botones, listas, cajas de edición, etc.

Efectos Especiales (11)

- Superficies de agua de gran realismo.
- Luces dinámicas.
- Sombras dinámicas usando *stencil buffer*.
- *Bumpmapping*.
- Objetos transparentes.
- Mapas de luces.
- Sistema de partículas para nieve, humo, fuego, entre otros.
- Animación de texturas.
- *Skyboxes* y *Skydomes*.
- Niebla

Shaders (11)

- *Vertex, Pixel, High Level.*
- Soporta la programación de *vertex shaders* y *pixel shaders* a bajo nivel.
- Soporte para GLSL: ARB Vertex Programs, ARB Pixel Programs, HLSL, GLSL 100 & 110, VS1.1 - 3.0, PS1.1 - PS3.0.

Gestión de Escena (11)

- General, BSP, *Octrees*.
- Grafo de escena fácil de extender.
- Soporta *picking*.

Formatos de texturas soportados (11)

- JPEG File Interchange Format (.jpg, r/w)
- Portable Network Graphics (.png, r/w)
- Truevision Targa (.tga, r/w)
- Windows Bitmap (.bmp, r/w)
- Adobe Photoshop (.psd, r)
- Zsoft Paintbrush (.pcx, r/w)

- Portable Pixmaps (.ppm, r/w)
- Quake 2 textures (.wal, r)

Formatos de mallas soportados (11)

- Irrlicht scenes (.irr, r/w)
- Irrlicht static meshes (.irrmesh, r/w)
- 3D Studio meshes (.3ds, r)
- B3D files (.b3d, r)
- Alias Wavefront Maya (.obj, r/w)
- Lightwave Objects (.lwo, r)
- COLLADA 1.4 (.xml, .dae, r/w)
- Microsoft DirectX (.x, r) (binary & text)
- Milkshape (.ms3d, r)
- OGRE meshes (.mesh, r)
- My3DTools 3 (.my3D, r)
- Pulsar LMTools (.lmts, r)
- Quake 3 levels (.bsp, r)
- Quake 2 models (.md2, r)
- Quake 3 models (.md3, r)
- DeleD (.dmf, r)
- FSRad oct (.oct, r)
- Cartography shop 4 (.csm, r)
- STL 3D files (.stl, r/w)

1.4. Sonido digital y su uso en aplicaciones de realidad virtual

1.4.1. Características fundamentales del sonido digital

Frecuencia: Es el número de vibraciones por segundo. La frecuencia de sonido se mide en Hercios (Hz). Un sonido que vibra una vez por segundo tiene una frecuencia de 1 Hz. Las frecuencias se escriben normalmente en kilohercios (Khz.), unidad que representa 1.000 Hz (12).

Una persona normal sin problemas auditivos puede oír sonidos que estén entre los 20 Hz y 20 Khz.(12).

Frecuencia de muestreo: es la cantidad de muestras de sonido capturadas en cada segundo. Su valor puede oscilar entre 8 Khz. y 48 Khz (12).

Amplitud: Es la cantidad de fuerza o energía de un sonido. Se mide en decibelios (db). Es la que influye en la intensidad o volumen con que se escucha el sonido. Por cada 3 db que se le aumente al sonido se duplica la intensidad del mismo. Por ejemplo si un sonido tiene 43 db y se le aumenta a 46 db se duplica su intensidad, y si se aumenta a 49 la intensidad será 4 veces mayor (12).

Precisión de las muestras: indica la escala de bits que se ha utilizado para guardar el sonido. Pueden ser 8 bits (256 valores posibles) o 16 bits (más de 65.000 valores posibles) (12).

Mono / Estéreo: El sonido pude grabarse en un solo canal (monoaural) o en dos (estereofónico) (12).

1.4.2. Formatos de sonido

A continuación se presentan algunos de los principales formatos de sonidos más populares en la actualidad.

WAV: Es un formato de audio digital sin comprimir. Fue el primer formato de audio que existió para PC. Fue desarrollado por IBM y Microsoft en 1991 para ser usado el sistema operativo Windows 3.1 (13). Admite archivos mono y estéreo a diversas resoluciones y velocidades de muestreo. Entre sus ventajas está que puede digitalizar los sonidos siendo totalmente fiel a la fuente original ya que es un formato sin compresión de datos, y además está su compatibilidad para convertirse en varios formatos por medio del software adecuado. Como principales desventajas tiene que debido a que precisamente es un formato sin compresión de datos los archivos wav son muy grandes (14), y estos archivos nada más pueden almacenar hasta 4 gigabytes debido a que en la cabecera del fichero se indica la longitud del mismo con un número de 32 bit, limitando el tamaño del fichero a 4 gigabytes.

MP3: MPEG-1 Audio Layer III, más conocido como MP3, es un formato de audio digital comprimido con pérdida, fue desarrollado por el *Moving Picture Experts Group* (MPEG). Los algoritmos utilizados en el desarrollo de MP3 se basan en la forma de escuchar que tiene el oído humano, pues las frecuencias que quedan fuera de la audición no son registradas en el archivo (las mayores de 20khz y las menores de 20hz). Esto se traduce en archivos mucho más pequeños sin una perdida perceptible de la calidad del sonido. La calidad de sonido del MP3 y su pequeño tamaño ha logrado que sea muy popular en Internet (15). El principal inconveniente con MP3 es que tiene patente, o sea es propietario, lo que hace prever que sea paulatinamente sustituido por otros formatos que tengan las mismas características de MP3 pero que sean libres de patentes.

Ogg: Este formato fue desarrollado por la fundación Xiph.org. Ogg es uno de los candidatos más firmes para sustituir al MP3. Presenta muchas ventajas con respecto al MP3 ya que hace que el sonido suene más natural y con una mayor calidad (13). El formato es libre de patentes y abierto, está diseñado para dar un alto grado de eficiencia en el *streaming* y la compresión de archivos. Además soporta audio de alta calidad y varios canales.

WMA: Windows Media Audio (WMA) es un formato de audio de compresión con pérdida, fue desarrollado por Microsoft, y está pensado especialmente para ser usado en el reproductor Windows Media Player, incluido en el popular sistema operativo Windows. Permite escuchar música mediante *streaming* manteniendo una alta calidad del sonido (13). WMA puede además gestionar los derechos digitales (16), logrando que si la canción está registrada con derechos de autor no se pueda copiar. Como principal desventaja tiene que es un formato propietario.

AAC: *Advanced Audio Coding* (AAC) es un formato de audio digital comprimido con pérdida es una extensión de MPEG-2 - un estándar creado por *Moving Pictures Expert Group*. Es una nueva y revolucionaria forma de codificar y reproducir ficheros desde el disco duro de un ordenador con una calidad que se puede asemejar al CD utilizando bastante menos espacio que el formato MP3. Ocupa casi un 30% menos de espacio que el MP3. Este sistema - al igual que MP3 - se aprovecha de las limitaciones del oído humano para descartar toda aquella información que no es perceptible, pero aventaja al formato MP3 en que elimina los defectos que se detectaron posteriormente en dicho formato (13).

AU: El formato AU es el formato nativo de las estaciones de trabajo Sun y similares. Es poco conocido fuera del ambiente UNIX. Este formato fue desarrollado por la empresa *NeXt Machine Sound System* y en la actualidad ha sido desplazado por otros formatos.

AIFF: *Audio Interchange File Format* (AIFF) es un formato de sonido muy simple y popular en Internet, es un formato originario para Macintosh parecido al WAV por su tamaño, también puede ser usado en otras plataformas.

VOC: Son similares a los archivos WAV, la diferencia es que traen marcadores de sincronización especialmente para ser usados con imágenes, videos u otros sonidos en aplicaciones multimedia.

1.4.3. Audio 3D

La síntesis del audio 3D es ampliamente utilizada en sonido en video juegos como en la música en algunos de los casos. El audio 3D funciona con la codificación HRTF *(Head Related Transfer Functions)* que simula procesos acústicos los cuales ocurren de forma natural en un espacio determinado produciendo una interacción entre la atmósfera simulada por el video juego y nuestros oídos. Dicho fenómeno surge como el resultado de reverberaciones y reflexiones con todas las variaciones que puedan tener estas (12).

1.4.4. SoundToolkit como biblioteca para el manejo de sonido en aplicaciones de realidad virtual

SoundToolkit es una biblioteca para el manejo de sonidos en SRV, surgida a partir de que se comienzan a realizar productos de esta rama de la informática dentro de la Universidad de las Ciencias Informáticas. Esta biblioteca brinda funcionalidades para el manejo de sonido de forma cómoda a los programadores, abstrayéndolos de los bajos niveles de programación, con una arquitectura orientada a objetos, configurable y multiplataforma. Fue desarrollada usando el Proceso Unificado de Software (RUP) e implementada en C++. Está basada en OpenAL. Soporta los formatos de sonido .ogg y .wav (12).

1.5. XML como formato para el almacenamiento de datos

XML (*e**X**tensible **M**arkup **L**anguage,* lenguaje de marcas extensible) es un metalenguaje extensible desarrollado por el *World Wide Web Consortium* (W3C). XML es un formato basado en texto, fue diseñado específicamente para almacenar y transmitir datos. Su principal novedad consiste en compartir los datos con los que trabajan todos los niveles, por todas las aplicaciones y soportes. Permitiendo compartir la información de una manera segura, fiable y fácil.

Los principales objetivos que se tuvieron en cuenta a la hora de desarrollar XML fueron los siguientes (17):

- Debe ser directamente utilizable sobre Internet.
- Debe soportar una amplia variedad de aplicaciones.
- Debe ser compatible con SGML.
- Debe ser fácil la escritura de programas que procesen documentos XML.
- El número de características opcionales en XML debe ser absolutamente mínimo, idealmente cero.
- Los documentos XML deben ser legibles por humanos y razonablemente claros.
- El diseño de XML debe ser preparado rápidamente.
- El diseño de XML debe ser formal y conciso.
- Los documentos XML deben ser de fácil creación.
- La concisión en las marcas XML es de mínima importancia.

1.5.1. Características

Entre las características de XML que han logrado que se convierta en un formato universal, se encuentran las siguientes (17):

- Es una arquitectura más abierta y extensible. No se necesita versiones para que pueda funcionar en futuros navegadores.
- Mayor consistencia, homogeneidad y amplitud de los identificadores descriptivos del documento con XML.

- Integración de los datos de las fuentes más dispares. Se puede hacer el intercambio de documentos entre las aplicaciones tanto la propia PC como en una red local o extensa.
- Datos compuestos de múltiples aplicaciones. La extensibilidad y flexibilidad de este lenguaje permite agrupar una amplia variedad de aplicaciones, desde páginas Web hasta bases de datos.
- Gestión y manipulación de los datos desde el propio cliente web.
- Se permitirá un comportamiento más estable y actualizable de las aplicaciones Web.
- Puede exportar a otros formatos de publicación. El documento maestro de la edición electrónica podría ser un documento XML que se integraría en el formato deseado de manera directa.

1.5.2. Ventajas

Entre las principales ventajas que ofrece XML se pueden citar las siguientes (17):

- Estandarización de la Información.
- Integración de aplicaciones.
- Portabilidad de Información.
- Compatibilidad entre sistemas.
- Mejora el acceso a la información.

1.6. Herramientas existentes para la configuración de sonidos en aplicaciones de realidad virtual

1.6.1. *Microsoft Cross Platform Audio Creation Tool*

Microsoft Cross Platform Audio Creation Tool (XACT) es una herramienta para la configuración de sonidos para juegos, forma parte de DirectX SDK (conjunto de bibliotecas, archivos de cabecera, herramientas de diagnóstico, utilitarios, etc., que posibilita el desarrollo de aplicaciones que utilicen DirectX). La configuración de los sonidos obtenida mediante XACT puede ser usada por aplicaciones que utilicen las bibliotecas de sonido XAudio o DirectSound. XACT se utiliza mayoritariamente en los juegos desarrollados

con XNA, que es una API desarrollada por Microsoft para el desarrollo de videojuegos para las plataformas Xbox 360 y Windows.

Entre las principales características de XACT tenemos las siguientes(18):

- Soporta los formatos de sonidos .wav, .aiff y .wma.
- Soporta sonidos estéreos y sonidos envolventes 5.1.
- Permite obtener una vista previa del resultado de la configuración de cada sonido.
- Permite organizar los archivos de sonidos en grupos denominados *Wave Banks*.
- Permite organizar la configuración de distintos sonidos en grupos denominados *Sound Banks*.

1.6.2. *Adventure Game Studio*

Adventure Game Studio (AGS) es un programa que contiene una interfaz gráfica mediante la cual se puede crear un juego de aventura gráfica sin ser necesario conocimientos de programación para ello, aunque cabe destacar que para usuarios con más experiencia incluye un editor de scripts para programar dentro del juego.

Dentro del grupo de funcionalidades que brinda AGS para el desarrollo de juegos se encuentran las referidas a la inclusión de sonidos, permitiendo agregar sonidos ambientales al juego o sonidos que se ejecuten cuando se vaya a realizar una acción determinada. Los formatos de archivos de sonidos que soporta son wav, ogg, voc, mp3, entre otros(19).

Conclusiones

En el transcurso de este capítulo se abordó todo lo relativo a las interfaces gráficas de usuario, presentándose los requisitos a tener en cuenta para desarrollar una correcta y agradable GUI. Se estudiaron las bibliotecas que permiten el desarrollo de interfaces gráficas de usuario. Se hizo un análisis de los motores gráficos libres de más prestaciones. Además se analizó el formato XML, mostrándose las ventajas que trae el uso de este. Todo esto encaminado a obtener los elementos necesarios para dar solución al objetivo de esta investigación.

Capítulo 2: Soluciones técnicas

Introducción

En este capítulo se expone la propuesta del sistema a obtener para lograr el objetivo de este trabajo. Además se analizan las estructuras de los ficheros donde se guardarán los datos persistentes y se hará una breve reseña de las herramientas a utilizar en el desarrollo de la aplicación.

2.1. Objeto de automatización

Con la realización de esta investigación se pretende automatizar el proceso de la adición de sonidos a escenas 3D, mediante la creación de una herramienta que permita agregar sonidos a una escena, configurar estos sonidos e ir en todo momento probando como se escuchan estos, pudiéndose de esta forma exportar esta configuración para que sea usada por terceros, minimizando el tiempo necesario para adicionar sonidos a una escena 3D e incrementando la productividad de los desarrolladores.

2.2. Información que se maneja

La información que se maneja es la referente a las propiedades del sonido y la posición del mismo en el entorno.

2.3. Propuesta del sistema

Se propone una aplicación de escritorio que permita al usuario cargar una escena 3D con el objetivo de adicionar y configurar sonidos, los cuales podrán ser ambientales o de objetos.

La aplicación permitirá además la realización de las siguientes acciones:

- Seleccionar el objeto al cual se le aplicará un sonido.
- Aplicar un sonido al objeto seleccionado.
- Configurar las propiedades del sonido aplicado al objeto.
- Eliminar un sonido aplicado al objeto seleccionado.
- Adicionar a la escena un sonido ambiental.
- Configurar las propiedades del sonido ambiental.
- Eliminar el sonido ambiental.
- Configurar las propiedades globales de los sonidos.
- Salvar las configuraciones de los sonidos.

- Salvar el proyecto, o sea salvar el entorno 3D con las configuraciones de sonido que se le hayan aplicado.
- Cargar un proyecto que fue salvado previamente.
- Rehacer y deshacer los cambios que se realicen mediante las salvas creadas a la vez que se realiza un cambio.

Las principales propiedades del sonido que se podrán configurar son el estado del sonido que puede ser play, pause o stop, el nivel al que pertenece el sonido, el volumen del sonido, el rango de volumen, la frecuencia del sonido, si es cíclico o no, entre otras.

Para la creación de la interfaz gráfica de usuario el sistema hará uso de la biblioteca Qt. La decisión de escoger Qt por encima de GTK+ estuvo basada fundamentalmente en la experiencia que se tenía con el uso de esta biblioteca y en la excelente documentación de Qt. Aclarar que GTK+ también cuenta con una buena documentación, pero sin dudas la documentación de Qt es una de la documentaciones más completa y fácil de usar que se pueda encontrar sobre cualquier software.

Se usará el motor gráfico OGRE para la representación de las escenas que se carguen en la aplicación. Esta decisión estuvo basada en que OGRE es un motor gráfico de gran rendimiento, y está diseñado para aprovechar al máximo las potencialidades del hardware moderno, superando en este aspecto tanto a Irrlicht como a OSG. La documentación de OGRE es buena aún cuando no se podría calificar de excelente, sin embargo cuenta con una muy activa comunidad en Internet que logra suplir la falta de documentación detallada que pueda existir sobre algún tema en particular. Por su parte la documentación de Irrlicht está casi a la altura de la de OGRE pero no cuenta con una comunidad tan activa como la de este. En este aspecto OSG es francamente inferior, ya que la documentación acerca de este motor gráfico es pobre y además la comunidad en Internet es menor y mucho menos activa que la de OGRE e Irrlicht, esta pobre documentación tiene como causa que los desarrolladores de este motor gráfico mayoritariamente cobran por los servicios de ayudas y documentación, por lo que para acceder a una mejor información o bien hay que comprar la documentación o bien hay que pagar el servicio de consultoría que los desarrolladores brindan. Por último - y esta fue la razón fundamental - OGRE actualmente es el motor gráfico con el cual están trabajando la mayoría de los proyectos en la facultad 5,

por lo que se decidió usar OGRE con la idea de llegar a dar en algún momento soporte para cargar escenas 3D en formato .scene que es el formato soportado por OGRE.

OGRE no permite cargar ficheros de modelos 3D en los formatos más usados, dígase .3ds, .obj, etc. Para solventar este problema se hará uso del motor gráfico OSG y de una serie de clases creadas por Assaf Raman –miembro del equipo desarrollador de OGRE–, el primero permite cargar ficheros de modelos 3D en los formatos más populares, contando con un poderoso sistema de *plugins* que posibilitan que de una manera muy fácil se pueda realizar esta operación; mientras que el segundo posibilita convertir los modelos cargados con OSG a mallas soportadas por OGRE.

Con el objetivo de hacer que la configuración de los sonidos se realice de la forma más amigable y sencilla posible se utilizará un grupo de clases creadas por Volker Wiendl –miembro del equipo desarrollador del motor gráfico Horde3D– las cuales haciendo uso de Qt simulan un editor de propiedades.

El fichero generado cuando se salve un proyecto o cuando se exporte la configuración de los sonidos estará en formato XML, para poder realizar esto se hará uso del *parser* TinyXml; este será usado además cuando se vaya a cargar un proyecto, pues es el que va a permitir leer la información del fichero donde se salvaron los datos del proyecto que se desea cargar.

2.4. Modelo de persistencia

2.4.1. Formato del fichero para almacenar la configuración de los sonidos

Luego de configurados los sonidos en la escena se necesita guardar esta información persistentemente, para ello se hará uso del formato XML. La principal razón por la que se determinó usar XML para salvar la configuración de los sonidos es la extensibilidad de XML, o sea, uno puede adicionar nuevas etiquetas sin que esto afecte a las aplicaciones que utilizaban la versión que no contaba con las nuevas etiquetas. En otras palabras si en versiones posteriores al fichero que guarda la configuración de los sonidos se le agregan otras etiquetas para almacenar otros datos de la configuración, las aplicaciones que cuenten con una versión anterior a la de los cambios podrán hacer uso de ese fichero, aunque como es lógico no usarán la nueva información.

La extensión del fichero donde se almacenará la configuración de los sonidos será **.cvs**, que es el acrónimo de ***C****onfiguration* ***V****isual* ***S****ound*.

2.4.1.1. Estructura del fichero

La principal etiqueta del fichero se denominará *sounds*, esta tendrá anidada toda la información referida a la configuración de los sonidos, dígase, configuración de sonidos aplicados a objetos de la escena, configuración de sonidos ambientales y la configuración de las propiedades globales de todos los sonidos. Primeramente aparecerá la información referida a la configuración de las propiedades globales de los sonidos y a continuación vendrá la información de los distintos sonidos.

2.4.1.2. Declaración de las propiedades globales de los sonidos

Las propiedades globales de los sonidos serán declaradas con la etiqueta *global_properties*, dentro de esta se encontrarán otras etiquetas que contendrán como tal la información de la configuración de las propiedades globales de los sonidos.

Ejemplo de definición de las propiedades globales

```
…
<global_properties>
    <doopler_factor>1</doopler_factor>
    <speed_sound>1</speed_sound>
    <gain>1</gain>
    <distance_model>INVERSE_DISTANCE_CLAMPED</distance_model>
</global_properties>
…
```

2.4.1.3. Declaración de los sonidos

Tanto la información de la configuración de los sonidos aplicados a objetos de la escena como la información de la configuración de los sonidos ambientales se almacenará en etiquetas *sound*. El primer atributo de esta etiqueta – *type* – será el que informe a qué tipo de sonido está referida la información que contiene el elemento, en caso de que sea un sonido ambiental el atributo *type* sería "*environmental*" y de ser un sonido aplicado a un objeto el valor asociado será "*object*"

Ejemplo de definición de un sonido ambiental

```
…
<sound type="environmental" sound_name="sound1" format="ogg" loop="yes">
    <name >Environmental6</name >
    <status>PLAY</status>
    <pitch>1</pitch>
    <gain min_gain="0" max_gain="1">1</gain>
    <level>0</level>
    <rool_of_factor>1</rool_of_factor>
    <direct_filter gain="1" gain_hf="1">NULL</direct_filter>
</sound>
…
```

Ejemplo de definición de un sonido de objeto

```
…
<sound type="object" sound_name="sound 2" format="ogg" loop="yes" relative="no">
        <name >Box03</name >
        <status>STOP</status>
        <pitch>1</pitch>
        <gain min_gain="0" max_gain="1">1</gain>
        <level>0</level>
        <rool_of_factor>1</rool_of_factor>
        <direct_filter gain="1" gain_hf="1">NULL</direct_filter>
        <position x="912.523" y="551.181" z="4744.32" />
        <direction x="0" y="0" z="0" />
        <reference_distance>2000</reference_distance>
        <max_distance>10000</max_distance>
        <cone_innner_angle>360</cone_innner_angle>
        <cone_outer_angle>360</cone_outer_angle>
    </sound>
…
```

2.4.2. Formato del fichero para almacenar la información de un proyecto

Para que el usuario tenga la posibilidad de guardar el estado del proyecto en el que esté trabajando - y de esa forma poder continuar la configuración de los sonidos de la escena en otro momento – se hace necesario igualmente guardar esta información persistentemente. En este caso se escogió igualmente el formato XML, por las razones anteriormente expuestas.

La extensión del fichero donde se almacenará la información del proyecto será **.vsop**, que es el acrónimo de ***V****isual* ***So****und* ***P****roject*.

2.4.2.1. Estructura del fichero

La principal etiqueta del fichero se denominará *visual_sound_project*, esta tendrá anidada toda la información del estado del proyecto. Primeramente aparecerán algunos elementos que almacenan la información que no está relacionada directamente con la configuración de los sonidos y a continuación vendrá la información de los sonidos.

2.4.2.2. Declaración de los elementos generales

Estos elementos contendrán la información relativa a la dirección donde se encuentra el modelo 3D – en caso de que se haya cargado alguno –, la velocidad con que se está desplazando el usuario por la escena y el último estado de reproducción que se le aplicó a todos los sonidos al mismo tiempo.

Ejemplo de elementos generales

```
<visual_sound_project>
   <dir_scene>C:/Documents and Settings/Admin/Escritorio/ city.3DS</dir_scene>
   <general_status>STOP</general_status>
   <speed>1500</speed>
  …
</visual_sound_project>
```

2.4.2.3. Declaración de la información de los sonidos

Para guardar la información referida a los sonidos se usa la misma estructura que se utiliza en los ficheros donde se guarda la configuración de los sonidos, con la diferencia de que en este caso se le agregarán otras etiquetas.

Dentro de la etiqueta *global_properties* se adicionan las etiquetas que permiten almacenar la posición y orientación del oyente en el momento de ser salvado el proyecto, y el total de sonidos ambientales que se han creado hasta ese instante, este último dato se guarda para lograr que los nombres que se le vayan asignando a los sonidos ambientales sean únicos.

Dentro de cada elemento *sound* se agrega una etiqueta que guardará la dirección donde se encuentra el fichero de sonido utilizado.

Ejemplo de información de los sonidos

```
…
<sounds>
    <global_properties>
        <doopler_factor>1</doopler_factor>
        <speed_sound>1</speed_sound>
        <gain>1</gain>
        <position x="606.328" y="647.938" z="2688.98" />
        <total_environmental>6</total_environmental>
        <orientation dir_x="0.99" dir_y="-0.031" dir_z="0.097" up_x="0.031" up_y="0.99" up_z="0.031" />
        <distance_model>INVERSE_DISTANCE_CLAMPED</distance_model>
    </global_properties>
    <sound type="environmental" sound_name="sound 5" format="ogg" loop="yes">
        <file>C:/Documents and Settings/Admin/Escritorio/ sound 1.ogg</file>
        <name>Environmental6</name>
        <status>PLAY</status>
        <pitch>1</pitch>
        <gain min_gain="0" max_gain="1">1</gain>
        <level>0</level>
        <rool_of_factor>1</rool_of_factor>
        <direct_filter gain="1" gain_hf="1">NULL</direct_filter>
    </sound>
    …
</sounds>
…
```

2.5. Herramientas de desarrollo

2.5.1. Microsoft Visual Studio 2008

Microsoft Visual Studio 2008 es un IDE para sistemas operativos Windows. Permite a los desarrolladores crear rápidamente aplicaciones de alta calidad y riqueza. Para ello cuenta con un conjunto de herramientas de desarrollo profesionales, las cuales conforman un sistema altamente productivo. Permite además, la adiciones de *plugins* que mejoran la funcionalidad en casi todos los niveles.

2.5.2. Code::Blocks 8.0.2

Code::Blocks es un IDE libre, multiplataforma y de código abierto para el desarrollo de programas en lenguaje C++. Está diseñado para ser muy extensible y totalmente configurable. Construido alrededor de una plataforma de *plugins*, lo que permite que le pueda ser adicionada cualquier tipo funcionalidad mediante la instalación de *plugins*. Soporta varios compiladores de C++ como MinGW (pre-determinado), GCC, Borland C++ Compiler, Digital Mars Compiler, Intel C++ Compiler y Open Watcom.

2.5.3. Rational Rose

Rational Rose es una herramienta de desarrollo del software para el Modelado Visual. Actualmente se encuentra a la cabeza en cuanto al desarrollo del *Unified Modeling Language* (UML), que se ha convertido en la notación estandarizada empleada en Rational Rose para especificar, visualizar y construir desarrollos de software y sistemas. Lo que permite que todo el equipo de desarrollo pueda comunicarse con un lenguaje y una herramienta. Rational Rose es el líder en el mercado en cuanto a herramientas para el análisis, modelado, diseño y construcción orientada a objetos se refiere. Brinda la posibilidad de visualizar, entender y refinar los requerimientos y arquitectura antes de enfrentar el código, evitando desperdiciar esfuerzo durante el ciclo de desarrollo. Permite especificar, analizar y diseñar el sistema antes de codificarlo. Además permite chequear la sintaxis UML, mantiene la consistencia de los modelos del sistema del software, genera la documentación automáticamente, posibilita la generación de código a partir de los modelos, entre otras funcionalidades.

2.5.4. Lenguaje de desarrollo

El lenguaje de programación a utilizar será C++, ya que es un lenguaje libre, portable y robusto, el cual esta soportado bajo el paradigma de la Programación Orientada a Objeto, además de ser el ideal para el desarrollo de aplicaciones gráficas.

Conclusiones

Con el capítulo presentado quedan trazadas las guías que servirán para realizar las siguientes fases del trabajo. Además, se tienen las bases por las cuales se regirá el desarrollo del sistema.

Capítulo 3: Características del sistema

Introducción

En este capítulo se define una visión más detallada y específica del sistema que se va a desarrollar, tomando como base las soluciones técnicas planteadas en el capítulo anterior. Se precisan las reglas del negocio y el modelo del dominio. Se determinan las capacidades o funciones que el sistema debe cumplir (requisitos funcionales) y las propiedades o cualidades que el producto debe tener (requisitos no funcionales). Se describen los procesos – en forma de casos de uso – que dan cumplimiento a los requisitos funcionales.

3.1. Reglas del negocio

- Los formatos de modelos 3D que se cargarán serán del tipo.3ds y .obj.
- Los formatos de los sonidos que se cargarán serán del tipo .wav y .ogg.
- Los sonidos en formato .wav que serán utilizados como sonidos de objetos deben ser monoaural. Solo serán estereofónico los sonidos destinados a ambientes o bandas musicales, que serán codificados como .ogg.
- En la dirección donde se guarde el fichero de la configuración de los sonidos también tendrán que estar los sonidos que se utilizan en dicha configuración.

3.2. Modelo del dominio

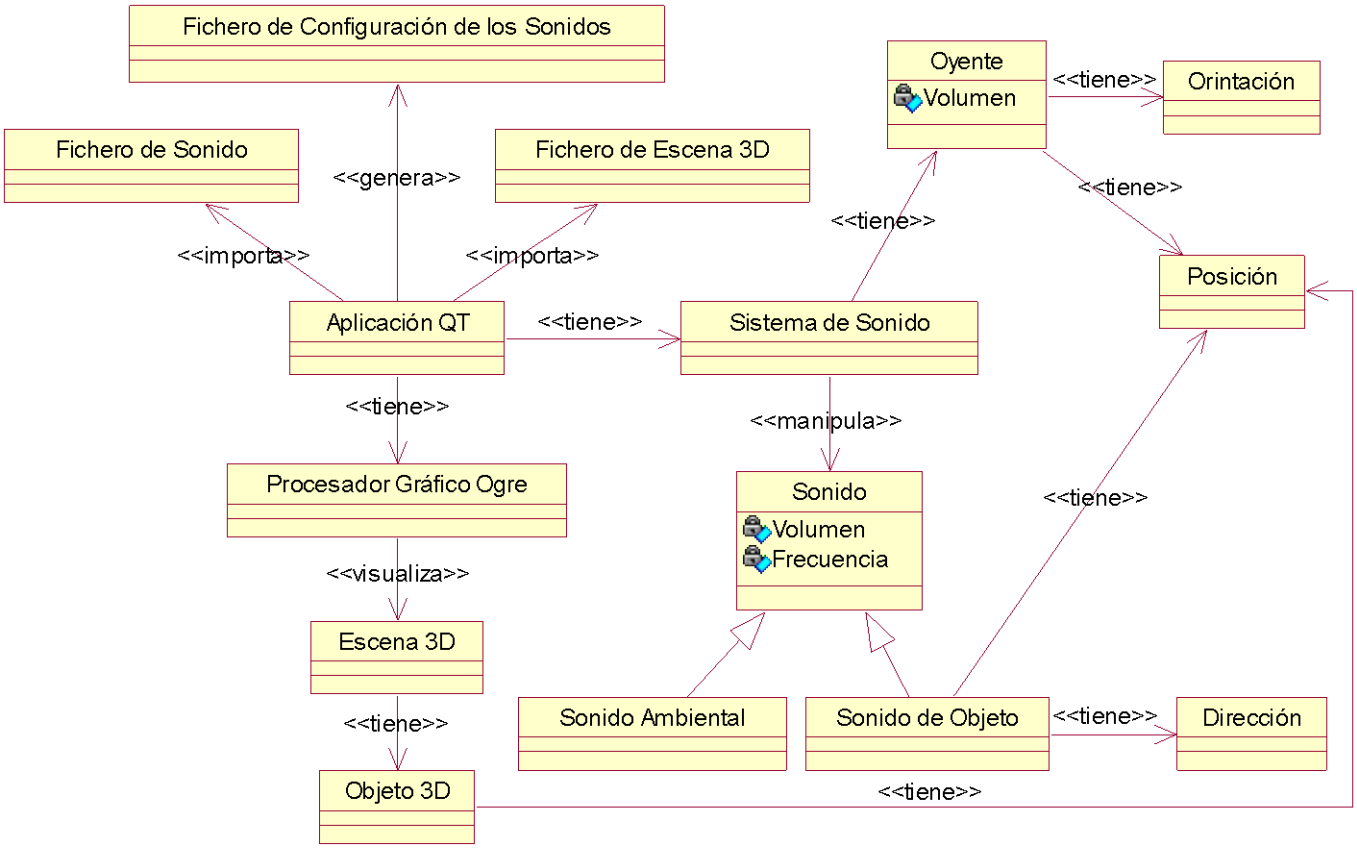

Figura 1: Modelo del dominio

3.3. Glosario de términos del modelo del dominio

Aplicación QT: Interfaz gráfica de usuario basada en Qt que posibilita que se ejecuten las distintas funcionalidades de la aplicación de una forma sencilla e intuitiva.

