Universidad de las Ciencias Informáticas.

Facultad 5.
Polo de Realidad Virtual.
Área Temática de Videojuego.

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

Título:

Herramienta de Configuración de Niveles para el Paquete de Juegos CNEURO.

Autor:

Diana Rosa Pérez Santiesteban.

Tutor:

Ing. Omar Correa Madrigal.

Junio de 2009.

"Soy lo suficientemente artista como para dibujar libremente sobre mi imaginación. La imaginación es más importante que el conocimiento. El conocimiento es limitado. La imaginación circunda el mundo..."

Albert Einstein

DATOS DE CONTACTO

Tutor: Ing. Omar Correa Madrigal.

Correo Electrónico: ocorrea@uci.cu

Graduado de Ingeniero en Ciencias Informáticas, en el 2007 en la Universidad de Ciencias Informáticas (UCI). Profesor desde el año 2007 en la UCI. Líder del proyecto CNEURO de la Facultad 5, desde el año 2008.

RESUMEN

El presente trabajo tiene como objetivo la implementación de un sistema que brinde la posibilidad de crear, modificar, eliminar y guardar en un fichero de datos la configuración para los componentes definidos en cada nivel del Paquete de Juegos CNEURO.

El proyecto se desarrollo a partir de un estudio de temas, tales como: proceso de configuración y formatos de fichero para almacenar información de videojuegos, bibliotecas multiplataforma para el desarrollo de interfaces gráficas de usuarios; así como, bibliotecas para el manejo de ficheros XML.

A partir de esta investigación se proponen las características técnicas de la solución, y se determina utilizar Qt en su versión 4.5 como marco de trabajo, Qt Creator como entorno de desarrollo integrado, RUP como metodología de desarrollo, UML como lenguaje de modelado orientado a objeto y C++ como lenguaje de programación.

Se realizaron actividades que comprende todo el ciclo de vida del producto como: captura de requisitos, agrupación de los requisitos funcionales en casos de uso del sistema y la realización de los casos de uso a través de los flujos de trabajo de análisis, diseño, implementación y prueba.

PALABRAS CLAVE

Configuración, Editor de Niveles, Herramienta, Gestión de Datos de un Juego, Motor Lógico, Nivel, Videojuego.

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN.

El éxito de los videojuegos no solo se centra en los gráficos y sonidos del juego, sino en los retos y reglas que estimulan la interacción con el usuario. (1) El cómo crear juegos para atraer a nuevos consumidores y a la vez cumplan con las expectativas de jugadores expertos, con un menor costo de recursos como: personas, tiempo, dinero, herramientas; es un principio de esta industria.

En los últimos años el mercado de los videojuegos se ha enfocado a dos tipos de jugadores: los que buscan realismo; aquellos para los que se crean juegos con alto realismo gráfico, sonoro y físico, y los que buscan jugabilidad; aquellos que desean una interacción más natural con el juego. (2)

El desarrollo de videojuegos no solo involucra la producción de este, sino también su publicación y venta. No obstante, muchas veces el trabajo no termina una vez que el juego fue publicado. Otros elementos que se crean son los contenidos descargables. Se trata de pequeñas ampliaciones del juego que se pueden ir comprando por separado y que añaden nuevas funcionalidades, como: nuevos mapas, personajes, vehículos, escenarios u opciones; además de generar un valor adicional sobre el mismo juego. (2)

También se puede trabajar en las continuaciones del juego, estas pueden producirse:

- En forma de expansión, cuando se utiliza el mismo motor y sólo se añade contenido extra.
- En forma de una nueva versión, cuando se modifica también el motor interno del mismo. (2)

Al reutilizar gran parte del mismo motor de juego para crear diferentes juegos del mismo tipo, se logra un proceso de desarrollo altamente económico ya que el motor está formado por módulos bien definidos y separables que pueden ser reutilizados independientemente, además, es una plataforma de desarrollo estable y bien conocida por los desarrolladores.

El Proyecto Juegos CNEURO del Área Temática de Videojuego del Polo de Realidad Virtual (RV) de la Facultad 5 de la Universidad de Ciencias Informáticas, Cuba; tiene como objetivo la realización de videojuegos terapéuticos.

El proyecto surge a partir de un estudio realizado en Cuba por el Instituto de Neurociencia Cognitiva, donde se dio a conocer que luego de examinar a 1.500 niños se pudo detectar una incidencia de un 3% a un 6% de casos de discalculia[1]. (3)

Los juegos que se desarrollan en el proyecto tienen su base en un motor de juego llamado: CNEUROGameEngine, este constituye una herramienta que brinda flexibilidad para desarrollar diversos productos usando el mismo motor de juego. En la creación del motor de juego se implementa una arquitectura en tres capas; denominadas: Soporte, Lógica y Aplicación.

En la capa Lógica se especifican los elementos que se manejan para cada uno de los niveles del juego como son: entorno, jugador, enemigos, aliados y tareas. Estos objetos están formados por datos como: imágenes, animaciones, sonidos, textos que indican los estados por los cuales puede transitar un personaje; estos datos suelen ser diferentes para cada nivel por tanto se hace necesario ahorrar tiempo de desarrollo en la gestión de configuración de niveles del juego.

Con el propósito de guiar el desarrollo del presente trabajo y delimitado por la ya expuesta situación problémica, el problema a resolver es: ¿Cómo automatizar la configuración de niveles del Paquete de Juegos CNEURO?

El objeto de estudio que se propone es el proceso de configuración de niveles del Paquete de Juegos CNEURO, y el campo de acción se centra en el proceso de elaboración de una herramienta de desarrollo que permita gestionar la configuración de niveles.

Como objetivo general del trabajo se propone, implementar una herramienta para automatizar el proceso de configuración de niveles del Paquete de Juegos CNEURO.

Con el fin de dar cumplimiento al objetivo planteado se hace indispensable realizar varias tareas de investigación resumidas en las siguientes:

- Análisis y recopilación de información sobre configuración de videojuegos.
- Estudio sobre bibliotecas multiplataforma para el desarrollo de interfaces de usuario.

[1] Discalculia: Dificultad específica para calcular o resolver operaciones aritméticas. No guarda relación con el nivel mental, con el método de enseñanza utilizado ni con trastornos efectivos, pero sí suele encontrarse asociado con otras alteraciones.

- Estudio sobre formatos de fichero XML.
- Elaboración de una solución técnica para dar cumplimiento al objetivo.
- Análisis y diseño de una herramienta de desarrollo que permita automatizar la configuración de niveles del Paquete Juego CNEURO.

Para dar solución a las tareas de investigación antes mencionadas se emplearan distintos métodos de investigación descritos a continuación:

Métodos Teóricos:
Método Analítico – Sintético.
Se escoge este método puesto que permite el análisis de teorías y documentos; con el fin de extraer los elementos más importantes y particulares que se relacionan con el objeto de estudio.

Método de Modelación
Modelación Teórica:
Se escoge este método para el desarrollo del análisis y diseño de una solución técnica propuesta, este método permite la creación de modelos, propuestas y diagramas para un mejor entendimiento del sistema a desarrollar.

El trabajo se ha estructurado en cuatro capítulos:
En el capítulo 1 se presentan los conceptos básicos que se manejan en relación al objeto de estudio. Además se justifican las herramientas que serán utilizadas para el desarrollo del sistema propuesto. En el capítulo 2 se plantea y detalla la solución propuesta, además se exponen las primeras características técnicas de la aplicación como: requisitos, casos de uso, y un prototipo inicial de interfaces de usuario. En el capítulo 3 se detalla el sistema durante los flujos de trabajo de análisis y diseño, obteniéndose un refinamiento de los requisitos funcionales y de la arquitectura del sistema. El capítulo 4 y final incluye la implementación del sistema, estándar de codificación empleado, así como las pruebas realizadas al software.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA.

INTRODUCCIÓN.

La intención de este capítulo es exponer al lector los principales conceptos relacionados con el proceso de configuración de niveles de un videojuego a partir de un estudio realizado en los libros: Introducción a los videojuegos y Lógica del videojuego.

El apartado le ofrece una breve reseña sobre las características y módulos que forman un motor de juego, luego se detalla a grandes rasgos el motor lógico. Se realiza una descripción del proceso de gestión de datos en un videojuego; se especifican los tipos de datos, sus relaciones. Se analiza cómo puede ser segmentado un juego y la forma más factible para organizar todos los elementos que compone cada segmento. Se establece una comparación entre los formatos de fichero .INI y .XML, como formatos de fichero descriptivos usados en videojuegos. Por último se establece una comparación entre entornos de desarrollo integrados para el framework Qt.

1.1. MOTOR DE JUEGO.

El motor de juego es el núcleo del videojuego. Por lo general un motor de juego puede ser descompuesto en partes pequeñas y reutilizables como son: entrada/salida, motor de renderizado para gráficos 2D o 3D, un motor de física o de detección de colisión, motor de sonido, inteligencia artificial, gestión de todos los datos de la partida y gestión de datos del usuario. Ver Figura 1.

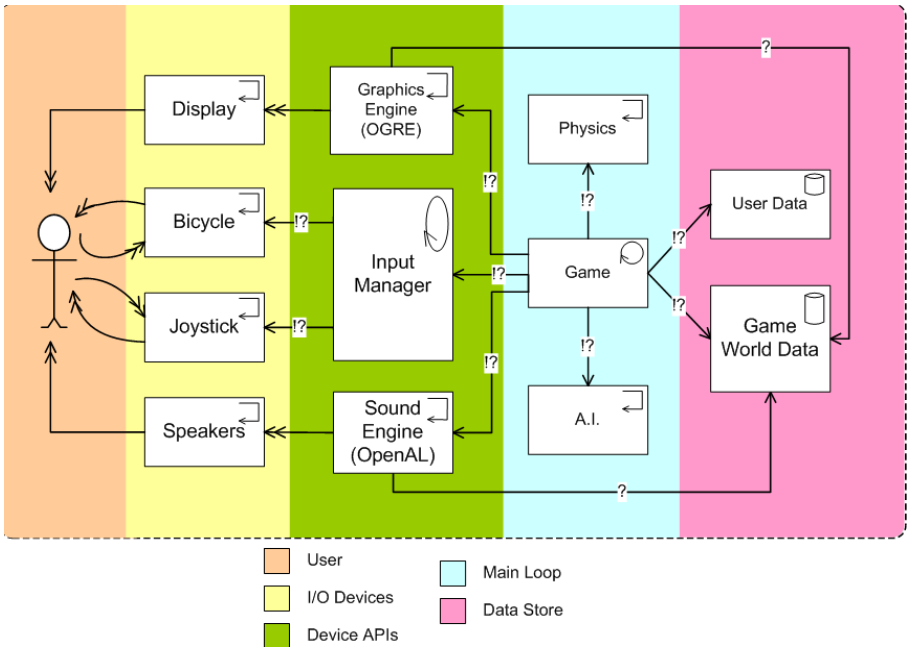


Figura 1. Modelo conceptual de un motor de juego

Algunos de los componentes de un motor de juego se construyen a partir de API[2] . Una API proporciona un sistema de abstracción que evita conocer los detalles técnicos del hardware del sistema. (2)

Las API son frecuentemente usadas cuando se necesita: ahorrar tiempo de desarrollo, alcanzar un alto realismo gráfico, sonoro y físico, y además cuando se quiere lograr un motor de juego flexible y reutilizable para nuevas versiones del juego.

Algunas de las API más usadas en el desarrollo de videojuegos son:

- **API gráficas**. Permiten acceder a todas las prestaciones del hardware. Las dos más utilizadas son OpenGL y DirectX. (2)
- **API físicas**. Se trata de API que describe el comportamiento físico real de los objetos cuando están sometidos a fuerzas y movimientos. Entre muchas otras características, facilitan el control de colisiones. Algunos ejemplos son Havok, Ode, Newton, Tomahawk, Novodex. (2)
- **API de red**. Permiten crear sockets de conexión entre dispositivos, enviar información y controlar el estado de las comunicaciones. Están adaptadas para soportar todo tipo de

[2] API: Application Programming Interface. Interfaces de Programación de Aplicaciones.

configuraciones, desde cliente-servidor hasta peer-to-peer. Algunos ejemplos son DirectPlay, Rakknet, SDL_net. (2)

- **API de sonido**. Permiten reproducir sonidos, canciones, trabajar con las ondas o bien posicionar elementos sonoros en un entorno tridimensional. Algunas de las bibliotecas más utilizadas son DirectSound, OpenAL, Fmod, PortAudio. (2)
- **API de interfaz**. Permiten la programación de los diferentes dispositivos de entrada, como ratones, joysticks, volantes... También permiten el control de algunas características de estos dispositivos, como la vibración o el force feedback. Las más utilizadas son DirectInput y SDL_Input. (2)
- **Otras API**. También podemos encontrar otras bibliotecas que evitarán la programación de algunas funciones específicas, como tareas rutinarias de inteligencia artificial, búsqueda de caminos o gestión de un grafo de estados (por ejemplo, PathLib), que permiten integrar scripting en el sistema (por ejemplo, LUA) y que permiten gestionar estructuras de datos complejas (por ejemplo, STL). (2)

1.2. MOTOR LÓGICO DEL VIDEOJUEGO.

La parte donde se controla el ciclo principal del juego se le denomina motor de lógica. Incluye la descripción de los atributos de todos los elementos que participan, y de todas las reglas y condiciones que se sitúan en el juego. Continuamente mira las acciones que han realizado los jugadores y los elementos controlados por la inteligencia artificial y decide si estas acciones se pueden llevar a cabo y cuál es el resultado de ejecutarlas. (1)

Dentro del motor lógico de un juego se juntan tres elementos fundamentales:

- La integración de todos los componentes que forman el motor de juego.
- La gestión de todos los datos de la partida.
- La aplicación de las reglas del juego.

1.3. GESTIÓN DE DATOS

El manejo de datos en un videojuego es una parte fundamental dentro de un motor lógico, estos datos pueden dividirse en dos grandes grupos:

- Los datos que se utilizan en los motores gráfico y sonoro; estos datos pueden incluir música, efectos o modelos tridimensionales, texturas o animaciones; estos se nombran entidades. (1)
- Los datos que describe todos los parámetros de los elementos que intervienen en el juego; nombrados objetos, los cuales pueden incluir desde posiciones, direcciones y velocidades, hasta atributos de los personajes; estos datos son utilizados por el motor lógico para poder evaluar continuamente el estado del juego. (1)

1.3.1. RELACIÓN OBJETO-ENTIDAD

Los objetos y las entidades normalmente se encuentran enlazados formando estructuras más complejas. Ver Figura 2. Esto permite mucha más flexibilidad, ya que se pueden combinar objetos y entidades para describir a los elementos que participan en el juego. (1)

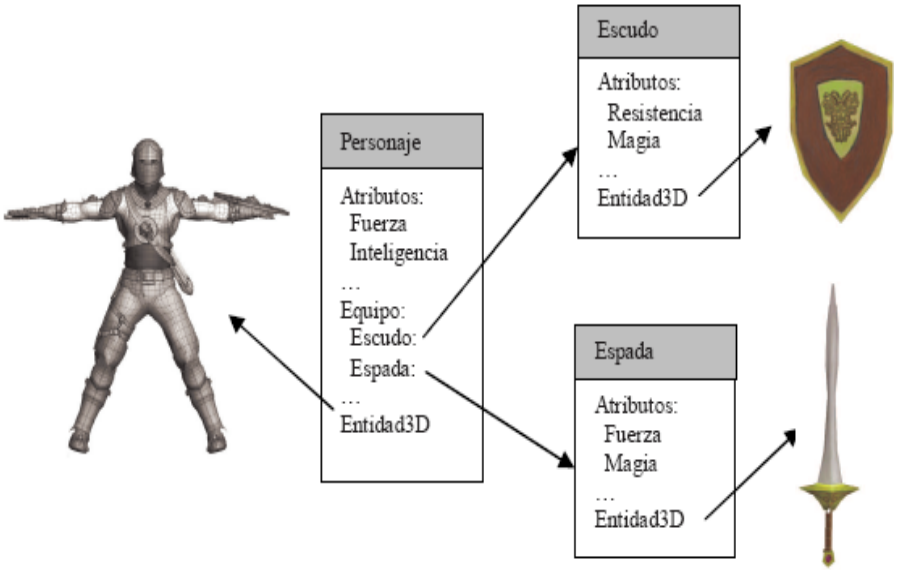


Figura 2. Ejemplo de la relación entre objetos y entidades.

