Universidad de las Ciencias Informáticas

Facultad 5

Interfaz gráfica de usuarios para la biblioteca SceneToolKit

Presentado en opción al título de

Ingeniero en Ciencias Informáticas

Autora: Yusimi Santiesteban Pérez

Tutor: MSc. Yanoski Camacho Román

Cotutor: Ing. Alfredo Quintana López

La Habana

Junio de 2011

Declaración jurada de autoría

Declaramos ser los únicos autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Y para que así conste, firmamos la presente declaración jurada de autoría en Ciudad de la Habana a los ___ días del mes de junio del año 2011.

Yusimi Santiesteban Pérez
Autora

MSc. Yanoski Camacho Román
Tutor

Ing. Alfredo Quintana López
Cotutor

Datos de contacto

Tutor: MSc. Yanoski Camacho Román.
Edad: 31.
Ciudadanía: Cubano.
Institución: Universidad de las Ciencias Informáticas (UCI).
Título: Máster en Informática Aplicada.
Categoría Docente: Profesor Asistente.
E-mail: rcamacho@uci.cu
Graduado de Ingeniería en Informática en la CUJAE, con siete años de experiencia en el tema de la Gráfica Computacional.

Cotutor: Ing. Alfredo Quintana López.
Edad: 24.
Ciudadanía: Cubano.
Institución: Universidad de las Ciencias Informáticas (UCI).
Título: Ingeniero en Ciencias Informáticas.
Categoría Docente: Recién Graduado en Adiestramiento.
E-mail: aqlopez@uci.cu
Graduado de Ingeniería en Ciencias Informáticas en la UCI, con 5 años de experiencia en el tema de la Gráfica Computacional.

Agradecimientos

Le agradezco a mi familia que siempre me ha apoyado,
principalmente a mi mamita Rosa, a mi papá Ariel,
a mis hermanos Gardenia, Pito y Arielito.
A mis sobrinos Arianna, Leosbel, Marcy y Marquito,
que son las personas que más quiero en este mundo.

A mi mamá, por apoyarme siempre y estar ahí cuando más
la necesitaba. A mi papá, por su perseverancia y por darme
ánimos para seguir adelante, si hoy estoy aquí es por ellos.

Le agradezco a Yanoski,
que me ha ayudado mucho en momentos muy difíciles.
Por haber sido tutor, amigo, padre y madre.
Simplemente, gracias por encaminarme y apoyarme siempre.

A todos mis amigos.

A todos los que de una forma u otra
me han ayudado en la realización de este trabajo.

A todos, gracias.

Síntesis

El desarrollo de motores gráficos y de motores de videojuegos, constituye una base importante para la realización de aplicaciones de realidad virtual mundialmente, por la disminución de esfuerzo y tiempo de desarrollo que representan, como forma de reutilización de código.

El presente trabajo aborda el desarrollo de un módulo para la creación de interfaces gráficas de usuarios sobre la biblioteca SceneToolKit, implementada en el Centro de Informática Industrial CEDIN de la facultad 5, de la Universidad de las Ciencias Informáticas, UCI.

La solución propuesta permite el uso de ventanas basadas en SDL sin componentes visuales, o el uso de componentes visuales de CEGUI dentro de dichas ventanas. En ambos casos, se abstrae a los programadores de la detección de la ocurrencia de eventos de periféricos, y de la inyección de dichos eventos a CEGUI. El módulo implementado facilita el desarrollo de aplicaciones con mayores niveles de usabilidad e interacción de los usuarios con los videojuegos y entrenadores.

Palabras clave: Interfaz gráfica de usuarios, componentes visuales, motor gráfico, motor de videojuego, realidad virtual, videojuegos, *graphic engine*, *game engine*.

Índice de contenidos

Índice de figuras

Índice de tablas

Introducción

La Realidad Virtual (RV) ha alcanzado la madurez tecnológica suficiente como para convertirse en una valiosa herramienta al servicio de la industria. Los Sistemas de Realidad Virtual (SRV) comienzan a popularizarse, muchos productos empiezan a invadir el mercado en forma de simuladores, videojuegos, así como en la televisión y el cine, entre otras esferas.

En la Universidad de las Ciencias Informáticas (UCI), se comenzó a trabajar en el tema de la Realidad Virtual desde el 2003 con el desarrollo de un simulador de conducción de auto. A partir de ese momento surgieron nuevos compromisos, tanto simuladores como videojuegos, siendo necesario reutilizar código y unificar las funcionalidades comunes en sistemas conocidos en este caso como herramientas o motores gráficos (en inglés *engines*), sobre los cuales se montan las aplicaciones finales. Por esta razón se trabajó en un motor gráfico propio (SceneToolKit, STK de forma abreviada), con las funcionalidades básicas para desarrollar videojuegos y simuladores.

Entre los retos identificados durante la concepción y desarrollo de la STK, siempre se ha tenido en cuenta la implementación de los mecanismos de interacción[1] que típicamente intervienen en un videojuego [9]. Uno de ellos es el relacionado con la interacción a través de Interfaces Gráficas de Usuarios (GUI, del inglés *Graphic User Interfaces*), donde se usan componentes visuales como botones, menús, cajas de textos, entre otros, para que los usuarios puedan registrarse, configurar la aplicación, seleccionar opciones, etc.

[1]Sobre estos mecanismos se aborda con más detalle en el epígrafe 1.2 de este documento.

En la STK se han hecho intentos de implementación de componentes visuales, como es el caso de un trabajo de diploma cuyo resultado se continuó perfeccionando, pero nunca resultó efectivo pues el uso de estos componentes era muy engorroso. Es por esto que en la versión actual de la STK (10.07), en la que se hizo un rediseño de los módulos, se eliminaron las funcionalidades relacionadas con los componentes visuales.

Entonces, la **situación problémica** de esta investigación se resume en que la versión actual de la STK carece de mecanismos de interacción que permitan la autenticación, configuración de las aplicaciones, etc. (es decir, interfaces gráficas de usuarios), lo cual limita la calidad, flexibilidad y usabilidad de los productos finales desarrolladas con dicha biblioteca.

De aquí se deriva como **problema científico** a resolver: ¿Cómo mejorar la interacción de los usuarios con las aplicaciones finales desarrolladas con la STK? Dicho problema se enmarca dentro de los mecanismos de interacción de usuarios con aplicaciones de realidad virtual, como **objeto de estudio**.

Se plantea entonces como **objetivo** de este trabajo, desarrollar un mecanismo de interacción de usuarios con las aplicaciones finales hechas con la STK, basado en el uso de Interfaces Gráficas de Usuarios, teniéndose como **campo de acción**, los mecanismos de interacción de usuarios con aplicaciones de realidad virtual, basados en Interfaces Gráficas de Usuarios.

Para dar cumplimiento a este objetivo se plantearon las siguientes tareas:

- Análisis de bibliotecas para desarrollo de GUI usadas en el ámbito de la realidad virtual, para su caracterización.
- Caracterización de la arquitectura de la SceneToolKit, para determinar los elementos a tener en cuenta para la integración de un módulo de GUI.
- Selección de las bibliotecas para desarrollo de GUI a emplear para dar solución al problema.
- Desarrollo de soluciones técnicas acorde a las características de la SceneToolKit, aplicando las fases fundamentales de RUP.

Con la incorporación de un módulo para el manejo de Interfaces Gráficas de Usuarios a la SceneToolkit, se espera mejorar los mecanismos de interacción de los usuarios con las aplicaciones finales, logrando una mayor calidad, flexibilidad y usabilidad de los productos hechos con dicha biblioteca.

El presente documento se estructura de la siguiente manera:

En el Capítulo 1 se hace un estudio del estado del arte, caracterizando la SceneToolKit y algunas bibliotecas para desarrollo de interfaces gráficas de usuarios, planteándose finalmente una solución teórica al problema planteado.

En el Capítulo 2 se hace una descripción técnica de la solución a implementar, basado en pruebas de códigos realizadas para hacer una correcta integración del módulo a desarrollar, atendiendo a las peculiaridades de la SceneToolKit. En dicho capítulo se presenta además el modelo del dominio y las reglas del negocio.

En el Capítulo 3 se presentan los diagramas y artefactos correspondientes a la ingeniería realizada para dar solución al problema.

Finalmente se exponen las conclusiones y recomendaciones del trabajo, así como los glosarios de términos y la bibliografía consultada.

Capítulo 1

Fundamentación teórica

En el presente capítulo, se introducen algunos conceptos necesarios para comprender el objeto de estudio, y se analizan las características de la SceneToolKit y algunas bibliotecas para desarrollo de interfaces gráficas de usuarios, planteándose finalmente una solución tentativa al problema planteado.

1.1. Conceptos generales

Realidad virtual:

Según el diccionario de la Real Academia de la Lengua Española, realidad virtual es: "Representación de escenas o imágenes de objetos producidos por un sistema informático, que da la sensación de su existencia real". [11]

La realidad virtual se podría definir como un sistema informático que genera en tiempo real representaciones de la realidad, que de hecho no son más que ilusiones ya que se trata de una realidad perceptiva sin ningún soporte físico y que únicamente se da en el interior de los ordenadores. [5]

Se puede definir entonces realidad virtual, como un sistema tecnológico basado en el empleo de ordenadores y otros dispositivos, que produce una realidad aparente que en la cual los usuarios tengan la sensación de estar presente en ella.

Motores de videojuegos [2]

La complejidad gráfica añadida a la hora de tratar con escenarios 3D, hace necesaria la creación de un módulo gráfico independiente, lo bastante potente para tratar toda la información que hay en él, así como su visualización en tiempo real. De esta necesidad surge el concepto de motor gráfico (*graphic engine*).

Si se enriquece este módulo con elementos como física, redes o sonido, por ejemplo, se tiene lo que se llama motor de videojuego (*game engine*). Estos otros modelos también pueden presentarse de forma independiente dando lugar a un motor físico (*physics engine*) u otras capas o bibliotecas como pueden ser una capa de red para el desarrollo de videojuegos en red.

Un motor de videojuego es el sistema operativo o el núcleo del videojuego. Se encarga de dirigir y coordinar cada uno de los aspectos tecnológicos ligados a él. Capta eventos de entrada, hace cálculos físicos (colisiones, proyectiles), reproduce sonidos, simula inteligencia artifical, pinta, entre otros.

Interfaces de usuarios:

La interfaz de usuario (UI, por las siglas del inglés *user interface*) es uno de los componentes más importantes de cualquier sistema computacional, pues funciona como el vínculo entre el humano y la máquina. Es un conjunto de protocolos y técnicas para el intercambio de información entre una aplicación computacional y el usuario, responsable de solicitar comandos al usuario, y de desplegar los resultados de la aplicación de una manera comprensible. La UI no es responsable de los cálculos de la aplicación, ni del almacenamiento, recuperación y transmisión de la información. [14]

1.2. Mecanismos de interacción de usuarios con aplicaciones de realidad virtual

Los mecanismos de interacción que intervienen en un videojuego, y de forma general en las aplicaciones de realidad virtual de este tipo, se pueden clasificar –según la dirección del flujo de información que se da en los diferentes tipos de interacción– en: [9]

- Usuario a videojuego.
- Videojuego a usuario.
- Videojuego a videojuego.

En la interacción **usuario a videojuego**, se tiene en cuenta los periféricos de entrada de datos como el teclado, el ratón y el *joystick*, además de otras clases de interacción como el micrófono y los gestos.

La interacción **videojuego a usuario** abarca la forma en que se le facilita la información al usuario, usando la pantalla (lo que se conoce por usabilidad), y de forma general las **interfaces gráficas de usuarios** (ver más en el epígrafe 1.3, página 7). Además incluye el audio y los mecanismos de *feedback*[1].

[1]Mecanismos que permiten una retroalimentación al usuario a través de fuerzas o vibraciones con dispositivos especiales.

En la interacción **videojuego a videojuego**, se tratan los mecanismos de comunicación de los juegos como procesos, básicamente a través de la red.

De estas tres clasificaciones, las dos primeras están referidas a la interacción de los usuarios con los videojuegos, siendo de especial atención para este trabajo los mecanismos de interacción videojuego a usuario, donde se inserta el uso de interfaces gráficas de usuarios.

1.3. Interfaces gráficas de usuarios (GUI)

Las interfaces gráficas de usuarios surgen dada la necesidad de hacer más accesible el uso de los ordenadores para los usuarios, limitación que fue salvada gracias al desarrollo de los entornos gráficos, que permitieron que las personas pudieran acceder a las PC sin tener que pasar por el proceso de tener que aprender a manejar un entorno a base de consolas o líneas de comandos.

La interfaz gráfica de usuario, conocida también como GUI (del inglés *graphical user interface*) es un programa informático que actúa como interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Su principal uso consiste en proporcionar un entorno visual sencillo, para permitir la comunicación con el sistema operativo de la PC. [1]

Otras definiciones de interfaz gráfica de usuario son: “Componente de una aplicación informática que el usuario visualiza y a través de la cual opera con ella. Está formada por ventanas, botones, menús e iconos, entre otros elementos” [6], y “Es la interacción entre máquina-persona, que permite que el artefacto pueda ser usado de manera más amigable para la persona, pudiendo hacer elecciones más adecuadas e intuitivas”. [8]

Algunas características que posee una GUI, son las siguientes: [1]

1. Posee un dispositivo apuntador (típicamente un ratón).
2. Los usuarios pueden ver en la pantalla los gráficos y textos tal como se verán impresos.

3. Sigue el paradigma de la interacción objeto-acción.
4. Se puede manipular en la pantalla directamente los objetos y la información.
5. Provee elementos de interfaz estándar como menús y diálogos.
6. Existe una muestra visual de la información y los objetos (iconos y ventanas).
7. Proporciona respuesta visual a las acciones del usuario.
8. Existe información visual de las acciones y modos del usuario/sistema (menús, paletas).
9. Existen controles gráficos (*widgets*) para la selección e introducción de la información.
10. Permite a los usuarios personalizar la interfaz y las interacciones.
11. Proporciona flexibilidad en el uso de dispositivos de entrada (teclado/ratón).

Estas características serán tenidas en cuenta para la solución final, para que sea estándar, principalmente lo relacionado con el paradigma de la interacción objeto-acción.

En el capítulo "Sonido, Interacción y Redes", del libro "Fundamentos y programación de videojuegos" [9], se expone una breve teoría del diseño de interfaces gráficas, así como el uso y ejemplo de "metáforas"[2], y modelos a seguir para un diseño coherente de las GUI. Proponen como componentes básicos de una interfaz gráfica:

- Etiqueta
- Cuadro de texto
- Cuadro de confirmación
- Grupo de selección

[2]En el diseño de interfaces se usa el concepto de **metáfora** para definir comportamientos de elementos que componen la aplicación. La metáfora más conocida es la del escritorio de usuario, aunque están también los iconos, menús... un escenario 3D es una metáfora visual de un mundo real, como también lo es las formas de interactuar con éste.

- Barra de desplazamiento
- Lista de elementos

Con estos componentes básicos los usuarios pueden hacer la mayoría de las acciones típicas (seleccionar opciones, configurar la aplicación, entre otros), por lo que la solución a implementar debe contar al menos con estos componentes.

1.4. Características de la SceneToolKit (STK) a tener en cuenta en la integración de una GUI

Como se había enunciado en la introducción de este documento, la SceneToolKit (STK) es un motor creado en la facultad 5 para el desarrollo de aplicaciones de realidad virtual[3].