Escena 3D: Entorno virtual. Contiene la información gráfica de varios objetos 3D.

Fichero de Configuración de los Sonidos: Fichero que contiene la información referente a la configuración de los sonidos en una escena 3D.

Fichero de Escena 3D: Fichero que contiene la información referente a una escena 3D.

Fichero de Sonido: Fichero que contiene la información referente a un sonido.

Objeto 3D: Representación gráfica tridimensional computarizada de un objeto del mundo real.

Orientación: Componente del estado geométrico de los cuerpos, que describe la variación en ángulo respecto a los ejes X, Y y Z.

Oyente: Referencia el punto donde se centra la percepción del audio.

Posición: Ubicación vectorial en un ambiente virtual (X, Y, Z).

Procesador Gráfico Ogre: Motor gráfico para la visualización de objetos 3D.

Sistema de sonido: Controlador o manipulador de los sonidos y sus propiedades.

Sonido: Codificación digital de una señal eléctrica que representa una onda sonora.

Sonido Ambiental: Sonido que se escucha de la misma forma en cualquier lugar de la escena 3D.

Sonido de Objeto: Sonido que se emite desde una posición especifica de la escena 3D asociado a un objeto de la escena.

3.4. Dependencias y relaciones con otros software

La aplicación no depende de ningún otro software para su funcionamiento.

3.5. Requisitos funcionales

1. Crear proyecto.
2. Cargar proyecto.
3. Salvar proyecto.
4. Cargar modelo 3D.
5. Seleccionar objeto.
6. Deseleccionar objeto.
7. Aplicar sonido a objeto.
8. Configurar sonido de objeto.
9. Eliminar sonido de objeto.
10. Crear sonido ambiental.
11. Configurar sonido ambiental.
12. Eliminar sonido ambiental.
13. Configurar propiedades globales de los sonidos.
14. Deshacer un cambio.
15. Rehacer un cambio.
16. Generar fichero de configuración de los sonidos.

3.6. Requisitos no funcionales

Requerimientos de software**:** Se utilizarán los IDE de desarrollo Microsoft Visual Studio 2008 y Code::Blocks 8.0.2; los motores gráficos OGRE 1.6.0 y OSG 2.8.0; la biblioteca de desarrollo de interfaces gráficas de usuario QT 4.5.0 y el parser de XML TinyXML 2.5.3.

Requerimientos de rendimiento**:** La aplicación debe correr en tiempo real con un alto grado de velocidad de procesamiento, tiempo de respuesta, recuperación del sistema y disponibilidad.

Requerimientos de Hardware: Se necesita un ordenador con CPU Intel Pentium IV, 1 Gb de HDD, 256 Mb de RAM. Compatibilidad con tarjetas gráficas de la familia NVIDIA (Geforce 3, Geforce 4, FX 5200).

Restricciones de Implementación: Debe ser implementado en el Leguaje C++. Se regirá por la filosofía de Programación Orientada a Objetos.

Requerimientos de Soporte: La aplicación debe de ser compatible con las plataformas Windows y GNU/Linux. En esta primera versión solo estará disponible para plataforma Windows.

Requerimientos de Usabilidad: Los usuarios del sistema no tienen que tener necesariamente conocimientos de programación, simplemente tienen que conocer las funcionalidades para la manipulación de los sonidos que brinda la biblioteca SoundToolkit.

3.7. Definición de los casos de uso del sistema

A continuación se reconocen los actores del sistema a desarrollar, y partir de los requisitos funcionales obtenidos anteriormente se concebirán los casos de uso del sistema y sus respectivas descripciones.

3.7.1. Actor del sistema

Actor	Justificación
Usuario	Es el que se encarga de configurar los sonidos en un entorno virtual.

Tabla 1: Actor del sistema

3.7.2. Casos de usos del sistema

Caso de uso	
CU-1	Crear proyecto
Actores	Usuario
Descripción	Permite crear un nuevo proyecto. En caso de que en el proyecto que estuviese abierto con anterioridad se haya realizado algún cambio que no haya sido salvado, se le preguntará al usuario si desea o no guardar los cambios.
Referencias	R1

Tabla 2: Caso de uso "Crear proyecto"

Caso de uso	
CU-2	Cargar proyecto
Actores	Usuario
Descripción	Permite cargar un proyecto que haya sido guardado con anterioridad. En caso de que en el proyecto que estuviese abierto en ese momento se haya realizado algún cambio que no haya sido salvado, se le preguntará al usuario si desea o no guardar los cambios. Si ocurriese algún error al cargar el proyecto, ya sea porque la dirección es incorrecta o porque los datos son incorrectos, se crearía un nuevo proyecto.
Referencias	R2

Tabla 3: Caso de uso "Cargar proyecto"

Caso de uso	
CU-3	Salvar proyecto
Actores	Usuario
Descripción	Permite salvar el estado en el que se encuentra el proyecto que este abierto.
Referencias	R3

Tabla 4: Caso de uso "Salvar proyecto"

Caso de uso	
CU-4	Cargar modelo 3D
Actores	Usuario
Descripción	Permite cargar un modelo 3D. Si anteriormente en el mismo proyecto había sido cargado otro modelo 3D, este es eliminado junto a los sonidos que se le hayan aplicado a los objetos de la escena. De ocurrir algún error al tratar de cargar el modelo, ya sea porque la dirección es incorrecta o porque los datos no son validos, se deja al proyecto en el estado en que se encontraba antes de intentar cargar el modelo.
Referencias	R4

Tabla 5: Caso de uso "Cargar modelo 3D"

Caso de uso	
CU-5	Seleccionar objeto
Actores	Usuario
Descripción	Permite seleccionar un objeto de la escena que se esté mostrando en ese momento. Si el objeto seleccionado tiene aplicado algún sonido se muestran los datos de dicho sonido. En caso de que existiese otro objeto seleccionado con anterioridad este es deseleccionado.
Referencias	R5

Tabla 6: Caso de uso "Seleccionar objeto"

Caso de uso	
CU-6	Deseleccionar objeto
Actores	Usuario
Descripción	Permite deseleccionar el objeto que este seleccionado en ese momento. En caso de que el objeto tuviese algún sonido aplicado, se dejan de mostrar los datos de dicho sonido.
Referencias	R6

Tabla 7: Caso de uso "Deseleccionar objeto"

Caso de uso	
CU-7	Aplicar sonido a objeto
Actores	Usuario
Descripción	Permite aplicarle un sonido al objeto que esté seleccionado en ese momento siempre y cuando este no tenga aplicado ningún sonido. En caso de producirse algún error, ya sea porque la dirección del sonido es incorrecta o porque la información del fichero de sonido es incorrecta, se retorna al proyecto al estado en que se encontraba con anterioridad.
Referencias	R7

Tabla 8: Caso de uso "Aplicar sonido a objeto"

Caso de uso	
CU-8	Configurar sonido de objeto
Actores	Usuario
Descripción	Permite modificar las propiedades del sonido que se le haya aplicado al objeto que esté seleccionado en ese momento.
Referencias	R8

Tabla 9: Caso de uso "Configurar sonido de objeto"

Caso de uso	
CU-9	Eliminar sonido de objeto
Actores	Usuario
Descripción	Permite eliminar el sonido que se le haya aplicado al objeto que esté seleccionado en ese momento.
Referencias	R9

Tabla 10: Caso de uso "Eliminar sonido de objeto"

Caso de uso	
CU-10	Crear sonido ambiental
Actores	Usuario
Descripción	Permite crear un sonido ambiental. En caso de que ocurra algún error, ya sea porque la dirección del sonido es incorrecta o porque la información del fichero de sonido no es válida, se retorna al proyecto al estado en que se encontraba con anterioridad.
Referencias	R10

Tabla 11: Caso de uso "Crear sonido ambiental"

Caso de uso	
CU-11	Configurar sonido ambiental
Actores	Usuario
Descripción	Permite modificar las propiedades del sonido ambiental que se seleccione.
Referencias	R11

Tabla 12: Caso de uso "Configurar sonido ambiental"

Caso de uso	
CU-12	Eliminar sonido ambiental
Actores	Usuario

Descripción	Permite eliminar el sonido ambiental que se seleccione.
Referencias	R12

Tabla 13: Caso de uso "Eliminar sonido ambiental"

Caso de uso	
CU-13	Configurar propiedades globales de los sonidos
Actores	Usuario
Descripción	Permite modificar las propiedades globales de los sonidos.
Referencias	R13

Tabla 14: Caso de uso "Configurar propiedades globales de los sonidos"

Caso de uso	
CU-14	Deshacer cambio
Actores	Usuario
Descripción	Permite deshacer el último cambio que haya sido realizado en el proyecto actual.
Referencias	R14

Tabla 15: Caso de uso "Deshacer cambio"

Caso de uso	
CU-15	Rehacer cambio
Actores	Usuario
Descripción	Permite rehacer el último cambio que haya sido desecho en el proyecto actual.
Referencias	R15

Tabla 16: Caso de uso "Rehacer cambio"

Caso de uso

CU-16	Generar fichero de configuración de los sonidos
Actores	Usuario
Descripción	Permite generar un fichero con la configuración de los sonidos en el proyecto actual.
Referencias	R16

Tabla 17: Caso de uso "Generar fichero de configuración de los sonidos"

3.7.3. Diagrama de casos de usos del sistema

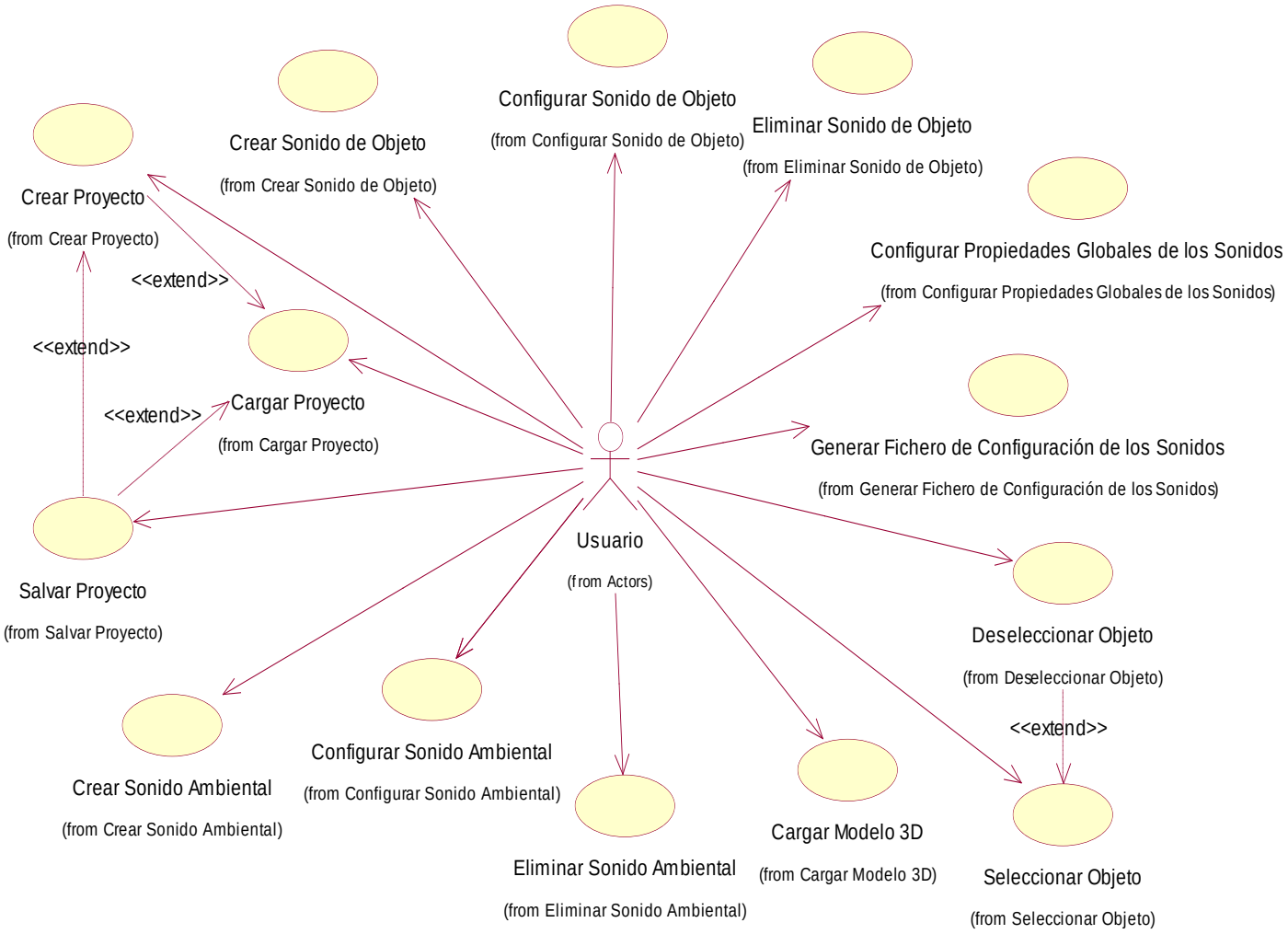

Figura 2: Diagrama de los casos de uso del sistema

3.7.4. Especificación de los casos de uso en formato expandido.

Caso de uso	Crear proyecto	
Actores	Usuario	
Resumen	El CU se inicia cuando el usuario decide crear un nuevo proyecto.	
Referencias	R1	
Precondiciones	-	
Flujo normal de eventos		
Acción del actor	**Respuesta del sistema**	
1. El usuario se dirige al menú principal y selecciona el submenú *File*.	1.1. El sistema muestra varias opciones entre las que se encuentra *New Project*.	
2. El usuario selecciona la opción *New Project*.	2.1. El sistema verifica si se ha realizado algún cambio en el proyecto actual que no haya sido salvado. 2.2. El sistema crea un nuevo proyecto.	
Flujo alterno 1		
Acción del actor	**Respuesta del sistema**	
	2.1. En caso positivo el sistema muestra un cuadro de diálogo con las opciones de guardar los cambios realizados, no guardar los cambios y cancelar la operación.	
3.1. El usuario decide guardar los cambios realizados.	3.1.1. El sistema salva el proyecto. 3.1.2. El sistema crea un nuevo proyecto.	
Flujo alterno 2		
3.2. El usuario decide no guardar los cambios realizados.	3.2.1. El sistema crea un nuevo proyecto.	
Flujo alterno 3		
3.3. El usuario decide cancelar la creación de un nuevo proyecto.	3.3.1. El sistema cierra el cuadro de diálogo.	
Puntos de extensión	Línea 3.1.1: Ver CU-3 Salvar proyecto.	
Poscondiciones	Es creado un nuevo proyecto.	

Tabla 18: Expansión del CU "Crear proyecto"

Caso de uso	Cargar proyecto
Actores	Usuario
Resumen	El CU se inicia cuando el usuario decide cargar un proyecto.
Referencias	R2
Precondiciones	-

Flujo normal de eventos	
Acción del actor	**Respuesta del sistema**
1. El usuario se dirige al menú principal y selecciona el submenú *File*.	1.1. El sistema muestra varias opciones entre las que se encuentra *Load Project*.
2. El usuario selecciona la opción *Load Project*.	2.1. El sistema verifica si se ha realizado algún cambio en el proyecto actual que no haya sido salvado. 2.2. El sistema muestra un cuadro de diálogo pre-configurado que permite especificar la ruta del proyecto a cargar o cancelar esta operación.
3. El usuario especifica la ruta del proyecto que desea cargar y pulsa abrir.	3.1. El sistema cierra el cuadro de diálogo. 3.2. El sistema verifica que la ruta sea válida. 3.3. El sistema verifica la validez de los datos del proyecto que se desea cargar. 3.4. El sistema carga el proyecto y se muestra la configuración de este.
Flujo alterno 1	
Acción del actor	**Respuesta del sistema**
	2.1. Si existe algún cambio en el proyecto actual que no haya sido salvado el sistema muestra un cuadro de diálogo con las opciones de guardar los cambios realizados, no guardar los cambios y cancelar la operación.
3.1. El usuario decide guardar los cambios realizados.	3.1.1. El sistema salva el proyecto. Ir al paso 2.2.
Flujo alterno 2	

3.2. El usuario decide no guardar los cambios realizados.	3.2.1. Ir al paso 2.2.
Flujo alterno 3	
3.3. El usuario decide cancelar la acción de cargar un proyecto.	3.3.1. El sistema cierra el cuadro de diálogo.
Flujo alterno 4	
3. El usuario decide cancelar la carga.	3.1. El sistema cierra el cuadro de diálogo.
Flujo alterno 5	
	3.2. Si la ruta no es correcta el sistema muestra un mensaje de error. 3.3. El sistema crea un nuevo proyecto.
Flujo alterno 6	
	3.3. Si los datos son incorrectos el sistema muestra un mensaje de error. 3.4. El sistema crea un nuevo proyecto.
Puntos de extensión	Línea 3.1.1: Ver CU-3 Salvar proyecto. Línea 3.3 (Flujo Alterno 5): Ver CU-1 Crear proyecto. Línea 3.4 (Flujo Alterno 6): Ver CU-1 Crear proyecto.
Poscondiciones	Es cargado un proyecto.

Tabla 19: Expansión del CU "Cargar proyecto"

Caso de uso	Salvar proyecto
Actores	Usuario
Resumen	El CU se inicia cuando el usuario decide salvar el proyecto actual.
Referencias	R3
Precondiciones	-
Flujo normal de eventos	
Acción del actor	**Respuesta del sistema**
1. El usuario se dirige al menú principal y selecciona el submenú *File*.	1.1. El sistema muestra varias opciones entre las que se encuentra *Save Project*.

2. El usuario selecciona la opción *Save Project*.	2.1. El sistema verifica si el proyecto ha sido salvado con anterioridad. 2.2. El proyecto es salvado.
Flujos alterno1	
Acción del actor	**Respuesta del sistema**
	2.1. Si el proyecto no ha sido salvado con anterioridad el sistema muestra un cuadro de diálogo pre-configurado que permite especificar la ruta de donde se va a salvar el proyecto o cancelar esta operación.
3.1. El usuario especifica la ruta donde desea salvar el proyecto y pulsa abrir.	3.1.1. El sistema verifica que la ruta sea válida. 3.1.2. El sistema salva los datos del proyecto en la ruta especificada por el usuario.
Flujo alterno 2	
3.1. El usuario decide cancelar la operación de salvar el proyecto.	3.1.1. El sistema cierra el cuadro de diálogo.
Flujo alterno 3	
	3.1.1. Si la ruta no es válida el sistema muestra un mensaje de error.
Puntos de extensión	-
Poscondiciones	Es salvado el proyecto actual.

Tabla 20: Caso de uso expandido "Salvar proyecto"

Caso de uso	Cargar modelo 3D
Actores	Usuario
Resumen	El CU se inicia cuando el usuario decide cargar un modelo 3D
Referencias	R4
Precondiciones	-
Flujo normal de eventos	
Sección "General"	

Acción del actor	Respuesta del sistema
1. El usuario se dirige al menú principal y selecciona el submenú *File*.	1.1. El sistema muestra varias opciones entre las que se encuentra *Load Model*.
2. El usuario escoge la opción *Load Model*.	2.1. El sistema muestra un cuadro de diálogo pre-configurado que permite escoger el formato de modelo 3D - .3ds o .obj - que desea cargar, especificar la ruta de donde se va a cargar el modelo o cancelar está operación.
3. El usuario especifica el tipo de modelo 3D que desea cargar, la ruta del modelo y pulsa abrir.	3.1 Si selecciona cargar un modelo 3D en formato .3ds ir a sección **Cargar 3ds**. Si selecciona cargar un modelo 3D en formato .obj ir a sección **Cargar obj**.
Sección "Cargar 3ds"	
4. El usuario especifica la ruta del fichero 3ds que desea cargar.	4.1. El sistema cierra el cuadro de diálogo. 4.2. El sistema verifica que la ruta sea válida. 4.3. El sistema verifica la validez de los datos del fichero 3ds. 4.4. El sistema elimina los objetos de la escena y los sonidos que hayan sido aplicados a estos. 4.5. El sistema crea y muestra los objetos obtenidos a partir de la información del archivo.
Sección "Cargar obj"	
4. El usuario especifica la ruta del fichero obj que desea cargar.	4.1. El sistema cierra el cuadro de diálogo. 4.2. El sistema verifica que la ruta sea válida. 4.3. El sistema verifica la validez de los datos del fichero obj. 4.4. El sistema elimina los objetos de la escena y los sonidos que hayan sido aplicados a estos. 4.5. El sistema crea y muestra los objetos obtenidos a partir de la información del archivo.
Flujos alternos	

Sección "General"	
Flujo alterno 1	
3. El usuario decide cancelar la carga del modelo 3D.	3.1. El sistema cierra el cuadro de diálogo.
Sección "Cargar 3ds"	
Acción del actor	**Respuesta del sistema**
Flujo alterno 2	
	4.2. Si la ruta no es válida el sistema muestra un mensaje de error.
Flujo alterno 3	
	4.3. Si los datos del fichero 3ds no son válidos el sistema muestra un mensaje de error.
Sección "Cargar obj"	
Flujo alterno 4	
	4.2. Si la ruta no es válida el sistema muestra un mensaje de error.
Flujo alterno 5	
	4.3. Si los datos del fichero obj no son válidos el sistema muestra un mensaje de error.
Puntos de extensión	-
Poscondiciones	Es cargado un modelo 3D y mostrado en la escena.

Tabla 21: Caso de uso expandido "Cargar modelo 3D"

Caso de uso	Seleccionar objeto
Actores	Usuario
Resumen	El CU se inicia cuando el usuario decide seleccionar un objeto de la escena.
Referencias	R5
Precondiciones	-
Flujo normal de eventos	

Acción del actor	Respuesta del sistema
1. El usuario da clic sobre un objeto de la escena.	1.1. Si existe algún objeto seleccionado anteriormente este es deseleccionado. 1.2. El sistema resalta el objeto sobre el cual el usuario dio clic.
Puntos de extensión	Línea 1.1: Ver CU-6 Deseleccionar Objeto
Poscondiciones	El objeto sobre el que se dio clic es resaltado.

Tabla 22: Caso de uso expandido "Seleccionar objeto"

Caso de uso	Deseleccionar objeto
Actores	Usuario
Resumen	El CU se inicia cuando el usuario decide deseleccionar un objeto de la escena.
Referencias	R6
Precondiciones	Existe algún objeto seleccionado.
Flujo normal de eventos	
Acción del actor	**Respuesta del sistema**
1. El usuario da clic derecho sobre el objeto seleccionado.	1.1. El sistema muestra varias opciones entre las que se encuentra *Deselected Object*.
2. El usuario escoge la opción *Deselected Object*.	2.1. El sistema deja de resaltar el objeto que había sido seleccionado.
Puntos de extensión	-
Poscondiciones	Se deja de resaltar al objeto que había sido seleccionado anteriormente.

Tabla 23: Caso de uso expandido "Deseleccionar objeto"

Caso de uso	Aplicar sonido a objeto
Actores	Usuario
Resumen	El CU se inicia cuando el actor decide aplicarle sonido a un objeto
Referencias	R7

Precondiciones	-
Flujo normal de eventos	
Acción del actor	**Respuesta del sistema**
1. El usuario selecciona un objeto.	1.1 El sistema comprueba que no tenga ningún sonido aplicado. 1.2 El sistema habilita la opción *Attach Sound Object.*
2. El usuario se dirige al menú principal y selecciona el submenú *Sound*.	2.1. El sistema muestra varias opciones entre las que se encuentra *Attach Sound Object*.
3. El usuario escoge la opción *Attach Sound Object*.	3.1. El sistema muestra un cuadro de diálogo pre-configurado que permite especificar la ruta de donde se va a cargar el fichero de sonido o cancelar está operación.
4. El usuario especifica la ruta del fichero de sonido que desea aplicarle al objeto y pulsa abrir.	4.1. El sistema cierra el cuadro de diálogo. 4.2. El sistema verifica que la ruta sea válida. 4.3. El sistema verifica la validez de los datos del fichero de sonido que desea cargar. 4.4. El sistema crea un sonido en la posición donde se encuentra el objeto seleccionado. 4.5. El sistema muestra los datos del sonido creado.
Flujo alterno 1	
Acción del actor	**Respuesta del sistema**
	1.1. Si el objeto seleccionado tiene aplicado algún sonido el sistema deshabilita la opción *Attach Sound Object*.
Flujo alterno 2	
4. El usuario decide cancelar la carga del sonido.	4.1. El sistema cierra el cuadro de diálogo.
Flujo alterno 3	
	4.2. Si la ruta no es válida el sistema muestra un mensaje de error.
Flujo alterno 4	

	4.3. Si los datos del fichero de sonido no son válidos el sistema muestra un mensaje de error.
Puntos de extensión	Línea 1: Ver CU-5 Seleccionar objeto
Poscondiciones	Es creado un sonido en la posición del objeto seleccionado.

Tabla 24: Caso de uso expandido "Aplicar Sonido a Objeto"

Caso de uso	Configurar sonido de objeto
Actores	Usuario
Resumen	El CU se inicia cuando el usuario decide configurar las propiedades del sonido que ha sido aplicado a un objeto
Referencias	R8
Precondiciones	-
Flujo normal de eventos	
Acción del actor	**Respuesta del sistema**
1. El usuario selecciona un objeto.	1.1 El sistema comprueba que tenga un sonido aplicado. 1.2 Si el objeto tiene aplicado un sonido el sistema muestra los datos del sonido aplicado al objeto seleccionado.
2. El usuario configura las propiedades del sonido.	2.1. El sistema verifica que el(los) dato(s) de la(s) propiedad(es) del sonido que se desea(n) configurar sea(n) correcto(s). 2.2. Si el(los) dato(s) es(son) correcto(s) el sistema modifica la(s) propiedad(es) del sonido.
Flujo alterno 1	
Acción del actor	**Respuesta del sistema**
	2.1. Si el(los) dato(s) no es(son) válido(s) el sistema restablece los valores de las propiedades del sonido que se mostraban antes del intento de configuración.

Puntos de extensión	Línea 1: Ver CU-5 Seleccionar objeto
Poscondiciones	Son modificadas las propiedades del sonido aplicado al objeto seleccionado.

Tabla 25: Caso de uso expandido "Configurar Sonido de Objeto"

Caso de uso	Eliminar sonido de objeto
Actores	Usuario
Resumen	El CU se inicia cuando el usuario decide eliminar el sonido que ha sido aplicado a un objeto
Referencias	R9
Precondiciones	-

Flujo normal de eventos	
Acción del actor	**Respuesta del sistema**
1. El usuario selecciona un objeto.	1.1 El sistema comprueba que tenga un sonido aplicado. 1.2 El sistema habilita la opción *Remove Sound Object.*
2. El usuario se dirige al menú principal y selecciona el submenú *Sound*.	2.1. El sistema muestra varias opciones entre las que se encuentra *Remove Sound Object*.
3. El usuario escoge la opción *Remove Sound Object*.	3.1. El sistema elimina el sonido aplicado al objeto seleccionado.

Flujo alterno 1	
Acción del actor	**Respuesta del sistema**
	1.1. Si el objeto seleccionado no tiene aplicado algún sonido el sistema deshabilita la opción *Remove Sound Object*.

Puntos de extensión	Línea 1: Ver CU-5 Seleccionar objeto
Poscondiciones	Es eliminado el sonido que había sido aplicado al objeto seleccionado.

Tabla 26: Caso de uso expandido "Eliminar Sonido de Objeto"

Caso de uso	Crear sonido ambiental
Actores	Usuario

Resumen	El CU se inicia cuando el usuario decide crear un sonido ambiental.	
Referencias	R10	
Precondiciones	-	
Flujo normal de eventos		
Acción del actor		**Respuesta del sistema**
1. El usuario se dirige al menú principal y selecciona el submenú *Sound*.		1.1. El sistema muestra varias opciones entre las que se encuentra *Add Environmental Sound*.
2. El usuario escoge la opción *Add Environmental Sound*.		2.1. El sistema muestra un cuadro de diálogo pre-configurado que permite especificar la ruta de donde se va a cargar el fichero de sonido o cancelar está operación.
3. El usuario especifica la ruta del fichero de sonido que desea cargar y pulsa abrir.		3.1. El sistema cierra el cuadro de diálogo. 3.2. El sistema verifica que la ruta sea válida. 3.3. El sistema verifica la validez de los datos del fichero de sonido que desea cargar. 3.4. El sistema crea un sonido ambiental. 3.5. El sistema muestra los datos del sonido ambiental creado.
Flujo alterno 1		
Acción del actor		**Respuesta del sistema**
3. El usuario decide cancelar la carga del sonido.		3.1. El sistema cierra el cuadro de diálogo.
Flujo alterno 2		
		3.2. Si la ruta no es válida el sistema muestra un mensaje de error.
Flujo alterno 3		
		3.3. Si los datos del fichero de sonido no son válidos el sistema muestra un mensaje de error.
Puntos de extensión	-	
Poscondiciones	Es creado un sonido ambiental.	

Tabla 27: Caso de uso expandido "Crear Sonido Ambiental"

Caso de uso	Configurar sonido ambiental
Actores	Usuario
Resumen	El CU se inicia cuando el usuario decide configurar las propiedades de un sonido ambiental
Referencias	R11
Precondiciones	-

Flujo normal de eventos	
Acción del actor	**Respuesta del sistema**
1. El usuario se dirige al menú principal y selecciona el submenú *View*.	1.1. El sistema muestra varias opciones entre las que se encuentra *Environmental Sound Window*.
2. El usuario escoge la opción *Environmental Sound Window*.	2.1. El sistema muestra una ventana con todos los sonidos ambientales existentes.
3. El usuario selecciona el sonido ambiental que desea configurar.	3.1 El sistema muestra los datos del sonido ambiental seleccionado
4. El usuario configura las propiedades del sonido ambiental.	4.1. El sistema verifica que el(los) dato(s) de la(s) propiedad(es) del sonido ambiental que se desea(n) configurar sea(n) correcto(s). 4.2. Si el(los) dato(s) es (son) correcto(s) el sistema modifica la(s) propiedad(es) el sonido ambiental.

Flujo alterno 1	
Acción del actor	**Respuesta del sistema**
	4.1. Si el(los) dato(s) no es (son) válido(s) el sistema restablece los valores de las propiedades del sonido ambiental que se mostraban antes del intento de configuración.

Puntos de extensión	-
Poscondiciones	Son modificadas las propiedades de un sonido ambiental.

Tabla 28: Caso de uso expandido "Configurar Sonido Ambiental"

Caso de uso	Eliminar sonido ambiental	
Actores	Usuario	
Resumen	El CU se inicia cuando el usuario decide eliminar un sonido ambiental.	
Referencias	R12	
Precondiciones	-	
Flujo normal de eventos		
Acción del actor		**Respuesta del sistema**
1. El usuario se dirige al menú principal y selecciona el submenú *View*.		1.1. El sistema muestra varias opciones entre las que se encuentra *Environmental Sounds Window*.
2. El usuario escoge la opción *Environmental Sounds Window*.		2.1. El sistema muestra una ventana con todos los sonidos ambientales existentes.
3. El usuario selecciona el sonido ambiental que desea eliminar.		3.1 El sistema habilita la opción Remove Environmental Sound.
4. El usuario se dirige al menú principal y selecciona el submenú *Sound*.		4.1. El sistema muestra varias opciones entre las que se encuentra Remove *Environmental Sound*.
5. El usuario escoge la opción Remove *Environmental Sound*.		5.1. El sistema elimina el sonido ambiental seleccionado.
Puntos de extensión	-	
Poscondiciones	Es eliminado un sonido ambiental.	