Para cualquier tipo de juego existen una serie de objetos comunes que normalmente son utilizados:

- **Objeto terreno o entorno**: Solo existe una copia de este objeto y define la base de nuestro juego. Normalmente este objeto está compuesto por una única entidad grande. (1)
- **Objetos estáticos**: Objetos que se encuentran encima del terreno con unas características (posición, tamaño… prefijadas desde el principio). Incluyen una o varias entidades para poderlos representar en la pantalla. Pueden estar animados o no. (1)

- **Objetos móviles**: Se trata del principal grupo de objetos del juego. Aquí incluiremos aquellos objetos que se desplazan e interactúan con el entorno, tanto si están controlados por un jugador como si están controlados por la inteligencia artificial. Están compuestos por una o varias entidades, además de tener incorporados varias animaciones para cada una de las acciones que realizan. Estos objetos incorporan varios niveles de información para el sistema lógico que dependen de su importancia. (1)
- **Objetos de control**: Los objetos de control son objetos que no están relacionados con ninguna entidad y por tanto son invisibles al usuario. Se trata de objetos que son necesarios para que sistemas como la inteligencia artificial (IA) o el motor lógico del juego puedan conocer más detalles sobre el mundo donde se desarrolla la partida. Por ejemplo, en un juego de carreras podemos tener un objeto de control que nos indique el camino óptimo dentro de la pista para que la IA pueda calcular las posiciones de los coches. (1)

La forma más simple de implementar los objetos es utilizando estructuras de datos. En una estructura de datos describimos todas las variables que están asociadas a un objeto, tanto las que se usan en la parte lógica como las que se usan para gráficos o sonido. (1)

1.3.2. NIVEL DE UN VIDEOJUEGO

Los juegos pueden ser segmentados por niveles, normalmente cada nivel tiene sus objetivos particulares, los cuales se tienen que llevar a cabo antes de poder pasar al siguiente nivel.

Los datos de cada nivel acostumbran a ser diferentes, tanto a nivel lógico como gráfico y sonoro. Por tanto, la segmentación de un juego en niveles nos permite organizar mucho mejor los datos y no tener que tratar con todo el volumen de datos a la vez. (1)

Un nivel es simplemente un conjunto de datos que recopila toda la información de todos los elementos que intervienen en esta sección del juego como:

- El terreno base para el nivel y todas las propiedades asociadas.
- Los objetos estáticos y dinámicos. Su colocación en la escena, sus atributos, sus propiedades lógicas.
- Los componentes adicionales para los gráficos (situación de las luces, cámaras,…).

- Los objetos de control y las reglas que determinan el funcionamiento global del nivel.

Para implementar un nivel dentro de un videojuego se definen dos formas: introduciendo todo como código fuente o combinando parte de código fuente con los datos de un fichero externo. (1)

1.4. EDITOR DE NIVELES

En el caso que queramos trabajar con los objetos y los niveles de forma independiente del código del programa, necesitaremos una aplicación para poder tratar con todos estos datos. A estas aplicaciones se les nombra editores de niveles. (1)

En un editor de niveles se debe poder organizar todos los elementos que compone un nivel, añadir nuevos modificarlos y quitarlos. Adicionalmente algunos editores permiten editar los objetos que tenemos definidos dentro del nivel. (1)

Un editor de niveles también se conoce como editor de mapas, campañas o escenario. A veces el editor de niveles se integra al videojuego o puede tratarse de un programa totalmente independiente. (1)

Como antecedente del editor de niveles se puede mencionar a los nombrados Construction Set, esto era una utilidad implementada en algunos videojuegos usada para crear niveles extras o crear todo el juego. Entre los primeros se puede hacer mención de Pinball Construction Set y Aventure Construction Set. Ambos publicados por Electronic Arts, desarrollados en 1983 y 1985, por Bill Budge y Stuart Smith respectivamente. (4) Ver Figura 3.

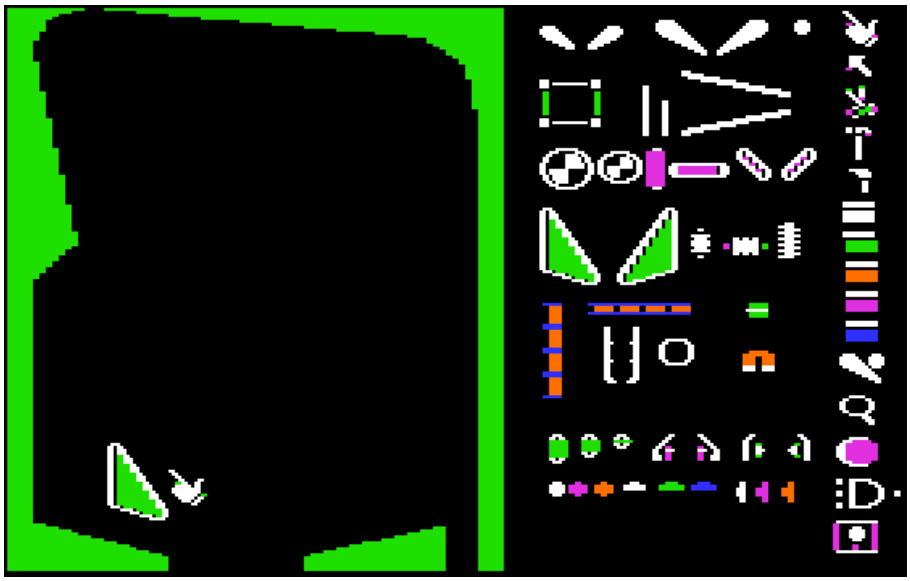


Figura 3. Pinball Construction Set.

Los editores de niveles son muy usados en el mundo de los videojuegos para crear niveles adicionales del juego, o para crear niveles personalizados.

El auge de estos ha fomentado la publicación de editores oficiales del juego y en otros casos por editores no oficiales. Entre los editores que se pueden mencionar se encuentran:

EA Graphics Editor, es un editor de gráficos de propósito general y editor de gráficos espectador, su uso es para ver y editar archivos gráficos de Electronic Arts. Fue escrito principalmente para la serie NBA Live, pero trabaja con un con un gran número de archivos de la NHL Hockey, FIFA Soccer, Need For Speed Triple Play. (5) Ver Figura 4.

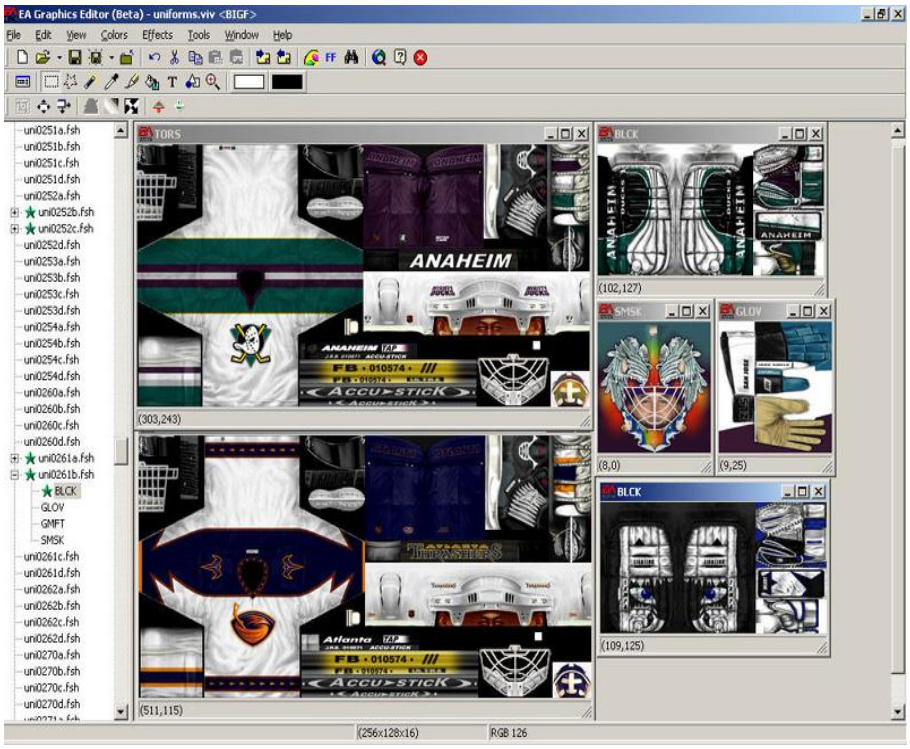


Figura 4. EA Graphics Editor.

GTKRadiant, es un editor de niveles para juegos deathmatch (popular tipo de partida al estilo "todos contra todos"). (6) Ver Figura 5.

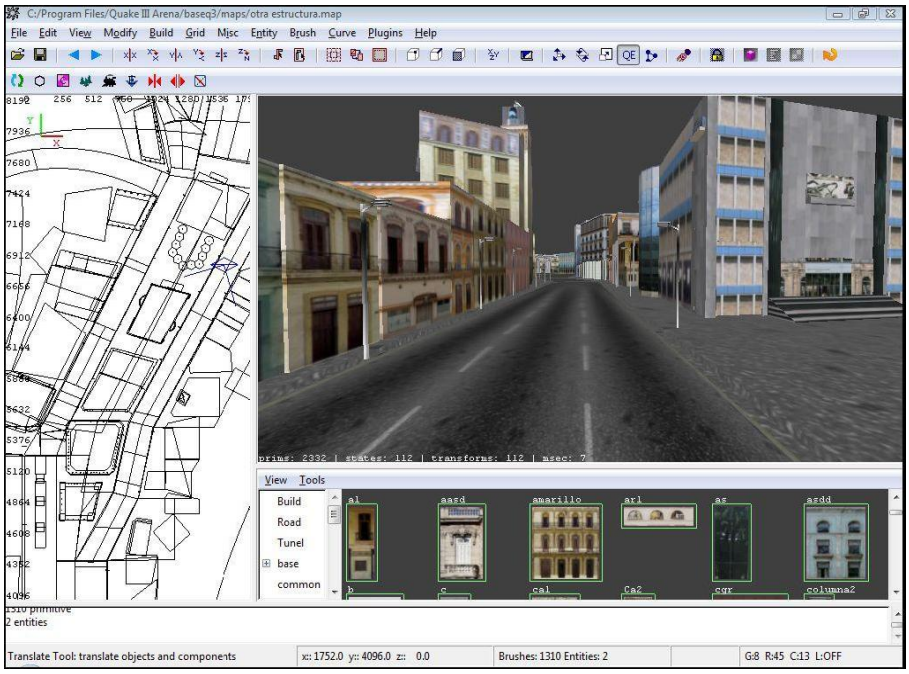


Figura 5. GTKRadiant

Dxtre3d, es un editor de niveles no oficial para el juego Tom Raider (TR), desarrollado por el grupo Turbo Pascal. Dxtre3d permite a los usuarios construir niveles desde TR1 al TR5. (7) Ver Figura 6.

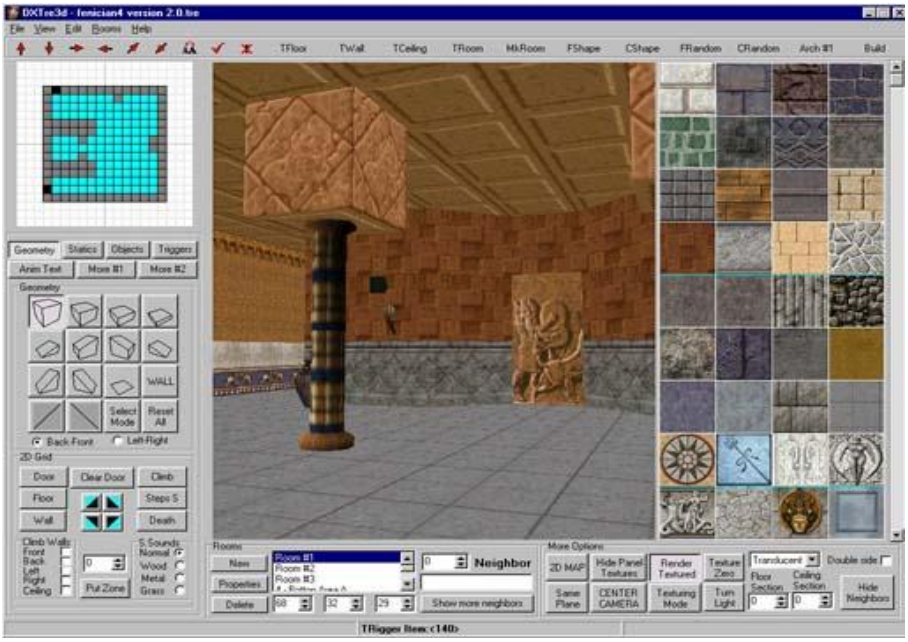


Figura 6. Dxtre3d.

1.5. FICHERO DE DATOS DE UN VIDEOJUEGO.

La comunicación entre el editor de niveles y el juego se realiza a través de ficheros de datos. Los datos almacenados pueden ser de dos tipos fundamentales: binarios (imágenes, sonidos, videos) y no binarios (descripciones de texto). (1)

El uso de los lenguajes descriptivos en la implementación de los ficheros es recomendable para garantizar que el compartimiento de la información entre los dos sistemas pueda ser realizado sin problemas. (1)

Con los lenguajes descriptivos podemos almacenar tanto los niveles como los objetos:

- De los objetos se pueden guardar sus entidades y como se estructuran jerárquicamente, y algunas variables propias del objeto que no cambian de un objeto a otro (o poner algunos valores por defecto).
- De los niveles se pueden guardar todos los objetos que intervienen en un determinado nivel, sus posiciones, atributos, y la lógica del sistema. (1)

1.5.1. FICHEROS .INI.

Un ejemplo de uso de lenguajes descriptivos en los ficheros de configuración de varios juegos es el .INI formato de fichero más común en la configuración de aplicaciones para Windows, el término proviene de "Windows **Ini**tialization File".

El .INI cuenta con una estructura muy sencilla, consiste en un simple archivo de texto ASCII que contiene dos tipos de entradas: secciones, permiten agrupar parámetros relacionados y valores, definen los parámetros y su valor; primero se define el nombre del parámetro y después su valor separado por el signo de igualdad (**=**). Además permite la utilización de comentarios con el propósito de explicar una sección o parámetro. Los comentarios comienzan con el carácter punto y coma (**;**). Ver Figura 7

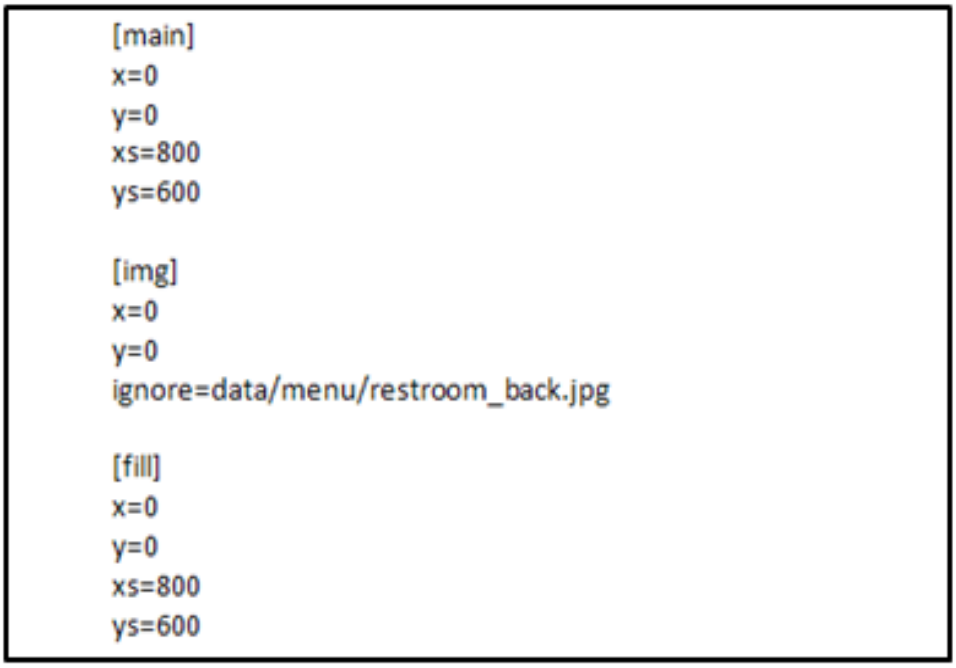


Figura 7. Fragmento del archivo levelup.ini del videojuego Turtle Odyesse 2. (8)

Debido a que las secciones y los valores no se encuentran estrictamente definidos; cada aplicación puede interpretar de una manera distinta: secciones duplicadas, parámetros duplicados, los valores pueden consistir en texto, números, listas separadas por comas, etc. Esto puede hacer de los ficheros .INI difíciles de leer y construir.