El mismo inicialmente se diseñó como un motor gráfico o un visualizador sencillo, pero posteriormente se le fueron añadiendo funcionalidades para el tratamiento de colisiones, sonidos, etc., por las necesidades prácticas de los primeros simuladores realizados. Con el tiempo tuvo que ser rediseñado como un motor de videojuego, con los módulos que típicamente incorporan éstos: módulo físico, inteligencia artificial, módulo de sonido, módulo de transmisión por redes, etc.[3]

En la versión actual (10.07), se tienen concebidos dos niveles, el motor gráfico y el motor de videojuego (ver figura 1.1), con la intención de separar las funcionalidades puramente gráficas del resto. Dentro del motor gráfico se tienen tres capas definidas: la capa **Núcleo**, que contiene todo el proceso de simulación: manejo de las estructuras de datos, actualización de objetos como geometrías, cámaras, luces, etc.; la capa ***Render***, que regula el proceso de dibujado; y la capa **Aplicación**, que actúa como interfaz con el programador de

[3]En realidad está concebido para ser un paquete de herramientas. El nombre se forma por *scene*, escena, y *toolkit*, "paquete de herramientas". El nombre completo de la STK es "Herramientas para Desarrollo de Sistemas de Realidad Virtual", y debe constar del motor de videojuego y otras herramientas utilitarias y de edición.

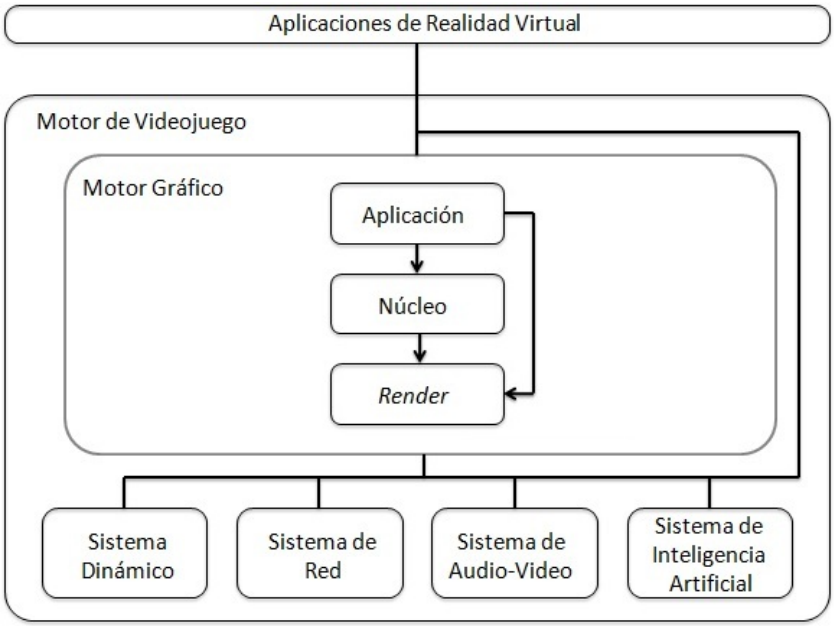

Figura 1.1: Arquitectura de la STK (imagen tomada del informe del proyecto).

aplicaciones finales, dándole acceso a las funcionalidades de la misma. A lo largo de estas capas se ejecuta un sistema para tratamiento de errores y generación de ficheros de eventos (*logs*).

La STK incluye además, en la capa de Aplicación, funcionalidades para el manejo de ventanas y de eventos de entrada de periféricos (teclado y *mouse*) sobre varios sistemas operativos, haciendo uso de la biblioteca SDL (*Simple DirectMedia Layer*)[4]. Dichas funcionalidades actualmente se encuentran desorganizadas.

Según la arquitectura definida en el proyecto, el módulo a desarrollar para las interfaces gráficas de usuarios debe insertarse en la capa de aplicación, manteniendo el uso de SDL. Además debe ser multiplataforma (es decir, ejecutarse al menos sobre Windows y GNU/Linux, y usando las bibliotecas OpenGL y DirectX), y debe mantenerse bajo los principios del *software* libre.

[4]SDL (*Simple DirectMedia Layer*) es una biblioteca multiplataforma desarrollada en C que proporciona funciones básicas para acceder al audio, teclado, *mouse*, *joystick*, y para realizar operaciones de dibujado 2D, gestión de efectos de sonido y música, y carga y gestión de imágenes. Ver más en http://www.libsdl.org/.

1.5. Algunas bibliotecas de GUI

A continuación se presentarán algunos ejemplos de bibliotecas para interfaces gráficas de usuarios (o que contienen funcionalidades para las mismas), populares por su fácil uso por parte de desarrolladores, y/o por los resultados que brindan en cuanto a aceptación por parte de usuarios finales.

1.5.1. Visual Components Library (VCL) de Borland

En 1995, Borland lanzó al mercado Delphi, que representó una revolución en la programación para Windows e inició el desarrollo rápido y sencillo de aplicaciones visuales, empleando como lenguaje base al Object Pascal. Desde el punto de vista técnico, resultó relevante la creación de la biblioteca VCL, un marco de trabajo para crear aplicaciones Windows diseñadas en torno al concepto de componente (propiedades, métodos y eventos). [4]

Aunque la VCL es privativa y no es multiplataforma, por lo que no será usada en la solución final de este trabajo, su estudio puede ser válido para una mejor comprensión del funcionamiento de las GUI.

El objetivo final de la VCL es crear clases que representan a componentes. Aunque algunas clases no hagan referencia a componentes concretos, realizan tareas de gestión interna y se emplean como clases bases para derivar mediante herencia otras clases. La VCL ha demostrado estar bien diseñada y el control que se tiene a través de los eventos de la misma es suficiente para la gran mayoría de aplicaciones. [4]

La figura 1.2 muestra la jerarquía de clases de la VCL de Borland. Las clases que conforman la parte superior de la jerarquía de la biblioteca VCL (TObject, TPersistent y TComponent), se denominan clases “abstractas” porque sirven para estructurar y agrupar comportamientos comunes de las clases de la biblioteca VCL. No se suelen crear objetos directamente a partir de ellas. En la parte inferior derecha se muestran clases relacionadas con componentes visuales, y en la izquierda las no visuales.

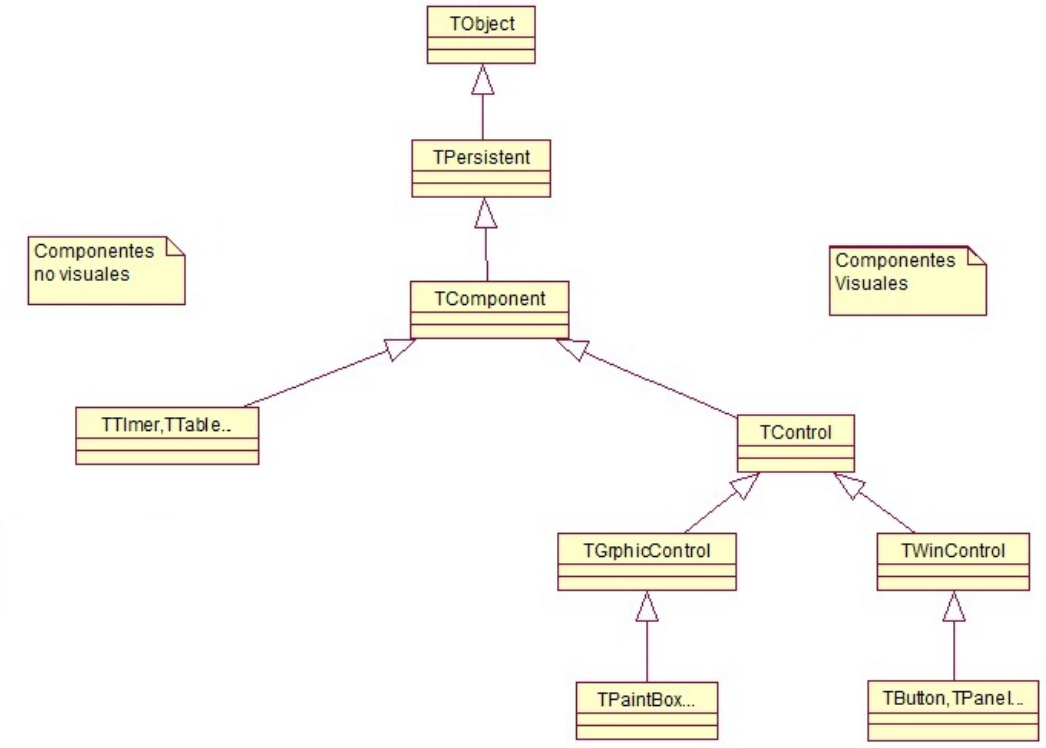

Figura 1.2: Principales clases de VCL de Borland. A la izquierda los componentes no visuales y a la derecha los visuales.

Los principales componentes de la VCL son:

TObject: Es el ancestro de todas las clases de la biblioteca VCL. Encapsula el comportamiento común de los objetos en C++ Builder.

TPersistent: Esta clase tiene que ver con la habilidad de un objeto de almacenarse en disco o en memoria.

TComponent: Esta clase proporciona toda la funcionalidad que requiere un componente básico.

TControl: Proporciona la funcionalidad de los componentes visuales (controles). Esta funcionalidad es principalmente el aspecto visual que deben ofrecer los componentes en tiempo de ejecución.

TWinControl: Son los controles típicos de Windows que pueden recibir el foco, contener otros controles y además poseen un manejador de ventana.

TApplication: Simplifica la interfaz entre el programador y el sistema operativo, ocupándose de tareas como gestionar el paso de mensajes, proporcionar la ayuda contextual, establecer los textos de sugerencia de los botones y barras de estado, procesamiento de "teclas rápidas", gestión de excepciones, ejecutar cuadros de mensajes, etc.

TForm: La clase TForm caracteriza a los formularios en la biblioteca VCL.

1.5.2. Microsoft Foundation Class Library (MFC) de Microsoft

La *Microsoft Foundation Class Library* (MFC), es el marco de trabajo desarrollado por *Microsoft* e incorporado a los compiladores *Microsoft Visual C++*. [4]. La MFC es una colección de clases que pueden ser usadas en la realización de programas de aplicación y también para todos los elementos de interfaz del usuario. Las clases en la biblioteca MFC están escritas en el lenguaje de programación C++. La biblioteca MFC le ahorra tiempo al programador pues le provee código que ya ha sido escrito. Esta biblioteca está compuesta por cientos de clases distintas, que desde el punto de vista funcional, se pueden dividir en cuatro grupos: [15]

1. Clases orientadas al interfaz de usuario, representan ventanas, menús, diálogos, etc.
2. Clases de propósito general, representando ficheros, strings, datos de fecha y hora, etc.
3. Clases orientadas a bases de datos, representando tanto bases de datos como conjuntos de registros seleccionados después de una consulta sobre una tabla, etc.
4. Clases para manejo de excepciones, para control avanzado de errores de ejecución.

Esta clasificación de clases puede ser útil para la solución final, pero no se podrá usar la MFC en dicha solución ya que es privativa y no es multiplataforma.

1.5.3. Gimp Tool Kit (GTK+)

GTK+ ("*GIMP Tool Kit*") [12], es un conjunto de bibliotecas para desarrollo de interfaces gráficas de usuario, inicialmente creada para desarrollar GIMP, aunque actualmente cuenta con varias bibliotecas del equipo GTK+ y de GNOME. En especial, la biblioteca GTK es la que realmente contiene objetos y funciones para crear la interfaz gráfica de usuario, maneja *widgets* como ventanas, botones, menús, etiquetas, pestañas, etc.

Otras bibliotecas de GTK+ son[5]:

GLib: Biblioteca de bajo nivel, proporciona manejo de estructura de datos para C, interfaces para funcionalidades de tiempo de ejecución como ciclos, hilos, carga dinámica, etc.

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

ATK: Biblioteca para crear interfaces más accesibles a personas discapacitadas o minusválidos, con utilerías como lupas de aumento, lectores de pantalla, o entradas de datos alternativas al clásico teclado o ratón.

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

Cairo: Biblioteca de renderizado avanzado de controles de aplicación.

GTK+ está concebida principalmente para GNU/Linux y Unix aunque también se puede usar en el escritorio de Windows y MacOSX. Es una de las bibliotecas más populares para X Window System.

Se ha diseñado para permitir programar con lenguajes como C, C++, C#, Java, Ruby, Perl, PHP o Python.[6]

La última versión liberada estable es la 2.22.0 (23 de septiembre de 2010), aunque se está probando la versión 2.91.4 (9 de noviembre de 2010). Está bajo licencia LGPL, es *software* libre y es parte del proyecto GNU.

[5] Ver más sobre estas bibliotecas, así como las versiones liberadas de cada una, en http://www.gtk.org/documentation.html.
[6] Ver todos los lenguajes en http://www.gtk.org/language-bindings.html.

Las principales clases de GTK+ [7] se muestran en la figura 1.3.

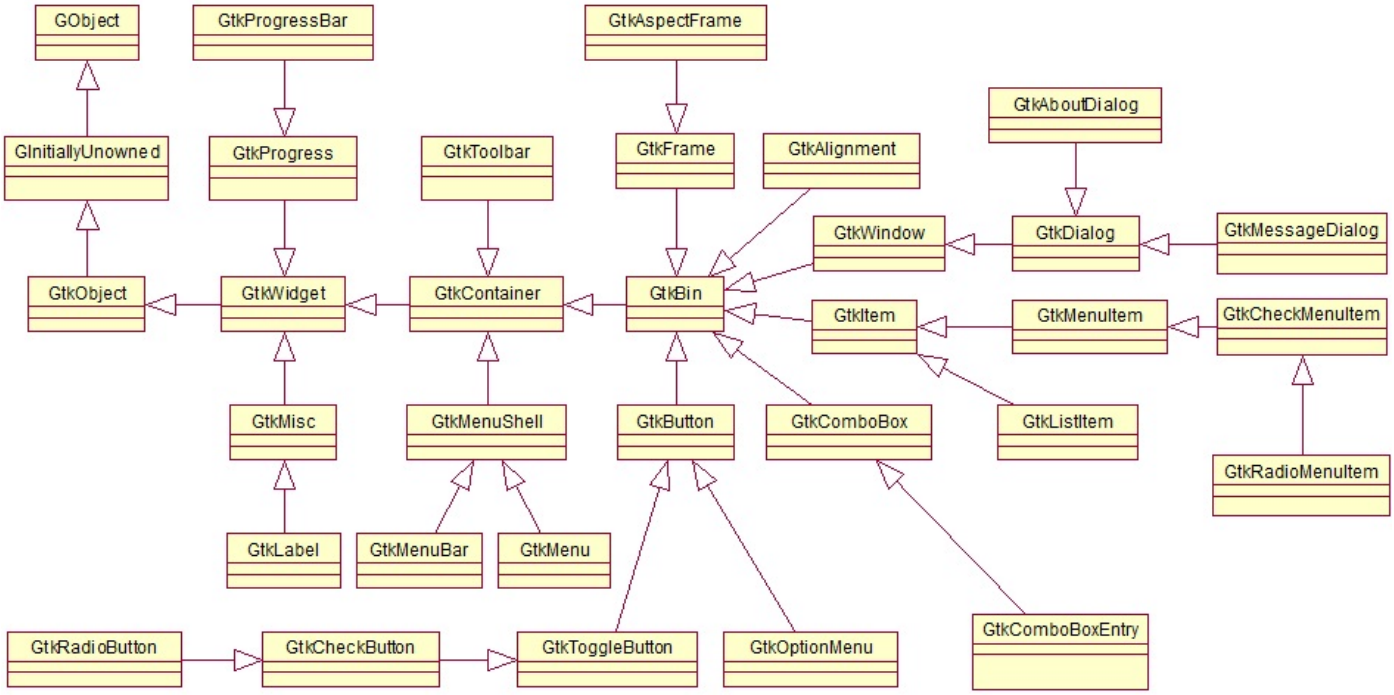

Figura 1.3: Principales clases de GTK+.