Tabla 29: Caso de uso expandido "Eliminar Sonido Ambiental"

Caso de uso	Configurar propiedades globales de los sonidos.
Actores	Usuario
Resumen	El CU se inicia cuando el usuario decide configurar las propiedades generales de los sonidos del entorno virtual.
Referencias	R13
Precondiciones	-
Flujo normal de eventos	

Acción del actor	Respuesta del sistema
1. El usuario se dirige al menú principal y selecciona el submenú *View*.	1.1. El sistema muestra varias opciones entre las que se encuentra *Global Properties Window*.
2. El usuario escoge la opción *Global Properties Window*.	2.1. El sistema muestra una ventana con las propiedades globales de los sonidos.
3. El usuario configura las propiedades globales de los sonidos del entorno virtual.	3.1. El sistema verifica que el(los) dato(s) de la(s) propiedad(es) que se desea(n) configurar sea(n) correcto(s). 3.2. Si el(los) dato(s) es(son) correcto(s) el sistema modifica la(s) propiedad(es) global(es) de los sonidos del entorno virtual.
Flujo alterno 1	
Acción del actor	**Respuesta del sistema**
	3.1. Si el(los) dato(s) no es (son) válido(s) el sistema restablece los valores de las propiedades globales de los sonidos del entorno virtual que se mostraban antes del intento de configuración.
Puntos de extensión	-
Poscondiciones	Son modificadas las propiedades generales de los sonidos del entorno virtual.

Tabla 30: Caso de uso expandido "Configurar Propiedades Globales de los Sonidos"

Caso de uso	Deshacer un cambio
Actores	Usuario
Resumen	El CU se inicia cuando el usuario decide deshacer un cambio realizado.
Referencias	R14
Precondiciones	Tiene que haberse realizado algún cambio en el proyecto actual.
Flujo normal de eventos	
Acción del actor	**Respuesta del sistema**

1. El usuario escoge la opción deshacer o pulsa Ctrl+z.	1.1. El sistema busca en el historial de cambios realizados el último cambio que se efectúo y deshace dicho cambio. 1.2. El sistema muestra la configuración del proyecto en su estado anterior.
Puntos de extensión	-
Poscondiciones	El proyecto retorna a su estado anterior.

Tabla 31: Caso de uso expandido "Deshacer Cambio"

Caso de uso	Rehacer un cambio
Actores	Usuario
Resumen	El CU se inicia cuando el usuario decide rehacer un cambio deshecho.
Referencias	R15
Precondiciones	La última acción del usuario tiene que haber sido deshacer un cambio.
Flujo normal de eventos	
Acción del actor	**Respuesta del sistema**
1. El usuario escoge la opción rehacer o pulsa Ctrl+y.	1.1. El sistema busca en el historial de cambios desechos el último cambio desecho y rehace dicho cambio. 1.2. El sistema muestra la configuración del proyecto en su estado posterior.
Puntos de extensión	-
Poscondiciones	El proyecto retorna a su estado posterior.

Tabla 32: Caso de uso expandido "Rehacer Cambio"

Caso de uso	Generar Fichero de Configuración de los Sonidos
Actores	Usuario
Resumen	El CU se inicia cuando el usuario decide salvar la configuración que tienen los sonidos hasta ese momento en el proyecto actual.
Referencias	R16
Precondiciones	-

Flujo normal de eventos	
Acción del actor	**Respuesta del sistema**
1. El usuario se dirige al menú principal y selecciona el submenú *File*.	1.1. El sistema muestra varias opciones entre las que se encuentra *Export*.
2. El usuario escoge la opción *Export*.	2.1. El sistema muestra un cuadro de diálogo pre-configurado que permite especificar la ruta donde se va a salvar la configuración de los sonidos y especificar el nombre que tendrá el archivo, o cancelar la operación.
3. El usuario especifica la ruta donde desea salvar la configuración de los sonidos y el nombre que tendrá el archivo y pulsa abrir.	3.1. El sistema cierra el cuadro de diálogo. 3.2. El sistema verifica que la ruta sea válida. 3.3. El sistema crea un archivo en la ruta especificada y salva la configuración de los sonidos en dicho archivo.
Flujo alterno 1	
Acción del actor	**Respuesta del sistema**
3. El usuario decide cancelar la operación de generar el fichero de configuración de los sonidos.	3.1. El sistema cierra el cuadro de diálogo.
Flujo alterno 2	
	3.2. Si la ruta no es válida el sistema muestra un mensaje de error.
Puntos de extensión	-
Poscondiciones	Se guarda la configuración de los sonidos hasta ese momento en el proyecto actual.

Tabla 33: Caso de uso expandido "Generar Fichero de Configuración de los Sonidos"

Conclusiones

En este capítulo se definieron las cualidades y características que espera el usuario que tenga el sistema. Para ello se definieron los requisitos funcionales y no funcionales. Además se describieron los casos de uso que dan cumplimiento a lo que el usuario desea que haga el sistema.

Capítulo 4: Diseño del sistema

Introducción

En este capítulo se presentan los diagramas de clases del diseño, constituyendo estos unos de los pasos más importantes dentro del desarrollo del sistema, ya que muestran una vista de los componentes estáticos del sistema (clases), indicando las relaciones entre estos y los atributos (datos) de las clases, así como sus métodos (código). Además se muestran los diagramas de secuencia de la realización de los casos de uso. Por último se especifican los patrones de diseño a utilizar en el desarrollo del sistema.

4.1. Diagramas de clases del diseño

4.1.1. Diagramas de clases del diseño por paquetes

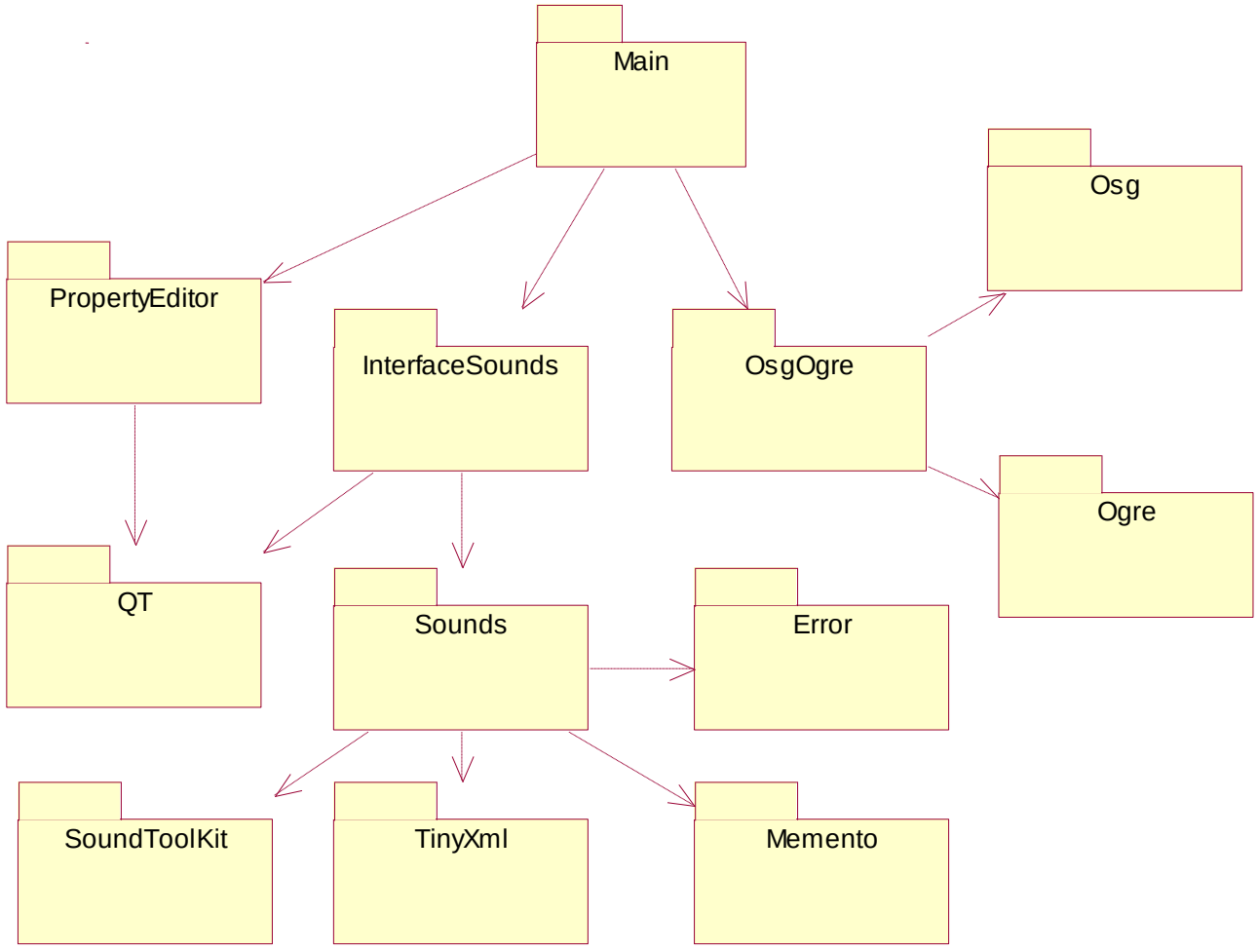

Figura 3: Diagrama de clases del diseño por paquetes

4.1.2. Paquete "Error"

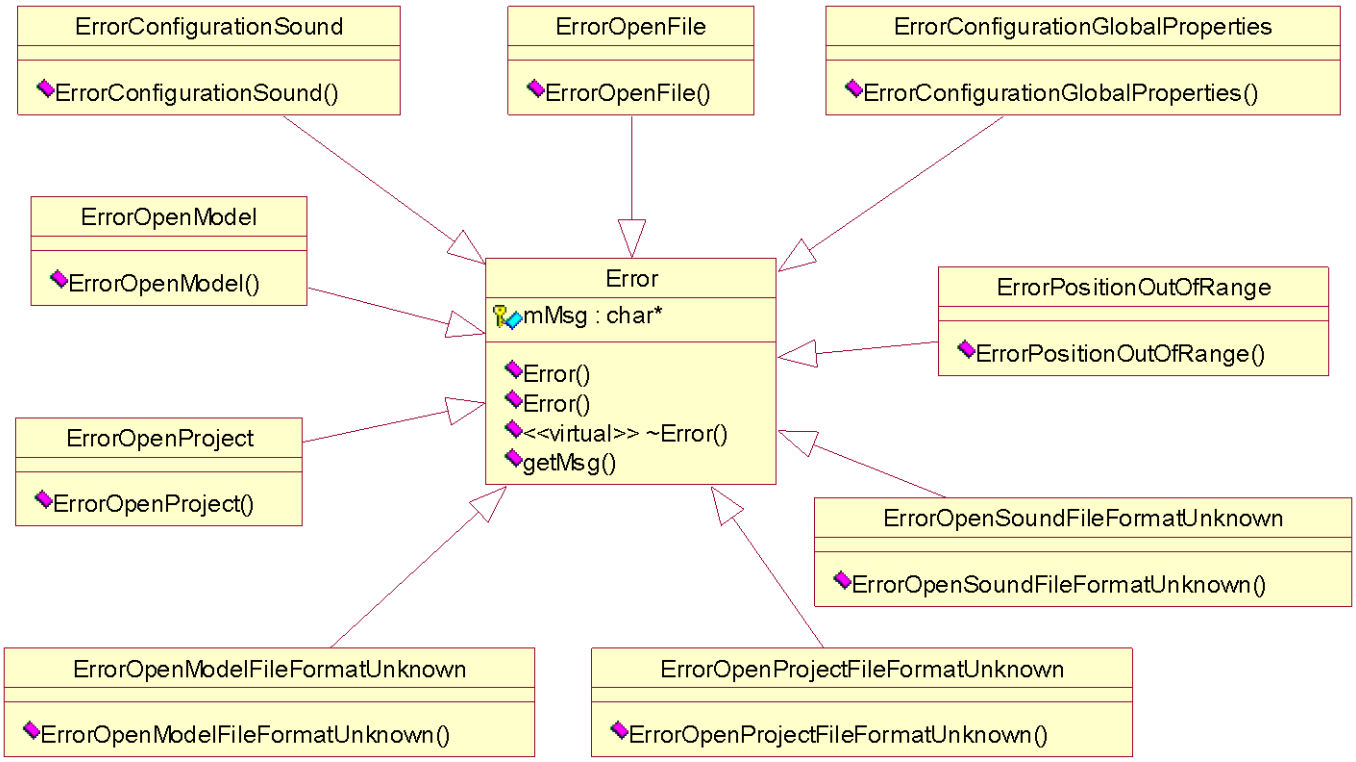

Figura 4: Diagramas de clases de diseño, paquete "Error"

4.1.3. Paquete "Osg"

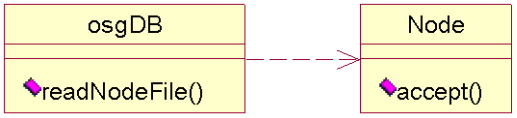

Figura 5: Diagramas de clases de diseño, paquete "Osg"

4.1.4. Paquete "OsgOgre"

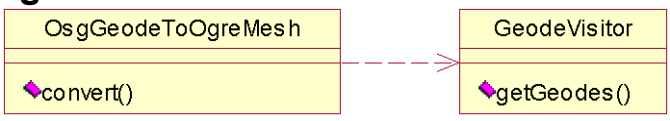

Figura 6: Diagramas de clases de diseño, paquete "OsgOgre"

4.1.5. Paquete “Memento”

4.1.5.1. Relaciones del paquete “Memento”

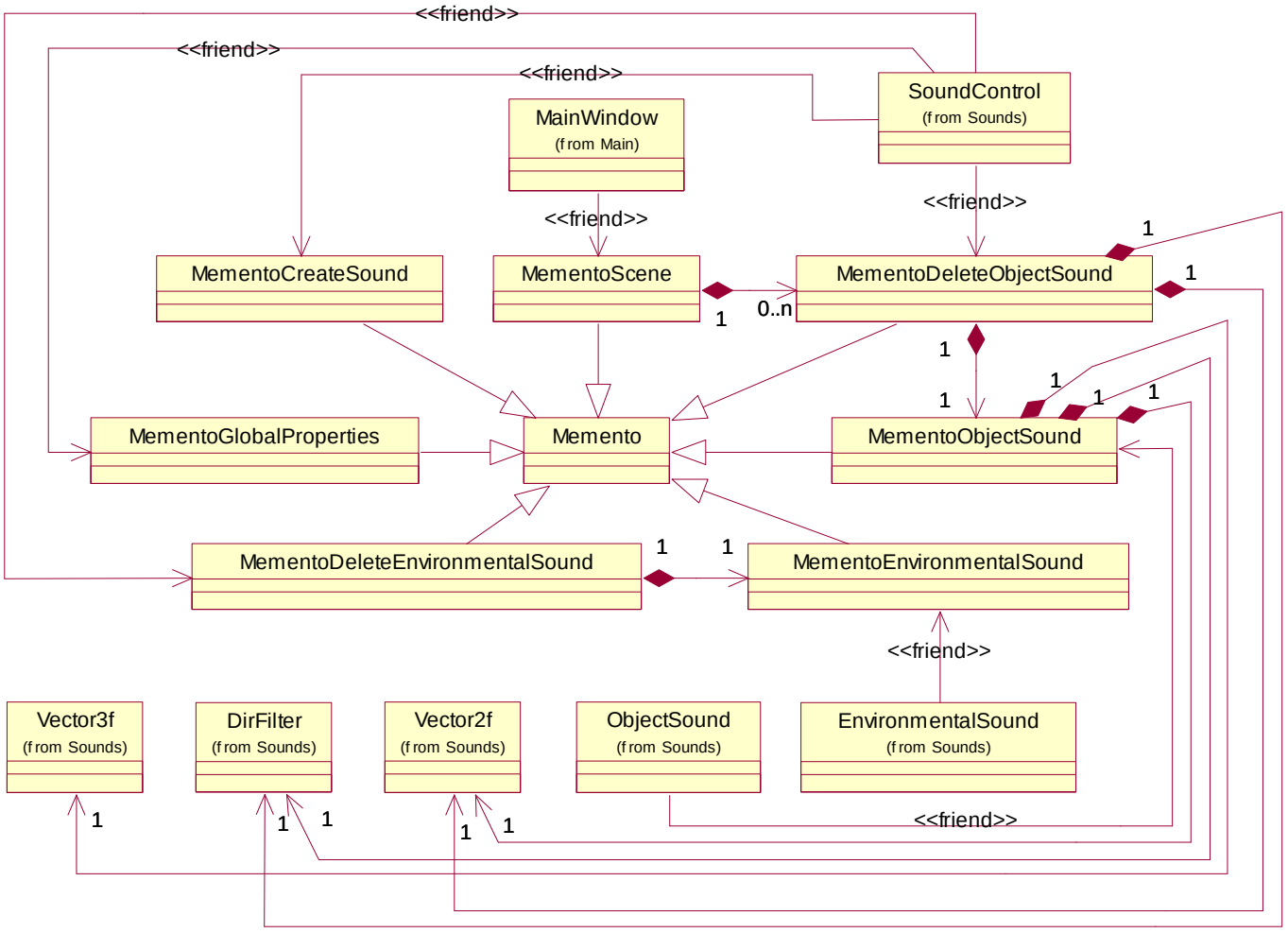

Figura 7: Relaciones del paquete “Memento”

4.1.5.2. Clases del paquete “Memento”

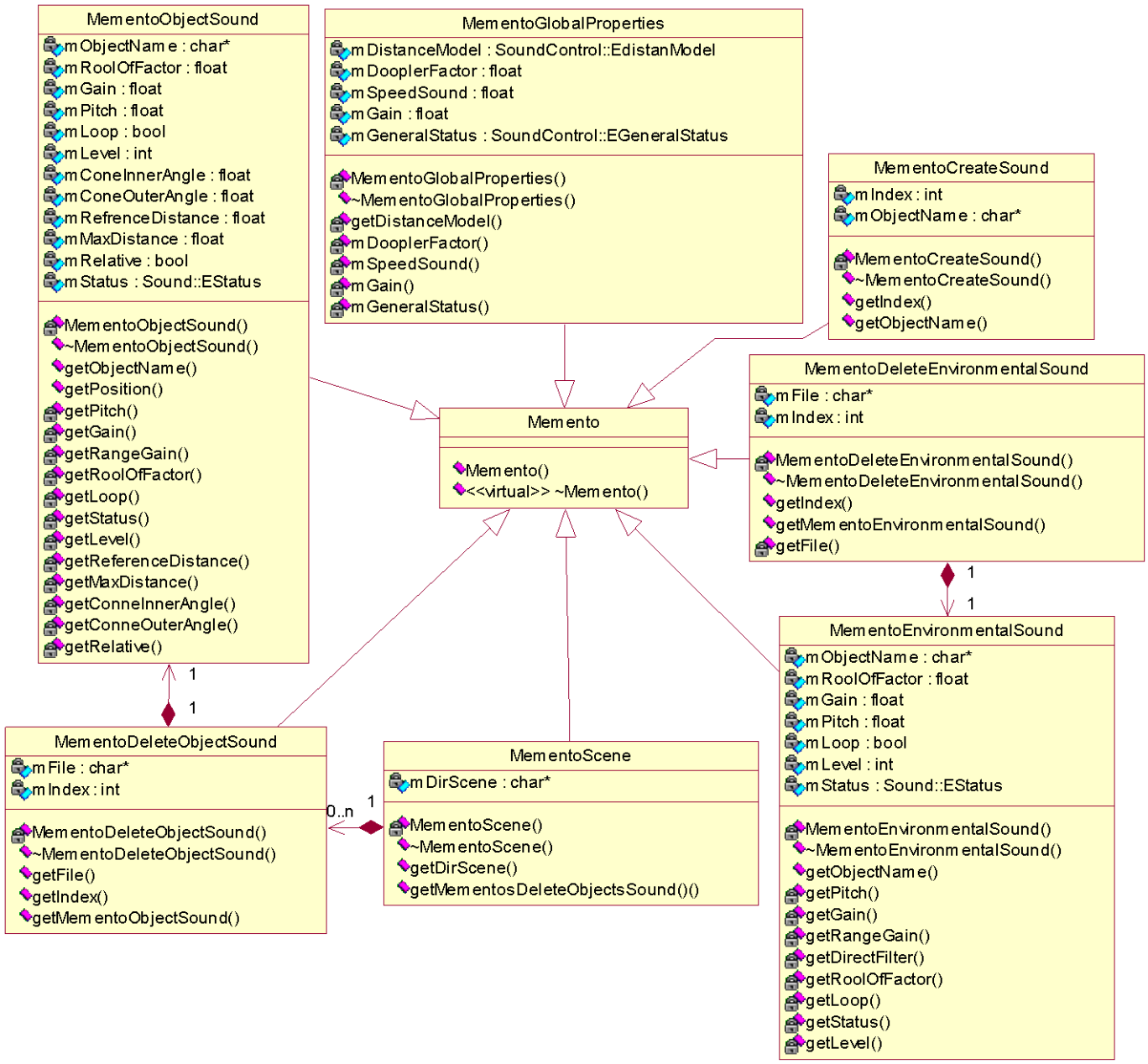

Figura 8: Diagramas de clases de diseño, paquete “Memento”

4.1.6. Paquete “Sounds”

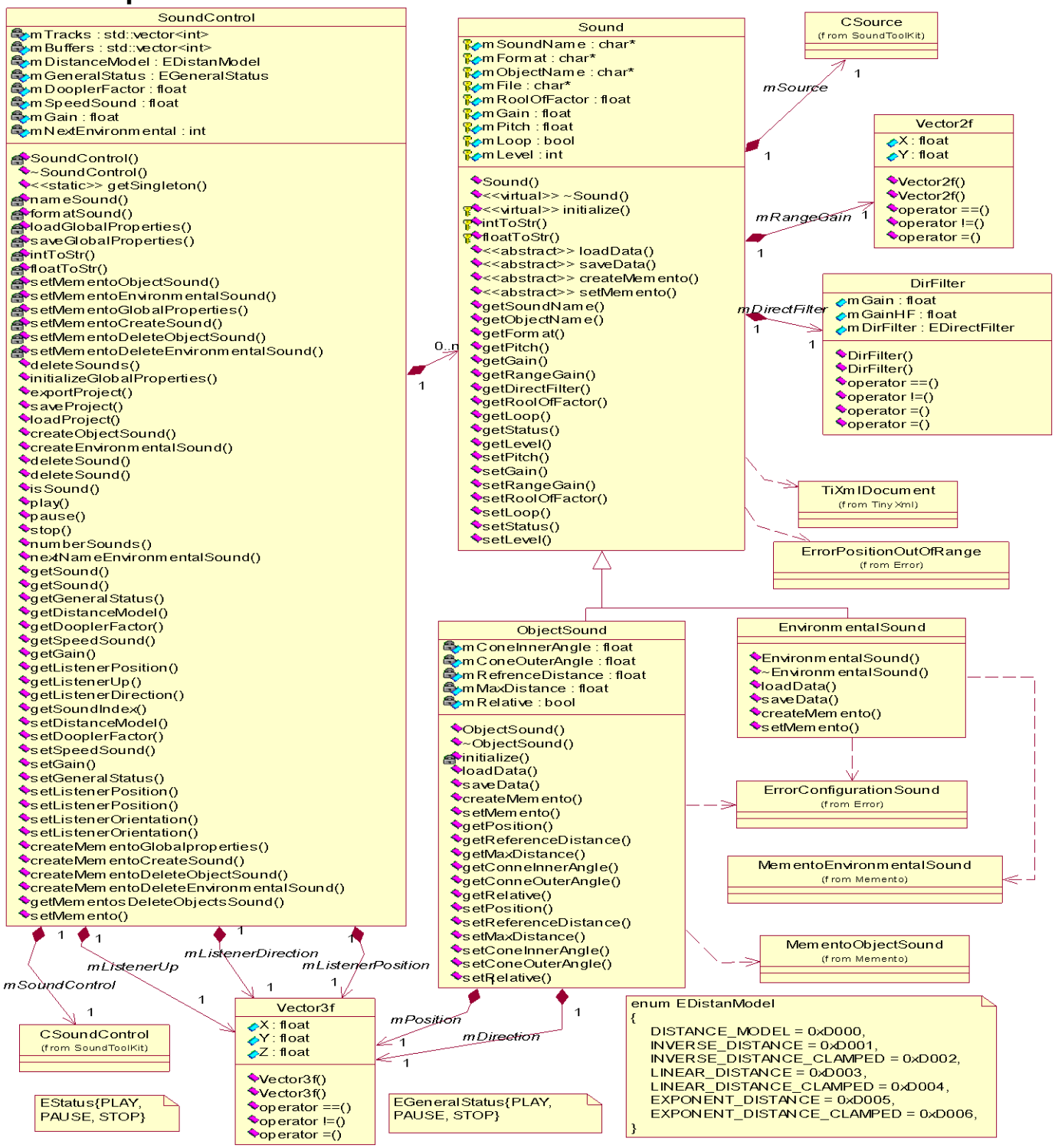

Figura 9: Diagramas de clases de diseño, paquete “Sounds”

4.1.7. Paquete “InterfaceSounds”

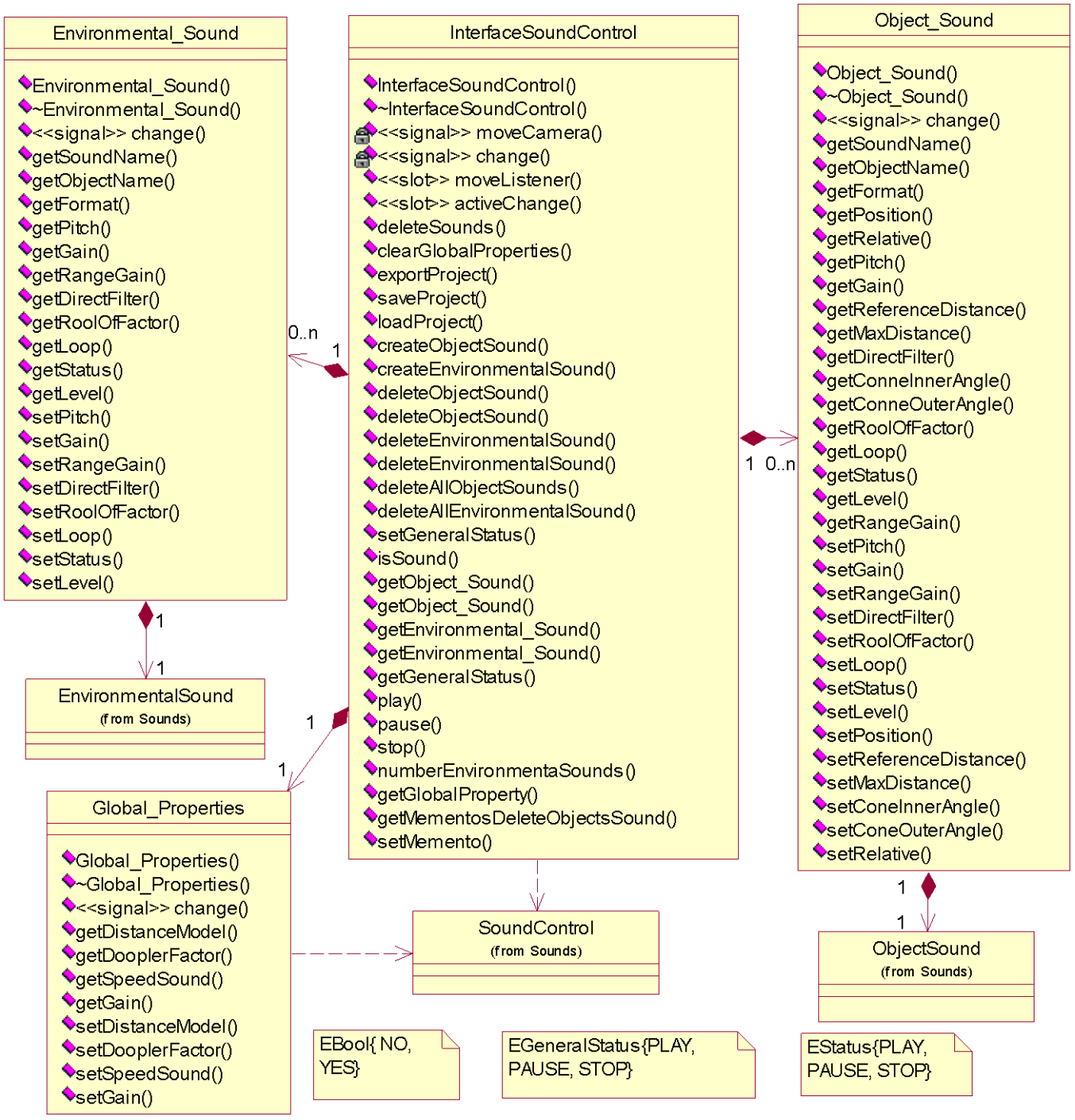

Figura 10: Diagramas de clases de diseño, paquete “InterfaceSounds”