Además es importante destacar que el .INI es un formato que solamente se usa en Windows y no es totalmente portable a otros sistemas operativos debido básicamente a que los caracteres de salto de línea no se interpretan igual en todos los sistemas operativos.

Por otra parte el .INI no es capaz de lograr una representación jerárquica de clases.

1.5.2. FICHEROS .XML.

Otro formato de fichero muy popular en el desarrollo no solo de videojuegos sino también de aplicaciones es el XML.

XML[3] acrónimo de e**x**tensible mar**k**up **l**anguage, una forma condensada de SGML[4]. La especificación de XML se publicó en forma de borrador por un grupo de trabajo del consorcio de Word Wide Web (w3c) y es soportada por varias compañías líderes en la industria de las computadoras. (9)

El lenguaje XML es una tecnología sencilla. Se compone de dos partes, por un lado un fichero DTD (Document Type Definition), que especifica la estructura que pueden tener las etiquetas, y por tanto el fichero, y por otro lado los ficheros XML que utilizan estas etiquetas para almacenar y describir la información. (1)

XML tiene algunas limitaciones sobre el SGML[5] por ejemplo que las marcas deben tener su correspondiente de cierre, deben estar incluida completamente en otra, de esta forma se genera un árbol donde se tiene un nodo inicial al que se llama “root”. (1)

Se puede destacar que permite crear etiquetas personalizadas que ofrecen gran flexibilidad para organizar y presentar información, proporciona un sistema de organización de la información que puede ser fácilmente interpretado por cualquier aplicación. Es un estándar portable a cualquier sistema operativo.

1.6. HERRAMIENTAS DE DESARROLLO.

Las herramientas no son más que programas, utilidades, librerías y otras ayudas tales como editores, compiladores y depuradores que se pueden utilizar para desarrollar programas. (9)

Definir cual utilizar en correspondencia a las necesidades de la aplicación a desarrollar es tarea difícil de acuerdo a la variedad existen, pero a la vez muy importante; puesto que pueden definir requisitos del producto final.

1.6.1. BIBLIOTECAS

[3] XML: Extensible Markup Language. Lenguaje de Marcado Extensible.

[4] SGML: Standard Generalized Markup Language. Lenguaje de Marcado Generalizado.

Una de las tareas de los desarrolladores de software es brindar una interfaz sencilla y amigable a sus usuarios. En el mundo existen innumerables bibliotecas que ofrecen soporte para este cometido; entre las que se pueden mencionar: GIMP[6] ToolKit más conocida por GTK+, wxWidgets y Qt. Ver Anexo 1

Hasta la versión 4.5 de Qt, una de las razones por la que algunos desarrolladores de software no usaban este framework era por el licenciamiento dual, a partir de esta versión; Qt también cuenta con licencia LGPL[7]. Es importante destacar el mecanismo desarrollado por Qt de "signal y slot" para la comunicación entre widgets. (12)

Por su parte wxWidgets hace uso de los widgets nativos de cada plataforma, lo que lo hace muy portable de una plataforma a otra. (10)

GTK+, es una de las librerías más usadas en el entorno GNOME para el desarrollo de interfaces graficas de usuario. (11)

1.6.2. ENTORNO DE DESARROLLO INTEGRADOS

Para manipular librarías como Qt existen varios entornos de desarrollo integrado comúnmente conocidos como IDE[8], entre los que se pueden mencionar Code Block, Visual Studio, Eclipse y Qt Creator.

Trolltech para los desarrolladores que gustan de QT ofrece una perfecta integración de herramientas de desarrollo de Qt para las plataformas Visual Studio y Eclipse.

Qt integración con Visual Studio. NET versión 1.4. Ver Figura 8. La integración permite a los desarrolladores: la edición de formas usando Qt Designer, un asistente para la creación de nuevos proyectos y clases Qt. Permite la importación y exportación de Qt proyecto (.pro) e integra la documentación de Qt. (13)

[6] GIMP: GNU Image Manipulation Program. Programa de Manipulación de Imágenes.

[7] LGPL: GNU Lesser General Public License. Licencia Pública General de GNU.

[8] IDE: Integrated Development Environment. Entorno de Desarrollo Integrado.

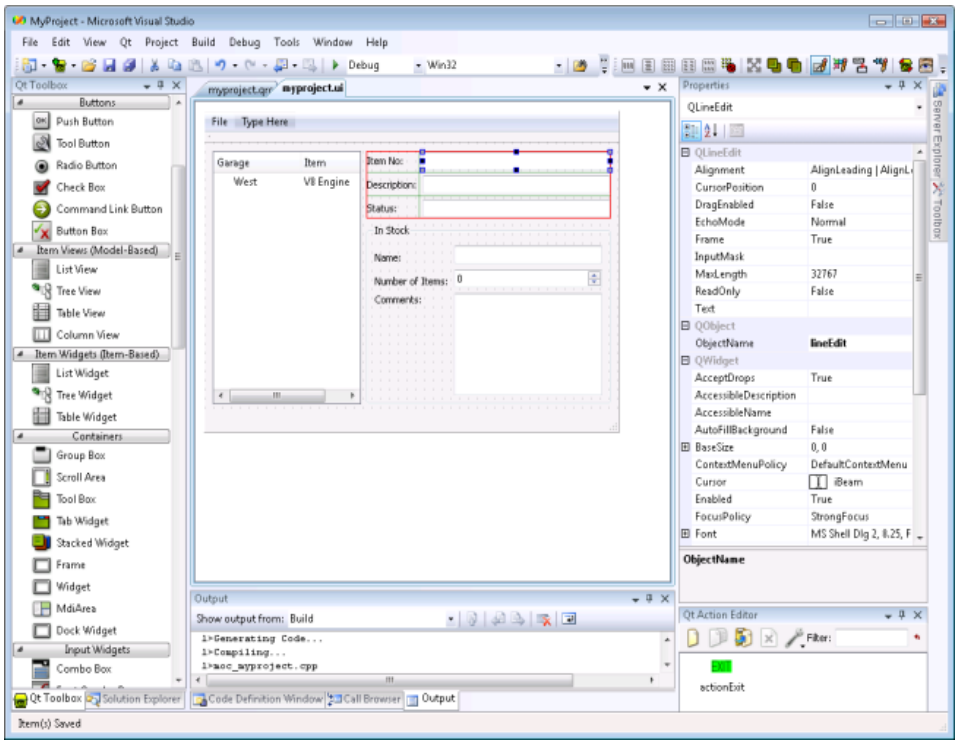


Figura 8. Interfaz de Visual Studio y su integración con Qt.

Qt integración con Eclipse versión 4.4.1, la integración facilita plenamente la unificación del editor Qt Designer, brinda un asistente para la creación de nuevos proyectos y clases Qt. Reúne toda la documentación de Qt. (13)

Qt Creator versión 1.0. Ver Figura 9. IDE para desarrollar proyectos con Qt. Multiplataforma, disponible para: Linux, Mac OS y Windows. Sus principales características son:

- Editor avanzado de código C++, con resaltado de sintaxis y completamiento de código.
- Diseñador de interfaces integrado Qt Designer.
- Herramientas para la administración de proyectos.
- Sistema integrado de documentación sobre Qt4.5 así como del propio IDE.
- Debugger visual e intuitivo.
- Navegación rápida a través del código fuente y clases. (13)

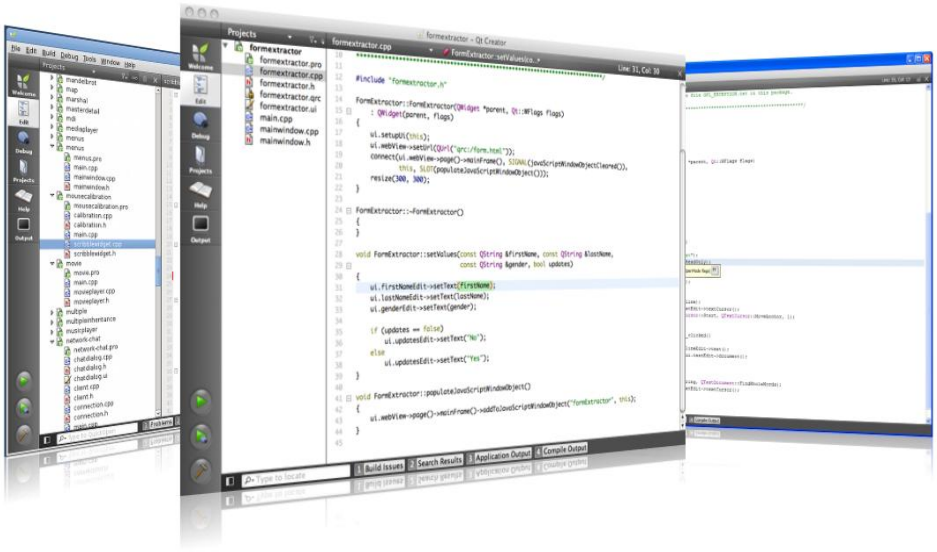


Figura 9. Qt Creator

CONCLUSIONES.

A lo largo de todo el capítulo, para el entendimiento del tema en que se desenvolverá el presente trabajo, se introdujeron conceptos relacionados con la gestión de datos de un videojuego, se hizo un análisis de la forma más factible para separar la implementación de un nivel del juego del código fuente, se presentaron formatos de fichero descriptivos para almacenar información de juegos.

Una conclusión importante del capítulo arrojada a partir del estudio de diversos editores de niveles es que: un editor de niveles debe responder a las necesidades propias del motor de juego para el que fue construido.

Se escoge XML como formato de fichero, para el almacenamiento de datos del software a desarrollar, por razones como: portabilidad, flexibilidad y forma jerárquica de representar la información.

Otra de las decisiones tomadas al término del capítulo es usar Qt como framework dada las potencialidades que brinda para el desarrollo de interfaces así como para el manejo de XML y como entorno de desarrollo integrado se escoge Qt Creator.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA.

INTRODUCCIÓN.

En este capítulo se propone una solución técnica que dé respuesta al problema planteado. Se expone la dinámica del sistema y se brindan soluciones específicas para el almacenamiento de datos.

Además se comienza a tener una visión práctica del sistema a desarrollar. Se definen las reglas del negocio y el modelo de dominio. Se obtienen los requisitos y se desarrolla el modelo de caso de uso del sistema.

2.1. SOLUCIÓN PROPUESTA.

La opción más factible para implementar un nivel es separar los datos del código principal, a las aplicaciones que permiten realizar esta funcionalidad se les nombran editores de niveles.

Un editor de niveles tiene como entrada, datos que definirán los objetos del nivel desde un punto de vista gráfico, sonoro o lógico, y como salida; un fichero de configuración del nivel, el cual podrá ser leído por el juego.

2.1.1. DESCRIPCIÓN DEL SISTEMA.

El sistema que se propone sigue la filosofía de los editores de niveles, pero no es una herramienta de diseño avanzada, puesto que centra su atención hacia la configuración de cada uno de los objetos que se manejan en el dominio del problema. Tiene una entrada de datos, y como salida; genera un fichero que maneja la información necesaria en cuanto a la configuración de niveles del juego. Ver Figura 10.

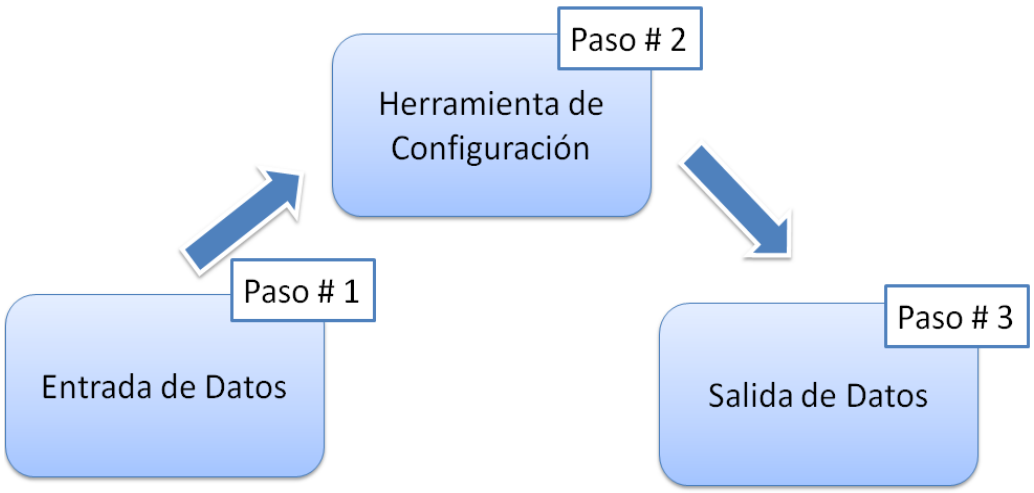


Figura 10. Dinámica de la herramienta propuesta.

2.1.2. FORMATO DEL FICHERO PROPUESTO.

Para realizar la tarea de almacenar los datos del nivel podemos usar un lenguaje descriptivo. Un lenguaje descriptivo incorpora etiquetas o marcas que contienen información adicional acerca de la estructura del texto o su presentación. De esta forma los datos guardados estarán bien estructurados y serán más fáciles de leer y de tratar.

El fichero de configuración que se escogió como salida de la implementación de la herramienta propuesta tendrá extensión .XML.