De estas, se pueden destacar:

GObject: Clase base de todas las demás.

GtkObject: Clase base de la jerarquía de clases del tipo GTK+.

GtkWidget: Clase base de todos los *widgets*.

GtkWindow: Clase que puede contener a otros *widgets*.

[7] Ver más clases de GTK+ en http://library.gnome.org/devel/gtk/stable/ch01.html.

El hecho de que GTK+ sea multiplataforma y cumpla con los principios del *software* libre, la convierte en una buena candidata para ser usada en la solución a implementar.

1.5.4. Qt

Qt [13] es una biblioteca multiplataforma destinada al desarrollo de interfaces gráficas de usuario. Creada por la compañía noruega Trolltech en 1992, siguió un desarrollo basado en código abierto pero no en código libre. Esta biblioteca es mayormente utilizada en entornos de escritorio KDE, llegando a alcanzar un notable éxito y rápida expansión entre 1996 y 1998, aunque también es perfectamente funcional en otros escritorios como Gnome. En el 2008 es adquirida por Nokia, y desarrollada por Qt Software, una división de Nokia.

Hace uso del lenguaje C++ de forma nativa, además existen interfaces para C, Python (PyQt), Java (QtJambi), Perl (PerlQt), Gambas (Gb.Qt), Ruby (QtRuby), PHP (PHP-Qt), Mono (Qtoto), entre otros. [13]

Actualmente cuenta con un sistema de triple licencia[8], GPL v2 y GPL v3 para el desarrollo de *software* de código abierto (*open source*) y *software* libre, y otra licencia de pago para el desarrollo de aplicaciones comerciales.

Recientemente Nokia está en proceso de venta de la parte comercial a Digia, dándole los derechos de vender licencias comerciales y ofrecer servicios profesionales y soporte a la comunidad de Qt. Esta parte comercial es importante pero no es la mayoría de Qt, por lo que Nokia sigue investigando y desarrollando en la misma bajo licencias de código abierto, y continúa con los derechos de "*copyright*".[9]

Actualmente Qt se encuentra en su versión 4.6.3[10], y además de las múltiples mejoras que se le han realizado, ahora las bibliotecas Qt son también liberadas bajo la licencia GPL para Sistemas Windows y Mac.

[8] Ver licencias de Qt en http://qt.nokia.com/products/licensing/

[9] Ver noticia en http://blog.qt.nokia.com/2011/03/14/qt-and-digia-facts-and-fiction/.

[10] La última versión liberada es la 4.6.3, del 8 de junio de 2010, disponible en: http://qt.nokia.com/about/news/nokia-releases-qt-4.6.3.

Es posible utilizar Qt para desarrollo de videojuegos[11], aunque no es el propósito principal de este *framework* y que esto puede traer algunos inconvenientes, y además se tienen poca documentación en este aspecto. No obstante, Qt ofrece herramientas y posibilidades como las siguientes:

Graphics View Framework: Ofrece wigdets que facilitan el dibujo y manipulación de gráficos 2D. También facilita la detección de colisiones.[12]

State Machine Framework: Un framework para la implementación de máquinas de estados. Un videojuego puede ser visto como una máquina de estados.[13]

Qt OpenGL Module: Facilita la integración de OpenGL en aplicaciones Qt.[14]

Qt Quick: Herramienta que ofrece y un lenguaje sencillo (similar a javascript), y que facilita la creación de transiciones y efectos. Está basada en el Graphics View Framework.[15]

En el sitio oficial de Qt se presentan algunos juegos que hacen uso de dicha biblioteca[16]

El uso de Qt en videojuegos le da ventaja sobre GTK+, pues aunque no se tenga mucho conocimiento al respecto, sirve al menos como garantía de que se puede usar en videojuegos.

Qt brinda un grupo de clases[17] según se muestra en la figura 1.4.De estas:

QObject: Es la clase base de todos los objetos de Qt.

QPaintDevice: Es la clase base de todos los objetos que pueden ser dibujados.

[11] http://www.zonaqt.com/foro/desarrollo-de-videojuegos-con-qt.
[12] http://doc.qt.nokia.com/latest/graphicsview.html.
[13] http://doc.qt.nokia.com/latest/statemachine-api.html.
[14] http://www.zonaqt.com/foro/doc.qt.nokia.com/latest/qtopengl.html.
[15] http://doc.qt.nokia.com/4.7-snapshot/declarativeui.html.
[16] http://developer.qt.nokia.com/wiki/Qt_Based_Games.
[17] Ver clases de Qt en http://doc.qt.nokia.com/4.0/mainclasses.html.

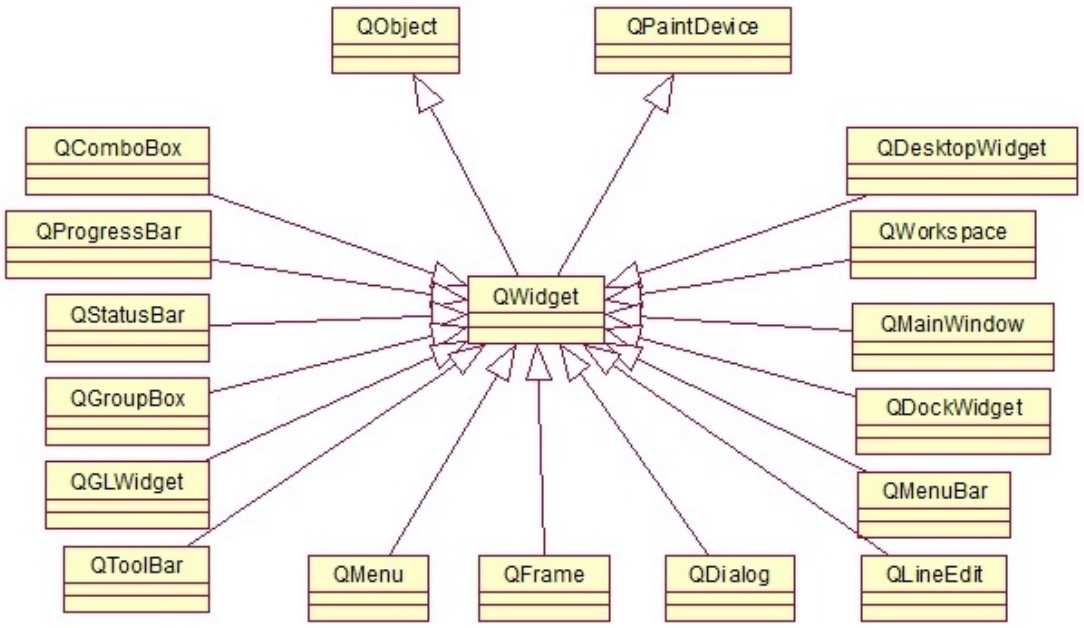

Figura 1.4: Principales clases de Qt.

QWidget: Es la base para todos los objetos de interfaces de usuarios[18]. QWidget hereda de QObject y de QPaintDevice.

QGLWidget: Es el *widget* para la representación de gráficos con OpenGL.

Los principales atributos de QWidget (accesibles a través de properties), son[19]: cursor (QCursor), enabled (bool), focus (bool), font (QFont), frameGeometry (QRect), frameSize (QSize), fullScreen (bool), geometry (QRect), height (int), isActiveWindow (bool), maximized (bool), maximumHeight (int), maximumSize (QSize), maximumWidth (int), minimized (bool), minimumHeight (int), minimumSize (QSize), minimumSizeHint (QSize), minimumWidth (int), palette (QPalette), pos (QPoint), rect (QRect), size (QSize), sizeHint

[18]Ver documentación de QWidget en: http://doc.qt.nokia.com/4.0/qwidget.html.
[19]Ver más sobre atributos de QWidget en http://doc.qt.nokia.com/4.0/qwidget.html.

(QSize), sizeIncrement (QSize), visible (bool), width (int), windowFlags (Qt::WindowFlags), windowIcon (QIcon), windowModified (bool), windowOpacity (double), windowTitle (QString), x (int), y (int).

Cada clase heredera incorpora atributos propios según su funcionamiento.

Para la **implementación de eventos** con Qt, se trabaja con los *signals* y *slots* [7][20]. Los ***signals*** son "señales" emitidas por los objetos cuando ocurre un cambio o evento. Solamente tienen acceso a éstos las clases en las que estén definidos así como las que hereden de éstas. Cuando un *signal* es emitido los *slots* asociados son ejecutados inmediatamente. Es decir, los *slots* responden a los cambios ocurridos y son "avisados" a través de los *signals*. Algunos *signals* predeterminados son: clicked(), fineshed() y rejected().

Los ***slots*** se ejecutan cuando es invocado un *signal*. Se usan normalmente de manera similar a como se usa un método en C++, es decir, pueden ser llamado para otras funcionalidades, ser virtuales, etc., con la diferencia de que pueden ser asociados a uno o varios *signals*. Para relacionar los *signals* y *slots* se utiliza el método ***connect***, al cual se le pasa como parámetros:

- El objeto del cual se va a recibir la señal "signal".
- La señal dentro de la palabra reservada SIGNAL.
- El objeto que va a ejecutar el "slot".
- El slot dentro de la palabra reservada SLOT.

Por ejemplo, para limpiar una caja de texto "lineEdit" cuando se da clic sobre el botón "PushButton" la sentencia sería:

```
connect(PushButton, SIGNAL(clicked()), lineEdit, SLOT(lineEdit->Clear()));
```

[20] Ver FAQ sobre *signals* y *slots* en http://developer.qt.nokia.com/search/tag/signal&slots y en http://developer.qt.nokia.com/search/tag/signals+and+slots.

Ejemplos de slots predeterminados son: exec(), reject(), done(), show() y close(). Los principales *slots* de QWidget son[21]:

- bool close ()
- void hide ()
- void lower ()
- void raise ()
- void repaint ()
- void setDisabled (bool disable)
- void setEnabled (bool)
- void setFocus ()
- void setHidden (bool hidden)
- virtual void setVisible (bool visible)
- void setWindowModified (bool)
- void show ()
- void showFullScreen ()
- void showMaximized ()
- void showMinimized ()
- void showNormal ()
- void update ()

[21] Ver descripción de los *slots* de QWidget en http://doc.qt.nokia.com/4.0/qwidget.html.

Se pueden crear *signals* y *slots* propios para funcionalidades que no vienen predefinidas. Para esto es necesario declararlos en sus respectivos bloques:

```
public slots:
  Metodo1();
  Metodo2();
signals:
  Metodo3();
  Metodo4();
```

El mecanismo de los *signals* y *slots* le da a Qt otra ventaja sobre GTK+, pues le brinda una mayor facilidad de uso, todo lo contrario a GTK+ que es muy difícil su aprendizaje.

1.5.5. Crazy Eddie's GUI System (CEGUI)

CEGUI (*Crazy Eddy Fraphic User Interface*)[10], es una biblioteca libre, gratuita y multiplataforma para el manejo de ventanas y *widgets* para las API gráficas y motores en los que esa funcionalidad no está disponible. No tiene su propio "*input*", es decir, no hace captura de eventos por lo que hay que inyectárselos para que se ejecuten las acciones sobre los componentes.

Fue creada por Paul Turner ("CrazyEddie") en el 2003, y la última versión liberada es la 0.7.4 de Octubre 2010[22]. Es orientada a objetos, escrita en C++, y orientada fundamentalmente al desarrollo de videjuegos.

Hasta la versión 0.4.1, CEGUI estaba bajo licencia LGPL, pero a partir de la 0.5 se distribuye bajo la licencia MIT, que es menos restrictiva que la LGPL[23].

[22] Ver http://www.cegui.org.uk/wiki/index.php/FAQ.
[23] Ver http://www.cegui.org.uk/wiki/index.php/FAQ.

Otras de sus características son[24]:

- Es compatible con Windows, Linux y MacOS.
- Brinda soporte para arquitecturas de 32 bits y 64 bits
- Maneja datos mediante archivos XML (*Extensible Markup Language*).
- Soporta Unicode - utf8 y UTF32.
- Posee *renderers* para OpenGL, DirectX 9, DirectX 10, DirectX 11, Irrlicht, Ogre3D, Crystal Space (proporcionado por el equipo de Crystal Space), OpenSceneGraph (proporcionado por el equipo de OpenSceneGraph) y DirectFB (versión experimental).

La figura 1.5 muestra las clases más comunes de CEGUI[25]. De ellas:

PropertyReceiver: Es la clase base que se asegura del *casting* correcto de los *receivers*.

PropertySet: Contiene una colección de objetos *Property*.

Font: Encapsula los tipos de letras (*typeface*).

XMLParser: Es una interface con las bibliotecas de parseo XML.

Window: Clase abstracta que brinda las funcionalidades comunes para las clases derivadas que conforman los principales controles visuales.

FrameWindow: Clase abstracta para ventanas móviles y de tamaño variable con una barra de título y un marco.

GUISheet: Clase para el manejo de ventanas simples y genéricas.

[24]Ver más características en http://www.cegui.org.uk/wiki/index.php/Features.
[25]Ver todas las clases y sus descripciones en http://www.cegui.org.uk/api_reference/functions_0x62.html#index_b.

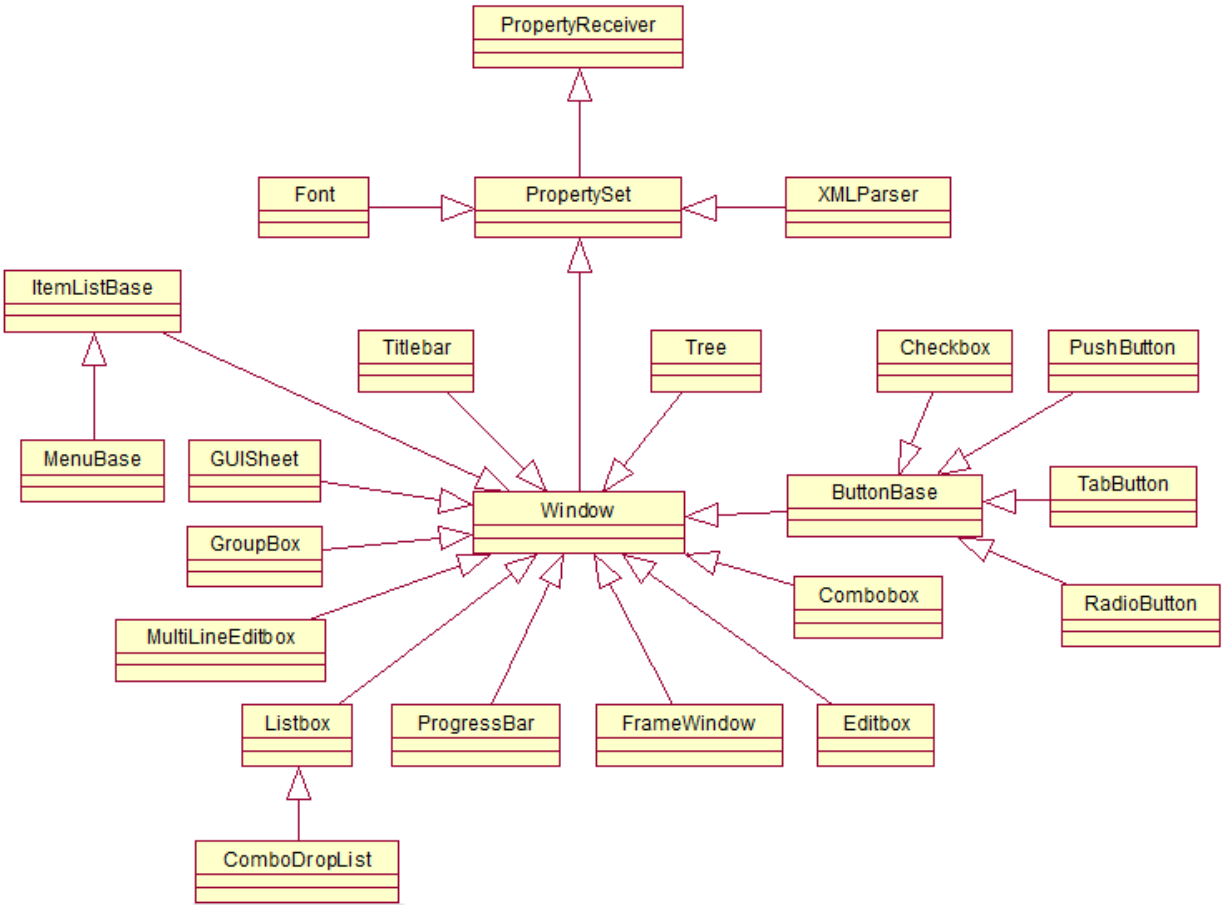

Figura 1.5: Principales clases de CEGUI.