4.1.8. Paquete “Main”

4.1.8.1. Clase “OgreView” y sus relaciones

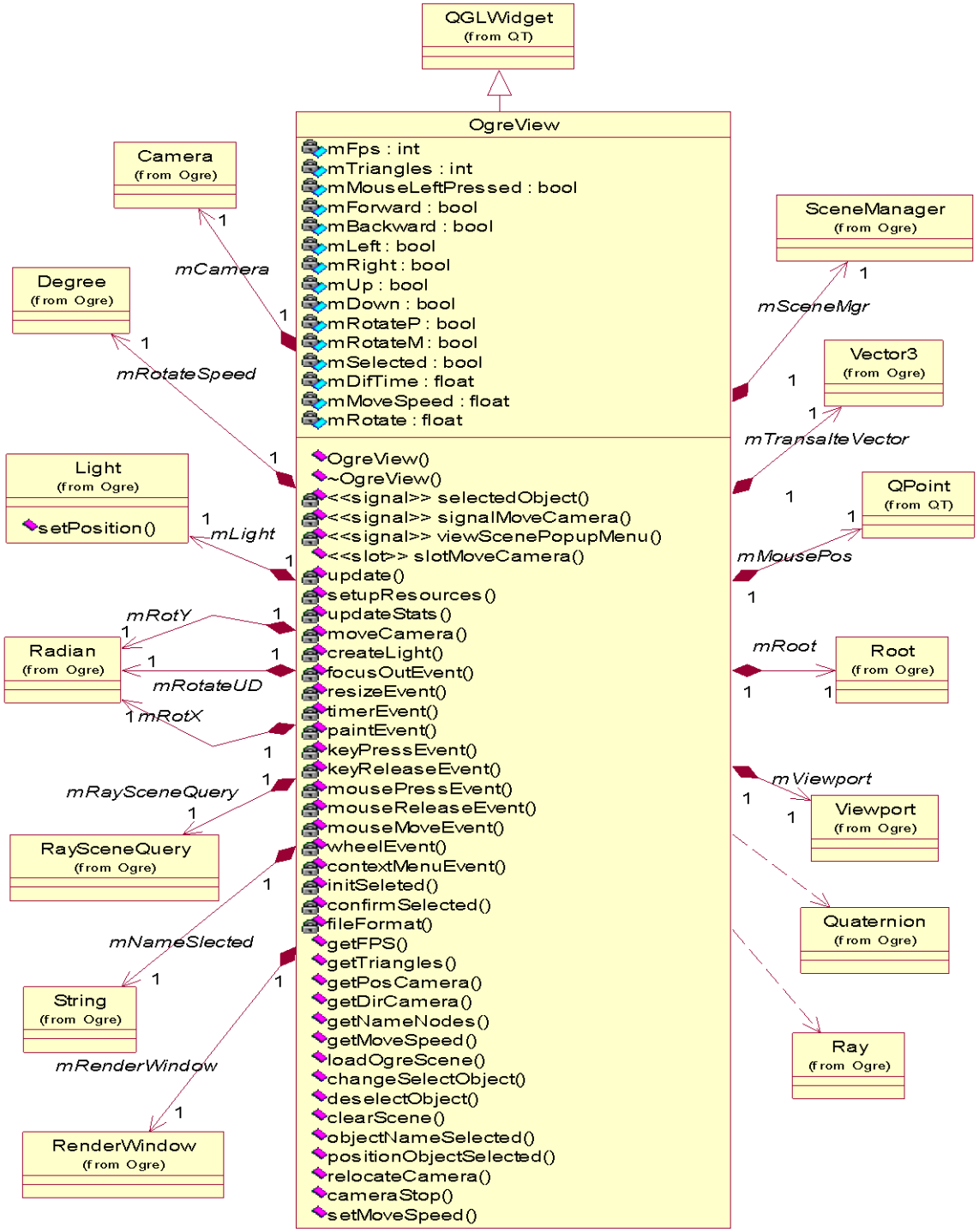

Figura 11: Clase “OgreView” y sus relaciones

4.1.8.2. Clase “MainWindow” y sus relaciones

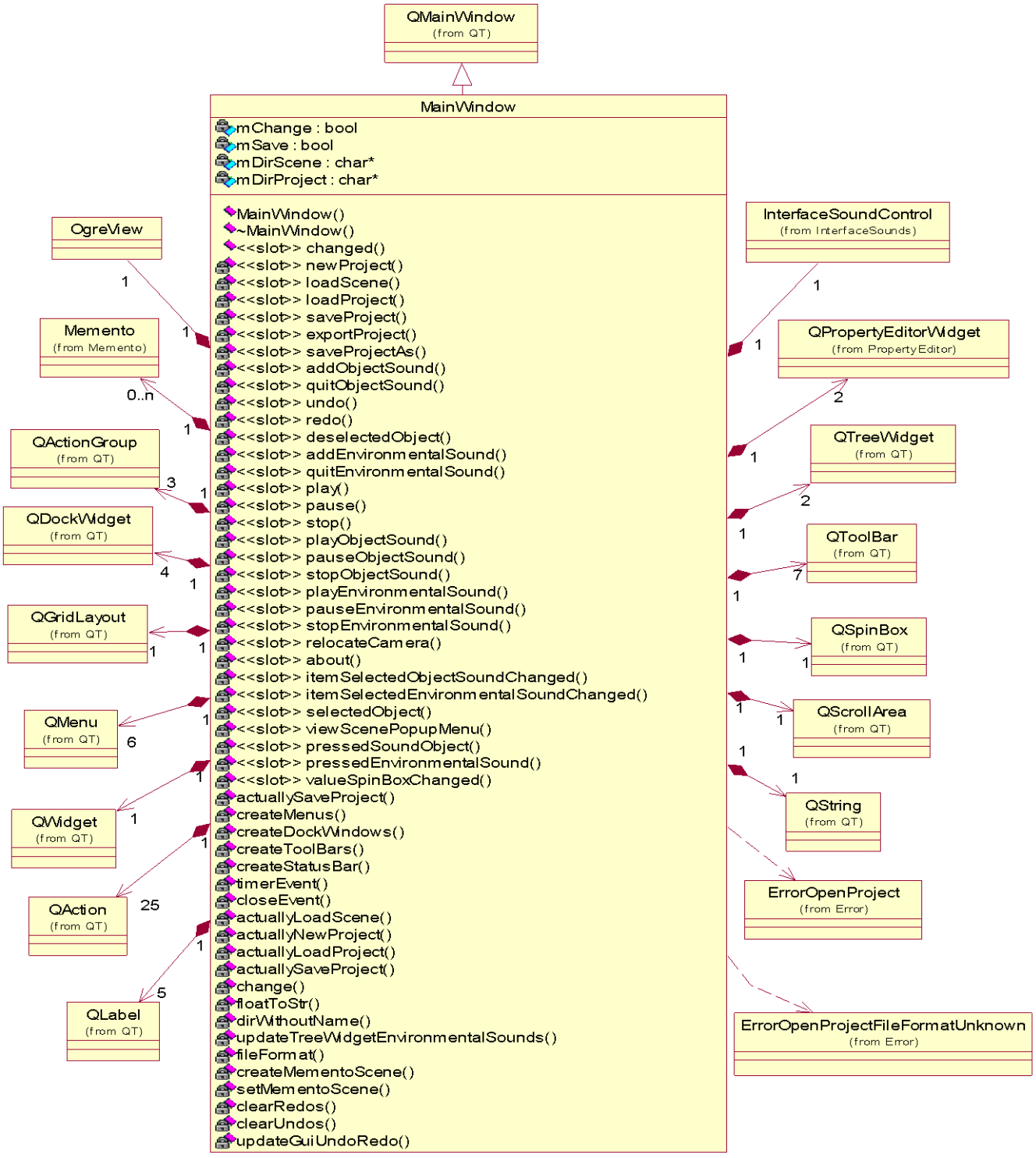

Figura 12: Clase “MainWindow” y sus relaciones

4.1.9. Paquete “PropertyEditor”

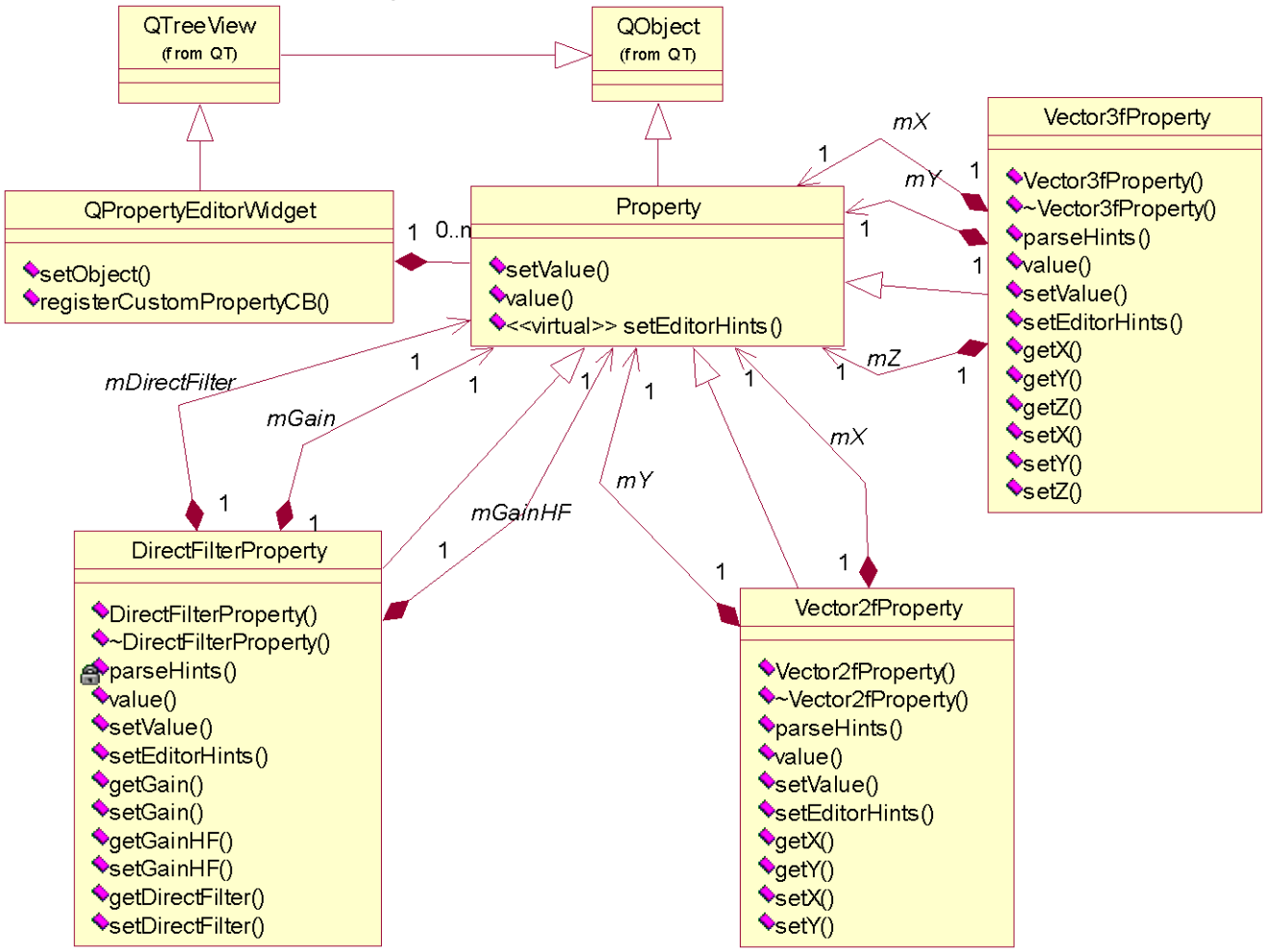

Figura 13: Diagramas de clases de diseño, paquete “PropertyEditor”

4.1.10. Paquete “SoundToolkit”

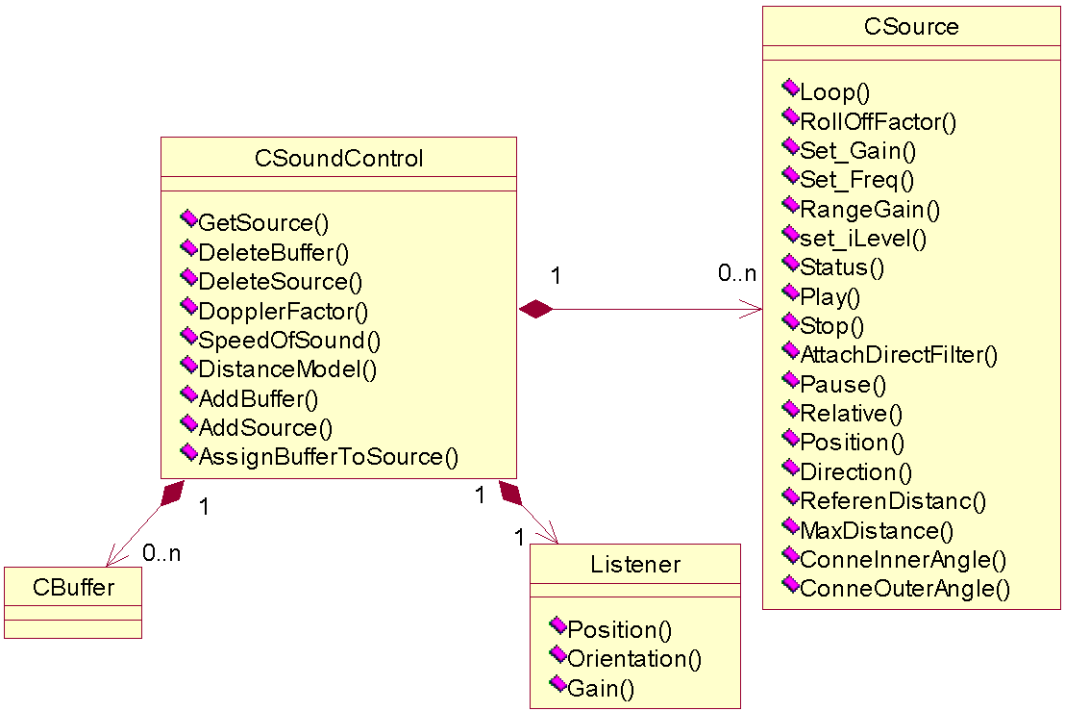

Figura 14: Diagramas de clases de diseño, paquete “SoundToolkit”

4.1.11. Paquete “TinyXml”

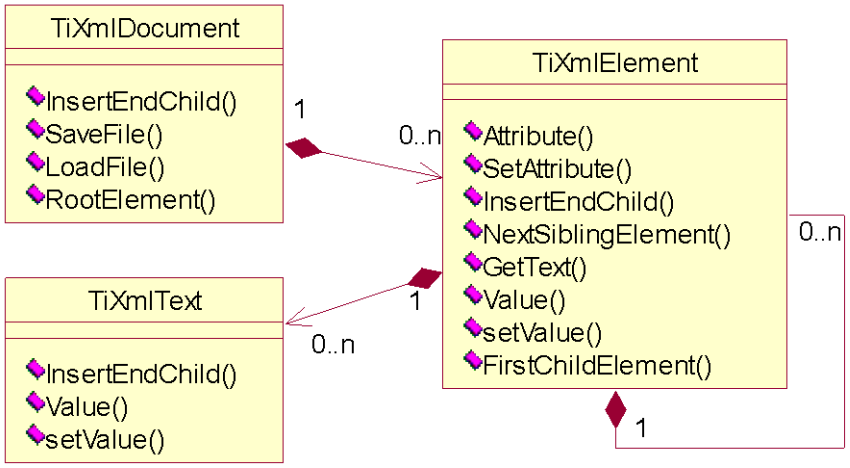

Figura 15: Diagramas de clases de diseño, paquete “TinyXml”

4.1.12. Paquete “Qt”

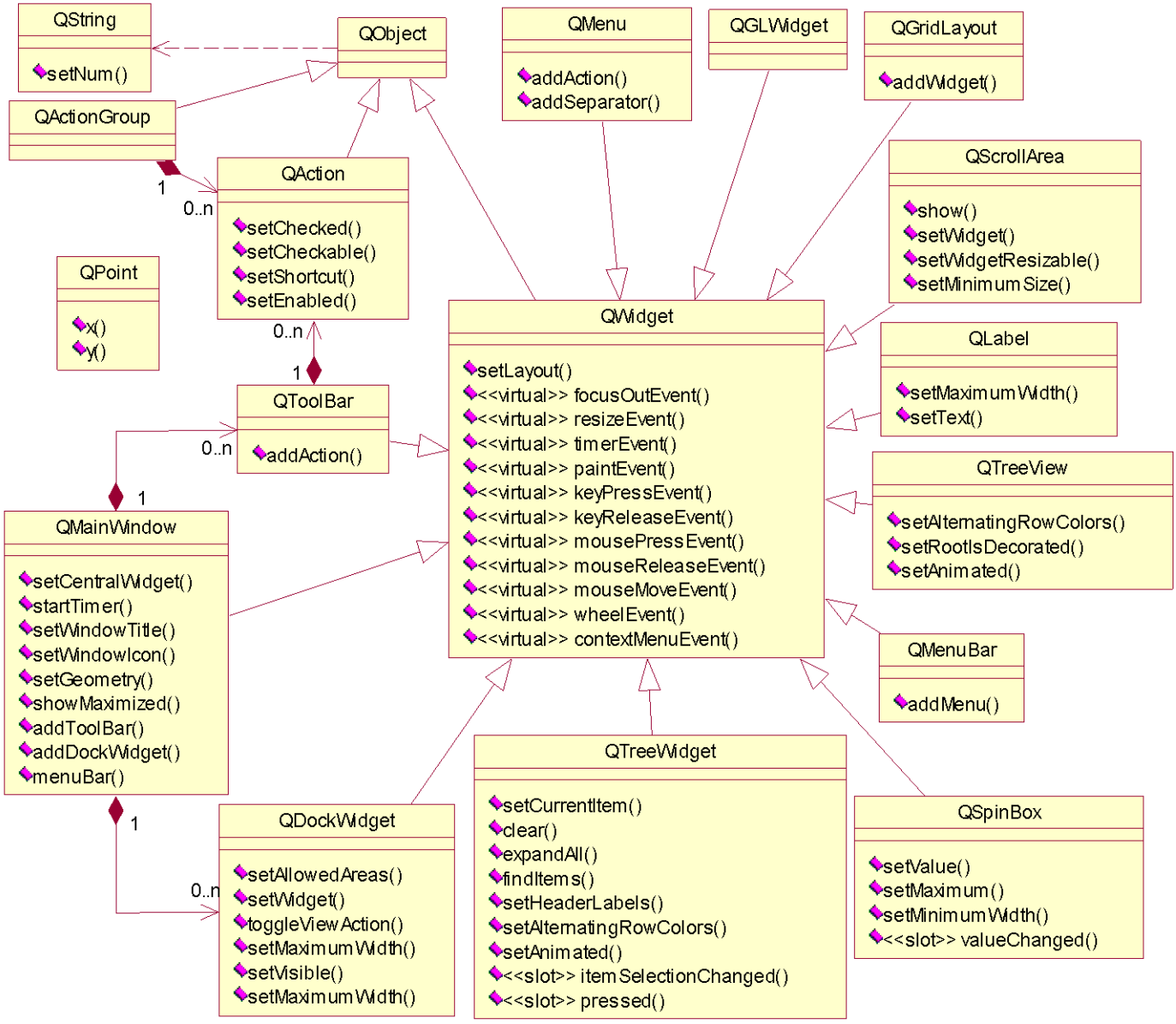

Figura 16: Diagramas de clases de diseño, paquete “Qt”

4.1.13. Paquete “Ogre”

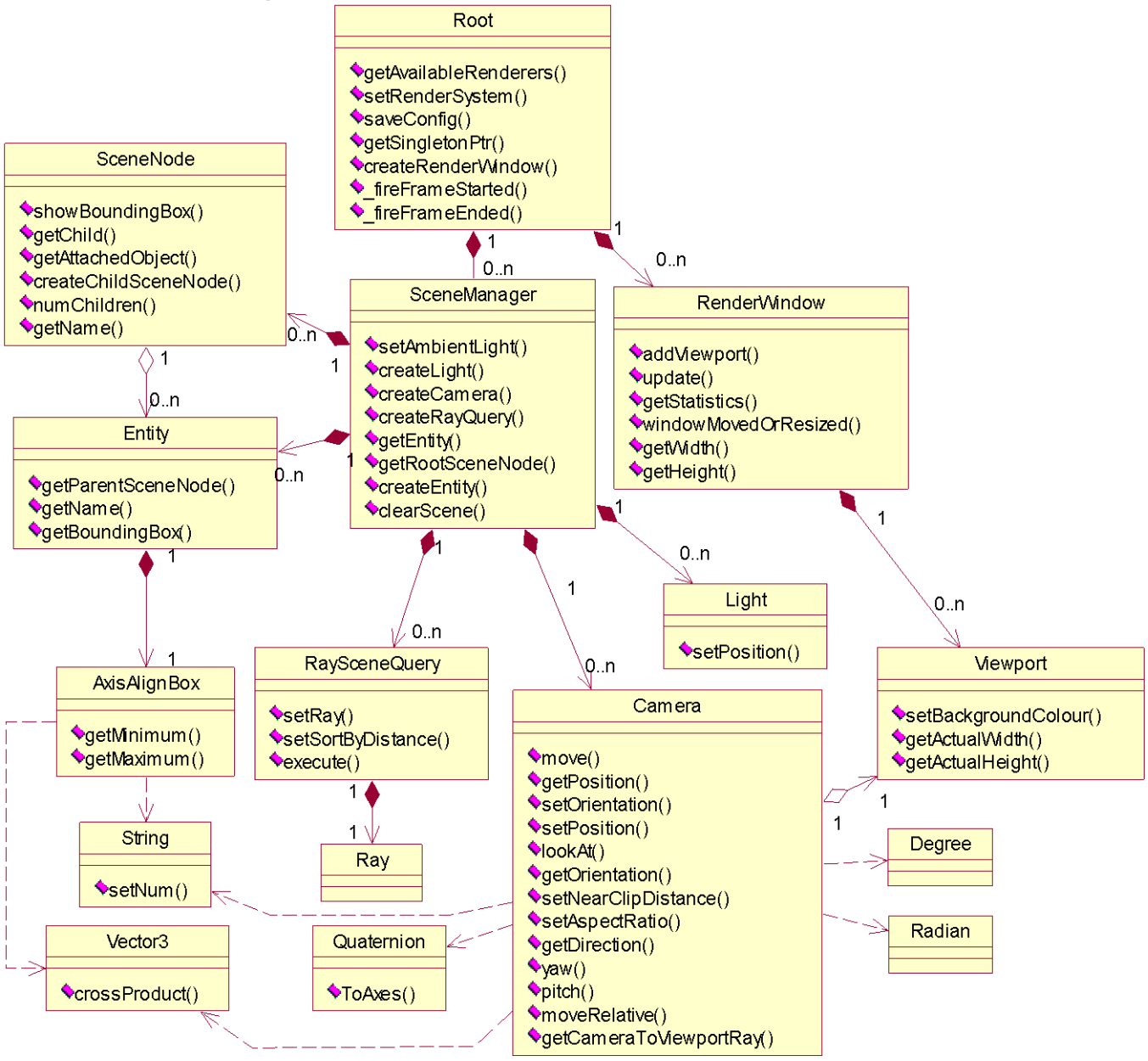

Figura 17: Diagramas de clases de diseño, paquete “Ogre”

4.2. Diagramas de secuencia

4.2.1. Diagrama de secuencia “Crear proyecto”

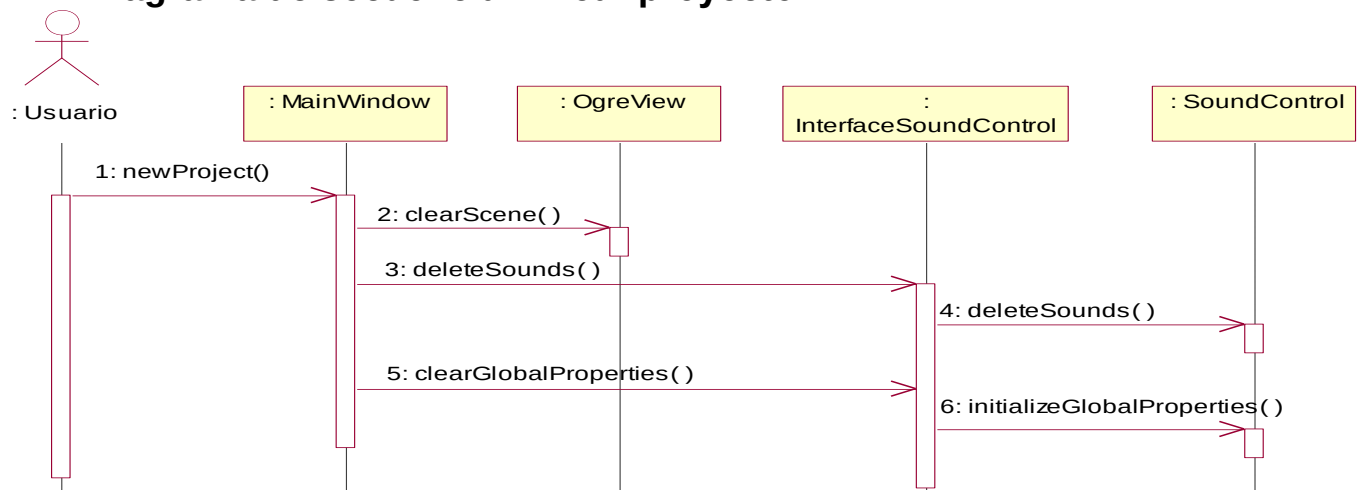

Figura 18: Diagrama de secuencia “Crear proyecto”

4.2.2. Diagrama de secuencia “Cargar proyecto”

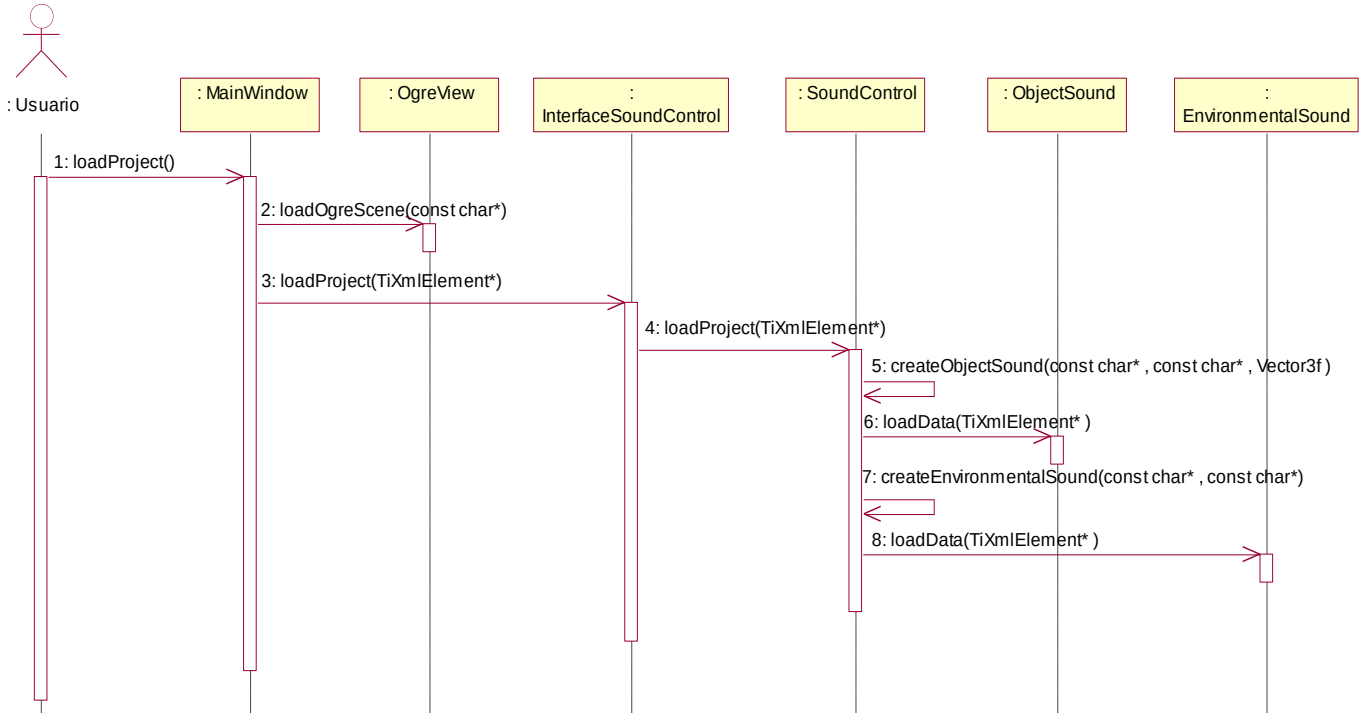

Figura 19: Diagrama de secuencia “Cargar proyecto”

4.2.3. Diagrama de secuencia “Salvar proyecto”

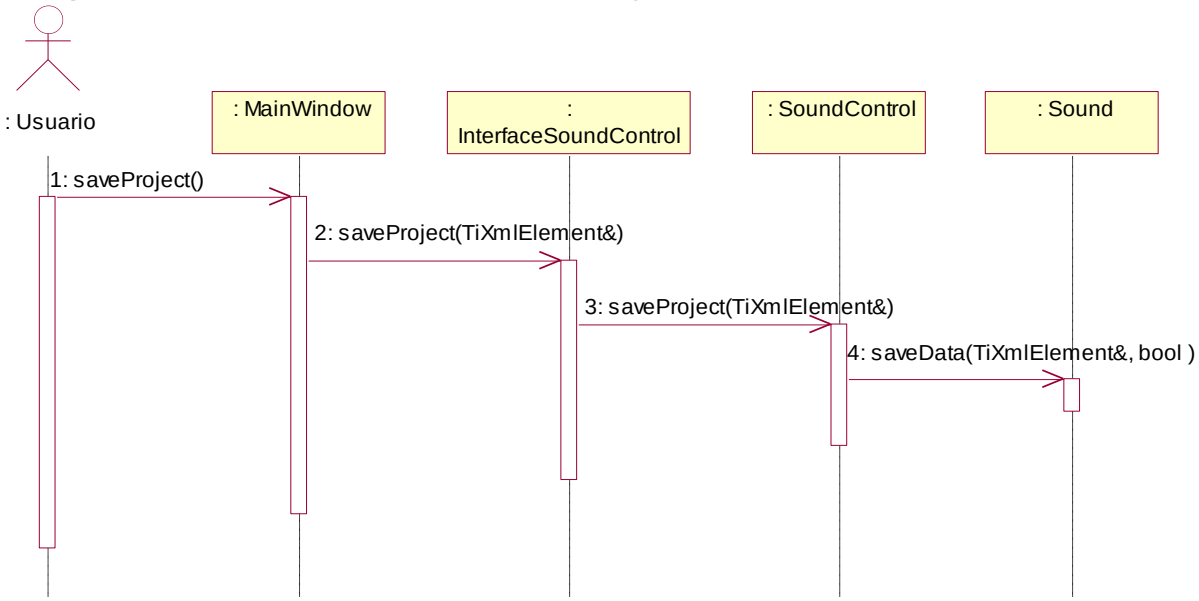

Figura 20: Diagrama de secuencia “Salvar proyecto”

4.2.4. Diagrama de secuencia “Cargar modelo 3D”

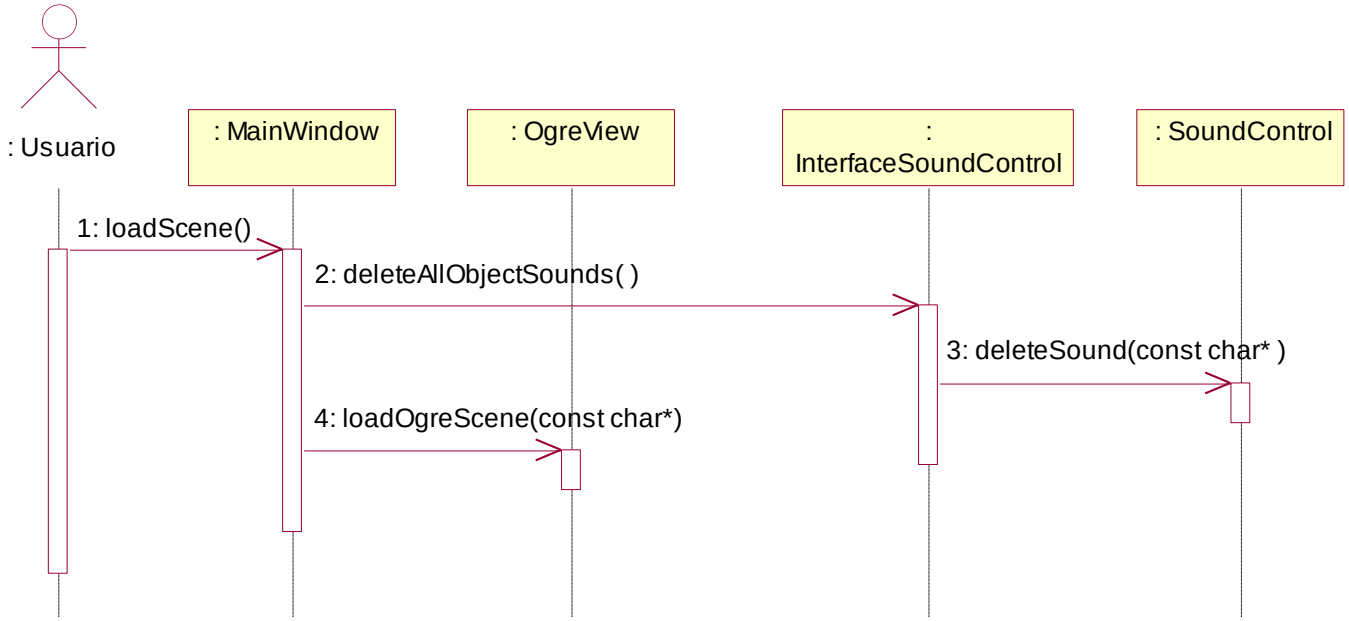

Figura 21: Diagrama de secuencia “Cargar modelo 3D”