2.1.2.1. DESCRIPCIÓN DE FORMATO DE FICHERO PROPUESTO.

El primer elemento que contiene el fichero es el encabezado XML. Luego una etiqueta que contiene las siglas del proyecto y un atributo que contiene la versión que se está desarrollando. Posteriormente la etiqueta <Level> la cual define un nivel y en cuyo interior irán todos los objetos pertenecientes a este nivel, dicha etiqueta contiene un atributo Id que hace referencia al número del nivel 1, 2,…. Ver Figura 11

También se definen etiquetas para representar cada uno de los objetos de cada nivel del juego estas se encuentran contenidas dentro de la etiqueta <Level>. Ver **¡Error! No se encuentra el origen de la referencia.**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cneuro>
<CNEURO version="1.0">
  <Level Id="1">
    <Environment></Environment>
    <Player></Player>
    <Opponent></Opponent>
    <Allies></Allies>
    <Tasks></Tasks>
  </Level>
</CNEURO>
```

Figura 11. Representación de la etiqueta Level.

En el sistema se definen etiquetas de uso general. La etiqueta <state> define los estados por los que puede transitar un objeto, contiene las etiquetas <animation> y <sound>. La etiqueta <sound> representa un sonido asociado al estado. La etiqueta <animation>, define una animación; esta contiene las etiquetas <element> y <frame>. La etiqueta <element> representa el conjunto de elementos de diseño 2D y 3D. La etiqueta <frame> define el primero y el último frame de una animación. Por último la etiqueta <name> es muy utilizada para identificar elementos mediante un nombre. Ver del Anexo 3 al Anexo 7.

La etiqueta <Environment> define el contenido de los ficheros de configuración de los niveles 2D y 3D. Estos contendrán todos los elementos de los entornos para un nivel. La etiqueta <Player> contiene toda la información visual y sonora del jugador para un nivel. Según la selección del usuario las entidades del jugador pueden variar, para ello se define la etiqueta <interface> la cual representa las diversas maneras en que puede ser visualizado un jugador. La etiqueta <Opponent> contiene toda la información visual y sonora del oponente. La etiqueta <Allies> contiene todos los aliados de un nivel a su vez, la etiqueta <Allied> representa la información relacionada con el aliado. La etiqueta <Tasks> contiene todas las tareas asignada a un nivel, la etiqueta <Task> encierra la información visual y sonora referente a una tarea. Ver del Anexo 3 al Anexo 7.

2.2. DESCRIPCIÓN DE LA SOLUCIÓN PROPUESTA.

A continuación se expone una descripción de algunos de artefactos definidos por RUP[9] para los flujos de trabajo de Modelamiento de Negocio y Requerimientos.

2.2.1. MODELO DE DOMINIO.

Un modelo de dominio captura los tipos más importantes de objetos en el contexto del sistema. Los objetos del dominio representan "cosas". (14)

El modelo de dominio es una representación visual estática de los objeto en el contexto del proyecto. Es decir, un diagrama con los objetos que existen relacionados con el proyecto y las relaciones que hay entre ellos. Se dice que es una representación estática porque no representa la interacción en el tiempo de los objetos, sino que representa una visión "parada" de las clases y sus interacciones. Ver Figura 12.

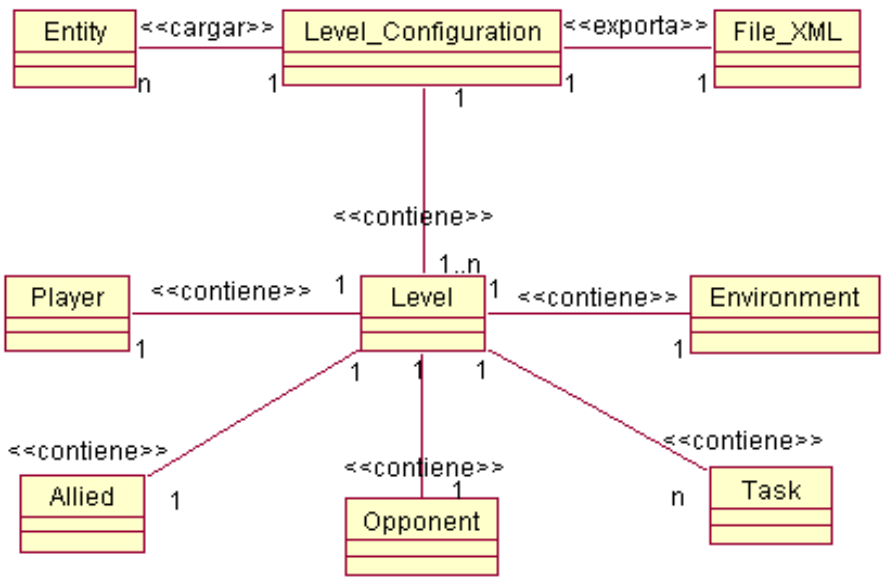


Figura 12. Modelo de Dominio.

2.2.2. GLOSARIO DE TÉRMINOS.

[9] RUP: Rational Unified Process. Proceso Unificado de Rational.

El glosario de términos define los principales vocablos usados en el proyecto. Permite establecer una terminología consensuada. (14)

- Entity: Se denomina a los datos que son utilizados en los motores gráfico y sonoro de un videojuego; estos datos pueden incluir música, efectos, o modelos tridimensionales, texturas o animaciones.
- Level_Configuration: Se denomina a la herramienta que se obtendrá como producto final y la cual manipulará el usuario.
- Level: Se le denomina a una sección del juego.
- Environment: Se le nombra al terreno base para cada uno de los niveles.
- Player: Se le denomina al ente o personaje que representa al jugador.
- Allied: se le denomina al ente o personaje que interactúa con el jugador ofreciéndole ayuda.
- Opponent: Se le denomina al ente o personaje que interactúa con el jugador en forma de reto.
- Task: Se le denomina a las operaciones que deben ser realizadas por el jugador y que son determinadas por el algoritmo adaptativo.
- File_XML: Se denomina al fichero que se exporta y que contendrá todas las entidades de los niveles del videojuego, con un formato XML.

2.2.3. REGLAS DEL NEGOCIO.

Las reglas del negocio describen políticas, normas, operaciones, definiciones y restricciones presentes en el dominio en el que se desempeñara el sistema. (14)

- Los ficheros a cargar con el contenido del entorno deben estar en formato CFG2D, CFG3D y cumplir con todas las especificaciones de este formato.
- Los ficheros a cargar de imágenes deben estar en formato PNG y cumplir con todas las especificaciones de este formato.
- Los ficheros a cargar de sonidos deben estar en formato OGG y cumplir con todas las especificaciones de este formato.
- El fichero XML que se exporta debe cumplir con las especificaciones que proponen en el capítulo anterior.

2.2.4. REQUISITOS DEL SISTEMA.

Los requisitos establecen que tiene que hacer exactamente el sistema que se construye. (14) Los requisitos en un sistema de software son el contrato que se debe cumplir, de modo que los usuarios finales tienen que comprender y aceptar los requisitos que se especifiquen. Los requisitos se dividen en dos grupos. Los requisitos funcionales y los no funcionales.

2.2.4.1. REQUISITOS FUNCIONALES

Un requisito funcional especifica una acción que debe ser capaz de realizar el sistema, sin considerar restricciones físicas; requisito que especifica condiciones de entrada/salida de un sistema. Los requisitos funcionales representan la funcionalidad del sistema. Se modelan mediante diagramas de Casos de Uso. (14)

R1.Configurar Nivel.
- R1.1. Crear Nivel.
- R1.2. Modificar Nivel.
- R1.3. Eliminar Nivel.

R2.Visualizar.
- R2.1. Visualizar Animación.

R3. Gestionar Fichero de Configuración del Nivel.
- R3.1. Cargar Fichero de Configuración.
- R3.2. Salvar Fichero de Configuración.

2.2.4.2. REQUISITOS NO FUNCIONALES

Los requisitos no funcionales representan aquellos atributos que debe exhibir el sistema, pero que no son una funcionalidad específica. Son requisitos que especifican propiedades del sistema. (14)

- Usabilidad.
  Los usuarios que utilizarán el sistema deberán tener conocimiento básico del manejo de la computadora, así como del trabajo con sistemas operativos visuales.

La aplicación deberá poseer una interfaz y navegación acorde y funcional, tanto para usuarios expertos, como para los que no tienen conocimientos profundos de Informática.

- Software.
  La aplicación podrá ser ejecutada, para los sistemas operativos: Windows XP o superior, Mac OS, y la familia Unix.

- Hardware.
  Los requisitos mínimos para la ejecución de la aplicación son: Procesador Intel Pentium IV de 3.0 MHz (o equivalente) y versiones posteriores, 256 de RAM y 200 MB de espacio en disco disponible.

- Restricciones en el Diseño e Implementación.
  Lenguaje de programación: C++.
  Paradigma de programación: Orientado a Objeto.
  Framework Qt 4.5 para construir la aplicación visual y el manejo de fichero XML.
  Entorno de Desarrollo Integrado: Qt Creator 1.0.

2.2.5. MODELO DE CASO DE USO DEL SISTEMA.

El modelo de Casos de Uso está formado por actores, casos de uso y las relaciones entre ambos; este modelo describe lo que el sistema debe hacer por sus usuarios y bajo qué restricciones. (14)

2.2.5.1. ACTOR DEL SISTEMA.

El actor del sistema representa el rol que juega una o varias personas, un equipo o un sistema automatizado. El cual interactúa con el sistema, pero no son parte de él. Ver Tabla 1

Tabla 1. Actor del sistema.

Actor	Descripción
Developer	Interactúa con el sistema, encargado de la configuración los niveles.

2.2.5.2. CASOS DE USO DE SISTEMA

Los casos de uso son fragmentos de funcionalidades que el sistema ofrece para aportar un resultado de valor para sus actores. Los casos de uso capturan requisitos potenciales del software. Cada caso de uso proporciona uno o más escenarios que indican cómo debería interactuar el sistema con el usuario o con otro sistema para conseguir un objetivo específico. (14) Ver Tabla 2 a la Tabla 4. Ver Anexo 8 y Anexo 9.

2.2.5.2.1. PATRÓN DE CASO DE USO.

El patrón de caso de uso que se escoge para modelar el sistema es CRUD (Create, Read, Update, Delete).

Tabla 2. Caso de Uso del Sistema Management Level.

CU 1	Management Level.
Actor	Developer
Referencia	R1, R1.1, R1.2, R1.3
Prioridad	Alta.
Descripción	El presente caso de uso está relacionado con la configuración de los datos del nivel, aquí converge todo lo referente a la creación, modificación y eliminación de un nivel.

Tabla 3. Caso de Uso del Sistema Visualise.

CU 2	Management View
Actor	Developer
Referencia	R2, R2.1
Prioridad	Baja.
Descripción	El presente caso de uso está relacionado con la visualización en el momento de la carga de las animaciones correspondiente a objetos móviles como Player, Opponent y Allied.

Tabla 4. Caso de Uso del Sistema Management File.

CU 3	Management File.

Actor	Developer
Referencia	R3,R3.1,R3.2
Prioridad	Alta.
Descripción	El presente caso de uso está relacionado con el almacenamiento de los datos del nivel, aquí converge todo lo referente a salvar y cargar el fichero de configuración de un nivel.

2.2.5.2.2. DIAGRAMA DE CASOS DE USO.

Los diagramas de casos de uso especifican la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios y otros sistemas. O lo que es igual, un diagrama que muestra la relación entre los actores y los casos de uso en un sistema. Los diagramas de casos de uso se utilizan para ilustrar los requerimientos del sistema al mostrar cómo reacciona una respuesta a eventos que se producen en el mismo. (14) Ver Figura 13.

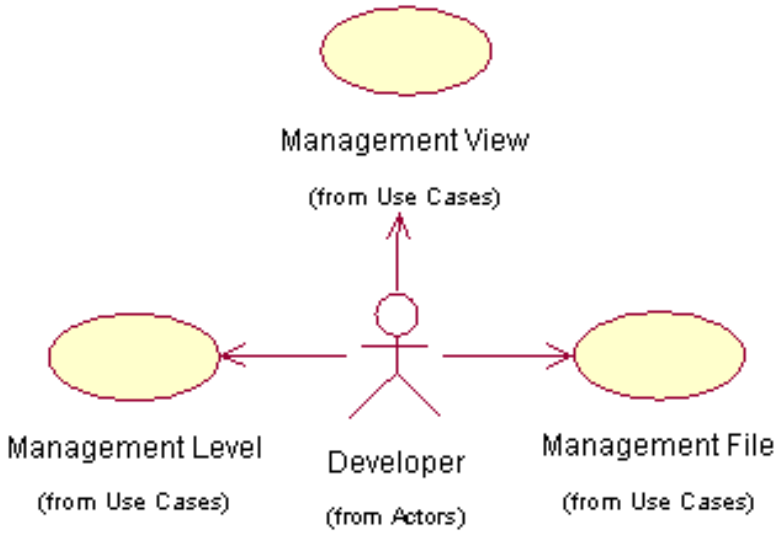


Figura 13. Diagrama de Caso de Uso del Sistema.

2.2.6. VISTA ARQUITECTÓNICA DEL MODELO DE CASOS DE USO.

Esta vista de la arquitectura abarca los casos de uso significativos desde un punto de vista arquitectónico. (14) En el sistema se definen como casos de uso crítico y significativo para la arquitectura: CU Management Level y CU Management File. Ver Figura 14

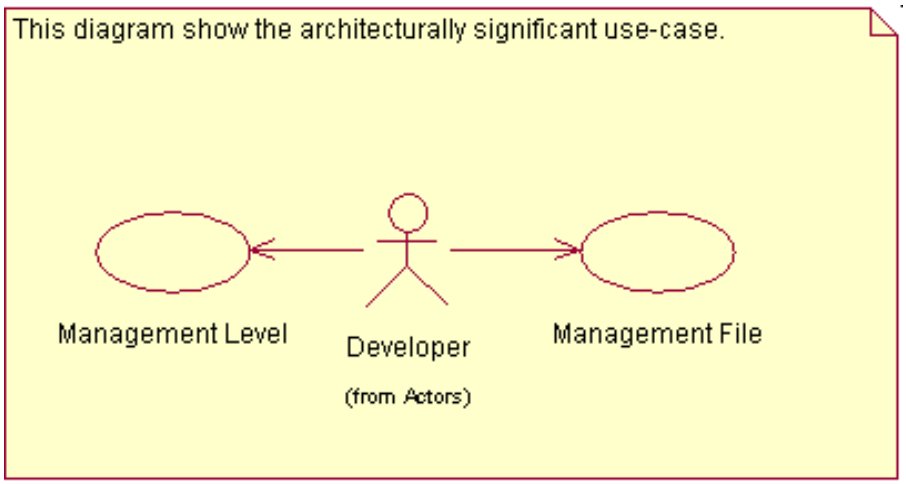


Figura 14. Vista arquitectónica del modelo de casos de uso

2.2.7. PROTOTIPO DE INTERFAZ DE USUARIO.

La interfaz de usuario (también conocida por sus siglas en inglés como GUI[10]) es la encargada de la interacción del usuario con el software. Un prototipo de interfaz de usuario, fundamentalmente es un prototipo ejecutable de una interfaz de usuario, pero que puede, en los momentos iniciales de desarrollo, constituir únicamente diseños de pantallas etc. Los prototipos ilustran cómo pueden utilizar el sistema los usuarios para ejecutar los casos de uso. (14)

Ver Figura 15.

[10] GUI: Graphical User Interface. Interfaz Gráfica de Usuario.

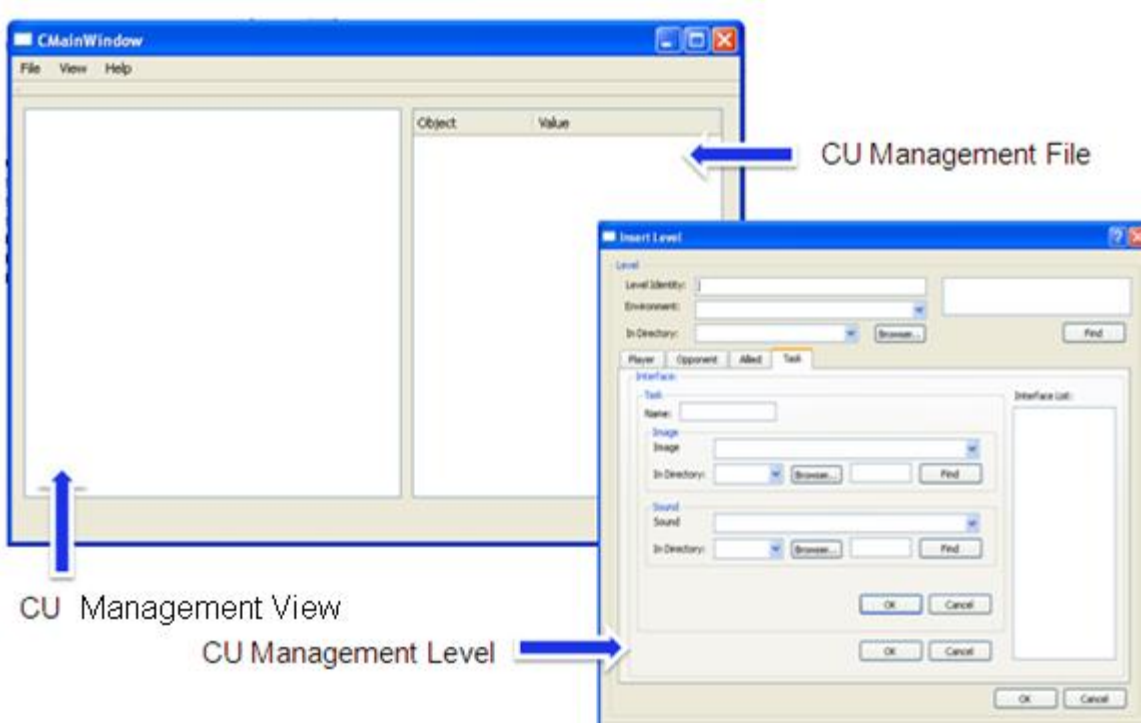


Figura 15. Prototipo de Interfaz de Usuario.

CONCLUSIONES.

El presente capítulo deja sentadas las bases técnicas sobre las cuales será implementada la herramienta de configuración de niveles, para dar solución al objetivo propuesto. Además como parte del segundo epígrafe del capítulo se definió qué espera el usuario con este sistema. Para ello quedaron establecidos sus requisitos funcionales y se describieron los casos de uso que permitirán al usuario obtener los resultados esperados.

CAPÍTULO 3: ANÁLISIS Y DISEÑO DEL SISTEMA

INTRODUCCIÓN

El capítulo se desarrolla a través de los flujos de trabajo de Análisis y Diseño. El modelo de análisis como traza directa del modelo de caso de uso permite razonar sobre aspectos internos del sistema; para una comprensión más precisa, los requisitos son estructurados en clases y paquetes de análisis. En el diseño se modela el sistema, se define una arquitectura que soporte los requisitos.

3.1. MODELO DE ANÁLISIS.

El modelo de análisis contiene las clases del análisis y sus objetos organizados en paquetes que colaboran. Es un modelo de objetos cuyos propósitos son: describir los requisitos de forma precisa; estructurarlos de manera que facilite su comprensión; y servir de punto de partida para dar forma al sistema durante su diseño e implementación, incluyendo su arquitectura. (14)

3.1.1. VISTA ARQUITECTÓNICA DEL MODEL O DE ANÁLISIS

La vista arquitectónica del modelo de análisis abarca las clases, paquetes y la realización del caso de uso del análisis; se refinan y estructuran los requisitos del sistema. (14)

3.1.1.1. PAQUETES Y CLASES DEL ANÁLISIS

Un paquete del análisis proporciona los medios para organizar los artefactos del modelo de análisis en piezas manejables. (14)

El sistema propuesto consta de tres paquetes que consisten en la división de las clases del análisis según el rol que desempeña. Ver Figura 16

El paquete Level Configuration GUI engloba las clases que tienen como responsabilidad modelar la interacción ente el sistema y sus actores, o sea, clases interfaz; para las cuales UML[11] define el estereotipo de “boundary”. Como clases interfaz se tienen: CI Create Level, CI Update Level, CI

[11] UML: Unified Model Language. Lenguaje Unificado de Modelado.

Delete Level, CI Load Level, CI Save Level y CI View Level Animation, cada una de estas clases responden a la realización de las funcionalidades del sistema.

El paquete Level Configuration Core comprende las clases que representan la coordinación, secuencia y control de otros objetos y se usan para encapsular control referido a un determinado caso de uso; clases controladoras para las que UML define el estereotipo de “control”. Entre las clases controladoras se encuentran: CC Management Level, CC Management File, CC Management View.

El paquete Level Configuration Object encierra las clases usadas para modelar la información de larga duración, clases entidad; para ellas UML define el estereotipo de “entity”. Se definen como clases entidad: CE Level, CE Environment, CE Player, CE Allied, CE Opponent, CE Task.

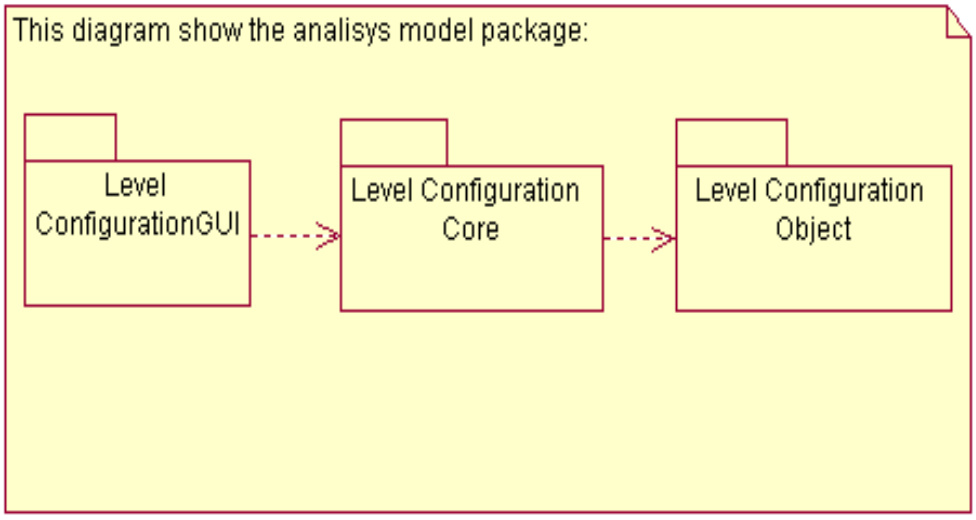


Figura 16. Diagrama de paquetes del análisis.

3.1.1.2. REALIZACIÓN DE CASOS DE USO-ANÁLISIS

Una realización de casos de uso-análisis es una colaboración dentro del modelo de análisis que describe cómo se llevan a cabo y se ejecutan los casos de uso determinado en términos de las clases de análisis y de sus objetos de interacción. Proporciona una traza directa hacia los casos de uso concretos del modelo de casos de uso. (14)

En el sistema que se propone fueron planteados tres casos de uso. Ver Figura 13. A continuación se realiza una descripción textual del flujo de sucesos y se identifican las clases cuyos objetos son necesarios para la realización los casos de uso-análisis.

En el CU Management Level intervienen las clases del análisis: CC Management Level la cual es responsable de controlar el flujo de datos desde las clases CI Create Level, CI Update Level y CI Delete Level las que permiten al usuario gestionar la información referente a cada uno de los niveles del juego; a las clases entidad CE Level, CE Environment, CE Player, CE Opponent, CE Allied, CE Task. Ver Anexo 10.

En el CU Management File involucra las clases del análisis: CC Management File la cual es responsable de coordinar la interacción entre las clases interfaz CI Load Level y CI Save Level que permiten al usuario cargar y salvar la información a un fichero de datos de extensión .XML; y las clases entidad CE Level, CE Environment, CE Player, CE Opponent, CE Allied, CE Task las que representan información de larga duración referente a cada uno de los objetos definidos para un nivel del juego. Ver Anexo 11.

En el CU Management View están presente las clases del análisis: CC Management View la cual es responsable de coordinar la forma en que será visualizada la información a través de la clase interfaz CI View Level Animation a partir de los datos almacenados en las clases CE Player, CE Opponent y CE Allied. Ver Anexo 12.

3.2. MODELO DE DISEÑO.

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema. Además, sirve de abstracción de la implementación del sistema y es, de ese modo, utilizada como una entrada fundamental de las actividades de implementación.

3.2.1. VISTA ARQUITECTÓNICA DEL MODELO DE DISEÑO.

Vista de la arquitectura de un sistema abarcando las clases, subsistemas, interfaces y realizaciones de casos de uso de diseño. (14)

La arquitectura sobre la que se desarrolla la aplicación es una arquitectura en tres capas. La carga se divide en tres partes con un reparto claro de funciones: una capa para la presentación (interfaz de usuario), otra para la gestión del flujo de datos (donde se encuentra modelado el dominio) y la administración del fichero de configuración (persistencia) y por ultimo una capa que es esta representada por la biblioteca de clases Qt (soporte). Ver Figura 17

3.2.1.1. PAQUETES Y CLASES DEL DISEÑO.

Los subsistema de diseño y clases son abstracciones del subsistema y componentes de la implementación del sistema. Estas abstracciones son directas, y representan una sencilla correspondencia entre el diseño y la implementación. (14)

A partir de la descomposición en paquetes de análisis, se realiza una traza directa a los subsistemas o paquetes de diseño los cuales se definen a continuación. Ver Figura 17.