Los principales atributos y métodos de CEGUI::Window son[26] (por cada "*set*" especificado existe el correspondiente "*get*"):

- bool isDisabled (bool localOnly), void setEnabled (bool setting)
- void enable (void), void disable (void)
- bool isVisible (bool localOnly=false), bool isActive ()
- uint getID, void setID (uint ID)
- size_t getChildCount (void)
- bool isChild (const String &name)
- Window* getChild (uint ID), Window* getActiveChild (void)
- Font* getFont (bool useDefault=true)
- Window* getParent (void)
- VerticalAlignment getVerticalAlignment ()
- HorizontalAlignment getHorizontalAlignment ()
- void rename (const String &new_name)
- void setVisible (bool setting)
- void show (void)
- void hide (void)
- void activate (void), void desactivate (void)

[26]Ver descripción de CEGUI::Window en http://www.cegui.org.uk/api_reference/classCEGUI_1_1Window.html#a58abe594a1ff0ff5c781d9379f3e4c77.

- void setText (const String &text)
- void setFont (Font* font)
- void setMouseCursor (const Image* image)
- void setVerticalAlignment (const VerticalAlignment alignment)
- void setHorizontalAlignment (const HorizontalAlignment alignment)
- void setArea: tiene varias implementaciones con diferentes parámetros
- void setPosition, void setXPosition, void setYPosition
- void setSize, void setWidth, void setHeight
- void setMaxSize, void setMinSize

Por sus características, CEGUI puede ser una buena solución al problema planteado en este trabajo, pues es libre, multiplataforma y muy usada en videojuegos. Además, tiene una característica muy importante y es que sus componentes visuales son de cierta forma "artísticos".

1.6. Solución tentativa

En la tabla 1.1 se presenta una pequeña comparación de los aspectos fundamentales de las bibliotecas estudiadas, como apoyo a la selección de las que se puedan usar para su integración con la STK.

Bajo el principio de que la solución debe ser libre y multiplataforma, quedan descartadas la VCL de Borland y la MFC de Microsoft, además de que no tienen un uso conocido en el desarrollo de videojuegos.

Teniendo en cuenta el aspecto visual de los controles brindados, se aprecia que tanto GTK+ como Qt brindan componentes tradicionales (como los de las ventanas estándares de los sistemas operativos, ver figura C.1 en

Aspecto	VCL	MFC	GTK+	Qt	CEGUI
Licencia	Privativa	Privativa	Libre	Libre	Libre
Uso de C++	Sí	Sí	Sí	Sí	Sí
Multiplataforma	Windows	Windows	Sí	Sí	Sí
Uso en VJ	Desconocido	Desconocido	Desconocido	Sí	Muy usado
Componentes	Tradicionales	Tradicionales	Tradicionales	Tradicionales	Artísticos

Tabla 1.1: Comparación de las GUI estudiadas.

la página 72), mientras que CEGUI, debido a que sus componentes se basan en imágenes, es más flexible para la creación de componentes visuales más artísticos y por tanto más apropiados para videojuegos (figura C.3).

En cuanto a GTK+ y Qt, no se encuentra información seria que permita hacer una comparación en cuanto al uso de uno o el otro, sin embargo en algunos foros[27] se favorece Qt, con razones como que:

- GTK+ no tiene soporte para renderizado directo ni indirecto como sí lo tiene Qt4, siendo notable la diferencia de velocidad de dibujado.
- GTK+, por ser en C es más flexible, pero implementa a través de C muchas cosas de C++ (como la herencia), y por ello puede ser incluso más engorroso de usar que empleando directamente C++.
- Qt tiene más funcionalidades y las herramientas son mejores.

Por otra parte, Nokia ha anunciado entre las nuevas prestaciones a implementar para Qt, la relacionada con el rendimiento:[28] "El rendimiento es otra de las prioridades, que ha hecho que los desarrolladores experimenten con un nuevo sistema OpenGL que permitirá un renderizado más rápido: mientras que ahora QML se basa en QGraphicsView y QPainter como los intermediarios para el renderizado, el uso de OpenGL podría ofrecer

[27] Ver: http://www.espaciolinux.com/foros/debates/gtk-t25147.html y http://softwarelibre.uca.es/node/822.
[28] Ver más en http://www.muylinux.com/2010/11/04/el-futuro-de-qt-al-descubierto/.

ventajas importantes". Este es un factor muy importante a considerar para la selección de una de estas bibliotecas para su integración con un sistema de realidad virtual, basado en el renderizado.

Además, la tendencia en el Departamento de Visualización y Realidad Virtual del CEDIN[29] es abandonar los desarrollos sobre GTK y orientarse más al uso de Qt, pues GTK+ a pesar de tener una extensa documentación, resulta complejo su aprendizaje. En cuanto al tratamiento de las señales, GTK+ es poco intuitivo, mientras que en Qt resulta más fácil de usar por los *signals* y los *slots*, por lo que finalmente se puede seleccionar Qt, comparada con GTK+, como mejor candidata para su integración con la STK.

Atendiendo a todo lo anterior, una solución ideal sería usar dos bibliotecas, Qt y CEGUI, y de esta manera brindarle al programador la posibilidad de desarrollar aplicaciones con componentes tradicionales o artísticos.

Conclusiones del capítulo

Partiendo del estudio realizado en este capítulo, se concluye que una buena solución sería el uso de Qt y GEGUI para incorporarle a la STK funcionalidades para el manejo de interfaces gráficas de usuarios.

[29]Departamento encargado de desarrollar soluciones para la visualización tridimensional y de realidad virtual, dentro del centro de Informática Industrial CEDIN de la facultad 5.

Capítulo 2

Descripción técnica y modelo del dominio

Partiendo de la posible solución planteada en el capítulo anterior, en este capítulo se hace una descripción de la propuesta final, donde se tiene en cuenta el uso de CEGUI, permitiendo también el trabajo con ventanas simples sin componentes visuales.

Se presenta además el modelo del dominio y el glosario de términos asociado, así como las reglas del negocio.

2.1. Descripción general

Existe una diferencia en cuanto al uso de Qt y CEGUI: Qt es un *framework* que controla a la aplicación y por tanto no se puede incluir en otra biblioteca, sino que se debe desarrollar sobre él, por lo que la STK no puede incluir a Qt para hacer una ventana haciendo uso de sus componentes. En cambio la biblioteca CEGUI sí se puede incluir dentro de la STK.

Por otra parte, partiendo de pruebas de código realizadas con el objetivo de comprobar la posible unión de CEGUI y de Qt con la STK, se determinó que en este último caso el acople resulta más complejo puesto que la STK usa a SDL para el trabajo con las ventanas y los eventos de periféricos, y al incluir la STK en una aplicación de Qt da error de redefinición del "*main*" [1], obligando a buscar la manera de no incluir la SDL dentro de la STK. Es por esto que finalmente se tendrán en cuenta dos elementos:

1. Se incluirá CEGUI en la STK: se usarán componentes visuales de CEGUI dentro de la ventana STK.
2. Se permitirá usar ventanas simples sin componentes visuales, es decir, sin usar CEGUI.

La solución con CEGUI se programará en un módulo, de forma que se pueda prescindir de dichas funcionalidades sin afectar el resto del funcionamiento de la STK.

2.2. Modelo del dominio

Para dar soporte a estas variantes, se propone el **modelo de dominio** representado en la figura 2.1:

[1] error LNK2005:_WinMain@16 already defined in qtmaind.lib(qtmain_win.obj), File SDLmain.lib

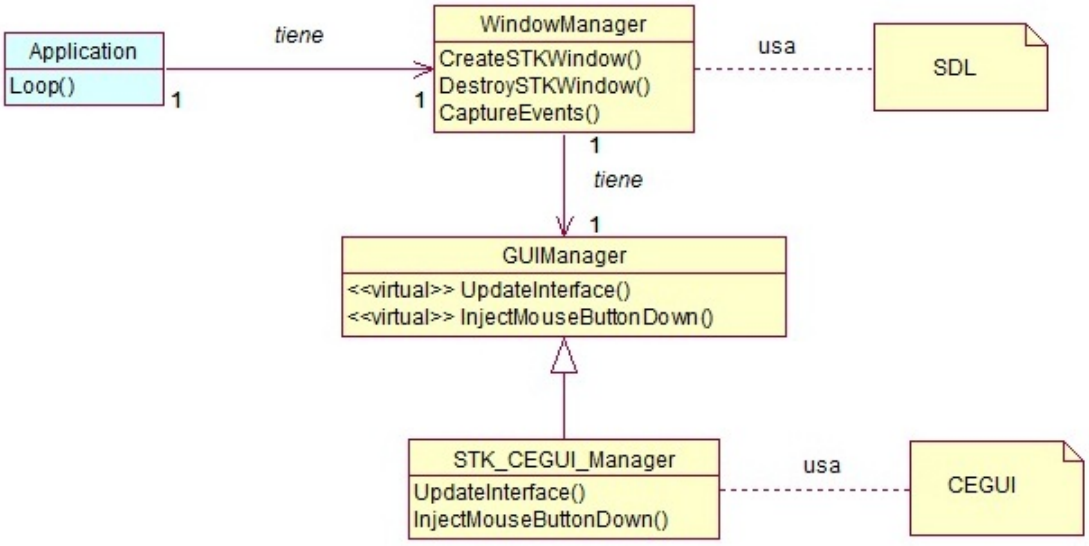

Figura 2.1: Modelo de dominio.

2.2.1. Glosario de términos del modelo del dominio

Application: es la clase de la STK que actúa como interfaz con los programadores, ya existe en la versión actual de la STK.

WindowManager: en Application existen algunas funcionalidades para crear ventanas y capturar eventos de teclado y *mouse* haciendo uso de SDL, pero estas funcionalidades están actualmente desorganizadas. Se propone entonces crear **WindowManager** para encapsular las funcionalidades de trabajo con la ventana (CreateSTKWindow) y los periféricos (CaptureEvents). WindowManager hace uso de la biblioteca externa SDL.

GUIManager: se especializa en la interacción con las interfaces gráficas de usuarios. Contiene métodos virtuales (UpdateInterface, InjectMouseButtonDown, etc.) que deberán ser reimplementados por clases herederas.

STK_CEGUI_Manager: actúa como interfaz con CEGUI, y hereda de GUIManager, reimplementando fun-

ciones virtuales de la clase padre, como actualizar los componentes visuales (UpdateInterface), e "inyectar" a CEGUI los eventos detectados por SDL (InjectMouseButtonDown, etc.)

2.3. Solución haciendo uso de CEGUI

- En esta variante, la escena virtual se dibuja sobre la ventana creada con SDL.
- STK_CEGUI_Manager permitirá introducir componentes visuales de CEGUI dentro de la ventana de SDL.
- El manejo de eventos se hará como muestra la figura 2.2: en el ciclo de escena de la STK (Loop), se chequea la existencia de eventos de periféricos a través de WindowManager que a su vez se informa con SDL. Si ha ocurrido un evento, se informa a CEGUI a través de STK_CEGUI_Manager, con lo cual se ejecutarán las acciones programadas en los eventos de los componentes de CEGUI. De todas maneras, se mantiene la ejecución del evento correspondiente de la STK, por si el programador desea hacer uso de ellas en la aplicación final como se hacía antes de incorporar el módulo de GUI.
- La actualización de la GUI se hará como muestra la figura 2.3: en el ciclo de escena de la STK (Loop), después de manejar los eventos de los periféricos, se manda a actualizar la interfaz de CEGUI (renderGUI) a través del UpdateInterface de STK_CEGUI_Manager (método virtual en GUIManager).

2.4. Solución sin CEGUI, ventanas simples

Con el modelo planteado en la figura 2.1, se puede cancelar el trabajo de STK_CEGUI_Manager, por lo que la ocurrencia de eventos será informada solamente a la aplicación final, y no se hará la actualización de la interfaz UpdateInterface (ver figura 2.4).

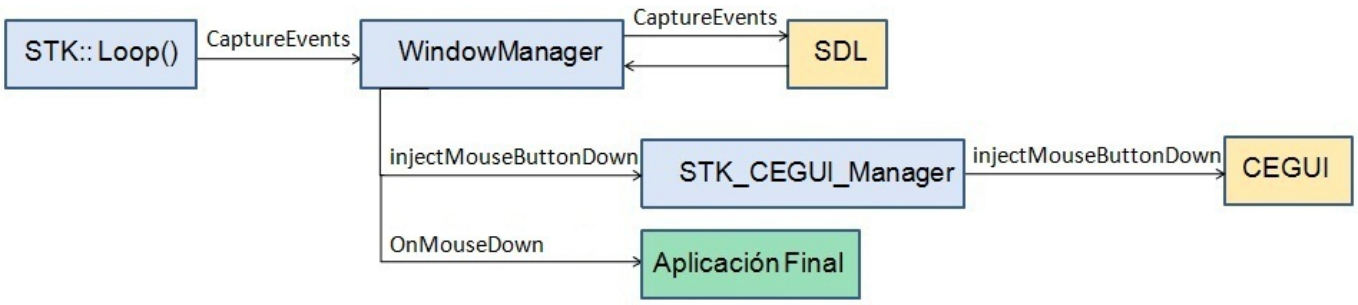

Figura 2.2: Manejo de eventos con CEGUI.

Figura 2.3: Refrescamiento de la interfaz de CEGUI.

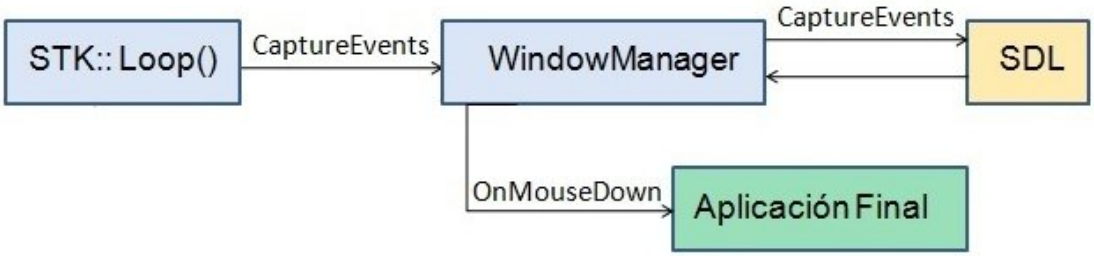

Figura 2.4: Trabajo con ventanas simples.