4.2.5. Diagrama de secuencia “Seleccionar objeto”

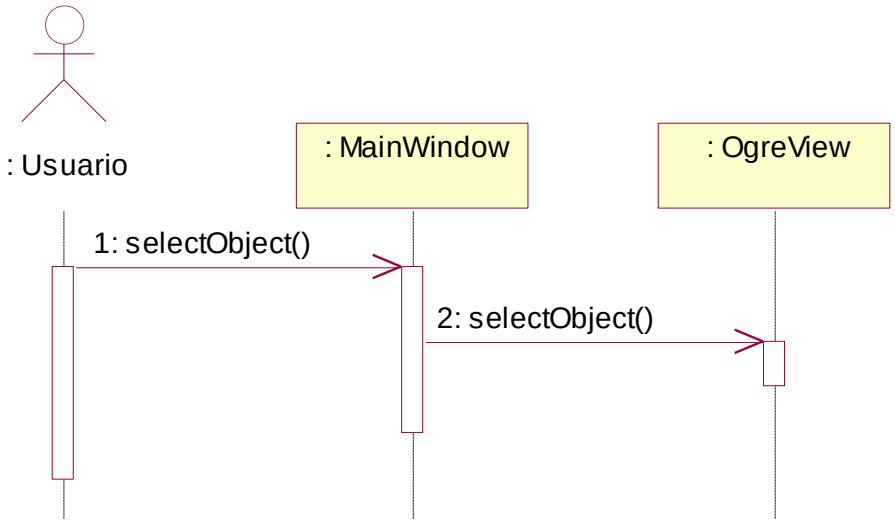

Figura 22: Diagrama de secuencia “Seleccionar objeto”

4.2.6. Diagrama de secuencia “Deseleccionar objeto”

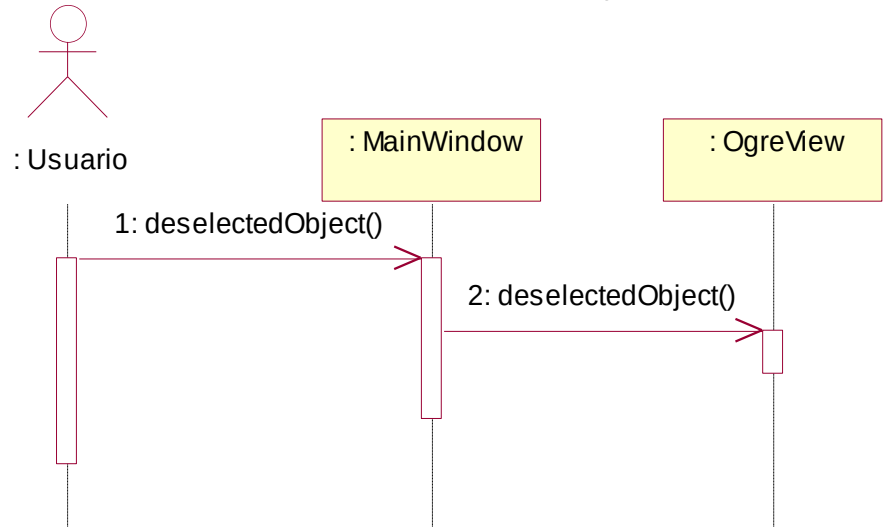

Figura 23: Diagrama de secuencia “Deseleccionar objeto”

4.2.7. Diagrama de secuencia “Aplicar sonido a objeto”

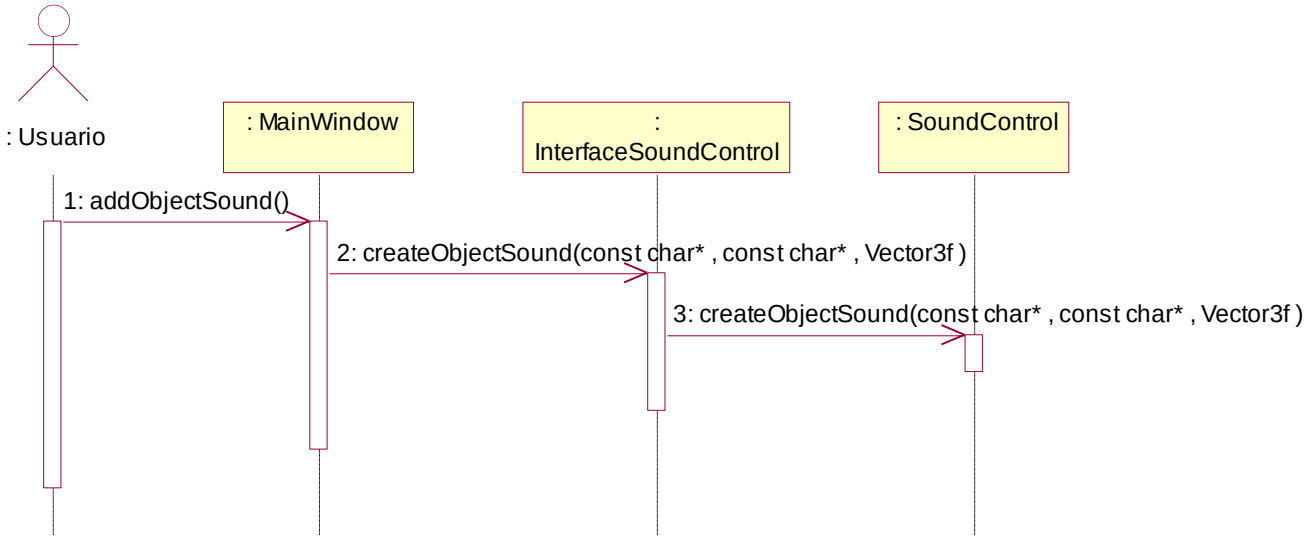

Figura 24: Diagrama de secuencia “Aplicar sonido a objeto”

4.2.8. Diagrama de secuencia “Configurar sonido de objeto”

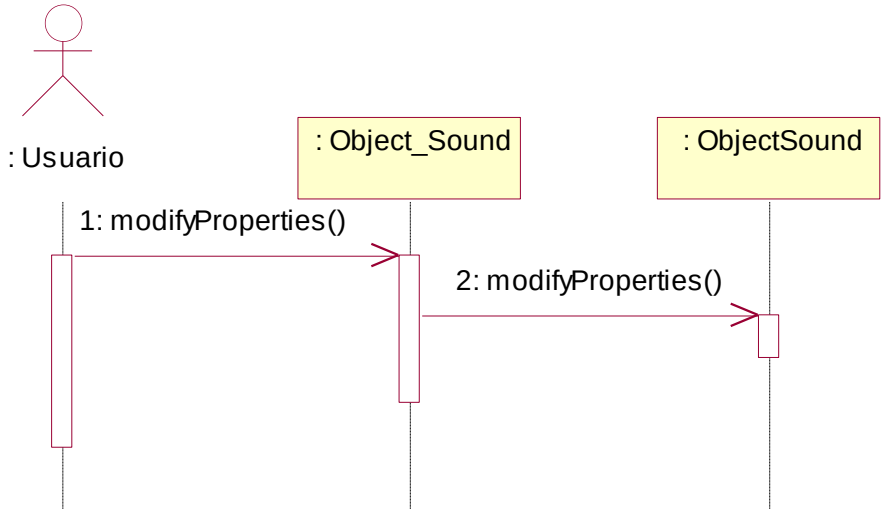

Figura 25: Diagrama de secuencia “Configurar sonido de objeto”

4.2.9. Diagrama de secuencia “Eliminar sonido de objeto”

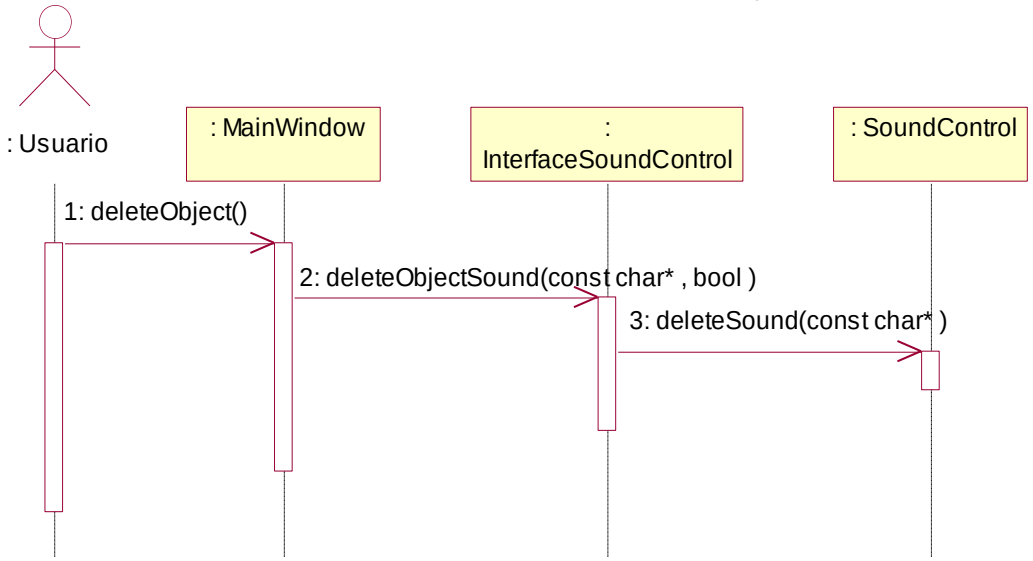

Figura 26: Diagrama de secuencia “Eliminar sonido de objeto”

4.2.10. Diagrama de secuencia “Crear sonido ambiental”

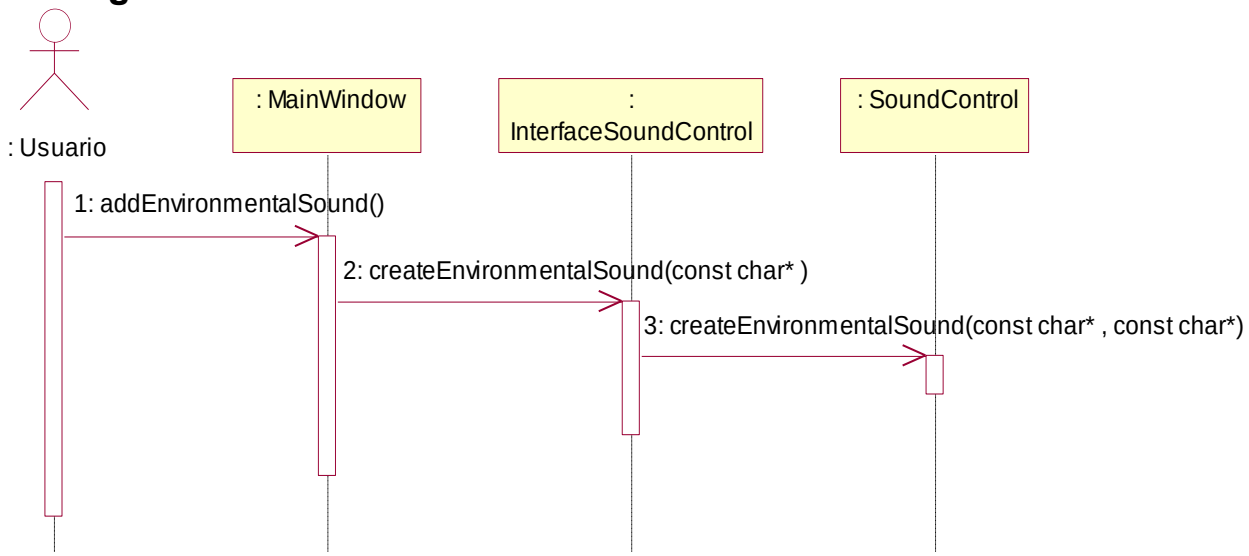

Figura 27: Diagrama de secuencia “Crear sonido ambiental”

4.2.11. Diagrama de secuencia “Configurar sonido ambiental”

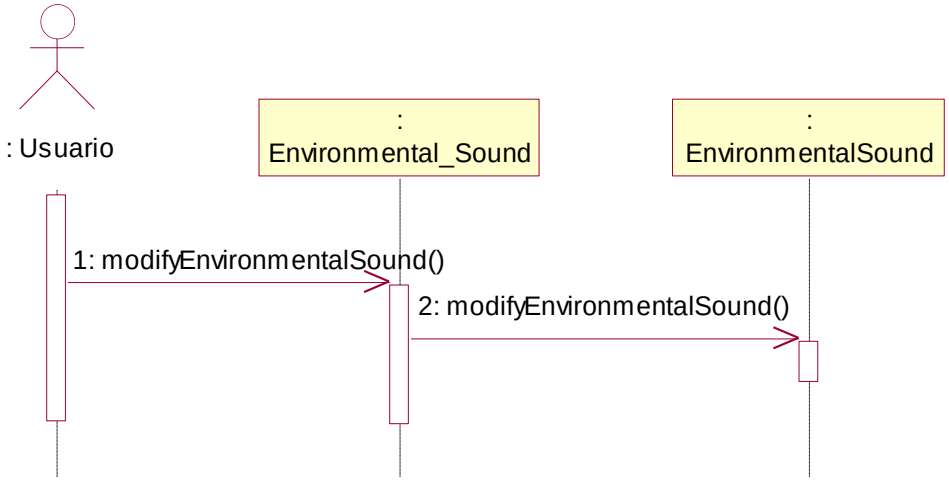

Figura 28: Diagrama de secuencia “Configurar sonido ambiental”

4.2.12. Diagrama de secuencia “Eliminar sonido ambiental”

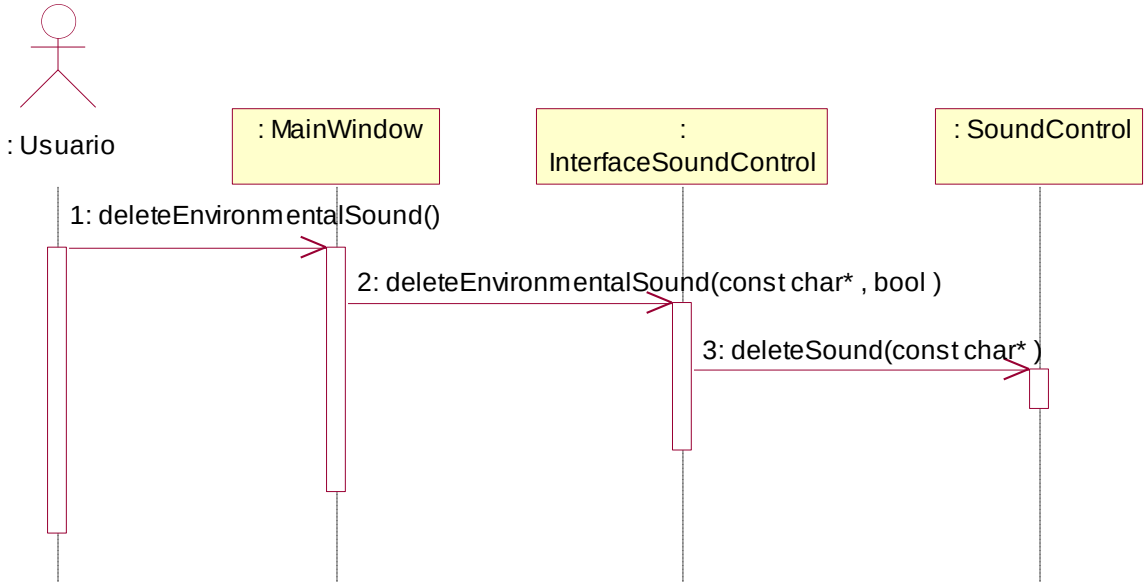

Figura 29: Diagrama de secuencia “Eliminar sonido ambiental”

4.2.13. Diagrama de secuencia “Configurar propiedades globales de los sonidos”

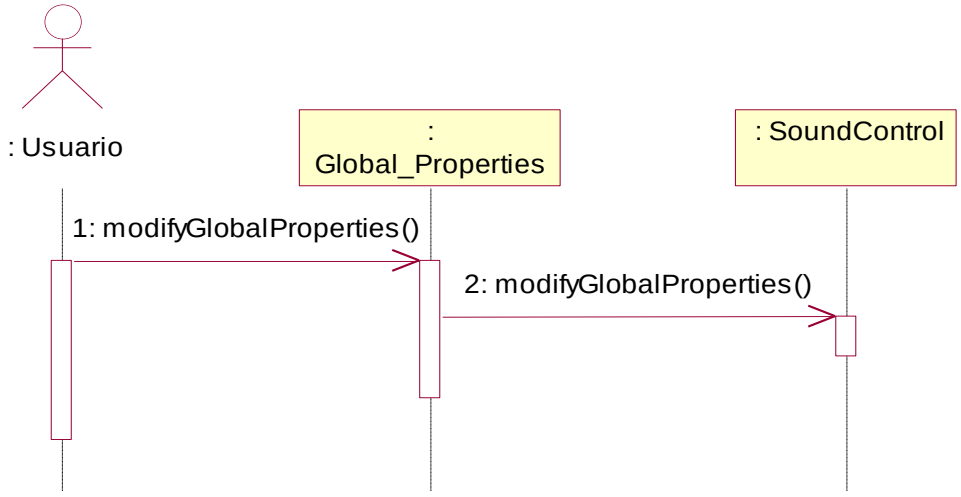

Figura 30: Diagrama de secuencia “Configurar propiedades globales de los sonidos”

4.2.14. Diagrama de secuencia “Deshacer cambio”

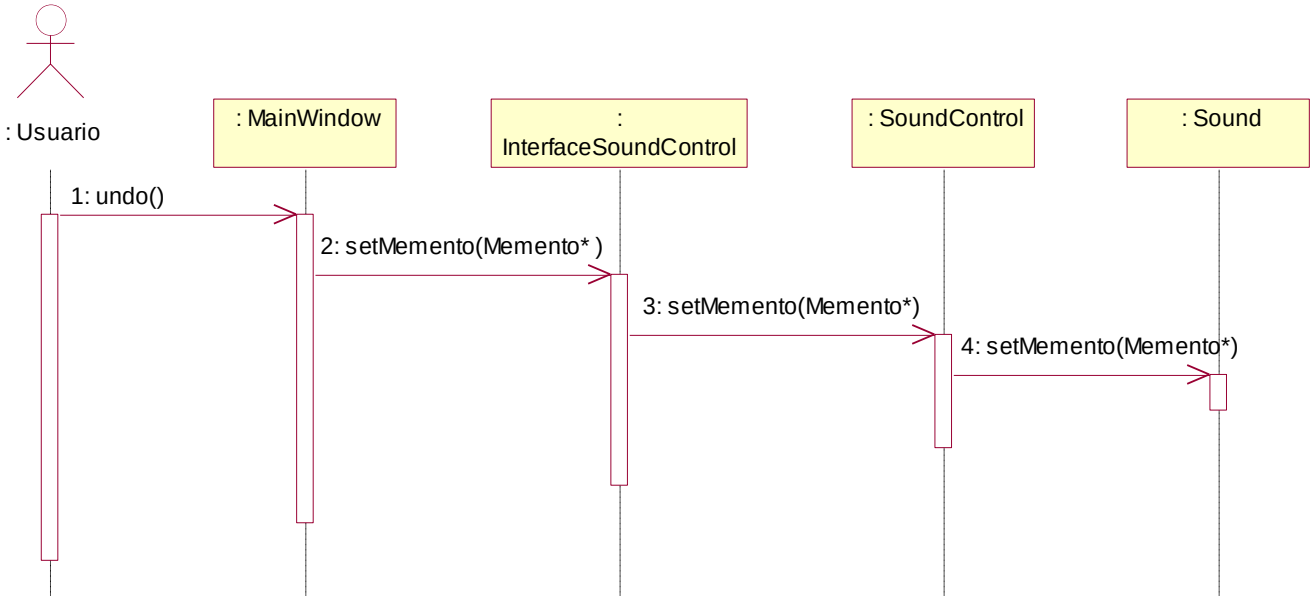

Figura 31: Diagrama de secuencia “Deshacer cambio”

4.2.15. Diagrama de Secuencia “Rehacer cambio”

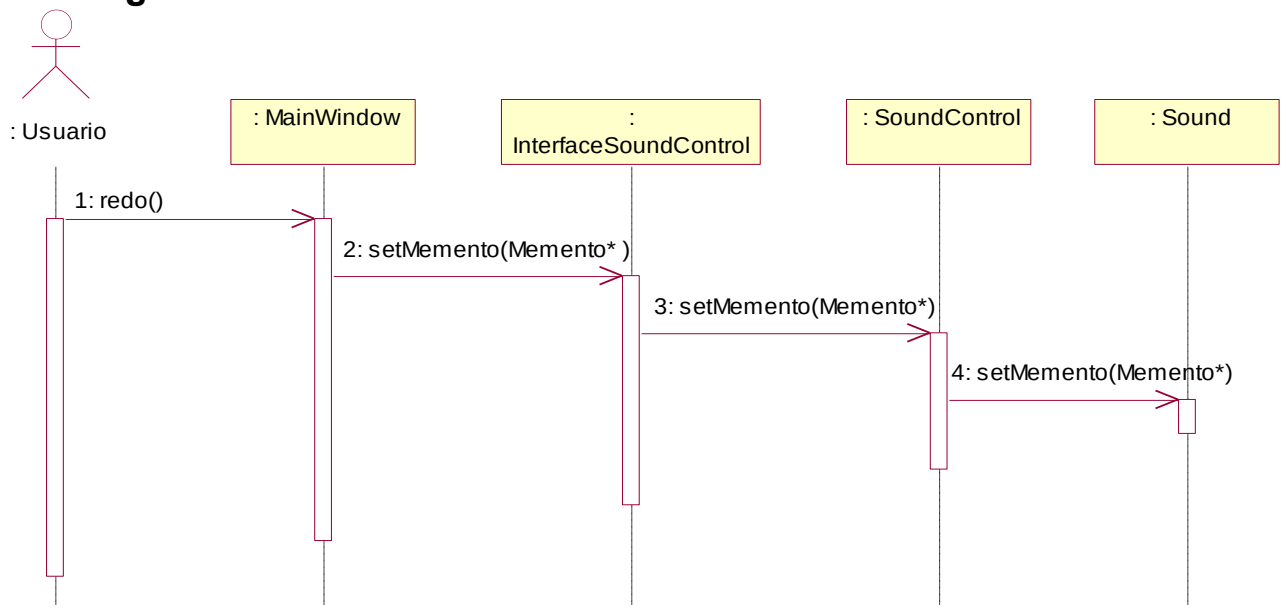

Figura 32: Diagrama de secuencia “Rehacer cambio”

4.2.16. Generar fichero de configuración de los sonidos

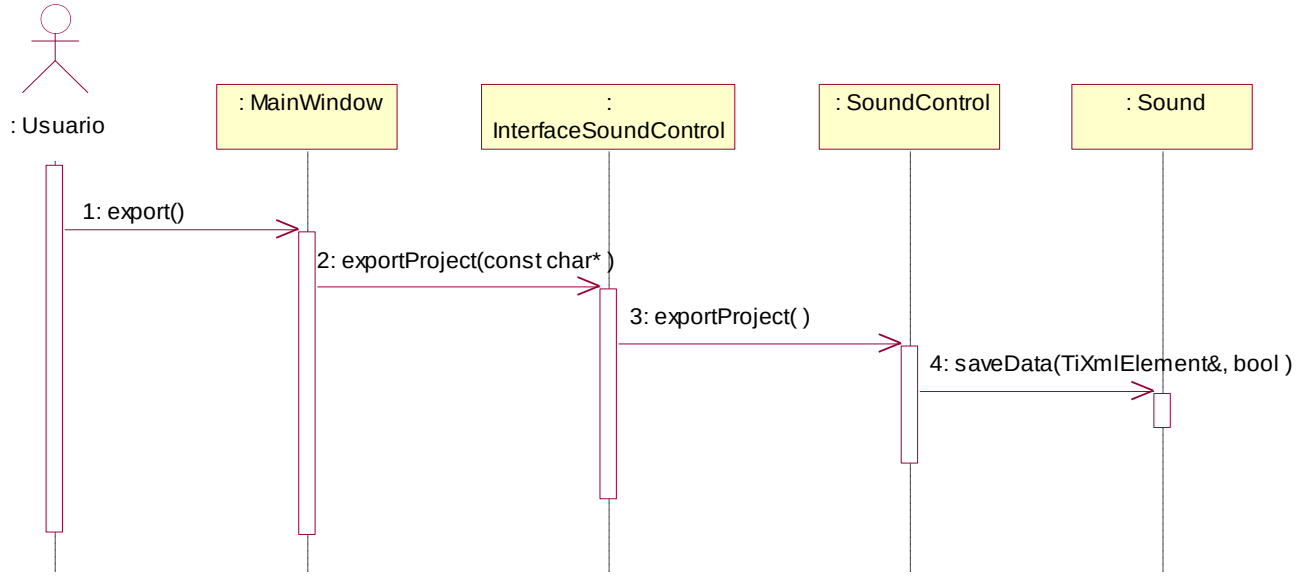

Figura 33: Diagrama de secuencia “Generar fichero de configuración de los sonidos”

4.3. Descripción de las clases del diseño

Nombre:	Sound
Tipo de clase:	Entidad
Atributo	**Tipo**
mObjectName	char*
mFormat	char*
mSoundName	char*
mFile	char*
mRoolOfFactor	float
mGain	float
mPitch	float
mLoop	bool
mRangeGain	Vector2f
mLevel	int
mDirectFilter	DirFilter
mSource	CSource
Para cada responsabilidad	
Nombre	`Sound(CSource* pSource,const char* pSoundName,const char* pFormat,const char* pFile,const char* pObjectName)`
Descripción	Constructor de la clase
Nombre	`~Sound()`
Descripción	Destructor de la clase
Nombre	`loadData(TiXmlElement* element)`
Descripción	Método abstracto que es implementado en las clases hijas, el cual se encargará de modificar los atributos del sonido con los datos obtenidos del objeto pasado por parámetro.
Nombre	`saveData(TiXmlElement& pElement, bool pSaveProject)`

Descripción	Método abstracto que es implementado en las clases hijas, el cual se encargará de salvar las propiedades del sonido en el objeto -`pElement`- pasado por parámetro. Los datos que se salven dependerán de si se está haciendo una salva para guardar el proyecto o exportando la configuración de los sonidos.
Nombre	`createMemento()`
Descripción	Método abstracto que es implementado en las clases hijas, el cual se encargará de crear una salva de los datos de la clase.
Nombre	`setMemento(Memento* pMemento)`
Descripción	Método abstracto que es implementado en las clases hijas, el cual se encargará de modificar los datos de la clase con los datos obtenidos del objeto pasado por parámetro.
Nombre	`initialize()`
Descripción	Método virtual el cual restablece los atributos de la clase a sus valores predeterminados.
Nombre	`intToStr(int pNum)`
Descripción	Devuelve como cadena el número natural pasado por parámetro.
Nombre	`floatToStr(float pNum)`
Descripción	Devuelve como cadena el número real pasado por parámetro.
Nombre	`getSoundName()`
Descripción	Devuelve el valor del atributo mSoundName.
Nombre	`getObjectName()`
Descripción	Devuelve el valor del atributo mObjectName.
Nombre	`getFormat()`
Descripción	Devuelve el valor del atributo mFormat.
Nombre	`getPich()`
Descripción	Devuelve el valor del atributo mPich.
Nombre	`getGain()`
Descripción	Devuelve el valor del atributo mGain.
Nombre	`getDirectFilter()`
Descripción	Devuelve el valor del atributo mDirectFilter.

Nombre	`getRoolOfFactor()`
Descripción	Devuelve el valor del atributo mRoolOfFactor.
Nombre	`getLoop()`
Descripción	Devuelve el valor del atributo mLoop.
Nombre	`getStatus()`
Descripción	Devuelve el estado en el que se encuentra el sonido (PLAY, PAUSE o STOP).
Nombre	`getLevel()`
Descripción	Devuelve el valor del atributo mLevel.
Nombre	`getRangeGain()`
Descripción	Devuelve el valor del atributo mRangeGain.
Nombre	`getFile()`
Descripción	Devuelve el valor del atributo mFile.
Nombre	`setPitch(float pPitch)`
Descripción	Modifica el valor del atributo mPitch
Nombre	`setGain(float pGain)`
Descripción	Modifica el valor del atributo mGain
Nombre	`setDirectFilter(DirFilter pDirectFilter)`
Descripción	Modifica el valor del atributo mDirectFilter
Nombre	`setRoolOfFactor(float pRoolOfFactor)`
Descripción	Modifica el valor del atributo mRoolOfFactor.
Nombre	`setLoop(EBool pLoop)`
Descripción	Modifica el valor del atributo mLoop
Nombre	`setStatus(EStatus pStatus)`
Descripción	Modifica el estado del sonido.
Nombre	`setLevel(int pLevel)`
Descripción	Modifica el valor del atributo mLevel.
Nombre	`setRangeGain(Vector2f pRangeGain)`
Descripción	Modifica el valor del atributo mRangeGain

Tabla 34: Descripción de la clase "Sound"

Nombre:	ObjectSound
Tipo de clase:	Entidad

Atributo	**Tipo**
mConeInnerAngle	float
mConnerOuterAngle	float
mReferenceDistance	float
mMaxDistance	float
mRelative	bool
mPosition	Vector3f
mDirection	Vector3f

Para cada responsabilidad	
Nombre	`ObjectSound(CSource* pSource,const char* pSoundName, const char* pFormat, const char* pFile, const char* pObjectName, Vector3f pPosition)`
Descripción	Constructor de la clase
Nombre	`~ObjectSound()`
Descripción	Destructor de la clase
Nombre	`loadData(TiXmlElement* element)`
Descripción	Método heredado y redefinido, el cual se encarga de modificar los atributos del sonido con los datos obtenidos del objeto pasado por parámetro.
Nombre	`saveData(TiXmlElement& pElement, bool pSaveProject)`
Descripción	Método heredado y redefinido, guarda en el objeto -`pElement`- pasado por parámetro los atributos del sonido. Los datos que se salven dependerán de si se está haciendo una salva para guardar el proyecto o exportando la configuración de los sonidos.
Nombre	`initialize()`
Descripción	Método heredado y redefinido, el cual restablece los atributos de la clase a sus valores predeterminados.
Nombre	`createMemento()`

Descripción	Método heredado y redefinido, el cual crea una salva de los datos de la clase.
Nombre	`setMemento(Memento* pMemento)`
Descripción	Método heredado y redefinido, el cual modifica los datos del sonido de objeto con los datos obtenidos del objeto pasado por parámetro.
Nombre	`getPosition()`
Descripción	Devuelve el valor del atributo mPosition
Nombre	`getDirection()`
Descripción	Devuelve el valor del atributo mDirection
Nombre	`getReferenceDistance()`
Descripción	Devuelve el valor del atributo mReferenceDistance
Nombre	`getMaxDistance()`
Descripción	Devuelve el valor del atributo mMaxDistance
Nombre	`getConeInnerAngle()`
Descripción	Devuelve el valor del atributo mConeInnerAngle
Nombre	`getConeOuterAngle()`
Descripción	Devuelve el valor del atributo mConeOuterAngle
Nombre	`getRelative()`
Descripción	Devuelve el valor del atributo mRelative
Nombre	`setPosition(Vector3f pPosition)`
Descripción	Modifica el valor del atributo mPosition
Nombre	`setDirection(Vector3f pDirection)`
Descripción	Modifica el valor del atributo mDirection
Nombre	`setReferenceDistance(float pReferenceDistance)`
Descripción	Modifica el valor del atributo mReferenceDistance
Nombre	`setMaxDistance(float pMaxDistance)`
Descripción	Modifica el valor del atributo mMaxDistance
Nombre	`setConeInnerAngle(float pConeInnerAngle)`
Descripción	Modifica el valor del atributo mConeInnerAngle
Nombre	`setConeOuterAngle(float pConeOuterAngle)`
Descripción	Modifica el valor del atributo mConeOuterAngle

Nombre	`setRelative(bool pRelative)`
Descripción	Modifica el valor del atributo mRelative

Tabla 35: Descripción de la clase "ObjectSound"

Nombre:	EnvironmentalSound	
Tipo de clase:	Entidad	
Atributo		**Tipo**
Para cada responsabilidad		
Nombre	`EnvironmentalSound(CSource* pSource,const char* pSoundName, const char* pFormat, const char* pFile, const char* pObjectName)`	
Descripción	Constructor de la clase	
Nombre	`~EnvironmentalSound()`	
Descripción	Destructor de la clase	
Nombre	`loadData(TiXmlElement* pElement)`	
Descripción	Método heredado y redefinido, el cual se encarga de modificar los atributos del sonido ambiental con los datos obtenidos del objeto pasado por parámetro.	
Nombre	`saveData(TiXmlElement& pElement, bool pSaveProject)`	
Descripción	Método heredado y redefinido, guarda en el objeto -`pElement`- pasado por parámetro los atributos del sonido ambiental. Los datos que se salven dependerán de si se está haciendo una salva para guardar el proyecto o exportando la configuración de los sonidos.	
Nombre	`createMemento()`	
Descripción	Método heredado y redefinido, el cual crea una salva de los datos de la clase.	
Nombre	`setMemento(Memento* pMemento)`	
Descripción	Método heredado y redefinido, el cual modifica los datos del sonido ambiental con los datos obtenidos del objeto pasado por parámetro.	