El paquete Level Configuration GUI contiene las interfaces de usuario para la realización de cada uno de los casos de uso. En el presente sistema solo se define una interfaz de usuario denominada CMainWindow, esta es una ventana la cual hereda todos los atributos propios de una clase QMainWindow, como son: QMenuBar, QToolBar y QStatusBar. Ver Anexo 16.

El paquete Level Configuration Core engloba algunas de las clases que modelan objetos definidos durante el modelado del dominio del proyecto, como son: CLevel, CEnvironment, CPlayer, COpponent, CAllied y CTask; clases que modelan información de cada uno de los objetos como CInterface, CState, CDesingElement, CAnimation, CSound. Las clases antes mencionadas heredan del la clases QTreeItemWidget. Por último y no menos importante una clase que controla el flujo de datos CLevelController la cual hereda del la clase QTreeWidget. Ver del Anexo 11 al Anexo 14.

El paquete Level Configuration File contiene clases que permiten la persistencia de los datos en un formato de fichero .XML, se definen las clases: CWriter la cual hereda de las clases

QXMLStreamWriter y la clase CReader que hereda de la clase QXMLStreamReader. Este paquete solo cuenta con dos interfaces bWriteFile (QIODevice *) y bReaderFile (QIODevice *). Ver Anexo 15.

Las principales interfaces de Qt que son utilizadas por la aplicación son QMainWindow, la cual provee de una ventana principal a la aplicación; por defecto contiene un QMenuBar, QToolBar y QStatusBar, a la que se incorporan otros componentes como QGroupBoxWidget, QTabWidget, QLineEdit, QLaber, QPushButton, QToolButton y QButtonBox. Se utiliza la clase QTreeWidget para modelas la estructura jerárquica de clases del sistema esta clase crea una vista de árbol a partir de los modelos brindados por la clase QTreeItemWidget. Para el trabajo con XML se emplean las clases QXMLStreamWriter y QXMLStreamReader.

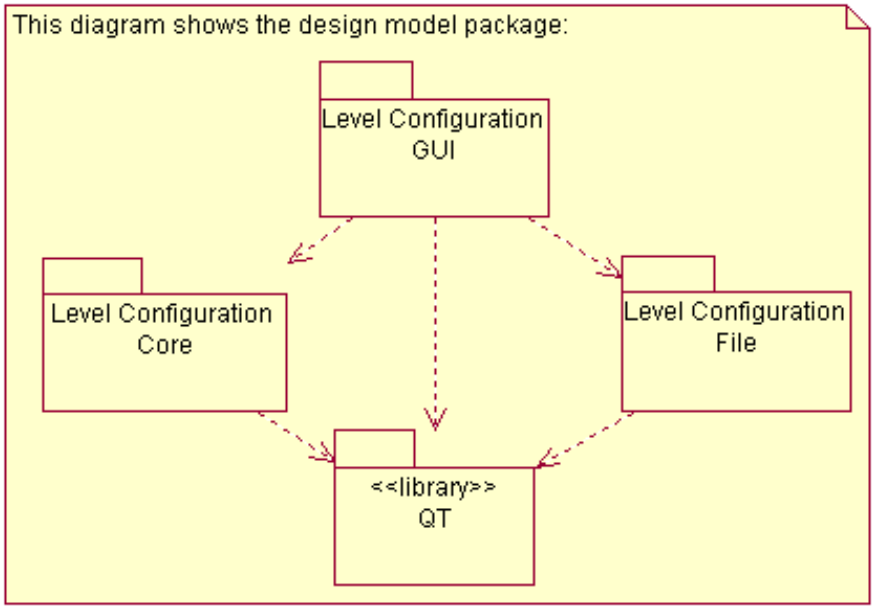


Figura 17. Diagrama de paquetes del diseño.

3.2.1.2. PATRONES DE DISEÑO.

Patrones de asignación de responsabilidades, GRASP:

Experto: El experto en información es el principio básico de asignación de responsabilidades. Nos indica que la responsabilidad de la creación de un objeto debe recaer sobre la clase que conoce toda la información necesaria para crearlo. (15)

Creador: El patrón creador ayuda a identificar quién debe ser el responsable de la creación de nuevos objetos o clases. (15)

Controlador: El patrón controlador es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado. (15)

Alta Cohesión y Bajo Acoplamiento:

Alta Cohesión: La información que almacena una clase debe de ser coherente y está en la mayor medida de lo posible relacionada con la clase. (15)

Bajo Acoplamiento: Consiste en tener las clases lo menos ligadas entre sí. De tal forma que en caso de producirse una modificación en alguna de ellas. (15)

3.2.1.3. REALIZACIÓN DE CASO DE USO-DISEÑO.

Una realización de caso de uso-diseño es una colaboración en el modelo de diseño que describe cómo se realiza un caso de uso específico, y como se ejecuta, en términos de clases de diseño y sus objetos. Una realización de caso de uso-diseño proporciona una traza directa a una realización de caso de uso-análisis.

A continuación se realiza una descripción textual del flujo de sucesos y se identifican las clases cuyos objetos son necesarios para la realización los casos de uso-diseño.

En el CU Management Level intervienen las clases del diseño: CLevelController la cual es responsable de controlar el flujo de datos desde la clase interfaz CMainWindow a las clases CLevel, CEnvironment, CPlayer, COpponent, CAllied, CTask.

En el CU Management File involucra las clases del diseño: CWriter la cual es responsable de escribir un XML y CReader el cual implementa un parser para XML estas clases permiten la persistencia de los datos manejados por la clase CLevelController; a las funcionalidades ofrecidas por estas clases se accede desde la interfaz principal QMainWindow a través de las opciones Open y Save.

En el CU Management View están presente las clases del diseño: CAnimation_Render la cual es responsable de coordinar la forma en que será visualizada la información a través de la clase interfaz CMainWindow a partir de la secuencia de imágenes que conforman una animación para las clases CPlayer, COpponent y CAllied.

3.3. VISTA ARQUITECTÓNICA DEL MODELO DE DESPLIEGUE.

Esta vista de la arquitectura del sistema abarca los nodos que forman la topología hardware sobre la que se ejecuta el sistema, vista que aborda la distribución, entrega e instalación de las partes que constituyen el sistema físico. (14)

Dado a que el sistema es una aplicación desktop no distribuida el diagrama de despliegue se representa con un único nodo. Ver Figura 18

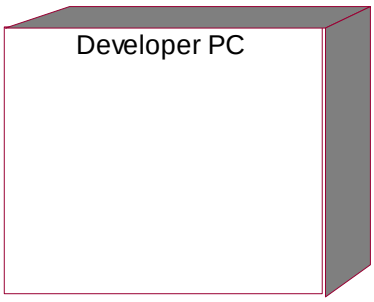


Figura 18. Vista del Modelo de Despliegue.

CONCLUSIONES

Al culminar el capítulo se tiene concebido detalladamente el análisis y diseño del sistema, a partir de la realización de casos de uso para cada uno de los flujos de trabajo se han refinado los requisitos funcionales del sistema y a partir de alguna decisiones como el uso del framework Qt se cumple requisitos no funcionales como la portabilidad. La arquitectura elegida es sencilla fácil de implementar y flexible a cambios en la estructura de clases. Con la conclusión del capítulo se puede pasar a la fase de construcción del producto final.

CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBA.

INTRODUCCIÓN.

Esta parte del trabajo alcanza su mayor desarrollo durante la fase de construcción definida por RUP, constituye el paso del diseño de clases a la creación de componentes físicos, que se traducen en ficheros .h, .cpp y .ui correspondientes a la implementación en C++ y Qt.

Además el capítulo incluye actividades de gestión de proyecto como es la determinación de un estándar de implementación.

También se incluyen actividades del flujo de trabajo de prueba como parte de la puesta en marcha de la aplicación; y de la validación y verificación del cumplimiento de los requisitos del sistema.

4.1. MODELO DE IMPLEMENTACIÓN.

Este modelo describe como se implementan los elementos del modelo de diseño. Las clases del diseño en términos de componentes representan archivos de código fuente y ejecutables. (14)

4.1.2. VISTA ARQUITECTÓNICA DEL MODELO DE IMPLEMENTACIÓN.

Vista de la arquitectura del sistema que abarca los componentes usados para el ensamblado y lanzamiento del sistema físico. (14)

4.1.3. SUBSISTEMAS Y COMPONENTES DE IMPLEMENTACIÓN.

Los subsistemas de implementación proporcionan una forma de organizar los artefactos del modelo de implementación en trozos más manejables. Ver Figura 19

Hay que destacar la relación del componente main.cpp; para el cual UML ofrece el estereotipo “Main Program”, este componente es el encargado de crear la aplicación a partir de un objeto de la clase QApplication.

El paquete Level Configuration GUI contiene las interfaces de usuario para la realización de cada uno de los casos de uso. En el presente sistema solo se define una interfaz de usuario denominada en diseño CMainWindow como traza de implementación esta clase se traduce en componentes como CMainWindow.h, CMainWindow.cpp y CMainWindow.ui

El paquete Level Configuration Core engloba los componentes que representan objetos del dominio como: CLevel.h y CLevel.cpp, CEnvironment.h y CEnvironment.cpp, CPlayer.h y CPlayer.cpp, COpponent.h y COpponent.cpp, CAllied.h y CAllied.cpp, y CTask.h y CTask.cpp; clases que modelan información de cada uno de los objetos como CInterface.h y CInterface.cpp, CState.h y CState.cpp, CDesingElement.h y CDesignElement.cpp, CAnimation.h y CAnimation.cpp; así como CSound.h y CSound.cpp. Por último la clase que controla el flujo de datos CLevelController.h y CLevelController.cpp.

El paquete Level Configuration File contiene clases para escribir y leer un XML: CWriter.h y CWriter.cpp, y CReader.h y CReader.cpp.

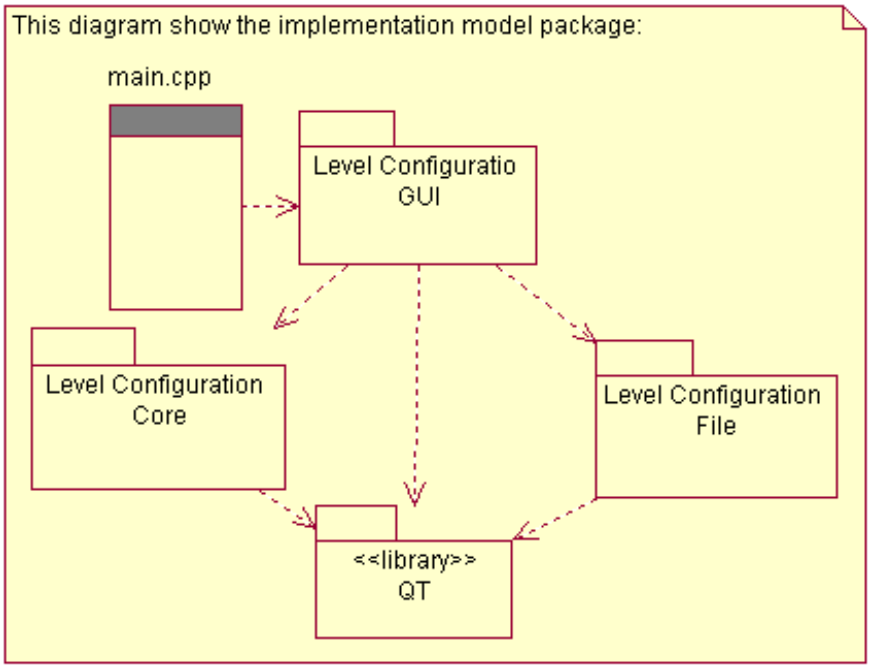


Figura 19. Diagrama de subsistemas de implementación.

4.2. ESTÁNDAR DE CODIFICACIÓN.

La codificación del sistema sigue algunos estándares propuestos por la CNEUROGameEngine. Se respetan los estándares de codificación de C++, lenguaje de desarrollo escogido.

Nombre de los ficheros:

Los nombres de los ficheros .h, .cpp y .ui utilizan las iniciales del nombre del proyecto (CNEURO).

Clases:

Se utiliza el indicador "**C**" para indicar que es una clase.

Ejemplo: class **CC**lassName.

Listas:

Para los tipos de datos utilizados de la biblioteca Qt (QList, QVector, QMap, QMultimap, etc.), se utiliza el indicador "**T**", con los sufijos **List**, **Map** y **MultiMap** según la estructura. Además el nombre lleva el tipo de dato a almacenar en la estructura en cuestión.

Ejemplo: QList<CLevel*> **T**level**List**.

Declaración de variables:

Los nombres de las variables comienzan con un identificador del tipo de dato al que correspondan, como se muestra a continuación. En el caso de que sean variables miembros de una clase, se le antepone el identificador "**m**_" (en minúscula) y en caso de ser argumentos de algún método, se les antepone el prefijo "**arg**_".

Ejemplos de tipos simples: bool **b**VarName, int **i**Name, unsigned int uiName, float **f**Name, char **c**Name, char* **ac**Name, char* **pc**Name, char** **aac**Name, char** **apc**Name, bool **m**_**b**MemberVarName, short **s**Name, void* **pv**Name.

Ejemplo de instancias de tipos creados**:** CclassName **k**ObjectName, CclassName* **pk**Name, CclassName* **ak**Name, CclassName* **m**_**ak**Name.

Métodos:

En el caso de los métodos, se les antepone el identificador del tipo de dato de devolución, y en caso de no tenerlo (void), no se les antepone nada. Los constructores y destructores, como lo exigen los compiladores, llevan el nombre de la clase.

Ejemplo de constructor y destructor: CclassName (bool arg_bVarName, float& arg_fVarName), ~CclassName ().

Métodos de acceso a miembros:

Los métodos de acceso a los miembros de las clases no se nombrarán “Gets” ni “Sets”, sino como los demás métodos, pero **con el nombre** de la variable a la que se accede y sin el prefijo “**m**_”:

Ejemplo para la variable: int **m**_**i**MyVar, los métodos de acceso serían: int **i**MyVar (), void MyVar (int arg_**i**MyVar).

4.3. MODELO DE PRUEBA.

Modelo que describe fundamentalmente cómo el componente ejecutable del modelo de implementación son probados. En este artefactos se incluyen los casos de prueba los cuales incluyen entrada o resultado con la que se ha de probar y las condiciones bajo las que ha de probarse. (14)

A continuación se exponen los casos de prueba que especifican cómo probar un caso de uso; estos casos de prueba tienen trazas con el modelo de casos de uso.

Un caso de prueba basado en casos de uso especifica típicamente una prueba del sistema como caja negra, es decir, una prueba del comportamiento observable externamente del sistema. (14)

Tabla 5. Caso de prueba "Adicionar entorno a un nivel donde ya existe este objeto"

Caso de Uso	Management Level.
Caso de Prueba	Adicionar entorno, cuando ya existe uno.
Entrada	Seleccionar la opción Adicionar Entorno.
Resultado	Muestra un mensaje de alerta, no se crea un nuevo entorno.
Condiciones	Para un nivel solo se contiene un objeto entorno.

Tabla 6 Caso de prueba "Cargar fichero XML que no cumpla con el formato especificado"

Caso de Uso	Management File.
Caso de Prueba	Cargar fichero XML que no cumpla con el formato especificado.
Entrada	Seleccionar la opción Cargar Nivel.
Resultado	Muestra un mensaje de alerta al usuario, no se carga el fichero.
Condiciones	El fichero debe cumplir con las especificaciones de un fichero CNEURO.

Tabla 7 Caso de prueba "Seleccionar un fichero de extensión incorrecta"

Caso de Uso	Management View.
Caso de Prueba	Seleccionar un fichero de extensión incorrecta.
Entrada	Seleccionar la opción Buscar fichero.
Resultado	El sistema no permite seleccionar una extensión diferente a PNG.
Condiciones	El fichero debe tener extensión PNG

CONCLUSIONES

Al término del capítulo están sentadas las bases para la implementación de los casos de uso del sistema, siguiendo un estándar de codificación. Se elaboro un diseño de casos de prueba que permitirán la validación de cada una de las funcionalidades del sistema.

CONCLUSIONES

Disponer de herramientas para la gestión de configuración de niveles es vital para un motor de juego. En el presente trabajo, en cumplimiento con el objetivo planteado, se construyó un sistema que se adapta perfectamente a las necesidades requeridas por el motor de juego CNEUROGameEngine y brindando la posibilidad de crear, modificar, eliminar y guardar en un fichero de datos la configuración para los componentes definidos en cada nivel del juego.

Para el cumplimiento del objetivo, se requirió primeramente hacer un estudio de varios temas, tales como: proceso de configuración y formatos de fichero para almacenar información de videojuegos, y principales características de las bibliotecas multiplataforma para el desarrollo de interfaces gráficas de usuarios; así como, bibliotecas para el manejo de ficheros XML.

A partir de esta investigación se proponen las características técnicas de la solución. Se realizó la captura de los requisitos funcionales y de los no funcionales, y la agrupación de los primeros en casos de uso del sistema. Luego se procede a la realización de los casos de uso a través de los flujos de trabajo de análisis, diseño e implementación.

RECOMENDACIONES

Este trabajo se realizó con el objetivo de desarrollar una herramienta que permitiera la gestión de datos de un juego. Se recomienda continuar profundizando en este tema. Como trabajos futuros se proponen las siguientes mejoras a la herramienta desarrollada:

- Incorporarle soporte para más idiomas mediante las bondades que brinda QT Linguist.
- Incorporar un manual de usuario para el fácil entendimiento de la aplicación.
- Crear interfaces más intuitivas, aumentado con ello la usabilidad del software.
- Incorporar reproducción de sonidos asociado a un estado.
- Incorporar una funcionalidad que permita salvar medias al usuario a una carpeta predefina.

BIBLIOGRAFÍA

IBM Corporation. 2006. *Rational Unificied Process versión 7.2.* 2006.

Michael, David y Chen, Sande. *Serious Game-Game that Educate, Train and Inform.* Canada : s.n.

Nokia. *User Interface Styling Made Simple.* Oslo : s.n., 2008.

Nokia. *Creating Cross-Platform Visualization UIs whit Qt and OpenGL.* Oslo : s.n., 2008.

Nokia. *Putting XML to Work in Your Application with XQuery.* Oslo : s.n., 2008.

Pressman, R. 1998. *Ingeniería de software. Un enfoque práctico.* Madrid : Mc Graw-Hill Interamericana de España S.A : s.n., 1998.

Watts, **Humphrey. 2000.** *Introduction to the Team Software Process.* Addison Wesley : s.n., 2000.

REFERENCIAS

1. **Gavaldà, Jordi Duch i y Tejedor Navarro, Heliodoro.** *Lógica de un videojuego.* Catalunya : s.n., 2008.

2. **Gavaldà, Jordi Duch i y Navarro, Heliodoro Tejedor.** *Introducción a los videojuegos.* Catalunya : s.n., 2008.

3. **Consorcio Periodístico de Chile S.A.** *La Tercera.* [En línea] 2008. http://www.latercera.cl/contenido/27_19623_9.shtml.

4. **Pinball Blog.** *Pinball.* [En línea] 2009. http://blogpinball.blogspot.com/2009/04/la-historia-de-la-pinball-construction.html.

5. **NBA Live Series Center.** *NBA Live Series Center.* [En línea] 2008. www.nba-live.com/eagraph/ .

6. **Marrero, Ismael Viamonte y Garay, Sádor Labrada.** *Editor de Pistas de Carrera para el proyecto Juego de Consola.* La Habana : s.n., 2008.

7. **DXTRE3D's Web Site.** *DXTRE3D.* [En línea] http://www.dxtre3d.com/.

8. **SOFTONIC.** *softonic.* [En línea] 2007. http://turtle-odyssey-2.softonic.com/.

9. **Microsoft Corporation.** *Diccionario de informática e Internet de Microsoft.* 2005.

10. **Nokia.** *Qt - A cross-platform application and UI framework* . [En línea] 2008. http://www.qtsoftware.com/.

11. **The GTK+ Team.** *The GTK+ Project.* [En línea] 2007. http://www.gtk.org/.

12. **Kevin Oliver.** *wxWidgets.* [En línea] http://www.wxwidgets.org/.

13. **Nokia.** *Online Reference Documentation.* [En línea] 2008. http://doc.trolltech.com/.

14. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** *El Proceso Unificado de Desarrollo de Software Volumen I.* La Habana : Felix Varela, 2004.

15. **Larman, Craig.** *UML y Patrones.* Mexico : Prentice Hall Hispanoamericana, 1999.

ANEXOS

ANEXO 1.

Tabla 8. Tabla comparativa entre las bibliotecas GTK+, wxWidgets y Qt

	GTK +	WxWidgets	Qt
Multiplataforma	Si	Si	Si
Lenguaje de desarrollo	C	C++	C++
Licencias	GNU Lesser General Public License (LGPL).	GNU Lesser General Public License (LGPL).	GNU Lesser General Public License (LGPL). GNU General Public License (GPL). Qt Commercial License Agreement.
Lenguajes que Soportan	C, C++, C#, Python, Perl, PHP, Java, Ruby	C++, C#, Python, Perl	C, C++, Python, Java, Perl, Gambas, Ruby, PHP y Mono
Manejo de Widgets	Si	Si	Si
Otros soportes	Biblioteca para gráficos; (GDK). Biblioteca para interfaces con características de una gran; (ATK). Biblioteca para el diseño y renderizado de texto; (Pango). Biblioteca de renderizado avanzado de controles de aplicación; (Cairo)		Multimedia. Gráfico 2D y 3D. XML. Bases de Datos, SQL. Renderizado de WebKit. Netscape Plug-in API.

ANEXO 2.

Tabla 9. Tabla comparativa entre las bibliotecas Boost y Qt.

	Boost	Qt
Flexibilidad	No	Si
Parse para SAX2	-	Si
Parse para DOM nivel 2	-	Si
Serialización	Si	Si

ANEXO 3.

Figura 20. Representación de la etiqueta Environment.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cneuro>
<CNEURO version="1.0">
  <Level Id="1">
    <Environment>environment.cfg.2d</Environment>
  </Level>
</CNEURO>
```