2.5. Herramientas a utilizar

Se trabajará sobre la SceneToolKit 10.07, con CEGUI v0.7.5.

Se desarrollará usando el IDE VisualStudio 2008. Se programará en C++ para permitir la migración del código a sistema operativo GNU/Linux.

La ingeniería se basará en lenguaje UML, siguiendo la metodología RUP, y usando el Rational Rose Interprise Edition 2003.06.00.436.000.

2.6. Reglas del negocio

- El programador final deberá implementar el "*main*" en su aplicación, y en dicho método deberá llamar el método principal de la STK ("*Run*").
- Los recursos de CEGUI (*schemes*, *imagesets*, *fonts*, *layouts*, *looknfeels* y *lua_scripts*) se deberán ubicar en una carpeta con nombre "datafiles" al mismo nivel del ejecutable de la aplicación final.
- En el trabajo con CEGUI, se deberá crear un objeto manejador de CEGUI y ser asignado al manejador de ventanas de la STK.

Conclusiones del capítulo

Se concluye que la solución propuesta presentada en este capítulo, satisface el objetivo planteado al inicio de este trabajo. Con el modelo del dominio y la descripción realizada, se podrá realizar la ingeniería del sistema, la cual se presenta en el próximo capítulo.

Capítulo 3

Características, diseño e implementación del sistema

De las fases de RUP, en este capítulo se presentarán la especificación de requisitos funcionales y no funcionales, y se definirán y describirán en formato expandido los casos de usos del sistema. Posteriormente se presentan los diagramas de secuencia y de clases del diseño, y por último los diagramas de despliegue y de componentes.

3.1. Especificación de los requisitos de *software*

3.1.1. Requisitos funcionales

1. Crear consola para mostrar mensajes.
2. Crear ventana SDL.
3. Especificar tamaño de ventana SDL.
4. Especificar si la ventana SDL se crea *fullscreen* o no.
5. Poner ventana *fullscreen*.
6. Quitar ventana *fullscreen*.
7. Cambiar de *fullscreen* a no *fullscreen*, y viceversa.
8. Destruir ventana SDL.
9. Capturar evento Quit.
10. Informar ocurrencia de evento Quit a la aplicación STK.
11. Capturar evento KeyDown.
12. Informar ocurrencia de evento KeyDown a la aplicación STK.
13. Capturar evento KeyUp.
14. Informar ocurrencia de evento KeyUp a la aplicación STK.
15. Capturar evento MouseMotion.
16. Informar ocurrencia de evento MouseMotion a la aplicación STK.
17. Capturar evento MouseButtonDown.

18. Informar ocurrencia de evento MouseButtonDown a la aplicación STK.
19. Capturar evento MouseButtonUp.
20. Informar ocurrencia de evento MouseButtonUp a la aplicación STK.
21. Capturar evento MouseWheel.
22. Informar ocurrencia de evento MouseWheel a la aplicación STK.
23. Capturar evento Resize.
24. Informar ocurrencia de evento Resize a la aplicación STK.
25. Conocer si una tecla está presionada.
26. Conocer el nombre de una tecla.
27. Conocer si un botón del *mouse* está presionado.
28. Conocer el estado del *mouse*.
29. Mostrar puntero del *mouse*.
30. Ocultar puntero del *mouse*.
31. Poner el puntero del *mouse* en una posición de la pantalla.
32. Inicializar sistema CEGUI.
33. Actualizar interfaz CEGUI.
34. Cargar directorio de grupos de recursos de *schemes*.
35. Especificar grupo de recurso de *schemes* a ser usado por defecto.
36. Cargar directorio de grupos de recursos de *imagesets*.
37. Especificar grupo de recurso de *imagesets* a ser usado por defecto.

38. Cargar directorio de grupos de recursos de *fonts*.
39. Especificar grupo de recurso de *fonts* a ser usado por defecto.
40. Cargar directorio de grupos de recursos de *layouts*.
41. Especificar grupo de recurso de *layouts* a ser usado por defecto.
42. Cargar directorio de grupos de recursos de *looknfeels*.
43. Especificar grupo de recurso de *looknfeels* a ser usado por defecto.
44. Cargar directorio de grupos de recursos de *lua_scripts*.
45. Especificar grupo de recurso de *lua_scripts* a ser usado por defecto.
46. Inyectar evento KeyDown a CEGUI.
47. Inyectar evento KeyUp a CEGUI.
48. Inyectar evento MouseMotion a CEGUI.
49. Inyectar evento MouseButtonDown a CEGUI.
50. Inyectar evento MouseButtonUp a CEGUI.
51. Inyectar evento MouseWheel a CEGUI.
52. Inyectar evento Resize a CEGUI.

3.1.2. Requisitos no funcionales

Usabilidad: El módulo deberá ser reusable por aplicaciones finales. Además, será utilizado por programadores con conocimientos básicos del trabajo con interfaces gráficas de usuarios y sistemas de realidad virtual, por lo que debe estar concebido para que el programador piense en qué desea hacer y no cómo hacerlo.

Rendimiento: Como el módulo formará parte de una biblioteca para desarrollo de aplicaciones de tiempo real, el mismo no debe afectar el desempeño de la aplicación final en cuanto a velocidad de procesamiento o cálculo, tiempo de respuesta y de recuperación, y disponibilidad.

Soporte: Debe ser multiplataforma: en una versión inicial se desarrollará sobre la plataforma Windows, pero debe ser compatible con GNU/Linux.

Legales: Para el desarrollo del módulo se usarán bibliotecas de código abierto y licencias libres.

Ayuda y documentación en línea: Se documentará el código siguiendo los estándares compatibles con el generador de documentación Doxygen.

3.2. Definición de casos de uso del sistema

3.2.1. Actor del sistema

Actor	Justificación
Aplicación final	La aplicación final es la que se beneficia con las funcionalidades para el trabajo con ventanas y la interacción con la interfaz gráfica de usuario.

Tabla 3.1: Actor del sistema.

3.2.2. Casos de uso del sistema

1. Crear consola.
2. Gestionar ventana.
3. Capturar eventos.
4. Detectar estado de periféricos.
5. Modificar puntero del *mouse*.
6. Inicializar CEGUI.
7. Actualizar interfaz de CEGUI.
8. Inicializar recursos de CEGUI.
9. Inyectar eventos a CEGUI.

3.2.3. Diagrama de CU y especificación en formato expandido

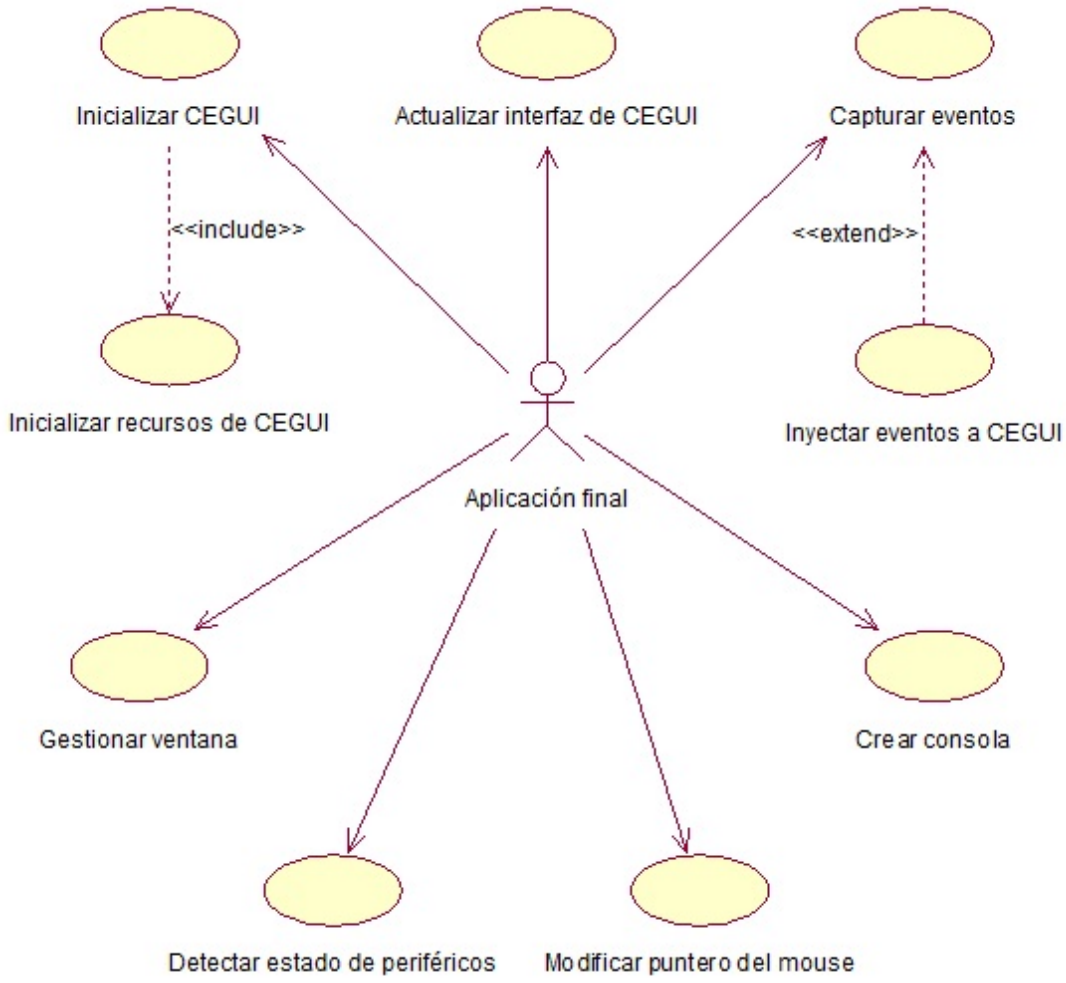

Figura 3.1: Casos de uso del sistema.

Caso de uso	
CU-1	Crear consola.
Propósito	Crear una consola donde se muestran textos (logs) de la STK
Actores	Aplicación final.
Resumen: El caso de uso se inicia cuando la aplicación final se inicializa e invoca la creación de una consola para mostrar los logs de la aplicación.	
Referencias	RF1
Flujo normal de los eventos	
Acción del actor	**Respuesta del sistema**
1- La aplicación final manda a crear la consola.	1.1- Crea la consola

Tabla 3.2: Especificación del CU Crear Consola.

Caso de uso	
CU-2	Gestionar ventana.
Propósito	Crear, configurar y destruir la ventana principal de la aplicación.
Actores	Aplicación final.
Resumen: El caso de uso se inicia cuando la aplicación final va a crear, modificar el estado de *fullscreen* de la ventana, o destruirla.	
Referencias	RF2, RF3, RF4, RF5, RF6, RF7, RF8
Flujo normal de los eventos	
Sección "Crear"	
Acción del actor	**Respuesta del sistema**
1- La aplicación final invoca crear una ventana, especificando tamaño de la misma y si es *fullscreen* o no.	1.1- Crea la ventana usando la biblioteca SDL. 1.2- Establece el modo de video según el tamaño de ventana y en estado de *fullscreen* especificado.
Sección "Cambiar a *fullscreen*"	
Acción del actor	**Respuesta del sistema**
1- La aplicación final invoca poner ventana en estado *fullscreen*.	1.1- Si la ventana no estaba *fullscreen*, la pone *fullscreen*, sino no hace nada.
Sección "Salir de *fullscreen*"	
1- La aplicación final invoca quitar ventana del estado *fullscreen*.	1.1- Si la ventana estaba *fullscreen*, la pone normal, sino no hace nada.
Sección "Alternar entre *fullscreen* y no *fullscreen*"	
1- La aplicación final invoca cambiar estado de *fullscreen* de la ventana (*toggle*)	1.1- Si la ventana estaba *fullscreen*, la pone normal, sino la pone *fullscreen*.
Sección "Destruir"	
Acción del actor	**Respuesta del sistema**
1- La aplicación final invoca destruir la ventana.	1.1- Se indica a la biblioteca SDL destruir la ventana.

Tabla 3.3: Especificación del CU Gestionar ventana.

Caso de uso	
CU-3	Capturar eventos.
Propósito	Detectar la ocurrencia de eventos de teclado y *mouse*, y avisar a la STK y a la manejadora de GUI de la ocurrencia del mismo.
Actores	Aplicación final.
Resumen: El caso de uso se inicia cuando la aplicación final indica detectar eventos para informar a la STK y a la GUI.	
Referencias	RF9, RF10, RF11, RF12, RF13, RF14, RF15, RF16, RF17, RF18, RF19, RF20, RF21, RF22, RF23, RF24. CU-9 (extendido).
Flujo normal de los eventos	
Acción del actor	**Respuesta del sistema**
1- La aplicación invoca la captura de eventos.	1.1- Se encuesta a SDL para determinar si existen eventos: Si el evento es Quit ir a sección QUIT; si es KeyDown ir a KEYDOWN; si es KeyUp ir a KEYUP; si es MouseMotion ir a MOUSEMOTION; si es Mouse ButtonDown ir a MOUSEBUTTONDOWN; si es Mouse ButtonUp ir a MOUSEBUTTONUP; si es MouseWheel ir a MOUSEWHEEL; si es Resize ir a RESIZE.
Sección “QUIT”	
Acción del actor	**Respuesta del sistema**
	1.2- Llama el Close de la STK.
Sección “KEYDOWN”	
Acción del actor	**Respuesta del sistema**
	1.2- Llama al CU-9, secc. KEYDOWN (pto de ext). 1.3- Llama al OnKeyDown de la STK.

Sección “KEYUP”	
Acción del actor	**Respuesta del sistema**
	1.2- Llama al CU-9, secc. KEYDOUP (pto de ext). 1.3- Llama al OnKeyUp de la STK.
Sección “MOUSEMOTION”	
Acción del actor	**Respuesta del sistema**
	1.2- Llama al CU-9, secc. MOUSEMOTION (pto de ext). 1.3- Llama al OnMouseMove de la STK.
Sección “MOUSEBUTTONDOWN”	
Acción del actor	**Respuesta del sistema**
	1.2- Llama al CU-9, secc. MOUSEBUTTONDOWN (pto de ext). 1.3- Llama al OnMouseDown de la STK.
Sección “MOUSEBUTTONUP”	
Acción del actor	**Respuesta del sistema**
	1.2- Llama al CU-9, secc. MOUSEBUTTONUP (pto de ext). 1.3- Llama al OnMouseUp de la STK.
Sección “MOUSEWHEEL”	
Acción del actor	**Respuesta del sistema**
	1.2- Llama al CU-9, secc. MOUSEWHEEL (pto de ext). 1.3- Llama al OnMouseWheel de la STK.
Sección “RESIZE”	
Acción del actor	**Respuesta del sistema**
	1.2- Llama al CU-9, secc. RESIZE (pto de ext). 1.3- Llama al OnResize de la STK.

Tabla 3.4: Especificación del CU Capturar eventos.