Tabla 36: Descripción de la clase "EnvironmentalSound"

Nombre:	SoundControl
Tipo de clase:	Controladora
Atributo	**Tipo**
mSounds	std:vector<Sound*>
mTracks	std:vector<int>
mBuffers	std:vector<int>
mDooplerFactor	float
mSpeedSound	float
mGain	float
mListenerPosition	Vector3f
mListenerUp	Vector3f
mListenerDirection	Vector3f
mGeneralStatus	EGeneralStatus
mDistanceModel	EDistanceModel
mNextEnvironmental	int
mSoundControl	CSoundControl*
Para cada responsabilidad	
Nombre	`SoundContro()`
Descripción	Constructor de la clase
Nombre	`~SoundContro()`
Descripción	Destructor de la clase
Nombre	`getSingleton()`
Descripción	Crea una instancia estática de la propia clase y la devuelve.
Nombre	`nameSound(const char* pFile)`
Descripción	Devuelve el nombre del fichero de sonido que se encuentra en la dirección pasada por parámetro.
Nombre	`formatSound(const char* pFile)`
Descripción	Devuelve el formato del fichero de sonido que se encuentra en la dirección pasada

	por parámetro.
Nombre	`loadGlobalProperties(TiXmlElement* pElement)`
Descripción	Modifica las propiedades globales de los sonidos con los datos obtenidos del objeto pasado por parámetro.
Nombre	`saveGlobalProperties(TiXmlElement& pElement, bool pExport = false)`
Descripción	Guarda en el objeto -`pElement`- pasado por parámetro las propiedades globales de los sonidos, las cuales serán posteriormente salvado en un fichero. Los datos que se salven dependerán de si se está haciendo una salva para guardar el proyecto o exportando la configuración de los sonidos.
Nombre	`intToStr(int pNum)`
Descripción	Devuelve como cadena el número natural pasado por parámetro.
Nombre	`floatToStr(float pNum)`
Descripción	Devuelve como cadena el número real pasado por parámetro.
Nombre	`setMementoObjectSound(MementoObjectName* pMemento)`
Descripción	Modifica los datos del sonido de objeto cuyo nombre sea igual al nombre almacenado en el objeto pasado por parámetro, devuelve una salva de dicho sonido antes de que este sea modificado.
Nombre	`setMementoEnvironmentalSound(MementoEnvironmentalSound* pMemento)`
Descripción	Modifica los datos del sonido ambiental cuyo nombre sea igual al nombre almacenado en el objeto pasado por parámetro, devuelve una salva de dicho sonido antes de que este sea modificado.
Nombre	`setMementoGlobalProperties(MementoGlobalProperties* pMemento)`
Descripción	Modifica las propiedades globales de los sonidos con los datos obtenidos del objeto pasado por parámetro.
Nombre	`setMementoCreateSound(MementoCreateSound* pMemento)`
Descripción	Modifica los datos del sonido de objeto cuyo nombre sea igual al nombre almacenado en el objeto pasado por parámetro.
Nombre	`setMementoDeleteObjectSound(MementoDeleteObjectSound*`

	pMemento)
Descripción	Crea un sonido de objeto a partir de los datos almacenados en el objeto pasado por parámetro.
Nombre	setMementoDeleteEnvironmentalSound(MementoDeleteEnvironmentalSound* pMemento)
Descripción	Crea un sonido ambiental a partir de los datos almacenados en el objeto pasado por parámetro.
Nombre	deleteSounds()
Descripción	Elimina todos los sonidos existentes.
Nombre	initializeGlobalProperties(bool pChangeValues = false)
Descripción	Modifica las propiedades globales de los sonidos con los valores almacenados en los atributos de la clase; en caso de que el parámetro sea verdadero, restablece los atributos a los valores predeterminados y luego es que modifica las propiedades globales de los sonidos.
Nombre	exportProject(const char* pFile)
Descripción	Salva la configuración de los sonidos en la dirección pasada por parámetro.
Nombre	saveProject(TiXmlElement& pElement)
Descripción	Guarda en el objeto pasado por parámetro los datos de los sonidos y las propiedades globales de estos.
Nombre	loadProject(TiXmlElement* pElement)
Descripción	Elimina los sonidos existentes. Con los datos obtenidos del objeto pasado por parámetro crea nuevos sonidos, los modifica y configura las propiedades globales de los sonidos.
Nombre	createObjectSound(const char* pFile, const char* pObjectName, Vector3f pPosition)
Descripción	Crea un nuevo sonido de objeto.
Nombre	createEnvironmentalSound(const char* pFile, const char* pObjectName = 0)
Descripción	Crea un nuevo sonido ambiental.
Nombre	deleteSound(const char* pObjectName)

Descripción	Elimina el sonido cuyo nombre sea igual a la cadena pasada por parámetro.
Nombre	`deleteSound(unsigned int pIndex)`
Descripción	Elimina el sonido que se encuentra en la posición pasada por parámetro.
Nombre	`isSound(const char* pObjectName)`
Descripción	Devuelve verdadero si existe algún sonido con el nombre igual a la cadena pasada por parámetro, falso en caso contrario.
Nombre	`play()`
Descripción	Se comienzan a reproducir todos los sonidos, excepto aquellos que ya estaban reproduciéndose.
Nombre	`pause()`
Descripción	Se pausa la reproducción de todos aquellos sonidos que estuvieran reproduciéndose en ese momento.
Nombre	`stop()`
Descripción	Se detiene la reproducción de todos los sonidos.
Nombre	`numberSounds()`
Descripción	Devuelve el número de sonidos existentes en ese momento.
Nombre	`nextNameEnvironmentalSound()`
Descripción	Devuelve el nombre que tendrá el próximo sonido ambiental que se cree.
Nombre	`getSound(const char* pObjectName)`
Descripción	Devuelve el sonido que tenga un nombre igual a la cadena pasada por parámetro.
Nombre	`getSound(unsigned int pIndex)`
Descripción	Devuelve el sonido que se encuentra en la posición pasada por parámetro de la lista de sonidos.
Nombre	`getSoundIndex(const char* pObjectName)`
Descripción	Devuelve la posición dentro de la lista de sonidos en la que se encuentra el sonido cuyo nombre sea igual a la cadena pasada por parámetro.
Nombre	`createMementoCreateSound()`
Descripción	Devuelve una salva de los valores de las propiedades globales de los sonidos.
Nombre	`createMementoCreateSound(const char* pObjectName)`
Descripción	Devuelve una salva del sonido cuyo nombre sea igual a la cadena pasada por

	parámetro.
Nombre	`createMementoDeleteObjectSound(int pIndex)`
Descripción	Devuelve una salva del sonido de objeto que se encuentre en la posición pasada por parámetro.
Nombre	`createMementoDeleteEnvironmentalSound(int pIndex)`
Descripción	Devuelve una salva del sonido ambiental que se encuentre en la posición pasada por parámetro.
Nombre	`getMementosDeleteObjectsSound()`
Descripción	Devuelve una lista con las salvas de todos los sonidos de objetos existentes.
Nombre	`setMemento(Memento* pMemento)`
Descripción	En dependencia de los datos que contenga el objeto pasado por parámetro modificará las propiedades de algún sonido, creará un sonido, eliminará un sonido o modificará las propiedades globales de los sonidos.
Nombre	`getGeneralStatus()`
Descripción	Devuelve el valor del atributo mGeneralStatus.
Nombre	`getDistanceModel()`
Descripción	Devuelve el valor del atributo mDistanceModel.
Nombre	`getDooplerFactor()`
Descripción	Devuelve el valor del atributo mDooplerFactor.
Nombre	`getSpeedSound()`
Descripción	Devuelve el valor del atributo mSpeedSound.
Nombre	`getGain()`
Descripción	Devuelve el valor del atributo mGain.
Nombre	`getListenerPosition()`
Descripción	Devuelve el valor del atributo mListenerPosition.
Nombre	`getListenerUp()`
Descripción	Devuelve el valor del atributo mListenerUp.
Nombre	`getListenerDirection()`
Descripción	Devuelve el valor del atributo mListenerDirection.
Nombre	`setDistanceModel(EDistanceModel pDistanceModel)`

Descripción	Modifica el tipo de atenuación de los sonidos.
Nombre	`setDooplerFactor(float pDooplerFactor)`
Descripción	Modifica el factor Doopler
Nombre	`setSpeedSound(float pSpeedSound)`
Descripción	Modifica la velocidad del sonido
Nombre	`setGain(float pGain)`
Descripción	Modifica el volumen de la escena.
Nombre	`setGeneralStatus(EGeneralStatus pGeneralStatus)`
Descripción	Modifica el estado del oyente.
Nombre	`setListenerPosition(Vector3f pListenerPosition)`
Descripción	Modifica la posición del oyente.
Nombre	`setListenerPosition(float pX, float pY, float pZ)`
Descripción	Modifica la posición del oyente.
Nombre	`setListenerOriantation(Vector3f pListenerDirection, Vector3f pListenerUp)`
Descripción	Modifica la orientación del oyente.
Nombre	`setListenerOriantation(float pDirX, float pDirY, float pDirZ, float pUpX, float pUpY, float pUpZ)`
Descripción	Modifica la orientación del oyente.

Tabla 37: Descripción de la clase "SoundControl"

Nombre:	Object_Sound
Tipo de clase:	Interfaz
Atributo	**Tipo**
mSound	ObjectSound*
Para cada responsabilidad	
Nombre	`Object_Sound(ObjectSound* pSound, InterfaceSoundControl* pSoundControl, QObject* pParent = 0)`
Descripción	Constructor de la clase

Nombre	`~Object_Sound()`
Descripción	Destructor de la clase
Nombre	`change(Memento * pMemento)`
Descripción	signal, es emitido cuando se modifica alguno de los atributos del objeto mSound.
Nombre	`getSoundName()`
Descripción	Devuelve el valor del atributo mSoundName del objeto mSound.
Nombre	`getObjectName()`
Descripción	Devuelve el valor del atributo mObjectName del objeto mSound.
Nombre	`getFormat()`
Descripción	Devuelve el valor del atributo mFormat del objeto mSound.
Nombre	`getPosition()`
Descripción	Devuelve el valor del atributo mPosition del objeto mSound.
Nombre	`getRelative()`
Descripción	Devuelve el valor del atributo mRelative del objeto mSound.
Nombre	`getPich()`
Descripción	Devuelve el valor del atributo mPich del objeto mSound.
Nombre	`getGain()`
Descripción	Devuelve el valor del atributo mGain del objeto mSound.
Nombre	`getReferenceDistance()`
Descripción	Devuelve el valor del atributo mReferenceDistance del objeto mSound.
Nombre	`getMaxDistance()`
Descripción	Devuelve el valor del atributo mMaxDistance del objeto mSound.
Nombre	`getConnerInnerAngle()`
Descripción	Devuelve el valor del atributo mConnerInnerAngle del objeto mSound.
Nombre	`getRoolOfFactor()`
Descripción	Devuelve el valor del atributo mRoolOfFactor del objeto mSound.
Nombre	`getLoop()`
Descripción	Devuelve el valor del atributo mLoop del objeto mSound.
Nombre	`getStatus()`
Descripción	Devuelve el valor del atributo mStatus del objeto mSound.

Nombre	`getLevel()`
Descripción	Devuelve el valor del atributo mLevel del objeto mSound.
Nombre	`getRangeGain()`
Descripción	Devuelve el valor del atributo mRangeGain del objeto mSound.
Nombre	`getConeOuterAngle()`
Descripción	Devuelve el valor del atributo mConeOuterAngle del objeto mSound.
Nombre	`setPitch(float pPitch)`
Descripción	En caso de que el valor pasado por parámetro sea distinto al valor del atributo mPitch del objeto mSound este toma el valor del parámetro y es emitido el signal change.
Nombre	`setGain(float pGain)`
Descripción	En caso de que el valor pasado por parámetro sea distinto al valor del atributo mGain del objeto mSound este toma el valor del parámetro y es emitido el signal change.
Nombre	`setPosition(Vector3f pGain)`
Descripción	En caso de que el valor pasado por parámetro sea distinto al valor del atributo mPosition del objeto mSound este toma el valor del parámetro y es emitido el signal change.
Nombre	`setRelative(EBool pRelative)`
Descripción	En caso de que el valor pasado por parámetro sea distinto al valor del atributo mRelative del objeto mSound este toma el valor del parámetro y es emitido el signal change.
Nombre	`setReferenceDistance(float pReferenceDistance)`
Descripción	En caso de que el valor pasado por parámetro sea distinto al valor del atributo mReferenceDistance del objeto mSound este toma el valor del parámetro y es emitido el signal change.
Nombre	`setMaxDistance(float pMaxDistance)`
Descripción	En caso de que el valor pasado por parámetro sea distinto al valor del atributo mMaxDistance del objeto mSound este toma el valor del parámetro y es emitido el signal change.

Nombre	`setDirectFilter(DirFilter pDirectFilter)`
Descripción	En caso de que el valor pasado por parámetro sea distinto al valor del atributo mDirectFilter del objeto mSound este toma el valor del parámetro y es emitido el signal change.
Nombre	`setConeInnerAngle(float pConeInnerAngle)`
Descripción	En caso de que el valor pasado por parámetro sea distinto al valor del atributo mConeInnerAngle del objeto mSound este toma el valor del parámetro y es emitido el signal change.
Nombre	`setRoolOfFactor(float pRoolOfFactor)`
Descripción	En caso de que el valor pasado por parámetro sea distinto al valor del atributo mRoolOfFactor del objeto mSound este toma el valor del parámetro y es emitido el signal change.
Nombre	`setLoop(EBool pLoop)`
Descripción	En caso de que el valor pasado por parámetro sea distinto al valor del atributo mLoop del objeto mSound este toma el valor del parámetro y es emitido el signal change.
Nombre	`setStatus(EStatus pStatus)`
Descripción	En caso de que el valor pasado por parámetro sea distinto al valor del atributo mStatus del objeto mSound este toma el valor del parámetro y es emitido el signal change.
Nombre	`setLevel(int pLevel)`
Descripción	En caso de que el valor pasado por parámetro sea distinto al valor del atributo mLevel del objeto mSound este toma el valor del parámetro y es emitido el signal change.
Nombre	`setRangeGain(Vector2f pRangeGain)`
Descripción	En caso de que el valor pasado por parámetro sea distinto al valor del atributo mRangeGain del objeto mSound este toma el valor del parámetro y es emitido el signal change.
Nombre	`setConeOuterAngle(float pConeOuterAngle)`
Descripción	En caso de que el valor pasado por parámetro sea distinto al valor del atributo

	mConeOuterAngle del objeto mSound este toma el valor del parámetro y es emitido el signal change.

Tabla 38: Descripción de la clase "Object_Sound"

Nombre:	Environmental_Sound	
Tipo de clase:	Interfaz	
Atributo		**Tipo**
mSound		EnvironmentalSound*
Para cada responsabilidad		
Nombre	`Environmental_Sound(EnvironmentalSound* pSound, InterfaceSoundControl* pSoundControl, QObject* pParent = 0)`	
Descripción	Constructor de la clase	
Nombre	`~Environmental_Sound()`	
Descripción	Destructor de la clase	
Nombre	`change(Memento * pMemento)`	
Descripción	signal, es emitido cuando se modifica alguno de los atributos del objeto mSound.	
Nombre	`getSoundName()`	
Descripción	Devuelve el valor del atributo mSoundName del objeto mSound.	
Nombre	`getObjectName()`	
Descripción	Devuelve el valor del atributo mObjectName del objeto mSound.	
Nombre	`getFormat()`	
Descripción	Devuelve el valor del atributo mFormat del objeto mSound.	
Nombre	`getPich()`	
Descripción	Devuelve el valor del atributo mPich del objeto mSound.	
Nombre	`getGain()`	
Descripción	Devuelve el valor del atributo mGain del objeto mSound.	
Nombre	`getDirectFilter()`	
Descripción	Devuelve el valor del atributo mDirectFilter del objeto mSound.	
Nombre	`getRoolOfFactor()`	

Descripción	Devuelve el valor del atributo mRoolOfFactor del objeto mSound.
Nombre	`getLoop()`
Descripción	Devuelve el valor del atributo mLoop del objeto mSound.
Nombre	`getStatus()`
Descripción	Devuelve el valor del atributo mStatus del objeto mSound.
Nombre	`getLevel()`
Descripción	Devuelve el valor del atributo mLevel del objeto mSound.
Nombre	`getRangeGain()`
Descripción	Devuelve el valor del atributo mRangeGain del objeto mSound.
Nombre	`setPitch(float pPitch)`
Descripción	En caso de que el valor pasado por parámetro sea distinto al valor del atributo mPitch del objeto mSound este toma el valor del parámetro y es emitido el signal change.
Nombre	`setGain(float pGain)`
Descripción	En caso de que el valor pasado por parámetro sea distinto al valor del atributo mGain del objeto mSound este toma el valor del parámetro y es emitido el signal change.
Nombre	`setDirectFilter(DirFilter pDirectFilter)`
Descripción	En caso de que el valor pasado por parámetro sea distinto al valor del atributo mDirectFilter del objeto mSound este toma el valor del parámetro y es emitido el signal change.
Nombre	`setRoolOfFactor(float pRoolOfFactor)`
Descripción	En caso de que el valor pasado por parámetro sea distinto al valor del atributo mRoolOfFactor del objeto mSound este toma el valor del parámetro y es emitido el signal change.
Nombre	`setLoop(EBool pLoop)`
Descripción	En caso de que el valor pasado por parámetro sea distinto al valor del atributo mLoop del objeto mSound este toma el valor del parámetro y es emitido el signal change.
Nombre	`setStatus(ESTATUS pStatus)`

Descripción	En caso de que el valor pasado por parámetro sea distinto al valor del atributo mStatus del objeto mSound este toma el valor del parámetro y es emitido el signal change.
Nombre	`setLevel(int pLevel)`
Descripción	En caso de que el valor pasado por parámetro sea distinto al valor del atributo mLevel del objeto mSound este toma el valor del parámetro y es emitido el signal change.
Nombre	`setRangeGain(Vector2f pRangeGain)`
Descripción	En caso de que el valor pasado por parámetro sea distinto al valor del atributo mRangeGain del objeto mSound este toma el valor del parámetro y es emitido el signal change.

Tabla 39: Descripción de la clase "Environmental_Sound"

Nombre:	Global_Properties
Tipo de clase:	Interfaz
Atributo	**Tipo**
Para cada responsabilidad	
Nombre	`Global_Properties(InterfaceSoundControl* pInterfaceSoundControl, QObject* pParent = 0)`
Descripción	Constructor de la clase
Nombre	`~Global_Properties()`
Descripción	Destructor de la clase
Nombre	`change(Memento* pMemento)`
Descripción	signal que es emitido cuando se modifica alguna de las propiedades globales de los sonidos.
Nombre	`getDistanceModel()`
Descripción	Devuelve el valor del atributo mDistanceModel del objeto estático del tipo SoundControl creado en la propia clase SoundControl.
Nombre	`getDooplerFactor()`

Descripción	Devuelve el valor del atributo mDooplerFactor del objeto estático del tipo SoundControl creado en la propia clase SoundControl.
Nombre	`getSpeedSound()`
Descripción	Devuelve el valor del atributo mSpeedSound del objeto estático del tipo SoundControl creado en la propia clase SoundControl.
Nombre	`getGain()`
Descripción	Devuelve el valor del atributo mGain del objeto estático del tipo SoundControl creado en la propia clase SoundControl.
Nombre	`setDistanceModel(EDistanceModel pDistanceModel)`
Descripción	En caso de que el valor pasado por parámetro sea distinto al valor del atributo mDistanceModel del objeto estático del tipo SoundControl - creado en la propia clase SoundControl - este toma el valor del parámetro y es emitido el signal change.
Nombre	`setDooplerFactor(float pDooplerFactor)`
Descripción	En caso de que el valor pasado por parámetro sea distinto al valor del atributo mDoopleFactor del objeto estático del tipo SoundControl - creado en la propia clase SoundControl - este toma el valor del parámetro y es emitido el signal change.
Nombre	`setSpeedSound(float pSpeedSound)`
Descripción	En caso de que el valor pasado por parámetro sea distinto al valor del atributo mSpeedSound del objeto estático del tipo SoundControl - creado en la propia clase SoundControl - este toma el valor del parámetro y es emitido el signal change.
Nombre	`setGain(float pGain)`
Descripción	En caso de que el valor pasado por parámetro sea distinto al valor del atributo mGain del objeto estático del tipo SoundControl - creado en la propia clase SoundControl - este toma el valor del parámetro y es emitido el signal change.

Tabla 40: Descripción de la clase "Global_Properties"

Nombre:	InterfaceSoundControl
Tipo de clase:	Controladora

Atributo	**Tipo**
mObjectSounds	std::vector<Object_Sound*>
mEnvironmentalSound	std::vector<Environmental_Sound*>
mGlobalProperty	Global_Properties

Para cada responsabilidad	
Nombre	`InterfaceSoundControl(QObject* pParentGlobalProperty = 0)`
Descripción	Constructor de la clase
Nombre	`~InterfaceSoundControl()`
Descripción	Destructor de la clase
Nombre	`moveCamera(Ogre::Vector3 pVectorMove, Ogre::Quaternion pOrintation)`
Descripción	signal, es emitido para cambiar la posición de la cámara cuando se carga un proyecto.
Nombre	`change(Memento* pMemento)`
Descripción	signal, es emitido cuando es modificado algún sonido o cuando se modifica alguna propiedad global de estos.
Nombre	`moveListener(Ogre::Vector3 pVector, Ogre::Quaternion pQuaternion)`
Descripción	slot, mueve al oyente para la posición y orientación pasadas por parámetros.
Nombre	`activeChange(Memento* pMemento)`
Descripción	slot, emite al signal change.
Nombre	`deleteSounds()`
Descripción	Elimina todos los sonidos y las interfaces correspondientes a estos.
Nombre	`clearGlobalProperties()`
Descripción	Asigna a las propiedades globales de los sonidos sus valores predeterminados.
Nombre	`exportProject(const char* pFile)`
Descripción	Salva la configuración de los sonidos en la dirección pasada por parámetro.

Nombre	`saveProject(TiXmlElement& pElement)`
Descripción	Guarda en el objeto pasado por parámetro los datos de los sonidos y las propiedades globales de estos.
Nombre	`loadProject(TiXmlElement* pElement)`
Descripción	Elimina los sonidos existentes. Con los datos obtenidos del objeto pasado por parámetro crea nuevos sonidos, los modifica y configura las propiedades globales de los sonidos.
Nombre	`createObjectSound(const char* pFile, const char* pObjectName, Vector3f pPosition)`
Descripción	Crea un nuevo sonido de objeto y la interfaz correspondiente a este.
Nombre	`createEnvironmentalSound(const char* pFile, const char* pObjectName = 0)`
Descripción	Crea un nuevo sonido ambiental y la interfaz correspondiente a este.
Nombre	`deleteObjectSound (const char* pObjectName)`
Descripción	Elimina el sonido de objeto y la interfaz correspondiente a este cuyo nombre sea igual a la cadena pasada por parámetro.
Nombre	`deleteObjectSound(unsigned int pIndex)`
Descripción	Elimina el sonido de objeto y la interfaz correspondiente a este que se encuentra en la posición pasada por parámetro.
Nombre	`deleteEnvironmentalSound (const char* pObjectName)`
Descripción	Elimina el sonido ambiental y la interfaz correspondiente a este cuyo nombre sea igual a la cadena pasada por parámetro.
Nombre	`deleteEnvironmentalSound (unsigned int pIndex)`
Descripción	Elimina el sonido ambiental y la interfaz correspondiente a este que se encuentra en la posición pasada por parámetro.
Nombre	`deleteAllObjectSounds()`
Descripción	Elimina todos los sonidos de objetos existentes y las interfaces correspondientes a estos.
Nombre	`deleteAllEnvironmentalSounds(unsigned int pIndex)`
Descripción	Elimina todos los sonidos ambientales existentes y las interfaces correspondientes

	a estos.
Nombre	`setGeneralStatus(SoundControl::EGeneralStatus pStatus)`
Descripción	Cambia el estado de todos los sonidos (PLAY, PAUSE o STOP);
Nombre	`isSound(const char* pObjectName)`
Descripción	Retorna verdadero si existe algún sonido con un nombre igual a la cadena pasada por parámetro, falso en caso contrario.
Nombre	`getObject_Sound(const char* pObjectName)`
Descripción	Devuelve la interfaz del sonido de objeto cuyo nombre sea igual a la cadena pasada por parámetro
Nombre	`getObject_Sound(unsigned int pIndex)`
Descripción	Devuelve la interfaz del sonido de objeto que se encuentre en la posición pasada por parámetro del arreglo: vector<Object_Sound*>
Nombre	`getEnvironmental_Sound(const char* pObjectName)`
Descripción	Devuelve la interfaz del sonido ambiental cuyo nombre sea igual a la cadena pasada por parámetro
Nombre	`getEnvironmental_Sound(unsigned int pIndex)`
Descripción	Devuelve la interfaz del sonido ambiental que se encuentre en la posición pasada por parámetro del arreglo: vector<Environmental_Sound*>
Nombre	`getGeneralStatus()`
Descripción	Devuelve el último estado (PLAY, PAUSE o STOP) que se le aplicó a todos los sonidos al mismo tiempo.
Nombre	`play()`
Descripción	Comienza a reproducir todos los sonidos, excepto aquellos que ya estaban reproduciéndose.
Nombre	`pause()`
Descripción	Se pause la reproducción de todos los sonidos que estuviesen reproduciéndose en ese momento.
Nombre	`stop()`
Descripción	Se detiene la reproducción de todos los sonidos.
Nombre	`numberEnvironmentalSounds()`

Descripción	Devuelve el número de sonidos ambientales existentes.
Nombre	`getGlobalProperty()`
Descripción	Devuelve la interfaz de las propiedades globales de los sonidos.
Nombre	`getMementosDeleteObjectsSound()`
Descripción	Devuelve una lista con las salvas de todos los sonidos de objeto existentes.
Nombre	`setMemento(Memento* pMemento)`
Descripción	En dependencia de los datos que contenga el objeto pasado por parámetro modificará las propiedades de algún sonido, creará un sonido, eliminará un sonido o modificará las propiedades globales de los sonidos.

Tabla 41: Descripción de la clase "InterfaceSoundControl"

Nombre:	OgreView
Tipo de clase:	Interfaz

Atributo	**Tipo**
mRenderWindow	Ogre::RenderWindow*
mSceneMgr	Ogre::SceneManager*
mCamera	Ogre::Camera*
mViewport	Ogre::Viewport*
mRoot	Ogre::Root*
mRaySceneQuery	Ogre::RaySceneQuery*
mLight	Ogre::Light*
mTranslateVector	Ogre::Vector3
mRotateUD	Ogre::Radian
mRotX	Ogre::Radian
mRotY	Ogre::Radian
mRotateSpeed	Degree
mNameSelected	Ogre::String

mMousePos	QPoint
mFps	int
mTriangles	int
mMouseLeftPressed	bool
mForward	bool
mBackward	bool
mLeft	bool
mRight	bool
mUp	bool
mDown	bool
mRotateP	bool
mRotateM	bool
mSelcted	bool
mDifTime	float
mMoveSpeed	float

Para cada responsabilidad	
Nombre	`OgreView(QWidget* pParent = 0)`
Descripción	Constructor de la clase.
Nombre	`~OgreView()`
Descripción	Destructor de la clase.
Nombre	`selectObject(const char* pName)`
Descripción	signal, es emitido cuando es seleccionado algún objeto de la escena.
Nombre	`signalMoveCamera(Ogre::Vector3 pPosicion, Ogre::Quaternion pOrientation)`
Descripción	signal, es emitido cuando se modifica la posición de la cámara.
Nombre	`viewPopupMenu(QPoint pPoint)`
Descripción	signal, es emitido cuando se da clic derecho en la escena.
Nombre	`slotMoveCamera(Ogre::Vector3 pPosicion, Ogre::Quaternion`

	`pOrientation)`
Descripción	slot, mueve la cámara para la posición y orientación pasadas por parámetro.
Nombre	`update()`
Descripción	Actualiza el estado de la escena.
Nombre	`setupResources()`
Descripción	Inicializa el nodo raíz.
Nombre	`updateStats()`
Descripción	Actualiza las variables que guardan los valores de los fotogramas por segundo a los que se está ejecutando la aplicación y los triángulos que se están dibujando en ese momento.
Nombre	`moveCamera()`
Descripción	Actualiza la posición de la cámara
Nombre	`createLight()`
Descripción	Crea una luz ambiental.
Nombre	`focusOutEvent(QFocusEvent* evt)`
Descripción	Evento que se ejecuta cuando la ventana pierde el foco.
Nombre	`resizeEvent(QResizeEvent* evt)`
Descripción	Evento que se ejecuta cuando se modifica el tamaño de la ventana.
Nombre	`timerEvent(QTimerEvent* evt)`
Descripción	Evento que se ejecuta cada cierto intervalo de tiempo definido previamente.
Nombre	`paintEvent(QPaintEvent* evt)`
Descripción	Evento que se ejecuta cada vez que se va a pintar la ventana.
Nombre	`keyPressEvent(QKeyEvent* evt)`
Descripción	Evento que se ejecuta cuando se presionada alguna tecla.
Nombre	`keyReleaseEvent(QKeyEvent* evt)`
Descripción	Evento que se ejecuta cuando se deja de presionar una tecla.
Nombre	`mousePressEvent(QMouseEvent* evt)`
Descripción	Evento que se ejecuta cuando se oprime alguno de los botones del mouse.
Nombre	`mouseReleaseEvent(QMouseEvent* evt)`
Descripción	Evento que se ejecuta cuando se deja de oprimir alguno de los botones del

	mouse.
Nombre	`mouseMoveEvent(QMouseEvent* evt)`
Descripción	Evento que se ejecuta cuando se mueve el cursor.
Nombre	`wheelEvent(QWheelEvent* evt)`
Descripción	Evento que se ejecuta cuando se mueve el scroll del mouse.
Nombre	`contextMenuEvent(QContextMenuEvent *evt)`
Descripción	Evento que se ejecuta cuando el usuario da clic derecho y se desea mostrar un menú.
Nombre	`initSeleted(float pX, float pY)`
Descripción	Determina el objeto sobre el cual el usuario dio clic.
Nombre	`confirmSelected(float pX, float pY)`
Descripción	Determina el objeto sobre el que el usuario dejó de dar clic, si este es el mismo objeto sobre el cual el usuario había dado clic, entonces este objeto es seleccionado y resaltado.
Nombre	`fileFormat(const char* pFile)`
Descripción	Devuelve el formato del archivo que se encuentra en la dirección pasada por parámetro.
Nombre	`getFPS()`
Descripción	Devuelve los fotogramas por segundo a los que se está ejecutando la aplicación.
Nombre	`getTriangles()`
Descripción	Devuelve los triángulos que están siendo dibujados en la escena.
Nombre	`getPosCamera()`
Descripción	Devuelve la posición de la cámara.
Nombre	`getDirCamera()`
Descripción	Devuelve la dirección de la cámara.
Nombre	`getNameNodes()`
Descripción	Devuelve una lista con los nombres de todos los objetos de la escena.
Nombre	`getMoveSpeed()`
Descripción	Devuelve la velocidad con la que se está trasladando el usuario por la escena.
Nombre	`loadOgreScene(const char* pFileName)`

Descripción	Carga el modelo que se encuentra en la dirección pasada por parámetro.
Nombre	`changeSelectObject(const char* pName)`
Descripción	Resalta el objeto cuyo nombre sea igual a la cadena pasada por parámetro.
Nombre	`deselectObject()`
Descripción	Si había algún objeto seleccionado este es deseleccionado.
Nombre	`clearScene()`
Descripción	Elimina todos los objetos de la escena.
Nombre	`ObjectNameSelected()`
Descripción	Devuelve el nombre del objeto que está seleccionado en ese instante.
Nombre	`positionObjectSelected()`
Descripción	Devuelve la posición del objeto que está seleccionado en ese instante.
Nombre	`relocateCamera()`
Descripción	Mueve la cámara para la posición inicial donde fue creada.
Nombre	`cameraStop()`
Descripción	Detiene el movimiento de la cámara.
Nombre	`setMoveSpeed(float pMoveSpeed)`
Descripción	Modifica la velocidad con la que el usuario se traslada por la escena.