ANEXO 4

Figura 21. Representación de la etiqueta Player

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cneuro>
<CNEURO version="1.0">
  <Level Id="1">
    <Player>
      <interface>
        <name>player1</name>
        <states>
          <state>
            <name>state1</name>
            <animation>
              <element>element1.png</element>
              <frame>
                <first>1</first>
                <last>2</last>
              </frame>
            </animation>
            <sound>sound1.ogg</sound>
          </state>
        </states>
      </interface>
    </Player>
  </Level>
</CNEURO>
```

ANEXO 5.

Figura 22. Representación de la etiqueta Opponent.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cneuro>
<CNEURO version="1.0">
  <Level Id="1">
    <Opponent>
      <name>opponent1</name>
      <states>
        <state>
          <name>state1</name>
          <animation>
            <element>element1.png</element>
            <frame>
              <first>1</first>
              <last>2</last>
            </frame>
          </animation>
          <sound>sound1.ogg</sound>
        </state>
      </states>
    </Opponent>
  </Level>
</CNEURO>
```

ANEXO 6.

Figura 23. Representación de la etiqueta Allies.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cneuro>
<CNEURO version="1.0">
  <Level Id="1">
    <Allies>
      <Allied>
        <name>allied1</name>
        <states>
          <state>
            <name>state1</name>
            <animation>
              <element>element1.png</element>
              <frame>
                <first>1</first>
                <last>2</last>
              </frame>
            </animation>
            <sound>sound1.ogg</sound>
          </state>
        </states>
      </Allied>
    </Allies>
  </Level>
</CNEURO>
```

ANEXO 7.

Figura 24. Representación de la etiqueta Tasks.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cneuro>
<CNEURO version="1.0">
  <Level Id="1">
    <Tasks>
      <Task>
        <name>task1</name>
        <elements>
          <element>element1.png</element>
        </elements>
        <sounds>
          <sound>sound1.ogg</sound>
        </sounds>
      </Task>
    </Tasks>
  </Level>
</CNEURO>
```

ANEXO 8.

Tabla 10. Especificación de CU Management Level.

Caso de Uso	Management Level
Actor	Developer
Propósito	Configurar Nivel
Resumen	El presente caso de uso está relacionado con la configuración de los datos del nivel, aquí converge todo lo referente a la creación, modificación y eliminación de un nivel.
Referencias	R1, R1.1, R1.2, R1.3
Precondiciones	
Flujo Normal de Eventos	
Sección “Gestionar Nivel”.	