Caso de uso	
CU-4	Detectar estado de periféricos.
Propósito	Detectar estado del teclado y el *mouse*.
Actores	Aplicación final.
Resumen: El caso de uso se inicia cuando la aplicación final invoca: conocer si una tecla está presionada, ir a sección Tecla presionada; conocer el nombre de una tecla, ir a sección Nombre tecla; conocer si un botón del *mouse* está presionado, ir a sección Botón *mouse*; conocer el estado del *mouse*, ir a sección Estado mouse.	
Referencias	RF25, RF26, RF27, RF28
Flujo normal de los eventos	
Sección "Tecla presionada"	
Acción del actor	**Respuesta del sistema**
1- La aplicación final pregunta si una tecla está presionada, especificando el código de la tecla.	1.1- Se encuesta a la biblioteca SDL. 1.2- Se responde verdadero o falso.
Sección "Nombre tecla"	
Acción del actor	**Respuesta del sistema**
1- La aplicación final pregunta el nombre de una tecla, especificando el código de la tecla.	1.1- Se encuesta a la biblioteca SDL. 1.2- Se retorna el nombre de la tecla.
Sección "Botón *mouse*"	
Acción del actor	**Respuesta del sistema**
1- La aplicación final pregunta si un botón del *mouse* está presionado, especificando el código del mismo.	1.1- Se encuesta a la biblioteca SDL. 1.2- Se responde verdadero o falso.
Sección "Estado mouse"	
Acción del actor	**Respuesta del sistema**
1- La aplicación final pregunta el estado del *mouse*.	1.1- Se encuesta a la biblioteca SDL. 1.2- Retorna la X,Y del *mouse*, y el estado de los botones.

Tabla 3.5: Especificación del CU Detectar estado de periféricos.

Caso de uso	
CU-5	Modificar puntero del *mouse*.
Propósito	Modificar la propiedad de visibilidad del *mouse*, o su posición.
Actores	Aplicación final.
Resumen: El caso de uso se inicia cuando la aplicación final invoca modificar las propiedades del puntero del *mouse*: mostrar el puntero, ir a sección Mostrar puntero; ocultar el puntero, ir a sección Ocultar puntero; poner el puntero del *mouse* en una posición de la pantalla, ir a sección Posicionar puntero.	
Referencias	RF29, RF30, RF31
Flujo normal de los eventos	
Sección "Mostrar puntero"	
Acción del actor	**Respuesta del sistema**
1- La aplicación final invoca Mostrar el puntero.	1.1- Se muestra el puntero y se retorna el estado anterior de visibilidad.
Sección "Ocultar puntero"	
Acción del actor	**Respuesta del sistema**
1- La aplicación final invoca Ocultar el puntero.	1.1- Se oculta el puntero y se retorna el estado anterior de visibilidad.
Sección "Posicionar puntero"	
Acción del actor	**Respuesta del sistema**
1- La aplicación final invoca poner el puntero en una posición de la pantalla, especificando las coordenadas X,Y.	1.1- Se ubica el puntero en la coordenada especificada.

Tabla 3.6: Especificación del CU Modificar puntero del *mouse*.

Caso de uso	
CU-6	Inicializar CEGUI.
Propósito	Inicializar el *render* de CEGUI.
Actores	Aplicación final.
Resumen: El caso de uso se inicia cuando la aplicación final invoca la inicialización del sistema CEGUI. Automáticamente se inicializan los recursos de CEGUI.	
Referencias	RF32, CU-8
Flujo normal de los eventos	
Acción del actor	**Respuesta del sistema**
1- La aplicación final invoca la inizialización de CEGUI.	1.1- Se inicia el sistema CEGUI especificando el *render*. 1.2- Llama al CU-8 “Inicializar recursos de CEGUI”.

Tabla 3.7: Especificación del CU Inicializar CEGUI.

Caso de uso	
CU-7	Actualizar interfaz de CEGUI.
Propósito	Actualizar componentes visuales de la GUI.
Actores	Aplicación final.
Resumen: El caso de uso se inicia cuando la aplicación final invoca la actualización de los componentes visuales de la GUI.	
Referencias	RF33
Flujo normal de los eventos	
Acción del actor	**Respuesta del sistema**
1- La aplicación final invoca actualizar interfaz de la GUI.	1.1- Se actualiza la interfaz de la GUI.

Tabla 3.8: Especificación del CU Actualizar interfaz de CEGUI.

Caso de uso	
CU-8	Inicializar recursos de CEGUI.
Propósito	Definir los directorios de recursos de CEGUI, y los recursos a usar por defecto.
Actores	CU-6 “Inicializar CEGUI”.
Resumen: Definir los directorios de recursos de CEGUI, y los recursos a usar por defecto.	
Referencias	RF34, RF35, RF36, RF37, RF38, RF39, RF40, RF41, RF42, RF43, RF44, RF45.
Flujo normal de los eventos	
Acción del actor	**Respuesta del sistema**
1- El caso de uso “ Inicializar CEGUI” invoca inicializar los recursos de CEGUI.	1.1- Se carga el directorio de grupos de recursos de *schemes* y se especifica el grupo a ser usado por defecto. 1.2- Se carga el directorio de grupos de recursos de *imagesets* y se especifica el grupo a ser usado por defecto. 1.3- Se carga el directorio de grupos de recursos de *fonts* y se especifica el grupo a ser usado por defecto. 1.4- Se carga el directorio de grupos de recursos de *layouts* y se especifica el grupo a ser usado por defecto. 1.5- Se carga el directorio de grupos de recursos de *looknfeels* y se especifica el grupo a ser usado por defecto. 1.6- Se carga el directorio de grupos de recursos de *lua_scripts* y se especifica el grupo a ser usado por defecto.

Tabla 3.9: Especificación del CU Inicializar recursos de CEGUI.

Caso de uso	
CU-9	Inyectar eventos a CEGUI.
Propósito	Informar a CEGUI de la ocurrencia de eventos de teclado y *mouse*.
Actores	CU-3 “Capturar eventos”.
Resumen: El caso de uso se inicia el caso de uso “Capturar eventos” indica informar a CEGUI de la ocurrencia de alguno.	
Referencias	RF46, RF47, RF48, RF49, RF50, RF51, RF52.
Flujo normal de los eventos	
Acción del actor	**Respuesta del sistema**
1- El caso de uso “Capturar eventos” invoca informar a CEGUI de la ocurrencia de un evento: • Si el evento es KeyDown ir a sección KEYDOWN. • Si el evento es KeyUp ir a sección KEYUP. • Si el evento es MouseMotion ir a sección MOUSEMOTION. • Si el evento es MouseButtonDown ir a sección MOUSEBUTTONDOWN. • Si el evento es MouseButtonUp ir a sección MOUSEBUTTONUP. • Si el evento es MouseWheel ir a sección MOUSEWHEEL. • Si el evento es Resize ir a sección RESIZE.	

Sección “KEYDOWN”	
Acción del actor	**Respuesta del sistema**
	1.1- Se inyecta el evento KeyDown y el código de la tecla presionada a CEGUI.
Sección “KEYUP”	
Acción del actor	**Respuesta del sistema**
	1.1- Se inyecta el evento KeyUp a CEGUI.
Sección “MOUSEMOTION”	
Acción del actor	**Respuesta del sistema**
	1.1- Se inyecta el evento MouseMove (MousePosition) a CEGUI.
Sección “MOUSEBUTTONDOWN”	
Acción del actor	**Respuesta del sistema**
	1.1- Se inyecta el evento MouseButtonDown a CEGUI.
Sección “MOUSEBUTTONUP”	
Acción del actor	**Respuesta del sistema**
	1.1- Se inyecta el evento MouseButtonUp a CEGUI.
Sección “MOUSEWHEEL”	
Acción del actor	**Respuesta del sistema**
	1.1- Se inyecta el evento MouseWheelChange a CEGUI.
Sección “RESIZE”	
Acción del actor	**Respuesta del sistema**
	1.1- Se inyecta el evento Resize a CEGUI.

Tabla 3.10: Especificación del CU Inyectar eventos a CEGUI.

3.3. Diseño del sistema

A continuación se muestran los diagramas de interacción en su dimensión temporal (diagramas de secuencia). Para cada caso se especifican los casos de usos relacionados.

3.3.1. Diagramas de secuencia

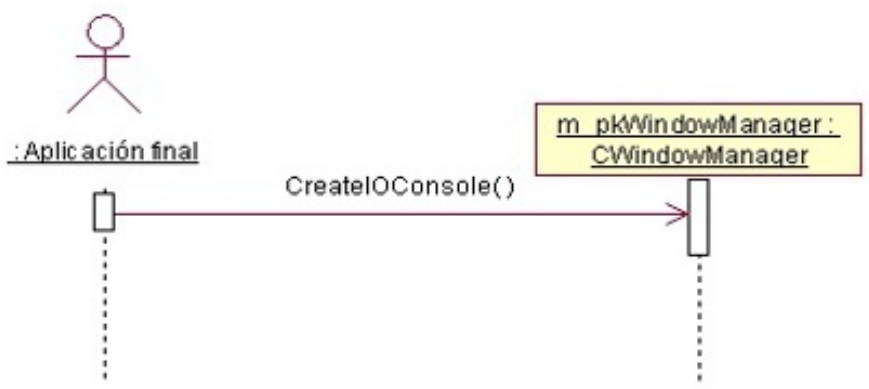

Figura 3.2: Diagrama de secuencia: Crear consola. Relacionado con el CU-1 Crear Consola.

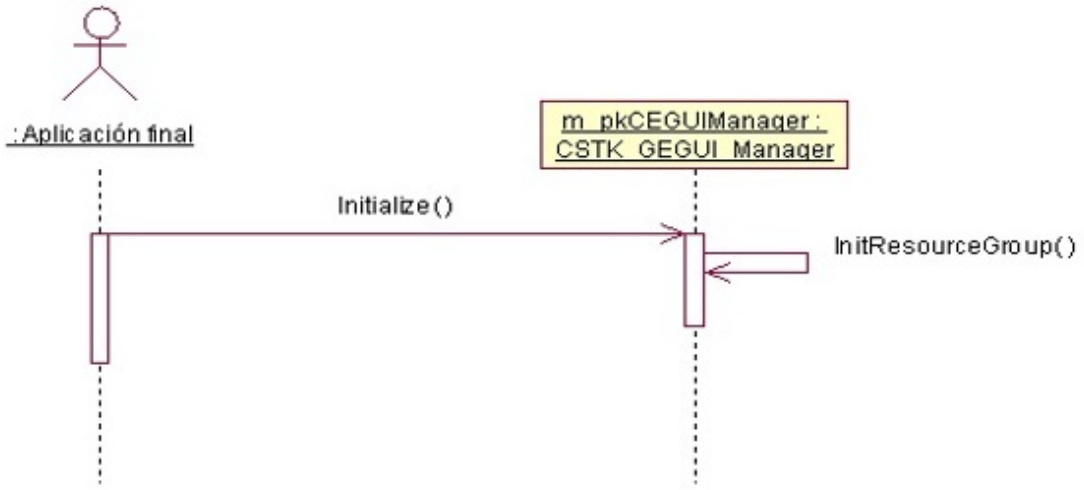

Figura 3.3: Diagrama de secuencia: Inicializar CEGUI. Relacionado con los CU-6 Inicializar CEGUI y CU-8 Inicializar Recursos de CEGUI.

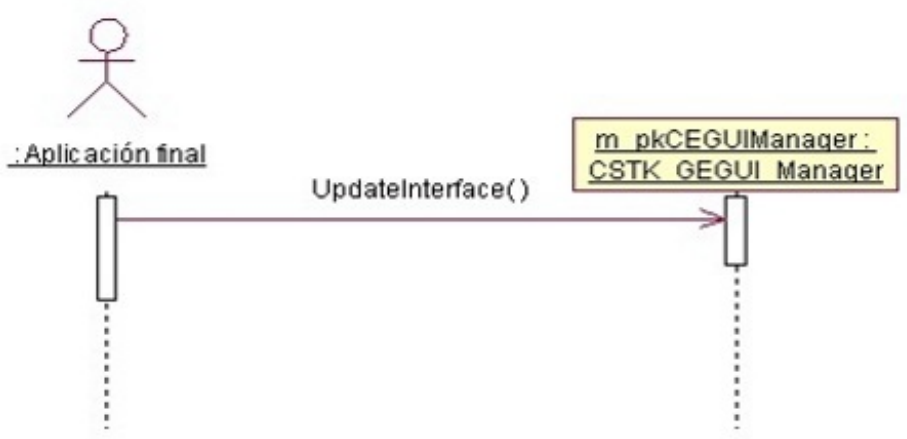

Figura 3.4: Diagrama de secuencia: Actualizar interfaz. Relacionado con el CU-7 Actualizar interfaz de CEGUI.

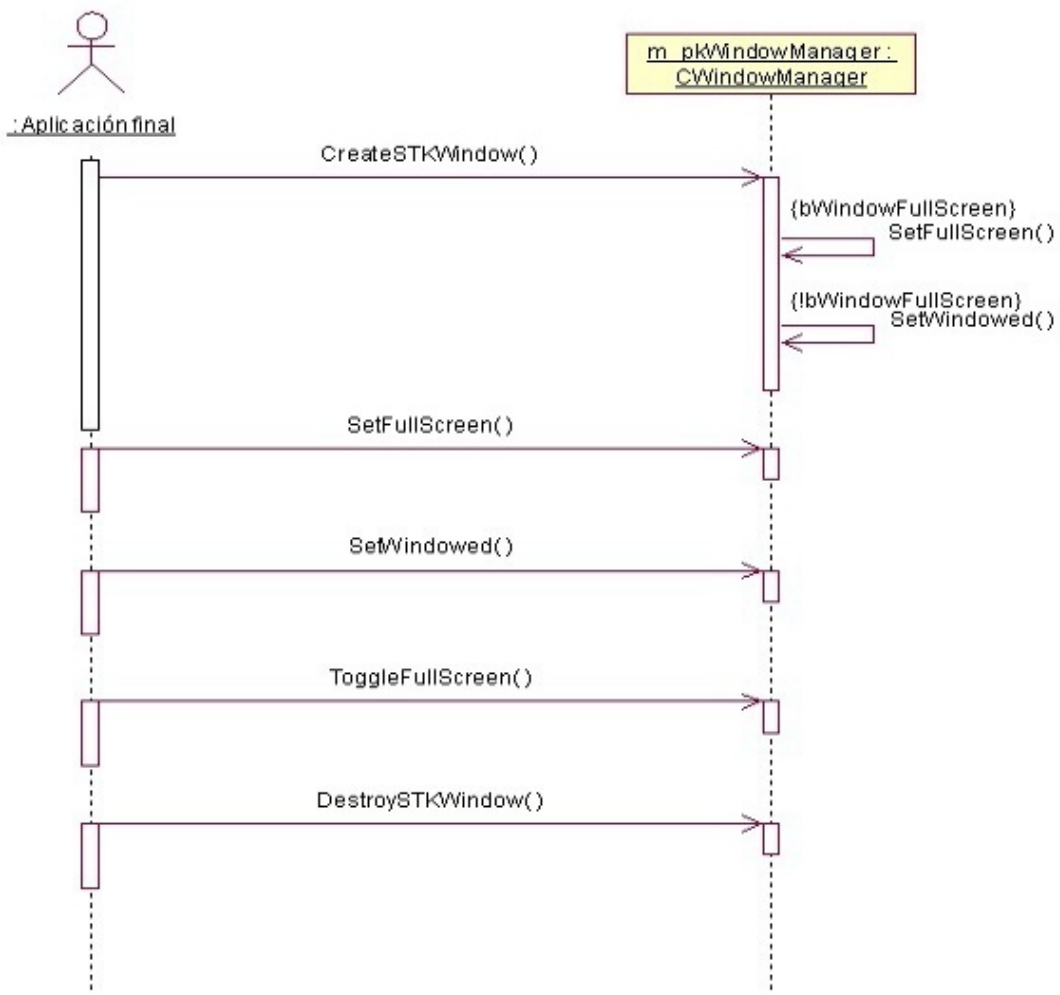

Figura 3.5: Diagrama de secuencia: Gestionar ventana. Relacionado con el CU-2 Gestionar ventana.