Tabla 42: Descripción de la clase "OgreView"

Nombre:	MainWindow
Tipo de clase:	Interfaz

Atributo	**Tipo**
mUndos	stack<Memento>*
mRedos	stack<Memento>*
mChange	bool
mSave	bool
mDirScene	char*
mDirProject	char*

mDirOpenFile	QString
mCentralWidget	QWidget*
mTreeWidgetObject	QTreeWidget*
mTreeWidgetEnvironmetalSounds	QTreeWidget*
mSoundEditor	QPropertyEditorWidget*
mGlobalPropertyEditor	QPropertyEditorWidget*
mSoundControl	InterfaceSoundControl*
mOgreArea	QScrollArea*
mOgreView	OgreView*
mCentralLayout	QGridLayout*
mMoveSpeedSpinBox	QSpinBox*
mFileMenu	QMenu*
mEditMenu	QMenu*
mViewMenu	QMenu*
mOptionMenu	QMenu*
mSoundMenu	QMenu*
mHelpMenu	QMenu*
mFileToolBar	QToolBar*
mEditToolBar	QToolBar*
mOptionToolBar	QToolBar*
mObjectSoundToolBar	QToolBar*
mEnvironmentalSoundToolBar	QToolBar*
mRelocateCameraToolBar	QToolBar*
mMoveSpeedToolBar	QToolBar*
mObjectDockWidget	QDockWidget*
mSoundDockWidget	QDockWidget*

mGlobalPropertiesDockWidget	QDockWidget*
mEnvironmentalSoundsDockWidget	QDockWidget*
mStatusActGroup	QActionGroup*
mObjectSoundStatusActGroup	QActionGroup*
mEnvironmentalSoundStatusGroup	QActionGroup*
mNewProjectAct	QAction*
mLoadSceneAct	QAction*
mLoadProjectAct	QAction*
mSaveProjectAct	QAction*
mExportAct	QAction*
mSaveProjectAsAct	QAction*
mQuitAct	QAction*
mUndoAct	QAction*
mRedoAct	QAction*
mAddObjectSoundAct	QAction*
mQuitObjectSoundAct	QAction*
mDeselectedObjectAct	QAction*
mAddEnvironmentalSoundAct	QAction*
mQuitEnvironmentalSoundAct	QAction*
mPlayAct	QAction*
mPauseAct	QAction*
mStopAct	QAction*
mPlayObjectSoundAct	QAction*
mPauseObjectSoundAct	QAction*
mStopObjectSoundAct	QAction*
mPlayEnvironmentalSoundAct	QAction*

mPauseEnvironmentalSoundAct	QAction*
mStopEnvironmentalSoundAct	QAction*
mRelocateCameraAct	QAction*
mAboutAct	QAction*
mFpsLabel	QLabel*
mTrianglesLabel	QLabel*
mPositionLabel	QLabel*
mDirectionLabel	QLabel*
mMoveSpeedLabel	QLabel*

Para cada responsabilidad	
Nombre	`MainWindow(const QString &fileName, QWidget *pParent = 0)`
Descripción	Constructor de la clase
Nombre	`~MainWindow()`
Descripción	Destructor de la clase
Nombre	`changed(Memento* tMemento)`
Descripción	slot, guarda la salva pasada parámetro.
Nombre	`newProject()`
Descripción	slot, inicia el proceso de crear un nuevo proyecto
Nombre	`loadScene()`
Descripción	slot, inicia el proceso de cargar una escena
Nombre	`loadProject()`
Descripción	slot, inicia el proceso de cargar un proyecto
Nombre	`saveProject()`
Descripción	slot, inicia el proceso de salvar el proyecto
Nombre	`exportProject()`
Descripción	slot, salva la configuración de los sonidos.
Nombre	`saveProjectAs()`
Descripción	slot, salva el proyecto actual con otro nombre.

Nombre	`addObjectSound()`
Descripción	slot, crea un sonido de objeto
Nombre	`quitObjectSound()`
Descripción	slot, elimina el sonido aplicado al objeto que este seleccionado en ese momento
Nombre	`undo()`
Descripción	slot, deshace la última acción realizada
Nombre	`redo()`
Descripción	slot, rehace la última acción deshecha
Nombre	`deselectedObject()`
Descripción	slot, desmarca al objeto de la escena que estuviese seleccionado en ese momento.
Nombre	`addEnvironmentalSound()`
Descripción	slot, crea un sonido ambiental
Nombre	`quitEnvironmentalSound()`
Descripción	slot, elimina el sonido ambiental que este seleccionado en ese momento
Nombre	`play()`
Descripción	slot, comienza la reproducción de todos los sonidos excepto aquellos que ya estuviesen reproduciéndose.
Nombre	`pause()`
Descripción	slot, pausa la reproducción de todos los sonidos que estuviesen reproduciéndose.
Nombre	`stop()`
Descripción	slot, detiene la reproducción de todos los sonidos.
Nombre	`playObjectSound()`
Descripción	slot, comienza la reproducción del sonido aplicado al objeto de la escena que esté seleccionado en ese momento.
Nombre	`pauseObjectSound()`
Descripción	slot, pausa la reproducción del sonido aplicado al objeto de la escena que esté seleccionado en ese momento.
Nombre	`stopObjectSound()`
Descripción	slot, detiene la reproducción del sonido aplicado al objeto de la escena que esté

	seleccionado en ese momento.
Nombre	`playEnvironmentalSound()`
Descripción	slot, comienza la reproducción del sonido ambiental que esté seleccionado en ese momento.
Nombre	`pauseEnvironmentalSound ()`
Descripción	slot, pausa la reproducción del sonido ambiental que esté seleccionado en ese momento.
Nombre	`stopEnvironmentalSound ()`
Descripción	slot, detiene la reproducción del sonido ambiental que esté seleccionado en ese momento.
Nombre	`relocateCamera()`
Descripción	slot, mueve la cámara para la posición inicial donde fue creada.
Nombre	`itemSelectObjectSoundChanged()`
Descripción	slot, muestra los datos del sonido de objeto que ha sido seleccionado y resalta el objeto de la escena al cual se le aplicó dicho sonido.
Nombre	`itemSelectEnvironmentalSoundChanged()`
Descripción	slot, muestra los datos del sonido ambiental que ha sido seleccionado.
Nombre	`selectedObject(const char* pName)`
Descripción	slot, muestra los datos del sonido – de tenerlo – aplicado al objeto seleccionado.
Nombre	`viewScenePopupMenu(QPoint pPoint)`
Descripción	slot, muestra un menú cuyos datos estarán en dependencia del lugar de la escena donde el usuario haya dado clic derecho.
Nombre	`pressedSoundOject(const ModelIndex& pIndex)`
Descripción	slot, muestra un menú en caso de que el usuario de clic derecho encima de la ventana que muestra los sonidos de objetos existentes.
Nombre	`pressedEnvironmentalOject(const ModelIndex& pIndex)`
Descripción	slot, muestra un menú en caso de que el usuario de clic derecho encima de la ventana que muestra los sonidos ambientales existentes.
Nombre	`valueSpinBoxChanged(int pValue)`
Descripción	slot, modifica la velocidad con que se va a mover el usuario en la escena.

Nombre	`createActions()`
Descripción	Inicializa todas las acciones (QAction).
Nombre	`createMenus()`
Descripción	Inicializa todos los menús (QMenu).
Nombre	`createToolBars()`
Descripción	Inicializa todas las barras de tareas (QToolBar).
Nombre	`createDockWindows()`
Descripción	Inicializa todas las ventanas de documentos (QDockWidget).
Nombre	`createStatusBar()`
Descripción	Inicializa la barra de estado (QStatusBar).
Nombre	`timerEvent(QTimerEvent* evt)`
Descripción	Evento que se ejecuta cada cierto intervalo de tiempo determinado previamente.
Nombre	`actuallyLoadScene(const char* pFile)`
Descripción	Carga la escena que se encuentra en la dirección pasada por parámetro
Nombre	`actuallyNewProject()`
Descripción	Crea un nuevo proyecto.
Nombre	`actuallyLoadProject()`
Descripción	Carga un proyecto.
Nombre	`actuallySaveProject(const char* pFile)`
Descripción	Salva el proyecto actual en la dirección pasada por parámetro.
Nombre	`change()`
Descripción	Devuelve verdadero si ha ocurrido algún cambio en el proyecto actual que no haya sido salvado, falso en caso contrario.
Nombre	`floatToStr(float pNum)`
Descripción	Devuelve como cadena el número real pasado por parámetro.
Nombre	`dirWithoutName(QString pDir)`
Descripción	Devuelve una cadena sin el nombre del archivo que se encuentra en la dirección pasada por parámetro.
Nombre	`updateTreeWidgetEnvironmentalSounds()`
Descripción	Actualiza la ventana donde se muestran los sonidos ambientales existentes.

Nombre	`fileFormat(const char* pFile)`
Descripción	Devuelve el formato del archivo que se encuentra en la dirección pasada por parámetro.
Nombre	`createMementoScene()`
Descripción	Devuelve una salva de la escena.
Nombre	`setMementoScene(MementoScene* pMemento)`
Descripción	Carga una escena y crea sonidos de objetos a partir de los datos obtenidos del objeto pasado por parámetro.
Nombre	`clearRedos()`
Descripción	Elimina todos los objetos almacenados en la pila mRedos.
Nombre	`clearUndos()`
Descripción	Elimina todos los objetos almacenados en la pila mUndos.
Nombre	`updateGuiUndoRedo(Memento* pMemento)`
Descripción	Actualiza los elementos que se muestran en la aplicación cuando se deshace o rehace una acción.

Tabla 43: Descripción de la clase "MainWindow"

Nombre:	Memento
Tipo de clase:	Entidad
Atributo	**Tipo**
Para cada responsabilidad	
Nombre	`Memento()`
Descripción	Constructor de la clase
Nombre	`~Memento()`
Descripción	Destructor de la clase

Tabla 44: Descripción de la clase "Memento"

Nombre:	MementoObjectSound
Tipo de clase:	Entidad

Atributo	**Tipo**
mObjectName	char*
mRoolOfFactor	float
mGain	float
mPitch	float
mLoop	bool
mRangeGain	Vector2f
mLevel	int
mConeInnerAngle	float
mConnerOuterAngle	float
mReferenceDistance	float
mMaxDistance	float
mRelative	bool
mPosition	Vector3f
mDirection	Vector3f
mStatus	Sound::EStatus

Para cada responsabilidad	
Nombre	`MementoObjectSound(const char* pObjectName, float pRoolOfFactor, float pGain, float pPitch, float pConeInnerAngle, float pConeOuterAngle, float pReferenceDistance, float pMaxDistance, int pLevel, bool pRelative, bool pLoop, Sound::EStatus pStatus, Vector2f pRangeGain, Vector3f pPosition, Vector3f pDirection)`
Descripción	Constructor de la clase
Nombre	`~MementoObjectSound`
Descripción	Destructor de la clase.

Nombre	`getObjectName()`
Descripción	Devuelve el valor del atributo mObjectName
Nombre	`getRoolOfFactor()`
Descripción	Devuelve el valor del atributo mRoolOfFactor
Nombre	`getGain()`
Descripción	Devuelve el valor del atributo mGain
Nombre	`getPitch()`
Descripción	Devuelve el valor del atributo mPitch
Nombre	`getLoop()`
Descripción	Devuelve el valor del atributo mLoop
Nombre	`getRangeGain()`
Descripción	Devuelve el valor del atributo mRangeGain
Nombre	`getLevel()`
Descripción	Devuelve el valor del atributo mLevel
Nombre	`getConeInnerAngle()`
Descripción	Devuelve el valor del atributo mConneInnerAngle
Nombre	`getConeOuterAngle()`
Descripción	Devuelve el valor del atributo mConeOuterAngle
Nombre	`getReferenceDistance()`
Descripción	Devuelve el valor del atributo mReferenceDistance
Nombre	`getMaxDistance()`
Descripción	Devuelve el valor del atributo mMaxDistance
Nombre	`getRelative()`
Descripción	Devuelve el valor del atributo mRelative
Nombre	`getPosition`
Descripción	Devuelve el valor del atributo mPosition
Nombre	`getDirection()`
Descripción	Devuelve el valor del atributo mDirection
Nombre	`getStatus()`
Descripción	Devuelve el valor del atributo mStatus

Tabla 45: Descripción de la clase "MementoObjectSound"

Nombre:	MementoEnvironmentalSound
Tipo de clase:	Entidad
Atributo	**Tipo**
mObjectName	char*
mRoolOfFactor	float
mGain	float
mPitch	float
mLoop	bool
mRangeGain	Vector2f
mLevel	int
mStatus	Sound::EStatus
Para cada responsabilidad	
Nombre	`MementoEnvironmentalSound(const char* pObjectName, float pRoolOfFactor, float pGain, float pPitch, int pLevel, bool pLoop, Sound::EStatus pStatus, Vector2f pRangeGain,)`
Descripción	Constructor de la clase
Nombre	`~MementoEnvironmentalSound`
Descripción	Destructor de la clase.
Nombre	`getObjectName()`
Descripción	Devuelve el valor del atributo mObjectName
Nombre	`getRoolOfFactor()`
Descripción	Devuelve el valor del atributo mRoolOfFactor
Nombre	`getGain()`
Descripción	Devuelve el valor del atributo mGain
Nombre	`getPitch()`
Descripción	Devuelve el valor del atributo mPitch

Nombre	`getLoop()`
Descripción	Devuelve el valor del atributo mLoop
Nombre	`getRangeGain()`
Descripción	Devuelve el valor del atributo mRangeGain
Nombre	`getLevel()`
Descripción	Devuelve el valor del atributo mLevel
Nombre	`getStatus()`
Descripción	Devuelve el valor del atributo mStatus

Tabla 46: Descripción de la clase "MementoEnvironmentalSound"

Nombre:	MementoGlobalProperties
Tipo de clase:	Entidad
Atributo	**Tipo**
mDistanceModel	SoundControl::EdistanModel
mDooplerFactor	float
mSpeedSound	float
mGain	float
mGeneralStatus	SoundControl::EGeneralStatus
Para cada responsabilidad	
Nombre	`MementoGlobalProperties(SoundControl::EDistanModel pDistanceModel, float pDooplerFactor, float pSpeedSound, float pGain, SoundControl::EGeneralStatus pGeneralStatus)`
Descripción	Constructor de la clase
Nombre	`~MementoGlobalProperties()`
Descripción	Destructor de la clase.
Nombre	`getDistanceModel()`
Descripción	Devuelve el valor del atributo mDistanceModel.
Nombre	`getDooplerFactor()`

Descripción	Devuelve el valor del atributo mDooplerFactor.
Nombre	`getSpeedSound()`
Descripción	Devuelve el valor del atributo mSpeedSound.
Nombre	`getGain()`
Descripción	Devuelve el valor del atributo mGain.
Nombre	`getGeneralStatus()`
Descripción	Devuelve el valor del atributo mGeneralStatus.

Tabla 47: Descripción de la clase "MementoGlobalProperties"

Nombre:	MementoCreateSound	
Tipo de clase:	Entidad	
Atributo		**Tipo**
mIndex		int
mObjectName		char*
Para cada responsabilidad		
Nombre	`MementoCreateSound(int pIndex, const char* pObjectName)`	
Descripción	Constructor de la clase	
Nombre	`~MementoCreateSound ()`	
Descripción	Destructor de la clase	
Nombre	`getIndex()`	
Descripción	Devuelve el valor del atributo mIndex	
Nombre	`getObjectName()`	
Descripción	Devuelve el valor del atributo mObjectName	

Tabla 48: Descripción de la clase "MementoCreateSound"

Nombre:	MementoDeleteObjectSound	
Tipo de clase:	Entidad	
Atributo		**Tipo**

mIndex	int
mFile	char*
mMementoObjectSound	MementoObjectSound*
Para cada responsabilidad	
Nombre	`MementoDeleteObjectSound(MementoObjectSound* pMementoObjectSound, const char* pFile, int pIndex)`
Descripción	Constructor de la clase
Nombre	`~MementoDeleteObjectSound()`
Descripción	Destructor de la clase
Nombre	`getIndex()`
Descripción	Devuelve el valor del atributo mIndex
Nombre	`getObjectName()`
Descripción	Devuelve el valor del atributo mObjectName
Nombre	`getMementoObjectSound()`
Descripción	Devuelve el valor del atributo mMementoObjectSound

Tabla 49: Descripción de la clase "MementoDeleteObjectSound"

Nombre:	MementoDeleteEnvironmentalSound
Tipo de clase:	Entidad
Atributo	**Tipo**
mIndex	int
mFile	char*
mMementoEnvironmentalSound	MementoEnvironmentalSound*
Para cada responsabilidad	
Nombre	`MementoDeleteEnvironmentalSound(MementoEnvironmentalSound* pMementoObjectSound, const char* pFile, int pIndex)`
Descripción	Constructor de la clase
Nombre	`~MementoDeleteEnvironmentalSound()`

Descripción	Destructor de la clase
Nombre	`getIndex()`
Descripción	Devuelve el valor del atributo mIndex
Nombre	`getObjectName()`
Descripción	Devuelve el valor del atributo mObjectName
Nombre	`getMementoEnvironmentalSound()`
Descripción	Devuelve el valor del atributo mMementoEnvironmentalSound

Tabla 50: Descripción de la clase "MementoDeleteEnvironmentalSound"

Nombre:	MementoScene
Tipo de clase:	Entidad
Atributo	**Tipo**
mDirScene	char*
mMementosDeleteObjectsSound	vector<MementoDeleteObjectSound*>*
Para cada responsabilidad	
Nombre	`MementoScene( const char* pDirScene, vector<MementoDeleteObjectSound*>* pMementoDeleteObjectSound)`
Descripción	Constructor de la clase
Nombre	`~MementoScene()`
Descripción	Destructor de la clase
Nombre	`getDirScene()`
Descripción	Devuelve el valor del atributo mDirScene
Nombre	`getMementosDeleteObjectsSound()`
Descripción	Devuelve el valor del atributo mMementosDeleteObjectsSound

Tabla 51: Descripción de la clase "MementoScene"

Nombre:	Vector2f	
Tipo de clase:	Entidad	
Atributo		**Tipo**
mX		float
mY		float
Para cada responsabilidad		
Nombre	`Vector2f()`	
Descripción	Constructor por defecto	
Nombre	`Vector2f( float pX, float pY)`	
Descripción	Constructor de la clase	
Nombre	`operator == (const Vector2f& other)`	
Descripción	Sobrecarga del operador de igualdad.	
Nombre	`operator != (const Vector2f& other)`	
Descripción	Sobrecarga del operador de desigualdad.	
Nombre	`operator = (const Vector2f& other)`	
Descripción	Sobrecarga del operador de asignación.	

Tabla 52: Descripción de la clase "Vector2f"

Nombre:	Vector3f	
Tipo de clase:	Entidad	
Atributo		**Tipo**
mX		float
mY		float
mZ		float
Para cada responsabilidad		
Nombre	`Vector3f()`	
Descripción	Constructor por defecto	
Nombre	`Vector3f( float pX, float pY, float pZ)`	

Descripción	Constructor de la clase
Nombre	`operator == (const Vector3f& other)`
Descripción	Sobrecarga del operador de igualdad.
Nombre	`operator != (const Vector3f& other)`
Descripción	Sobrecarga del operador de desigualdad.
Nombre	`operator = (const Vector3f& other)`
Descripción	Sobrecarga del operador de asignación.

Tabla 53: Descripción de la clase "Vector3f"

Nombre:	DirFilter
Tipo de clase:	Entidad
Atributo	**Tipo**
mGain	float
mGainHF	float
mDirectFilter	EDirectFilter
Para cada responsabilidad	
Nombre	`DirFilter()`
Descripción	Constructor por defecto
Nombre	`DirFilter( float pGain, float pGainHF, EDirectFilter pDirFilter)`
Descripción	Constructor de la clase
Nombre	`operator == (const DirFilter& other)`
Descripción	Sobrecarga del operador de igualdad.
Nombre	`operator != (const DirFilter& other)`
Descripción	Sobrecarga del operador de desigualdad.
Nombre	`operator = (const DirFilter& other)`
Descripción	Sobrecarga del operador de asignación.

Tabla 54: Descripción de la clase "DirFilter"

Nombre:	Error	
Tipo de clase:	Entidad	
Atributo		**Tipo**
mMsg		char*
Para cada responsabilidad		
Nombre	`Error()`	
Descripción	Constructor por defecto	
Nombre	`Error(const char* pMsg)`	
Descripción	Constructor de la clase	
Nombre	`~Error()`	
Descripción	Destructor de la clase	
Nombre	`getMsg()`	
Descripción	Devuelve el valor del atributo mMsg.	

Tabla 55: Descripción de la clase "Error"

Nombre:	ErrorConfigurationGlobalProperties	
Tipo de clase:	Entidad	
Atributo		**Tipo**
Para cada responsabilidad		
Nombre	`ErrorConfigurationGlobalProperties()`	
Descripción	Constructor de la clase	

Tabla 56: Descripción de la clase "ErrorConfigurationGlobalProperties"

Nombre:	ErrorConfigurationSound	
Tipo de clase:	Entidad	
Atributo		**Tipo**
Para cada responsabilidad		
Nombre	`ErrorConfigurationSound(const char* pDirSound)`	

Descripción	Constructor de la clase

Tabla 57: Descripción de la clase "ErrorConfigurationSound"

Nombre:	ErrorOpenFile
Tipo de clase:	Entidad
Atributo	**Tipo**
Para cada responsabilidad	
Nombre	`ErrorOpenFile(const char* pDirFile)`
Descripción	Constructor de la clase

Tabla 58: Descripción de la clase "ErrorOpenFile"

Nombre:	ErrorOpenModel
Tipo de clase:	Entidad
Atributo	**Tipo**
Para cada responsabilidad	
Nombre	`ErrorOpenModel()`
Descripción	Constructor de la clase

Tabla 59: Descripción de la clase "ErrorOpenModel"

Nombre:	ErrorOpenModelFileFormatUnknown
Tipo de clase:	Entidad
Atributo	**Tipo**
Para cada responsabilidad	
Nombre	`ErrorOpenModelFileFormatUnknown(const char* pFile)`
Descripción	Constructor de la clase

Tabla 60: Descripción de la clase "ErrorOpenModelFileFormatUnknown"

Nombre:	ErrorOpenProject	
Tipo de clase:	Entidad	
Atributo		**Tipo**
Para cada responsabilidad		
Nombre	`ErrorOpenProject()`	
Descripción	Constructor de la clase	

Tabla 61: Descripción de la clase "ErrorOpenProject"

Nombre:	ErrorOpenProjectFileFormatUnknown	
Tipo de clase:	Entidad	
Atributo		**Tipo**
Para cada responsabilidad		
Nombre	`ErrorOpenProjectFileFormatUnknown(const char* pFile)`	
Descripción	Constructor de la clase	

Tabla 62: Descripción de la clase "ErrorOpenProjectFileFormatUnknown"

Nombre:	ErrorPositionOutOfRange	
Tipo de clase:	Entidad	
Atributo		**Tipo**
Para cada responsabilidad		
Nombre	`ErrorPositionOutOfRange()`	
Descripción	Constructor de la clase	

Tabla 63: Descripción de la clase "ErrorPositionOutOfRange"

4.4. Patrones de diseño

4.4.1. *Memento*

Sin violar el encapsulamiento, captura y externaliza el estado interno de un objeto de manera que dicho objeto puede ser restaurado a este estado posteriormente. En el caso de la aplicación es necesario ir almacenando los cambios que sean realizados por el usuario, para que este tenga la posibilidad en todo momento de deshacer algún cambio que haya realizado o rehacer un cambio que hubiese desecho. Los objetos normalmente encapsulan todo su estado, haciéndolo inaccesible e imposibles de externalizar. Exponer este estado violaría el encapsulamiento, lo cual compromete la fiabilidad y la extensibilidad de la aplicación.

Participantes(20)**:**

- ***Memento***
    - Almacena el estado interno de un objeto *Originator*. El *Memento* puede almacenar mucho o parte del estado interno de *Originator*.
    - Tiene dos interfaces. Una para *Caretaker*, que le permite manipular el *Memento* únicamente para pasarlo a otros objetos. La otra interfaz sirve para que *Originator* pueda almacenar/restaurar su estado interno, sólo *Originator* puede acceder a esta interfaz, al menos en teoría.

- ***Originator***
    - *Originator* crea un objeto *Memento* conteniendo una fotografía de su estado interno.
    - *Originator* usa a *Memento* para restaurar su estado interno.

- ***Caretaker***
    - Es responsable por mantener a salvo a *Memento*.
    - No opera o examina el contenido de *Memento*.

4.4.2. *Singleton*

En el caso del sistema es necesario asegurar que exista una única instancia de la clase SoundControl y proveer un punto global para acceder a esta. Una variable global hace un objeto accesible, pero no limita la instanciación de múltiples objetos de un tipo. Una mejor solución es hacer que la clase misma sea responsable de garantizar su única instancia, interceptando solicitudes para crear nuevos objetos y facilitando además una vía para acceder a la instancia. Este es el patrón *Singleton*.

Participantes(21)**:**

- ***Singleton***
    - Define una operación Instancia que permite que los clientes accedan a su única instancia. Instancia es una operación de clase.
    - Puede ser responsable de crear su única instancia.

Aplicabilidad(21):

- Deba haber exactamente una instancia de una clase y ésta deba ser accesible a los clientes desde un punto de acceso conocido.
- La única instancia debería ser extensible mediante herencia y los clientes deberían ser capaces de utilizar una instancia extendida sin modificar su código.

Consecuencias(21):

- Acceso controlado a la única instancia. Puede tener un control estricto sobre cómo y cuándo acceden los clientes a la instancia.
- Espacio de nombres reducido. El patrón *Singleton* es una mejora sobre las variables globales.
- Permite el refinamiento de operaciones y la representación. Se puede crear una subclase de *Singleton*.
- Permite un número variable de instancias. El patrón hace que sea fácil cambiar de opinión y permitir más de una instancia de la clase *Singleton*.
- Más flexible que las operaciones de clase.

Conclusiones

En este capítulo se presentaron los diagramas de clases del diseño y los diagramas de secuencia de la realización de los casos de uso. Además se determinaron los patrones de diseño que se utilizarán en la construcción del sistema. Permitiendo todo esto que se pase a la fase de implementación del sistema.

Capítulo 5: Implementación

Introducción

En este capítulo se muestran los componentes físicos por los que estará compuesto el sistema (ficheros .h y .cpp correspondientes a la implementación en C++), se define el estándar de codificación a utilizar y se elabora el diagrama de despliegue para el sistema.

5.1. Estándar de codificación

Constantes: Los nombres de las constantes se escriben completamente con mayúsculas, en caso de que el nombre este compuesto por más de una palabra, estas son separadas por el carácter "_".

Variables: Las variables comenzarán con una letra minúscula, la cual dependerá del tipo de variable, y a continuación la palabra o palabras que conformen el nombre de la variable, comenzando cada una con letra mayúscula y el resto con minúsculas. Las variables miembro de una clase tendrán como prefijo el carácter "m"; las variables temporales tendrán como prefijo una "t"; las variables globales tendrán como prefijo una "g" y las variables que son argumentos de una función tendrán como prefijo una "p".

Clases: Los nombres de las clases se escriben con letra inicial mayúscula, en caso de tener un nombre conformado por más de una palabra estas se escribirán una a continuación de la otra comenzando siempre cada palabra con una letra mayúscula y el resto en minúscula.

Métodos miembros de clases: Los nombres de los métodos miembros de una clase comenzarán con letra inicial minúscula, en caso de que el nombre este compuesto por más de una palabra la primera palabra será escrita en minúscula completamente y el resto comenzará con letra mayúscula.

Nombres de enumerados: Los enumerados comenzarán con la letra "E" y a continuación la palabra o palabras que conformen el nombre, cada una comenzando con letra mayúscula y el resto en minúsculas. Los valores de los enumerados cumplen con las mismas reglas que las constantes.