Acciones del Actos	Respuestas del sistema
1. El usuario desea Configurar un Nivel: • Crear Nivel. • Actualizar Nivel. • Eliminar Nivel.	2. El sistema ejecuta algunas de las acciones siguientes: a) Si el usuario desea crear un nivel ir a la sección Crear Nivel b) Si el usuario desea modificar información de un nivel ir a la sección Modificar Nivel c) Si el usuario desea eliminar un nivel ir sección Borrar Perfil
Sección “Crear Nivel”	
Acciones del Actos	Respuestas del sistema
3.El usuario desea Crear un Nivel 5.El usuario desea agregar componentes al nivel	4. El sistema le brinda la opción nuevo para crear un nuevo nivel. 6. El sistema le brinda la opción de adicionar componentes al nivel.
Sección “Modificar Nivel”	
7. El usuario desea Modificar un Nivel. 9.El usuario desea modificar los componentes del nivel	8. El sistema le brinda la opción de modificar el identificador del nivel 10. El sistema le brinda la opción de modificar cualquier valor para un componente determinado.
Sección “Eliminar Nivel”	
11. El usuario desea Eliminar un Nivel. 13. El usuario desea eliminar un componente	12. El sistema le brinda la opción de eliminar el nivel. 14. El sistema le brinda la opción de eliminar componentes
Poscondiciones	El nivel es creado El usuario puede variar la información del nivel El usuario puede eliminar el nivel

ANEXO 9.

Tabla 11. Especificación de CU Management File.

Caso de Uso	Management File
Actor	Developer

Propósito	Gestionar Fichero de Configuración
Resumen	El presente caso de uso está relacionado con el almacenamiento de los datos del nivel, aquí converge todo lo referente a salvar y cargar el fichero de configuración de un nivel.
Referencias	R3,R3.1,R3.2
Precondiciones	
Flujo Normal de Eventos	
Sección “Gestionar Fichero de Configuración”.	
Acciones del Actos	Respuestas del sistema
1. El usuario desea Gestionar el Fichero de Configuración: • Salvar a Fichero XML. • Cargar desde Fichero XML.	2. El sistema ejecuta algunas de las acciones siguientes: a) Si el usuario desea salvar los datos ir a la sección Salvar b) Si el usuario desea cargar los datos de un ir a la sección Cargar.
Sección “Salvar”	
Acciones del Actos	Respuestas del sistema
3.El usuario desea Salvar la Configuración de los Niveles	4. El sistema le brinda la opción de salvar a un fichero XML. 5. El sistema solicita una dirección y nombre para el fichero.
Sección “Cargar”	
6.El usuario desea Cargar la Configuración de los Niveles	7. El sistema le brinda la opción de cargar desde un fichero XML. 8. El sistema solicita una dirección y nombre del fichero. 9. El sistema verifica que el formato de fichero es válido.
Poscondiciones	Los datos fueron salvados a un fichero XML. Los datos son cargados desde un fichero XML.

ANEXO 10.

Figura 25. Diagrama de clases del análisis del CU Management Level.

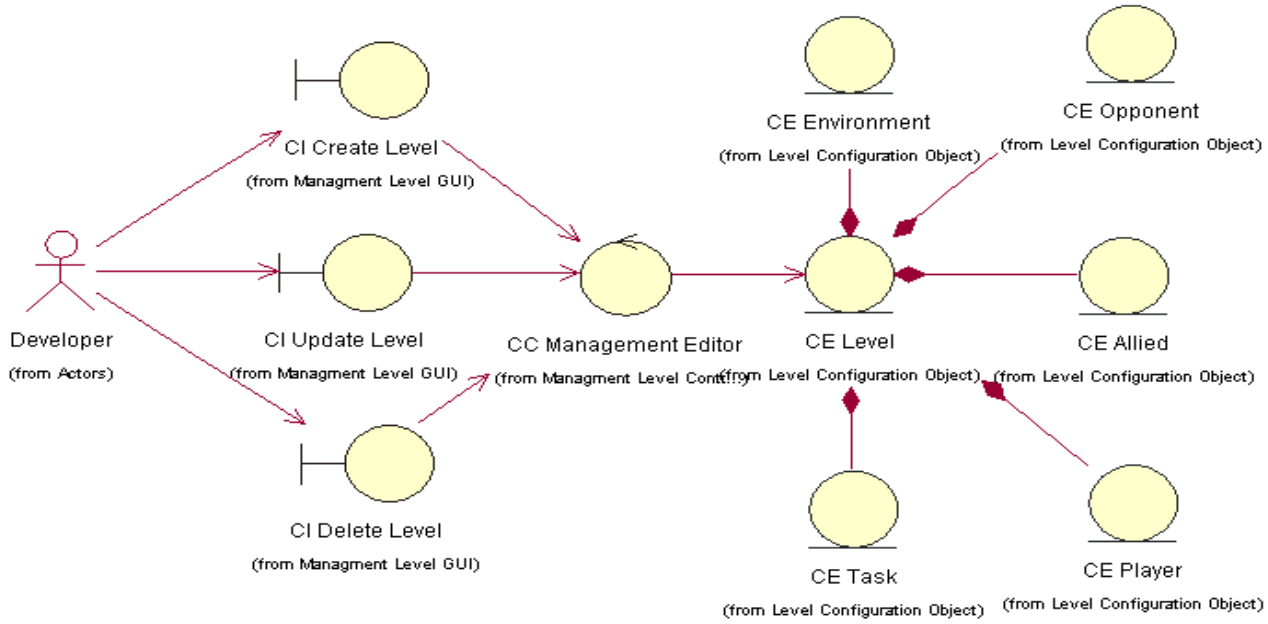


Figura 26. Diagrama de colaboración para la sección Crear Nivel del CU Management Level.

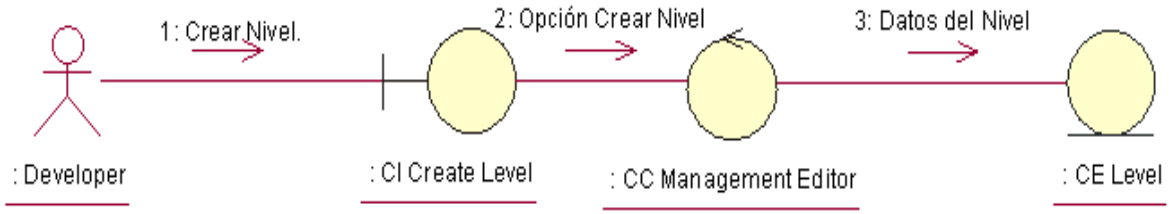


Figura 27. Diagrama de colaboración para la sección Modificar Nivel del CU Management Level.

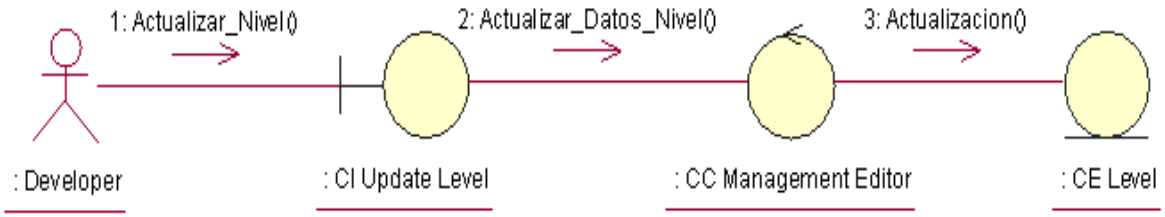
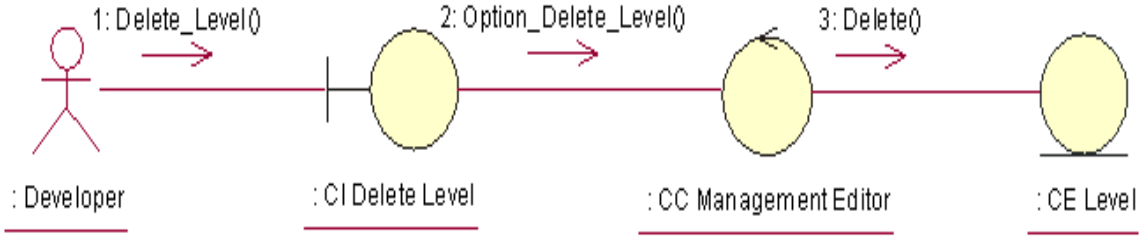


Figura 28. Diagrama de colaboración para la sección Eliminar Nivel del CU Management Level.



ANEXO 11.

Figura 29. Diagrama de clases del análisis del CU Management File.

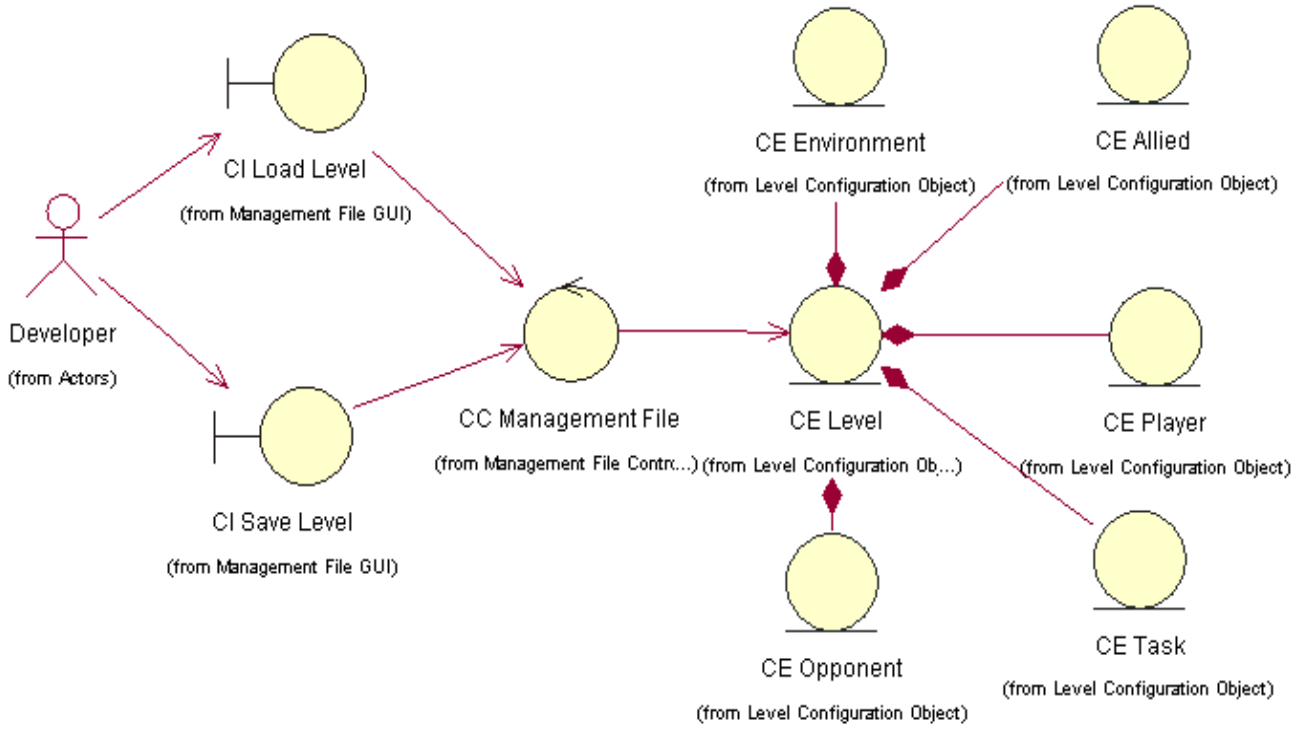


Figura 30. Diagrama de colaboración para la sección Salvar Nivel del CU Management File.

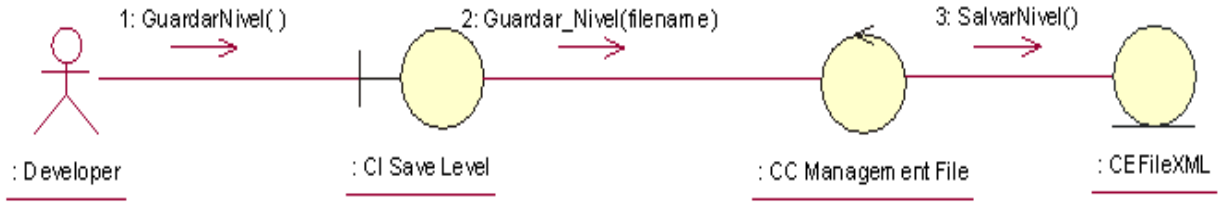
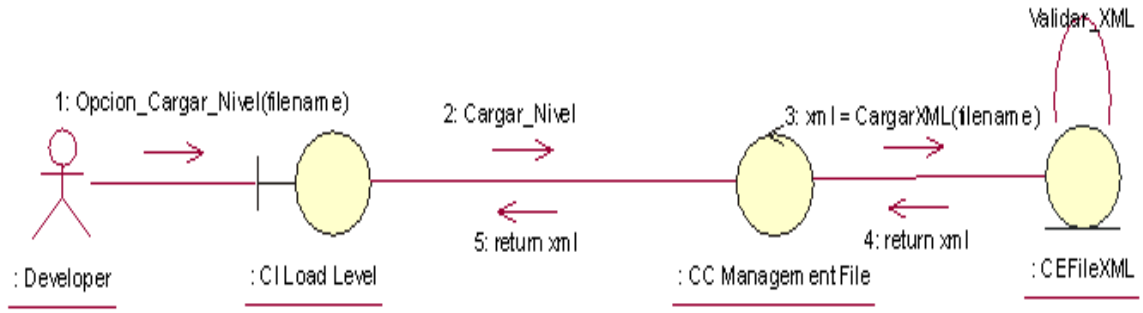
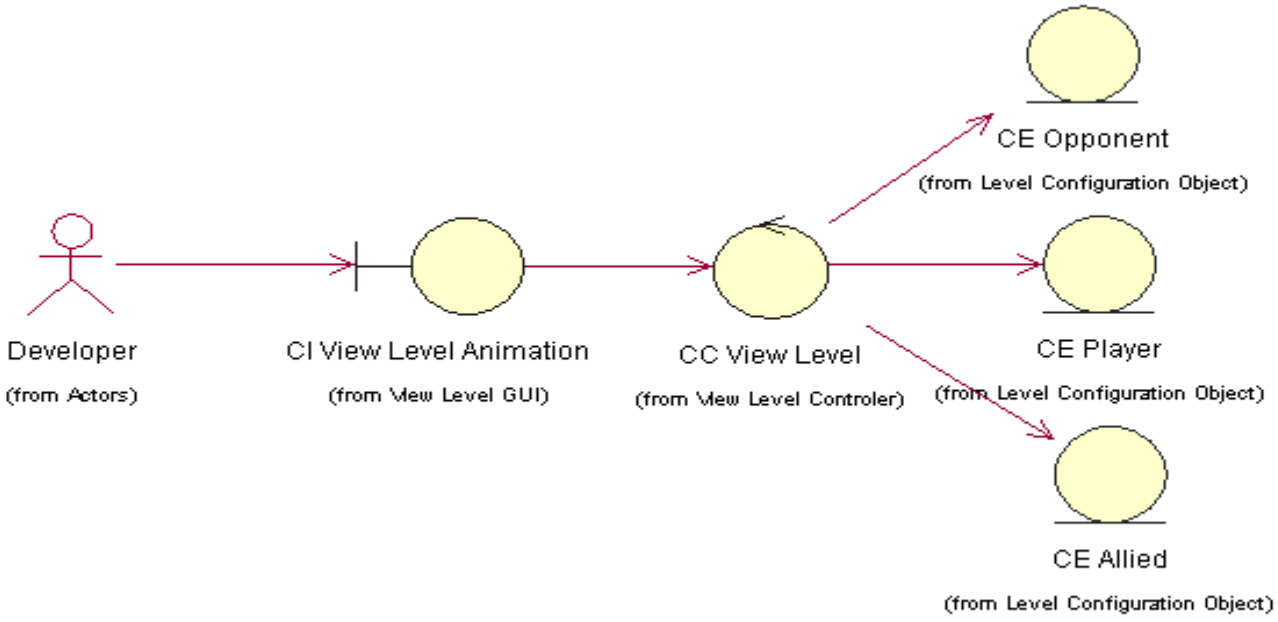


Figura 31. Diagrama de colaboración para la sección Cargar Nivel del CU Management File.