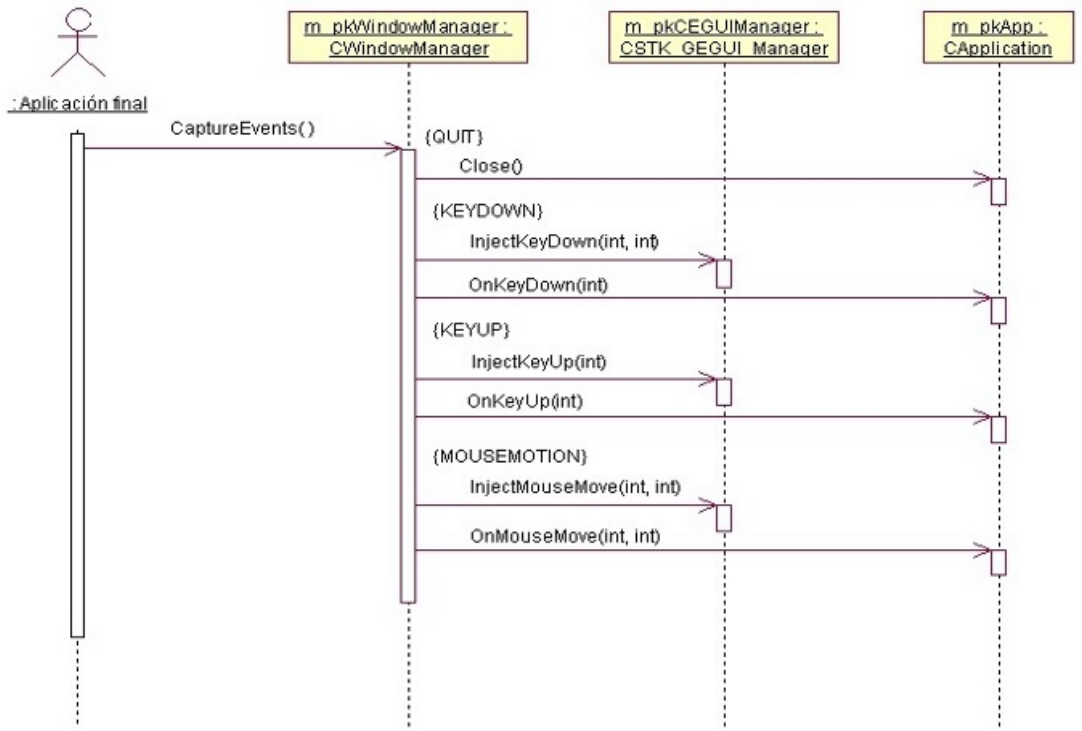

Figura 3.6: Diagrama de secuencia: Capturar e inyectar eventos (A). Relacionado con los CU-3 Capturar ventos y CU-9 Inyectar eventos a CEGUI.

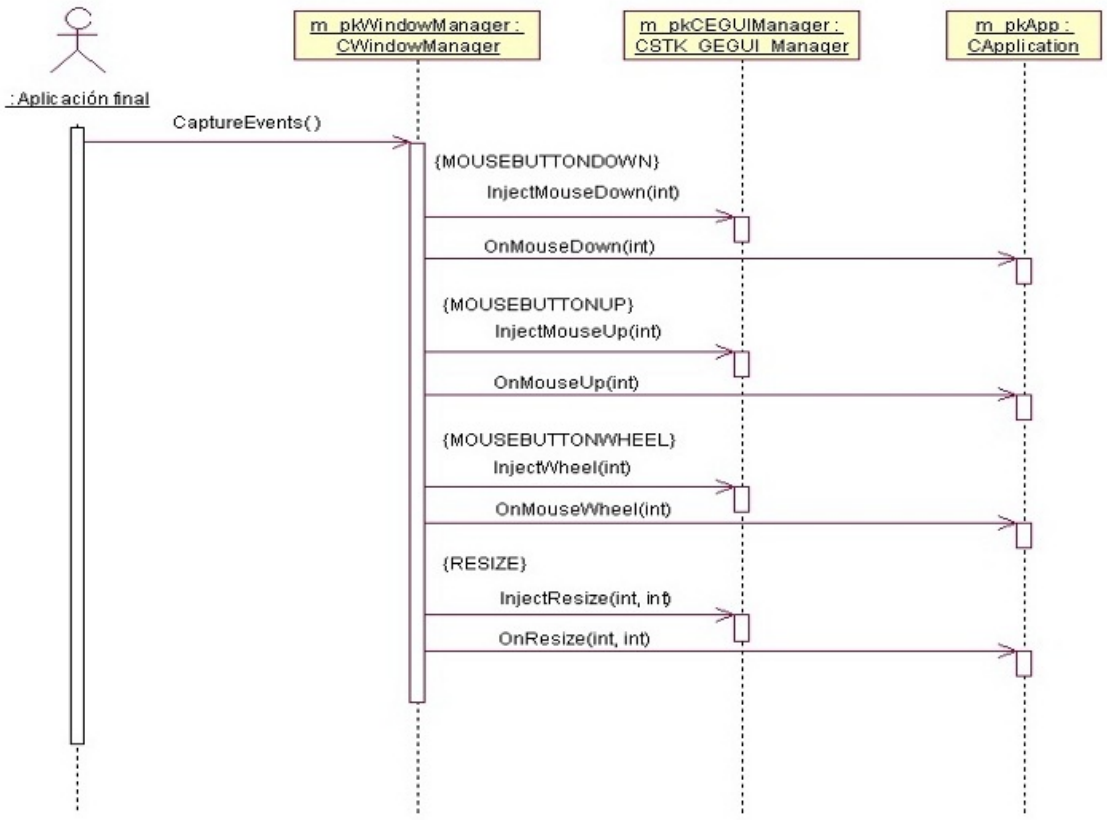

Figura 3.7: Diagrama de secuencia: Capturar e inyectar eventos (B). Relacionado con los CU-3 Capturar ventos y CU-9 Inyectar eventos a CEGUI.

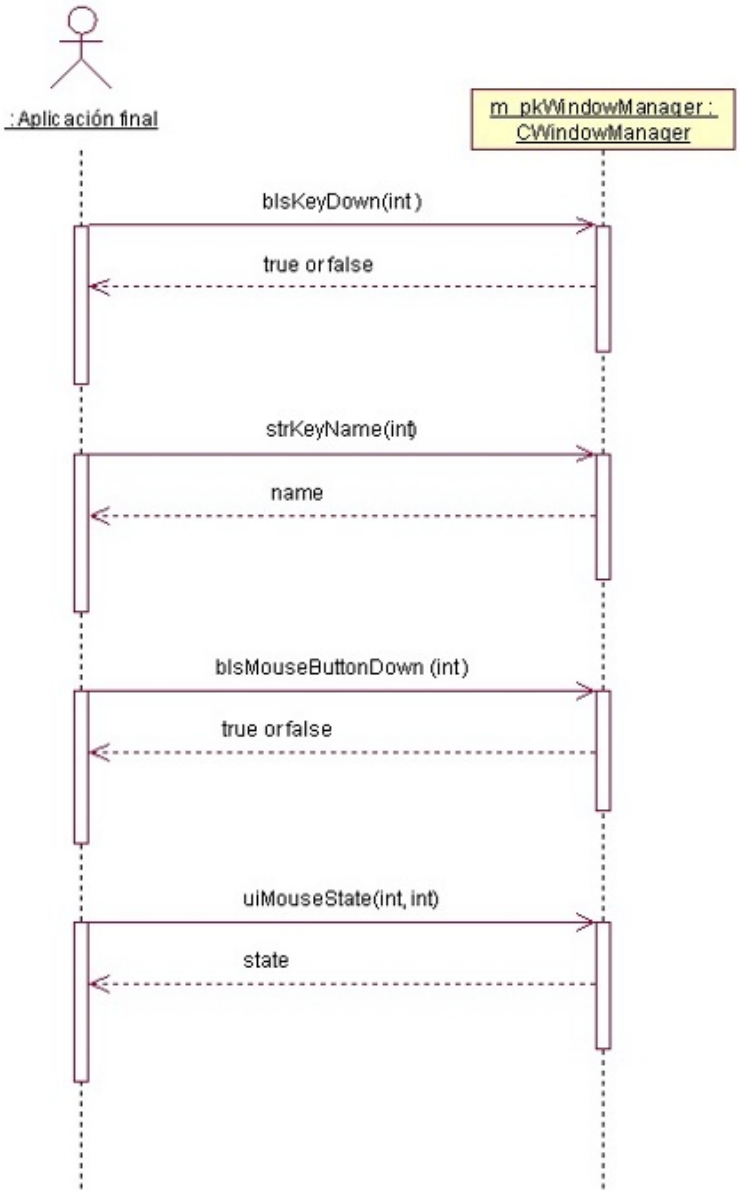

Figura 3.8: Diagrama de secuencia: Detectar estado de periféricos. Relacionado con el CU-4 Detectar estado de periféricos.

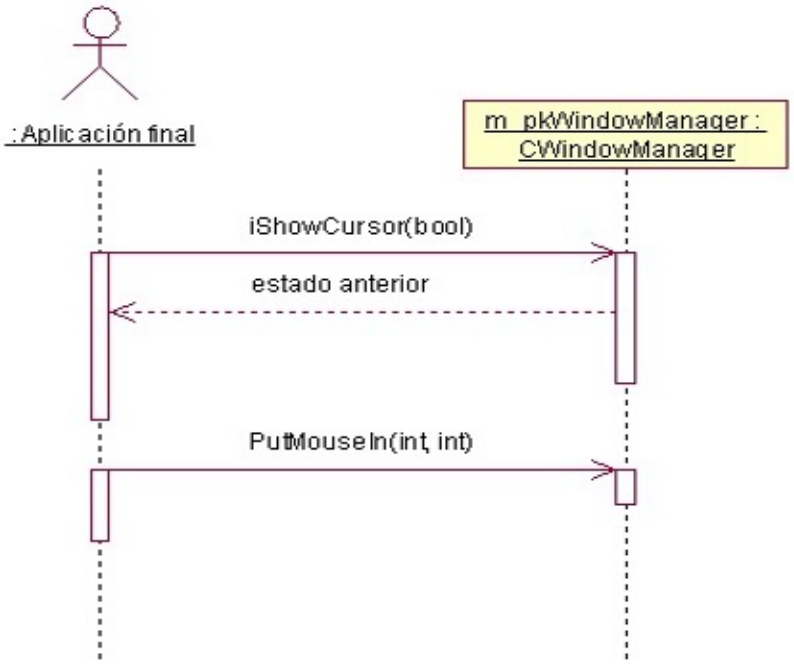

Figura 3.9: Diagrama de secuencia: Modificar puntero del *mouse*. Relacionado con el CU-5 Modificar puntero del *mouse*.

3.3.2. Diagrama y descripción de clases de diseño

Figura 3.10: Diagrama de clases de diseño.

Descripción de clase de diseño	
Nombre: CWindowManager	
Tipo de clase: Controladora	
Atributos	**Tipos**
m_bCurrentFullScreen	bool
m_iWindowHandler	int
m_pkGUIManager	CGUIManager*
m_pkApp	CApplication*
Responsabilidades	
Nombre:	CreateIOConsole
Descripción:	Para crear una consola donde se muestran textos (*logs*) de la STK.
Nombre:	CreateSTKWindow
Descripción:	Para crear la ventana.
Nombre:	DestroySTKWindow
Descripción:	Para destruir la ventana.
Nombre:	CaptureEvents
Descripción:	Para capturar enventos de entrada con SDL.
Nombre:	SetFullScreen
Descripción:	Para poner la ventana *FullScreen.*
Nombre:	SetWindowed
Descripción:	Para poner la ventana *Windowed* (No *FullScreen*).

Nombre:	ToggleFullScreen
Descripción:	Para alternar entre *FullScreen* y *Windowed*.
Nombre:	bIsKeyDown
Descripción:	Para conocer si una tecla está presionada. Recibe la tecla arg_iKey.
Nombre:	strKeyName
Descripción:	Devuelve el nombre una tecla dada arg_ikey.
Nombre:	bIsMouseButtonDown
Descripción:	Para conocer si un botón del *mouse* está presionado. Recibe el botón arg_iButton.
Nombre:	uiMouseState
Descripción:	Para conocer la posición del puntero del mouse y si algún botón está presionado.
Nombre:	iShowCursor
Descripción:	Para activar-desactivar la visualización del cursor en pantalla. Recibe si se desea visualizar o no arg_bShow.
Nombre:	PutMouseIn
Descripción:	Para colocar el puntero del ratón en una posición determinada. Recibe las coordenadas en pantalla arg_uiX, arg_uiY.

Tabla 3.11: Descripción de la clase CWindowManager.

Descripción de clase de diseño	
Nombre: CGUIManager	
Tipo de clase: Controladora	
Atributos (no tiene)	
Responsabilidades	
Nombre:	UpdateInterface (virtual)
Descripción:	Para actualizar la interfaz de usuario.
Nombre:	InjectKeyDown (virtual)
Descripción:	Para inyectar un evento de KeyDown en el sistema de GUI. Recibe el código de la tecla arg_uiScanCode.
Nombre:	InjectKeyUp (virtual)
Descripción:	Para inyectar un evento de KeyUp en el sistema de GUI. Recibe el código de la tecla arg_uiScanCode.
Nombre:	InjectMouseMove (virtual)
Descripción:	Para inyectar un evento de MouseMove en el sistema de GUI. Recibe la coordenada del puntero en pantalla arg_iX, arg_iY.
Nombre:	InjectMouseDown (virtual)
Descripción:	Para inyectar un evento de InjectMouseDown en el sistema de GUI. Recibe el botón del *mouse* arg_iSTKButton que provocó el evento.
Nombre:	InjectMouseUp (virtual)
Descripción:	Para inyectar un evento de InjectMouseUp en el sistema de GUI. Recibe el botón del *mouse* arg_iSTKButton que provocó el evento.

Nombre:	InjectWheel (virtual)
Descripción:	Para inyectar un evento de InjectWheel en el sistema de GUI. Recibe la variación de la ruedita arg_iSTKWheelValue.
Nombre:	InjectResize (virtual)
Descripción:	Para inyectar un evento de InjectResize en el sistema de GUI. Recibe el ancho y alto de la ventana arg_iNewWidth, arg_iNewHeight

Tabla 3.12: Descripción de la clase CGUIManager.

Descripción de clase de diseño	
Nombre: CSTK_GEGUI_Manager	
Tipo de clase: Controladora	
Atributos (no tiene)	
Responsabilidades	
Nombre:	Initialize
Descripción:	Inicializa el sistema de *render*. Automáticamente llama a InitResourceGroup.
Nombre:	InitResourceGroup
Descripción:	Inicializa el grupo de recursos de CEGUI por defecto.

Nombre:	UpdateInterface
Descripción:	Para actualizar la interfaz de usuario.
Nombre:	InjectKeyDown
Descripción:	Para inyectar un evento de KeyDown en el sistema de GUI. Recibe el arg_uiScanCode.
Nombre:	InjectKeyUp
Descripción:	Para inyectar un evento de KeyUp en el sistema de GUI. Recibe el arg_uiScanCode.
Nombre:	InjectMouseMove
Descripción:	Para inyectar un evento de MouseMove en el sistema de GUI. Recibe la coordenada del puntero en pantalla arg_iX, arg_iY.
Nombre:	InjectMouseDown
Descripción:	Para inyectar un evento de InjectMouseDown en el sistema de GUI. Recibe el botón del *mouse* arg_iSTKButton que provocó el evento.
Nombre:	InjectMouseUp
Descripción:	Para inyectar un evento de InjectMouseUp en el sistema de GUI. Recibe el botón del *mouse* arg_iSTKButton que provocó el evento.
Nombre:	InjectWheel
Descripción:	Para inyectar un evento de InjectWheel en el sistema de GUI. Recibe la variación de la ruedita arg_iSTKWheelValue.
Nombre:	InjectResize
Descripción:	Para inyectar un evento de InjectResize en el sistema de GUI. Recibe el ancho y alto de la ventana arg_iNewWidth, arg_iNewHeight.