5.2. Diagrama de despliegue

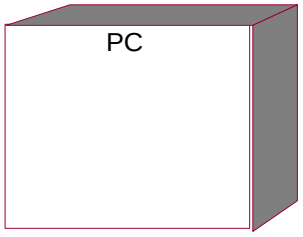

Figura 34: Diagrama de despliegue

5.3. Diagramas de componentes

5.3.1. Diagrama de componentes por paquetes

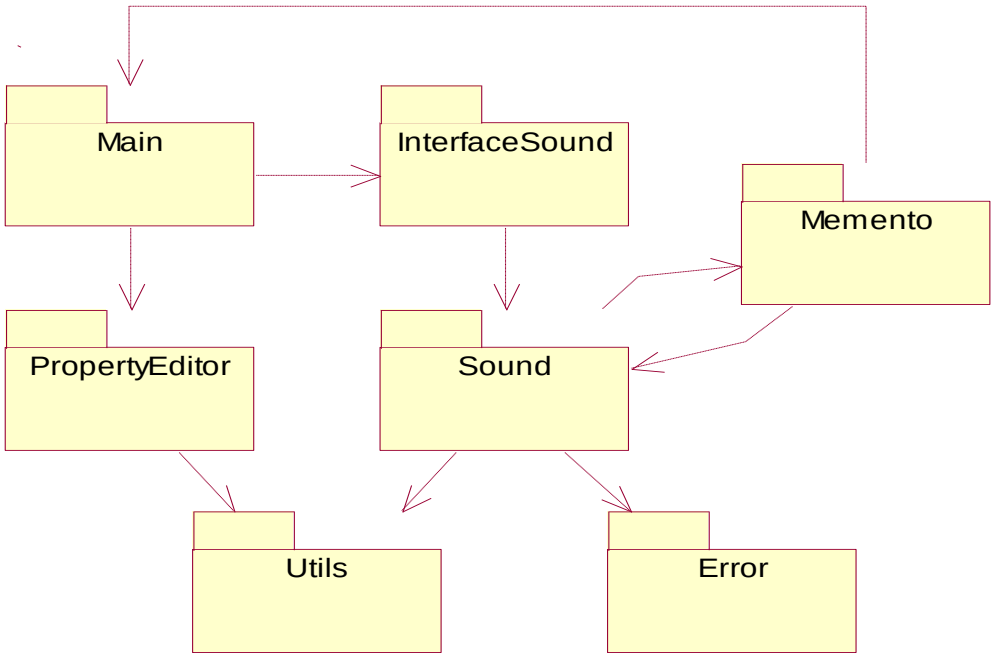

Figura 35: Diagrama de componentes por paquetes

5.3.2. Diagrama de componentes, paquete "Main"

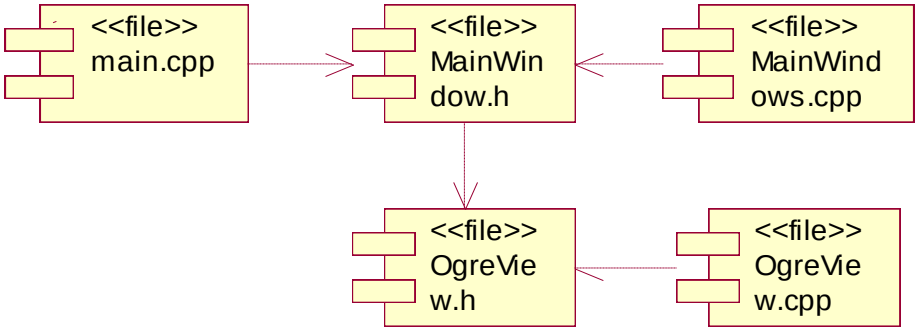

Figura 36: Diagrama de componentes, paquete "Main"

5.3.3. Diagrama de componentes, paquete “InterfaceSound”

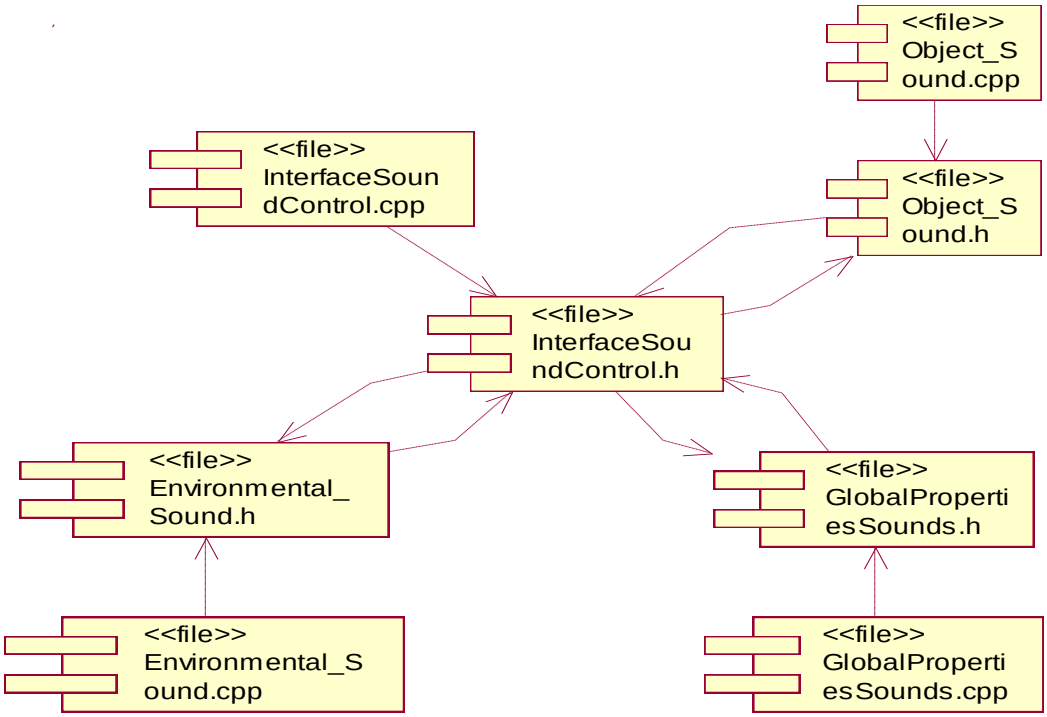

Figura 37: Diagrama de componentes, paquete “InterfaceSounds”

5.3.4. Diagrama de componentes, paquete “PropertyEditor”

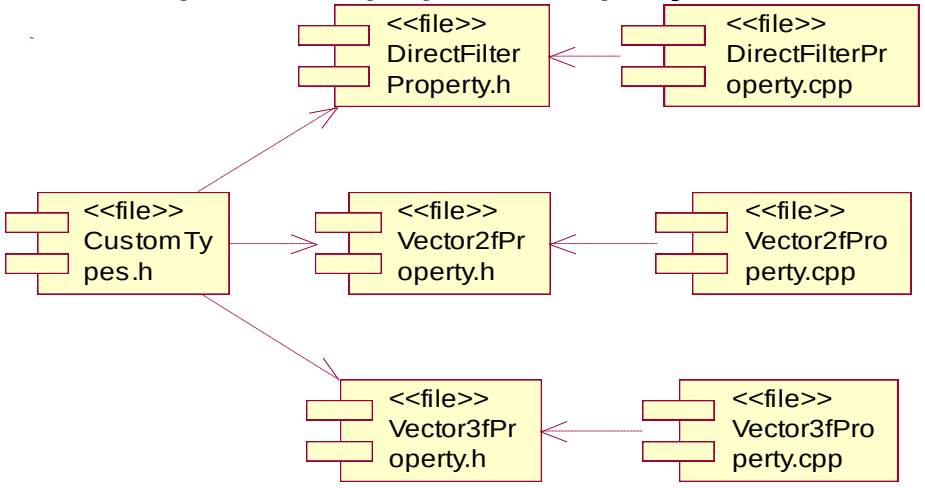

Figura 38: Diagrama de componentes, paquete “PropertyEditor”

5.3.5. Diagrama de componentes, paquete “Sound”

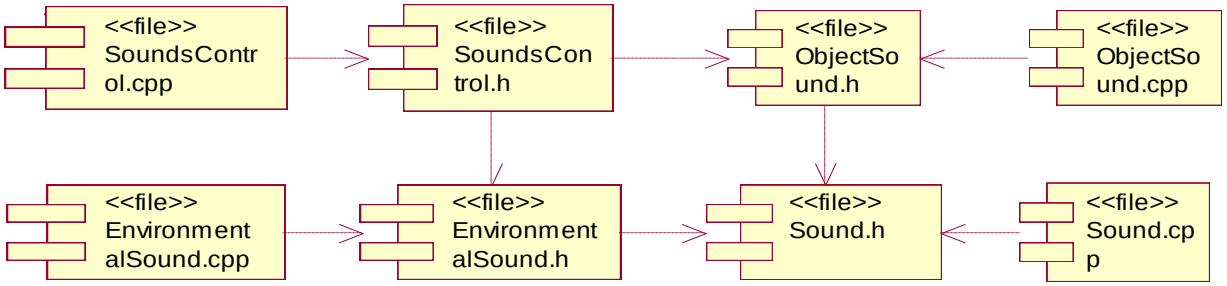

Figura 39: Diagrama de componentes, paquete “Sound”

5.3.6. Diagrama de componentes, paquete “Utils”

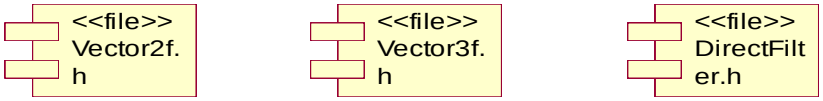

Figura 40: Diagrama de componentes, paquete “Utils”

5.3.7. Diagrama de componentes, paquete “Error”

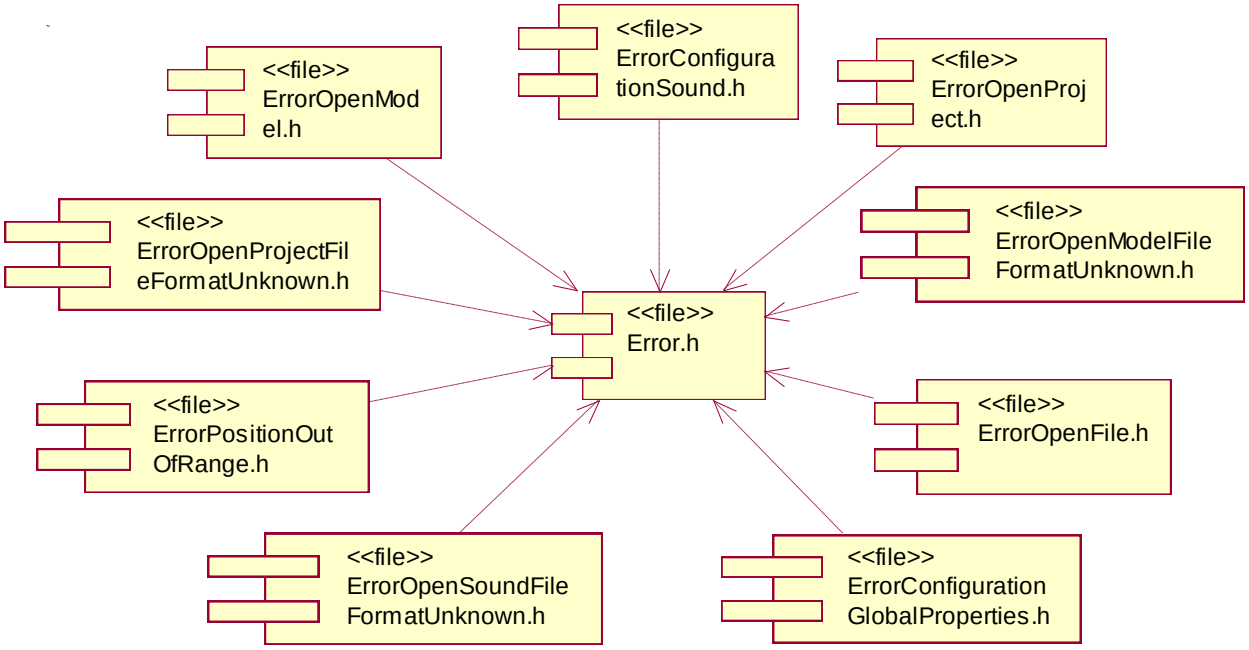

Figura 41: Diagrama de componentes, paquete “Error”

5.3.8. Diagrama de componentes, paquete “Memento”

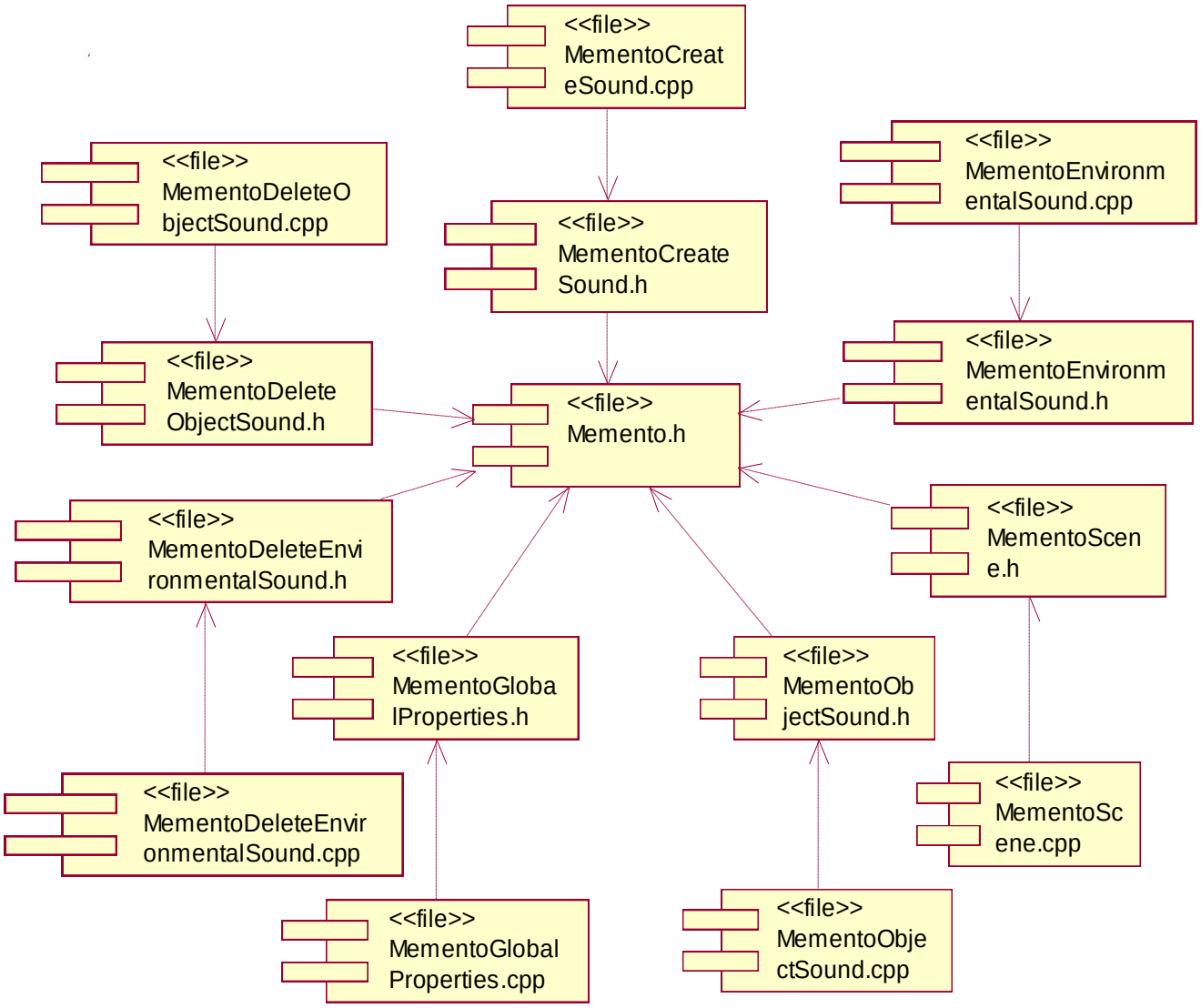

Figura 42: Diagrama de componentes, paquete “Memento”

Conclusiones

En este capítulo se dieron a conocer los diagramas más importantes del flujo de trabajo de implementación (diagrama de despliegue y diagrama de componentes). Con esto queda todo listo para pasar a la programación de los casos de uso correspondientes al primer ciclo de desarrollo.

Conclusiones

Con la realización de este trabajo, y en cumplimiento de los objetivos inicialmente planteados se obtuvo una aplicación que satisface las necesidades de la biblioteca de sonido SoundToolkit, brindando las siguientes funcionalidades:

- Cargar una escena 3D.
- Adicionar, modificar o eliminar sonidos referidos a objetos.
- Adicionar, modificar o eliminar sonidos ambientales.
- Modificar las propiedades globales de los sonidos.
- Guardar la configuración de los sonidos para que posteriormente sea usada por terceros.

Además se brindan otras funcionalidades auxiliares (crear/salvar/cargar un proyecto, deshacer/rehacer, etc.) con el objetivo de hacer que el trabajo con la aplicación sea lo más fácil y agradable posible, dándole al usuario un mayor grado de libertad y seguridad cuando interactúa con el sistema.

Con esta aplicación se garantiza el incremento de la productividad de los desarrolladores, disminuyendo drásticamente el tiempo necesario para configurar los sonidos del entorno 3D que se esté desarrollando (juego, simulador, etc.), además de permitir delegar esta responsabilidad en personas que no tengan conocimientos de programación, permitiendo al equipo de desarrollo concentrarse en otras tareas.

Esta aplicación puede ser portada a otros sistemas operativos.

Recomendaciones

Durante el desarrollo de este trabajo han surgido algunas ideas para lograr crear una aplicación con un mayor número de funcionalidades y más fácil de usar, por lo que se recomienda:

- Añadir la funcionalidad de adicionar sonidos a la escena en lugares donde no se encuentre ningún objeto.
- Añadir la funcionalidad de poder seleccionar múltiples objetos para poder adicionar/eliminar sonidos al unísono.
- Añadir la funcionalidad de adicionar sonidos a objetos que tengan movimiento.
- Salvar el estado (configuración) de la interfaz visual cuando se cierre la aplicación y restablecer dicho estado cuando esta vuelva a ser ejecutada.
- Incorporar soporte para más idiomas haciendo uso de Qt Linguist.

Bibliografía consultada

1. **Blanchette, Jasmin y Summerfield, Mark.** *C++ GUI Programming with Qt 4.* Massachusetts : Trolltech AS, 2006. ISBN 0-13-187249-4..

2. **Boggs, Wendy y Boggs, Michael.** *Mastering UML with Rational Rose 2002.* Alameda : SYBEX, 2002. ISBN: 0−7821−4017−3.

3. **Griffith, Arthur.** *KDE 2/Qt Programming Bible.* New York : IDG Books, 2001. ISBN: 0-7645-4682-1.

4. grinninglizard. *TinyXml Documentation.* [En línea] http://www.grinninglizard.com/tinyxmldocs/index.html.

5. **Hiebert, Garin.** *Creative OpenAL Programmer's Reference.* 2001.

6. **Junker, Gregory.** *Pro OGRE 3D Programming.* New York : apress, 2006. ISBN-13: 978-1-59059-710-1.

7. **Katrib, Miguel, y otros.** *Visual Studio 2008. Desafía todos los retos.* Ciudad de la Habana : Editorial Capitán San Luis, 2008. ISBN: 978-959-211-329-9.

8. **Marrero, Dagoberto y Abreu, Lazaro.** *SoundToolkit, Módulo Para el Manejo de Sonidos en Sistemas de Realidad Virtual.* Ciudad de la Habana : s.n., 2007.

9. **Martz, Paul.** *OpenSceneGraph Quick Start Guide.* California : Skew Matrix Software, 2007.

10. Ogre3d. [En línea] http://www.ogre3d.org.

11. OpenSceneGraph. [En línea] http://www.openscenegraph.org/.

12. qtsoftware. *qt-4.4-whitepaper.* [En línea] http://www.qtsoftware.com/products/files/pdf/qt-4.4-whitepaper.

13. Trolltech. *Qt Reference Documentation.* [En línea] http://doc.trolltech.com/4.5/index.html.

Referencias bibliográficas

1. **Aimacaña Toledo, Carlos.** wikiciencia. *Interfaz de usuario.* [En línea] [Citado el: 12 de Diciembre de 2008.] http://www.wikiciencia.org/informatica/programacion/iusuario/index.php.

2. **Sebastián Gómez, Leopoldo.** [En línea] [Citado el: 12 de Enero de 2009.] http://200.14.84.223/apuntesudp/showDoc.php?id=1209&ramo=ICI2423.

3. **Marrero Expósito, Carlos.** Mente Expansiva. *Interfaz Gráfica de Usuario: aproximación semio-cognitiva.* [En línea] 2006. [Citado el: 12 de Diciembre de 2008.] http://www.chr5.com/investigacion/investiga_igu/index_igu.html.

4. The GTK+ Project. *features.* [En línea] [Citado el: 8 de Enero de 2009.] http://www.gtk.org/features.html.

5. qtsoftware. *products.* [En línea] [Citado el: 15 de 1 de 2009.] http://www.qtsoftware.com/products.

6. DevMaster.net. *3D Engines Database.* [En línea] [Citado el: 14 de Enero de 2009.] http://www.devmaster.net/engines/engine_details.php?id=2.

7. ogre3d. *features.* [En línea] [Citado el: 12 de Diciembre de 2008.] http://www.ogre3d.org/about/features.

8. OpenSceneGrph. *Introduction.* [En línea] [Citado el: 12 de Diciembre de 2008.] http://www.openscenegraph.org/projects/osg/wiki/About/Introduction.

9. Encuentros Java. *grafos de escena.* [En línea] [Citado el: 13 de Enero de 2009.] http://encuentrosjava.uji.es/materiales/workshop_3d.pdf.

10. DevMaster.net. *3D Engines Database.* [En línea] [Citado el: 14 de Enero de 2009.] http://www.devmaster.net/engines/engine_details.php?id=4.

11. Irrlicht. *features.* [En línea] [Citado el: 15 de Enero de 2009.] http://irrlicht.sourceforge.net/features.html.

12. **Marrero, Dagoberto y Abreu, Lazaro.** *SoundToolkit, Módulo Para el Manejo de Sonidos en Sistemas de Realidad Virtual.* Ciudad de la Habana : s.n., 2007.

13. Asociación de Músicos en Internet. *Formatos de compresión de audio.* [En línea] [Citado el: 15 de Diciembre de 2008.] http://www.asociacionmusica.com/t_audio.asp.

14. wisegeek. *What is a WAV File?* [En línea] [Citado el: 15 de Diciembre de 2008.] http://www.wisegeek.com/what-is-a-wav-file.htm.

15. Laboratorio de Procesado de Imagen. *Formatos de compresión de audio digital.* [En línea] [Citado el: 14 de Diciembre de 2008.]

http://www.lpi.tel.uva.es/~nacho/docencia/ing_ond_1/trabajos_01_02/digitalizacion_compresion_audio/bin/Formatos.htm.

16. All Streaming Media.Com. *Windows Media audio format (.WMA files).* [En línea] [Citado el: 14 de Diciembre de 2008.] http://all-streaming-media.com/faq/streaming-media/Format-Windows-Media-audio-WMA.htm.

17. **Pérez, Fabienny y Romero, Mario Orlando.** *Interacción entre elementos virtuales a través de estándares XML y X3D.* Ciudad de la Habana : s.n., 2008.

18. **Cawood, Stephen**. Google libros. [En línea] 9 de Junio de 2007. [Citado el: 13 de Enero de 2009.] http://books.google.com.cu/books?id=nF7vmDOf8UcC&pg=PT363&lpg=PT363&dq=xact+%2B+Cross-platform+Audio+Creation+Tool&source=bl&ots=wf4fVL8LDl&sig=E811e3fjY7m-vcmRc_8uxQy8Hk4&hl=es&ei=Wzw8Sp2vF9qntgf65fEE&sa=X&oi=book_result&ct=result&resnum=9.0071614060.

19. Adventure Game Studio. *features*. [En línea] [Citado el: 15 de Diciembre de 2008.] http://www.adventuregamestudio.co.uk/acfeat.htm.

20. **Caituiro, Hillary.** *Memento.* [En línea] Noviembre de 2001. [Citado el: 2 de Febrero de 2009.] www.freewebz.com/amanecer/personal/papers/paper.memento.pdf.

21. **Welicki, León.** MSDN. *El Patrón Singleton.* [En línea] [Citado el: 27 de Enero de 2009.] http://msdn.microsoft.com/en-us/library/bb972272(es-es).aspx#EDAA.

Anexos

Anexo 1. Ejemplo de fichero .cvs

```
<sounds>
   <global_properties>
      <doopler_factor>1</doopler_factor>
      <speed_sound>1</speed_sound>
      <gain>1</gain>
      <distance_model>INVERSE_DISTANCE_CLAMPED</distance_model>
   </global_properties>
   <sound type="environmental" sound_name="sound 1" format="ogg" loop="yes">
      <name >Environmental1</name >
      <status>PLAY</status>
      <pitch>1</pitch>
      <gain min_gain="0" max_gain="1">1</gain>
      <level>0</level>
      <rool_of_factor>1</rool_of_factor>
      <direct_filter gain="1" gain_hf="1">NULL</direct_filter>
   </sound>
   <sound type="object" sound_name="sound 2" format="ogg" loop="yes" relative="no">
      <name >Box03</name >
      <status>STOP</status>
      <pitch>1</pitch>
      <gain min_gain="0" max_gain="1">1</gain>
      <level>0</level>
      <rool_of_factor>1</rool_of_factor>
      <direct_filter gain="1" gain_hf="1">NULL</direct_filter>
      <position x="912.523" y="551.181" z="4744.32" />
      <direction x="0" y="0" z="0" />
      <reference_distance>2000</reference_distance>
```

```
    <max_distance>1e+006</max_distance>
    <cone_innner_angle>360</cone_innner_angle>
    <cone_outer_angle>360</cone_outer_angle>
  </sound>
</sounds>
```

Anexo 2. Ejemplo de fichero .vsop

```
<visual_sound_project>
   <dir_scene>C:/Documents and Settings/Admin/Escritorio/ city.3DS</dir_scene>
   <general_status>STOP</general_status>
   <speed>1500</speed>
   <sounds>
      <global_properties>
         <doopler_factor>1</doopler_factor>
         <speed_sound>1</speed_sound>
         <gain>1</gain>
         <position x="606.328" y="647.938" z="2688.98" />
         <total_environmental>6</total_environmental>
         <orientation dir_x="0.99" dir_y="-0.031" dir_z="0.097" up_x="0.031" up_y="0.99" up_z="0.031" />
         <distance_model>INVERSE_DISTANCE_CLAMPED</distance_model>
      </global_properties>
      <sound type="environmental" sound_name="sound 5" format="ogg" loop="yes">
         <file>C:/Documents and Settings/Admin/Escritorio/ sound 1.ogg</file>
         <name>Environmental6</name>
         <status>PLAY</status>
         <pitch>1</pitch>
         <gain min_gain="0" max_gain="1">1</gain>
         <level>0</level>
         <rool_of_factor>1</rool_of_factor>
         <direct_filter gain="1" gain_hf="1">NULL</direct_filter>
      <sound type="object" sound_name="sound 2" format="ogg" loop="yes" relative="no">
         <file>C:/Documents and Settings/Admin/Escritorio/ 2.ogg</file>
         <name>Box03</name>
         <status>STOP</status>
         <pitch>1</pitch>
         <gain min_gain="0" max_gain="1">1</gain>
```

```
        <level>0</level>
        <rool_of_factor>1</rool_of_factor>
        <direct_filter gain="1" gain_hf="1">NULL</direct_filter>
        <position x="912.523" y="551.181" z="4744.32" />
        <direction x="0" y="0" z="0" />
        <reference_distance>2000</reference_distance>
        <max_distance>1e+006</max_distance>
        <cone_innner_angle>360</cone_innner_angle>
        <cone_outer_angle>360</cone_outer_angle>
      </sound>
   </sounds>
</visual_sound_project>
```

Anexo 3. Aplicación obtenida

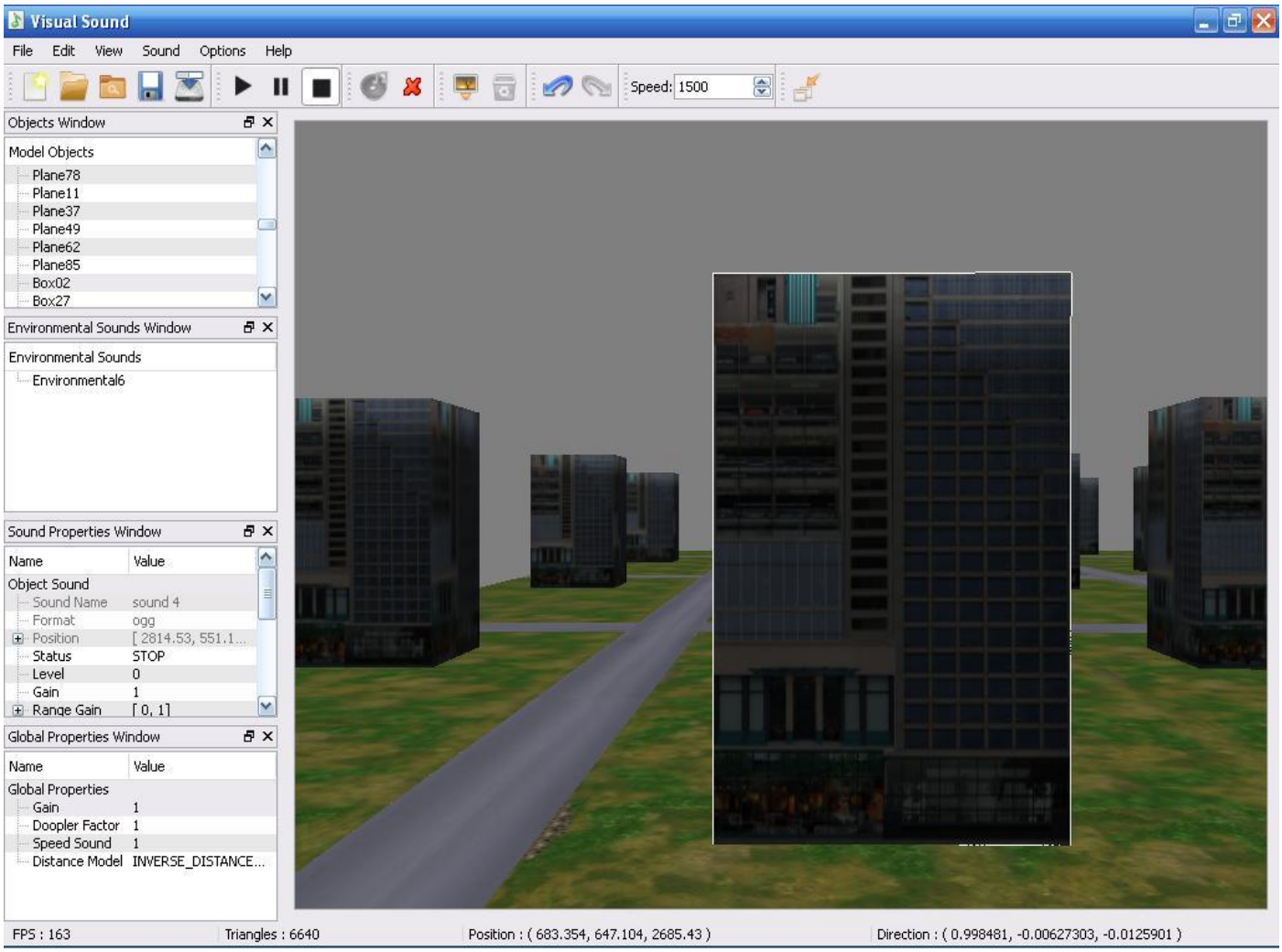

Figura 43: Aplicación obtenida

Glosario de términos

DirectX: es una colección de APIs creadas para facilitar las complejas tareas relacionadas con multimedia, especialmente programación de juegos y vídeo en la plataforma Microsoft Windows.

Doppler: consiste en la variación de la longitud de onda de cualquier tipo de onda emitida o recibida por un objeto en movimiento.

Entornos virtuales: se trata de la simulación de mundos o entornos, denominados virtuales, en los que el hombre interacciona con la máquina en entornos artificiales semejantes a la vida real.

Estéreo: Sistema de registro y reproducción que utiliza dos canales de información sonora: izquierdo y derecho.

Escalabilidad: es la propiedad deseable de un sistema que indica su habilidad de estar preparado para hacerse más grande sin perder calidad en los servicios ofrecidos. Capacidad del sistema informático de cambiar su tamaño o configuración para adaptarse a las circunstancias cambiantes.

***Framework*:** estructura de soporte definida, en la cual otro proyecto de software puede ser organizado y desarrollado.

Mono: Sistema de registro y reproducción que utiliza solo un canal de información sonora.

Multiplataforma: Que la aplicación corra en varios sistemas operativos.

OpenAL: *Open Audio Library*, en castellano significa: Biblioteca Abierta de Audio. Pretende ser una extensión de OpenGL que provea herramientas para el manejo de audio.

OpenGL: es una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D.

Plugin: un plugin (o plug-in, también conocido como addin, add-in, addon o add-on) es una aplicación informática que interactúa con otra aplicación para aportarle una funcionalidad o utilidad, generalmente muy específica.

Rendering: crear en forma automática una imagen de acuerdo al modelo tridimensional que existe en el ordenador.

Shader: conjunto de instrucciones gráficas destinadas para el acelerador gráfico, estas instrucciones dan el aspecto final de un objeto. Los *shaders* determinan materiales, efectos, color, luz, sombra.

Signals y slots: mecanismo de comunicación entre objetos, exclusivo de Qt.

Glosario de abreviaturas

3D: De tres dimensiones.

API: *Application Programming Interface* o Interfaz de Programación de Aplicaciones.

CUI: *Command Line User Interface* o Interfaz de Usuario de Línea de Comando.

GUI: *Graphic User Interface* o Interfaz Grafica de Usuario.

IDE: *Integrated Development Environment* o Entorno Integrado de Desarrollo.

GIMP: GNU *Image Manipulation Program* o Programa de Manipulación de Imágenes de GNU, es un programa de edición de imágenes.

GNOME: es un entorno de escritorio e infraestructura de desarrollo para sistemas operativos Unix/GNU/Linux, compuesto enteramente de software libre.

GNU: Proyecto iniciado por Richard Stallman con el objetivo de crear un sistema operativo completamente libre.

GPL: La GNU *General Public License* o Licencia Pública General de GNU, es una licencia creada por la *Free Software Foundation*.

GTK: GTK+ o *The GIMP Toolkit* es un grupo importante de bibliotecas o rutinas para desarrollar interfaces gráficas de usuario (GUI).

LGPL: La GNU *Lesser General Public License* o Licencia Pública General Reducida de GNU, es una licencia creada por la *Free Software Foundation*.

Qt: Es una biblioteca multiplataforma para desarrollar interfaces gráficas de usuario.

OGRE: *Object-Oriented Graphics Rendering Engine,* es un motor gráfico 3D.

SGML: *Standard Generalized Markup Language* o Lenguaje de Marcación Generalizado.

SRV: Sistema de realidad virtual.

XML: ***Ex****tensible* ***M****arkup* ***L****anguage* o Lenguaje de Marcas Extensible, es un metalenguaje extensible de etiquetas desarrollado por el *World Wide Web Consortium* (W3C).

Índice de figuras

Índice de tablas