Tabla 3.13: Descripción de la clase CSTK_CEGUI_Manager.

3.4. Implementación del sistema

Todos los componentes del *software* (figura 3.12) serán desplegados en en un solo nodo físico (figura 3.11).

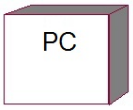

Figura 3.11: Diagrama de despliegue.

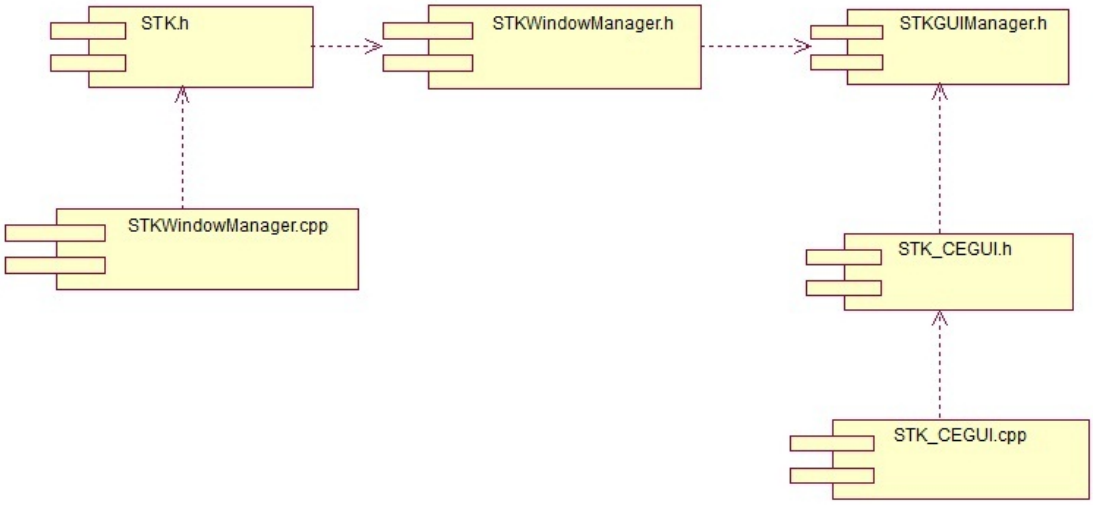

Figura 3.12: Diagrama de componentes.

Conclusiones del capítulo

Con las especificaciones ingenieriles expuestas en este capítulo, se podrá pasar a la codificación de la solución propuesta.

Conclusiones

Con la realización de este trabajo, se elaboró un módulo de interfaz gráfica de usuarios para la biblioteca STK, que permite la incorporación de componentes visuales, abstrayendo a los programadores de la detección de la ocurrencia de eventos de periféricos. De esta forma, se facilita el desarrollo de aplicaciones con mayores niveles de usabilidad e interacción de los usuarios con los videojuegos y entrenadores.

De las bibliotecas estudiadas, se determinó que CEGUI y Qt son las mejores candidatas para el desarrollo de componentes visuales para aplicaciones de realidad virtual. Sin embargo, partiendo de pruebas de código realizadas, se llegó a la conclusión de que el uso de Qt resulta engorroso bajo las caracerísticas actuales de la STK, por lo que no se tiene en cuenta su uso para esta versión del módulo.

Se demostró la factibilidad de usar componentes de CEGUI sobre ventanas de SDL, integrados a la arquitectura de la STK.

Recomendaciones

Se recomienda incorporar el uso de Qt al módulo para brindar la posibilidad de trabajar con componentes tradicionales, y lograr así una mayor aceptación por parte de la comunidad de desarrolladores que prefieren el uso de Qt sobre CEGUI.

Se recomienda además incorporar, en la captura de eventos, los relacionados con los *joysticks*, pues en la solución actual solamente se tienen en cuenta los eventos de teclado y *mouse*.

Bibliografía

[1] Carlos Aimacaña Toledo. Interfaz de usuario, 2005.

[2] J. Alonso Alonso. *Fundamentos y programación de videojuegos*, chapter VideoJuegos 3D, pages 7–30. UOC, Barcelona, 2008.

[3] Yanoski Camacho Román and Fernando Jiménez López. Informe final de proyecto, Febrero 2009.

[4] Francisco José Cortijo Bon and Fernando Berzal Galiano. Curso de c++ builder. http://elvex.ugr.es/decsai/builder/index.html, Julio 2010.

[5] Facultad de Informática de Barcelona (FIB) de la Universidad Politécnica de Cataluña (UPC). Retro informática, el pasado del futuro. http://www.fib.upc.edu/retro-informatica/avui/realitatvirtual.html, Junio 2010.

[6] Definición.org. Definición de interfaz gráfica de usuario. http://www.definicion.org/interfaz-grafica-de-usuario, Julio 2010.

[7] Vladimir del Corte. Informe sobre librerías qt. http://www.elai.upm.es:8009/spain/Investiga/GCII/personal/vcorte/informeqt.PDF, 2010.

[8] Diccionario informático. Definición de interfaz gráfica de usuario. http://www.softzone.es/glosario/g-h-e-i/, Julio 2010.

[9] Jordi Duch i Gavaldà and Heliodoro Tejedor Navarro. *Fundamentos y programación de videojuegos*, chapter Sonido, Interacción y Redes, pages 5–103. UOC, Barcelona, 2008.

[10] Crazy Eddie. Sitio web oficial Crazy Eddie's GUI System. http://www.cegui.org.uk/wiki/index.php/Main_Page, Julio 2010.

[11] Real Academia Española. Dicionario de la Real Academia Española. http://buscon.rae.es/draeI/SrvltGUIBusUsual?LEMA=realidad, Junio 2010.

[12] GTK+ Project. Sitio web oficial gtk. http://www.gtk.org/, Diciembre 2010.

[13] Qt-Nokia. Sitio web oficial Qt-Nokia. http://qt.nokia.com, Julio 2010.

[14] Cuauhtémoc Rivera Loaiza. Utilización de redes de Petri para la elaboración de una interfaz de usuario. http://www.fismat.umich.mx/~crivera/tesis/node11.htmll, Febrero 2010. Facultad de Ingeniería Eléctrica, Universidad Michoacana de San Nicolás de Hidalgo.

[15] Whatis.com. Definición de Microsoft Foundation Class Library (MFC). http://whatis.techtarget.com/definition/0,,sid9_gci214094,00.html, Julio 2010.

Apéndice A

Glosario de términos

***Game engine*:** Es el sistema operativo de un videojuego, el núcleo. Se encarga de dirigir y coordinar cada uno de los aspectos tecnológicos ligados a él. Capta eventos de entrada, computa física, reproduce sonidos, simula inteligencia artificial, pinta, etc.

***Graphic engine*:** (motor gráfico) Módulo gráfico independiente lo bastante potente para poder tratar toda la información que hay en él, así como su visualización en tiempo real. Este concepto surge por la complejidad gráfica añadida a la hora de tratar con escenarios 3D.

***Graphical user interface*:** (GUI) Es un programa informático que actúa como interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Su principal uso, consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina o computador.

Herramienta: (en gráficos por computadora) Ver *engine*, *graphic engine* y *game engine*. Está mayormente referido al motor de videojuego: herramienta de videojuegos. Usada en este documento con letra inicial mayúscula como sinónimo del motor de videojuego que forma parte del paquete **SceneToolKit**.

Interfaz de usuarios: Ver *user interface*.

Interfaz gráfica de usuarios: Ver *graphical user interface*.

Motor gráfico - Motor de videojuego: Ver *graphic engine* y *game engine* respectivamente.

Realidad Virtual: Representación de escenas o imágenes de objetos producidos por un sistema informático, que da la sensación de su existencia real.

***Render*:** (*rendering*) Crear en forma automática una imagen de acuerdo al modelo tridimensional que existe en el ordenador.

SceneToolKit: (STK) Alias del paquete de herramientas desarrollado por el proyecto, formada por *scene*, escena, y *toolkit*, "paquete de herramientas". El nombre completo es "Herramientas para Desarrollo de Sistemas de Realidad Virtual", y debe constar del *game engine* y otras herramientas utilitarias.

***Signals*:** En Qt, son "señales" emitidas por los objetos cuando ocurre un cambio o evento.

***Slots*:** En Qt, son métodos que se ejecutan cuando es invocado un *signal*. Responden a los cambios ocurridos y son "avisados" a través de los *signals*.

Simuladores: Un simulador es un aparato que permite la simulación de un sistema, reproduciendo su comportamiento. Los simuladores reproducen sensaciones que en realidad no están sucediendo.

Sistemas de Realidad Virtual: Aplicaciones informáticas basadas en técnicas de realidad virtual (ver Realidad Virtual).

***User interface*:** (UI) Es un conjunto de protocolos y técnicas para el intercambio de información entre una aplicación computacional y el usuario, responsable de solicitar comandos al usuario, y de desplegar los resultados de la aplicación de una manera comprensible.

Videojuego: Programa informático normalmente asociado a un hardware específico, que recrea un ejercicio sometido a reglas, se debe lograr uno o varios objetivos, donde los jugadores pueden interactuar y tomar decisiones.

***Widgets*:** Elementos básicos que constituyen una interfaz gráfica de usuario: botones, barras de desplazamiento, pestañas, etc.

Apéndice B

Glosario de abreviaturas

API: *Application Programming Interface*, Interfaz de programación de aplicaciones.

CEGUI: *Crazy Eddy Graphic User Interface*, Biblioteca para programación de interfaces gráficas de usuarios.

GTK+: *Gimp Tool Kit*, Conjunto de bibliotecas para desarrollo de interfaces gráficas de usuario.

GUI: *Graphical user Interface*, interfaz gráfica de usuarios.

MFC: *Microsoft Foundation Class Library*, Biblioteca de clases de Microsoft.

Qt: Biblioteca multiplataforma para desarrollar interfaces gráficas de usuario.

RUP: *Rational Unified Process*, metodología de desarrollo de software.

RV - SRV: Realidad Virtual, Sistemas de Realidad Virtual.

STK: SceneToolKit.

UCI: Universidad de las Ciencias Informáticas.

UI: *User Interface*, interfaz de usuarios.

VCL: *Visual Components Library*, Biblioteca de Componentes Visuales de Borland.

XML: *Extensible Markup Language*, metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C)

Apéndice C

Anexos

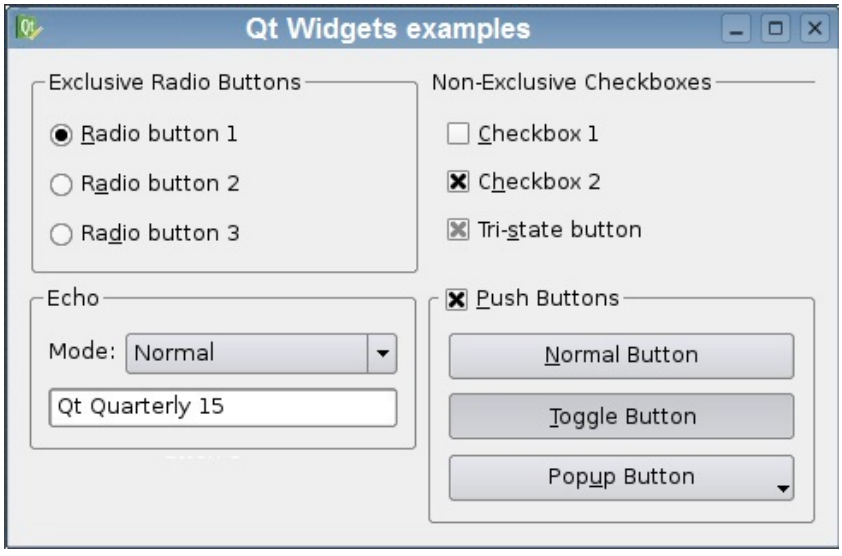

Figura C.1: Componentes visuales tradicionales.

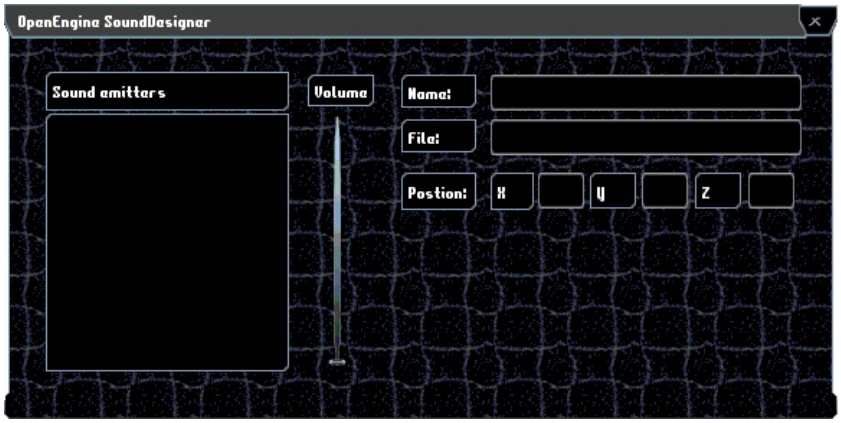

Figura C.2: Componentes visuales artisticos.

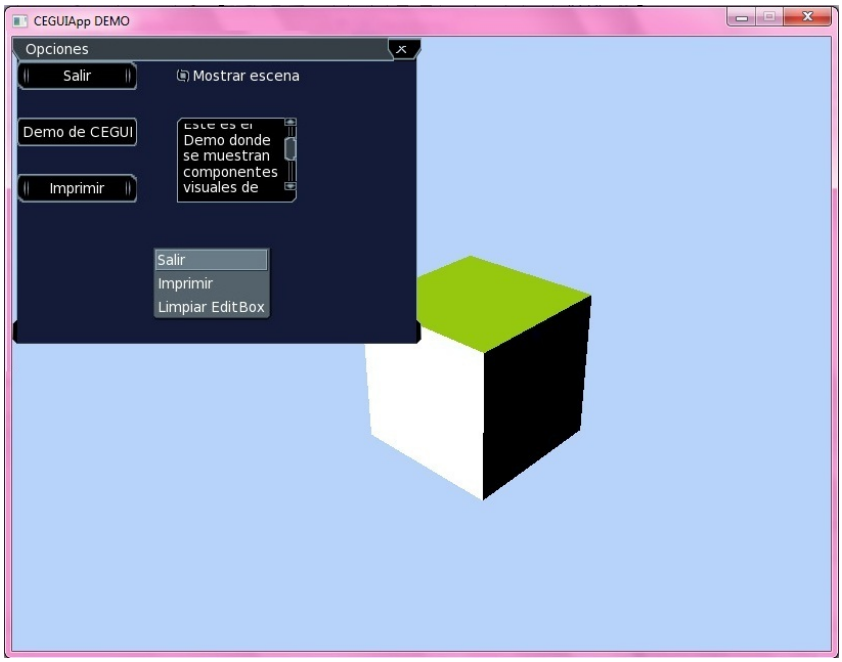

Figura C.3: Captura de pantalla de la aplicación demostrativa desarrollada con las funcionalidades del módulo.