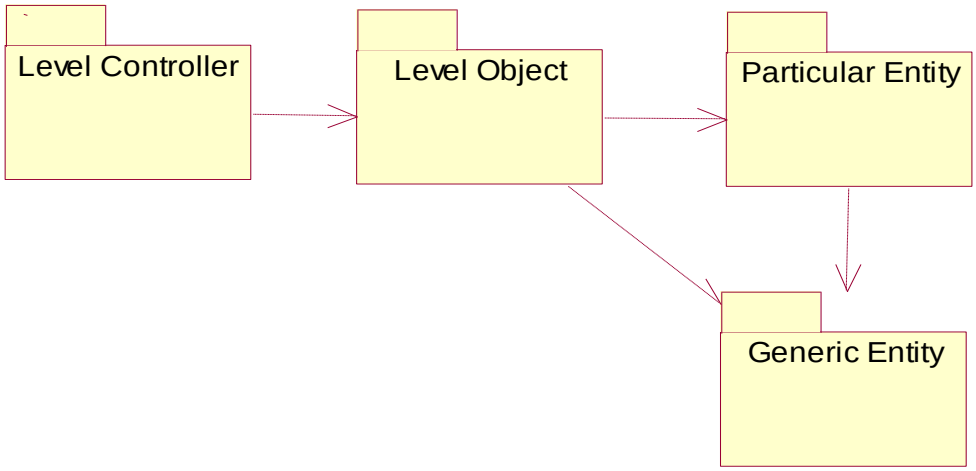
ANEXO 12.

Figura 32. Diagrama de clases del análisis para el CU Management View.



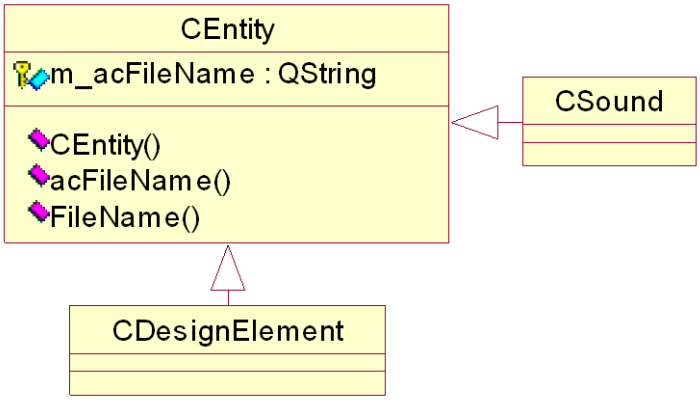
ANEXO 13.

Figura 33. Diagrama de paquetes del paquete de diseño Level Configuration Core.



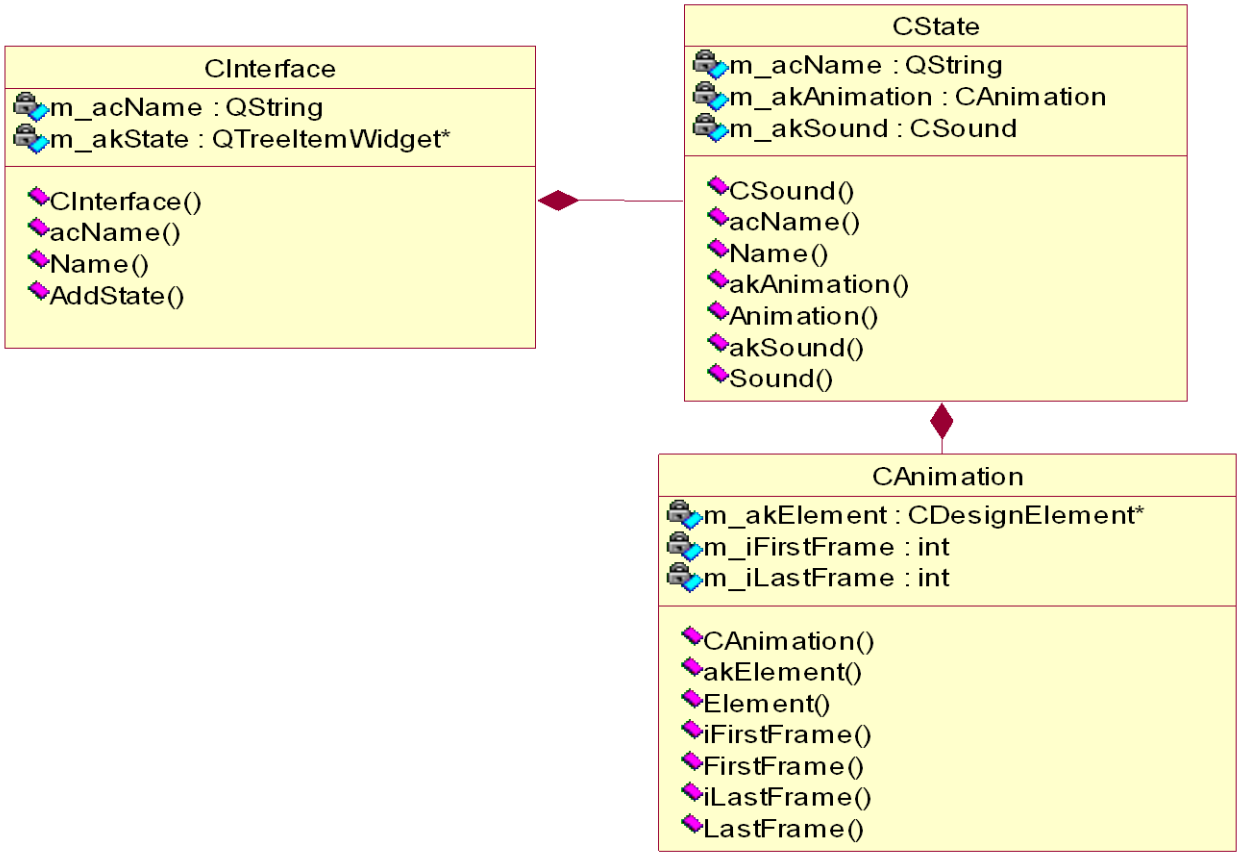
ANEXO 14.

Figura 34.Diagrama de clases del diseño del paquete Generic Entity.



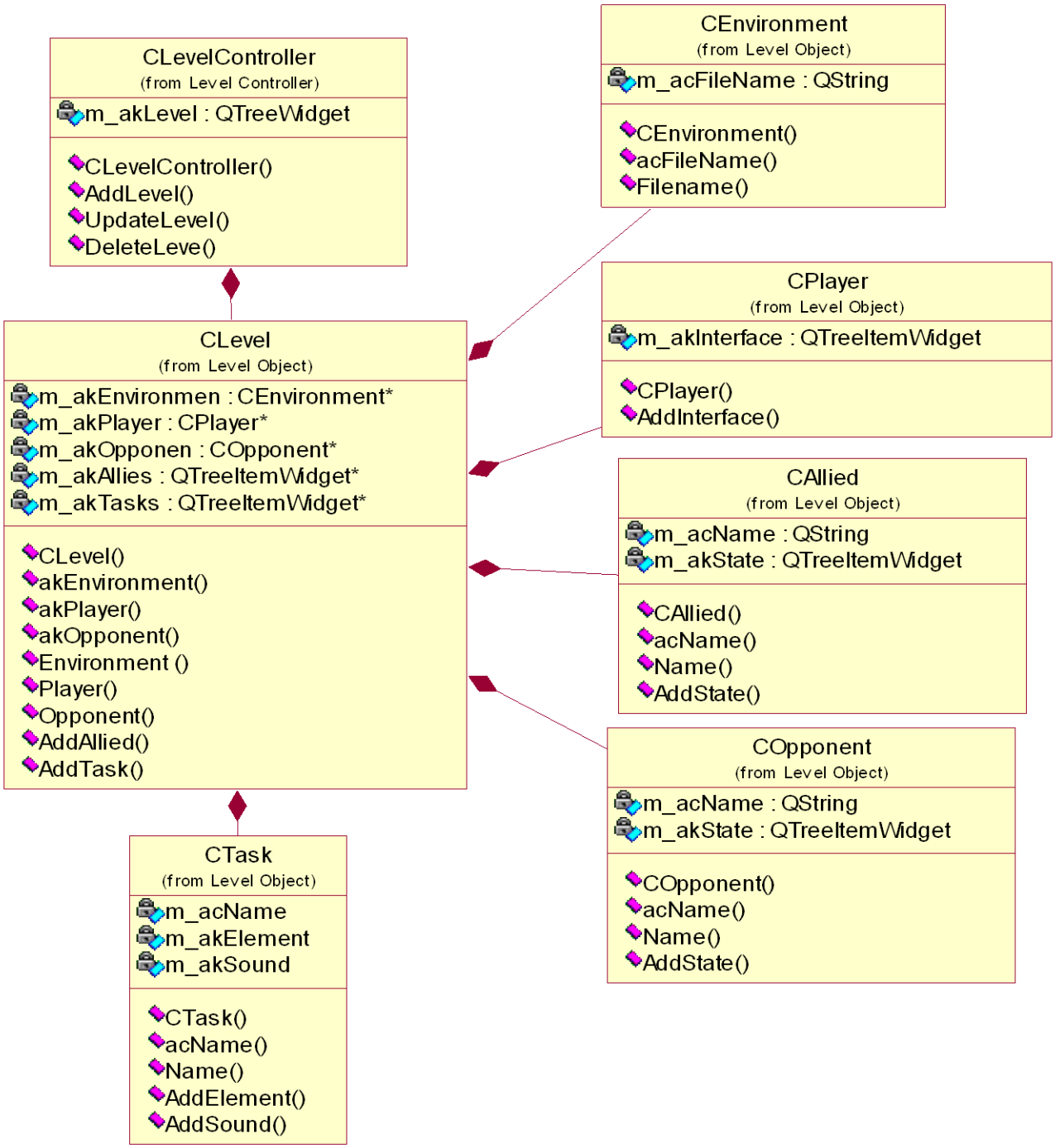
ANEXO 15.

Figura 35. Diagrama de clases del diseño del paquete Particular Entity.



ANEXO 16.

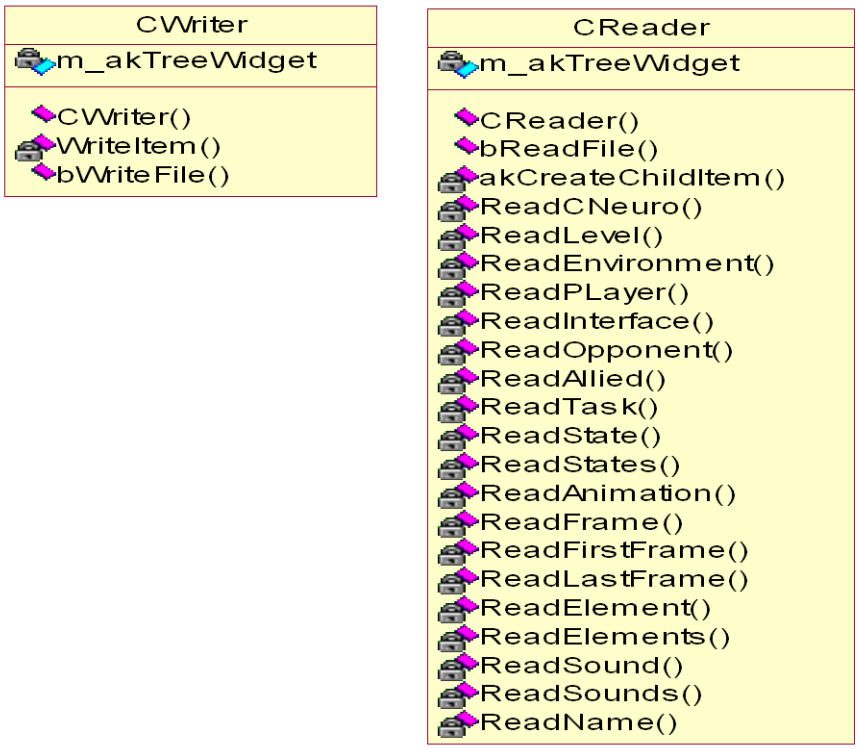
Figura 36.Diagrama de clases del diseño de los paquetes Level Controller y Level Object.



.

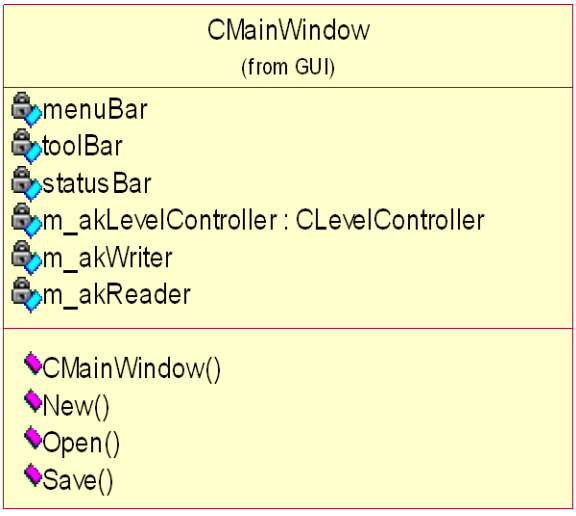
ANEXO 17.

Figura 37. Diagrama de clases de diseño del paquete Level Configuration File.



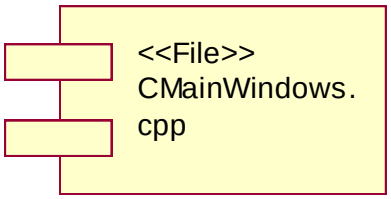
ANEXO 18.

Figura 38. Diagrama de clases de diseño para el paquete Level Configuration GUI.



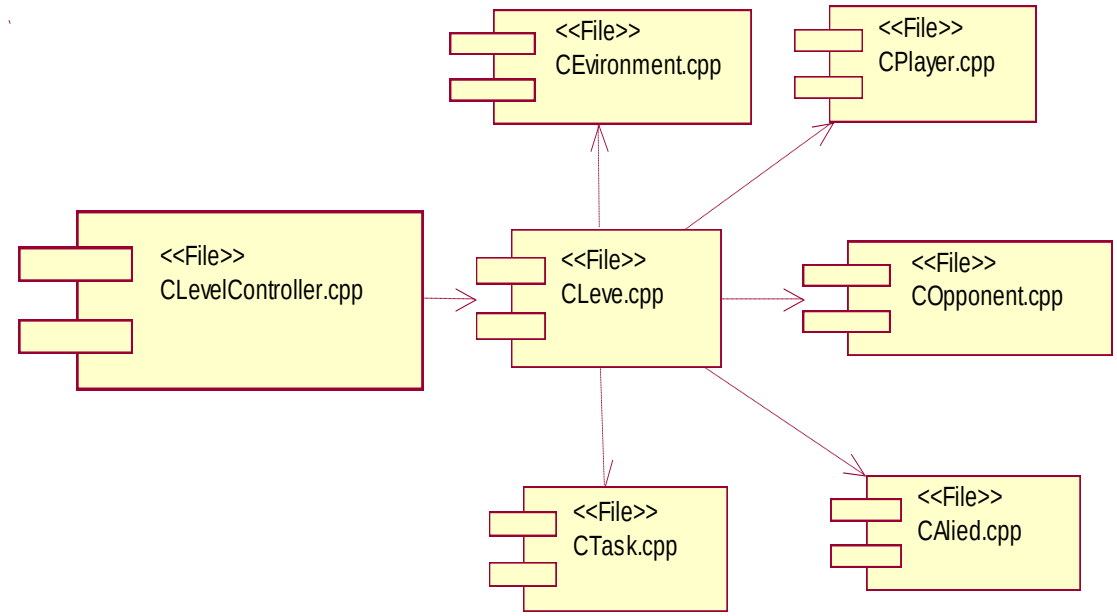
ANEXO 19.

Figura 39. Diagrama de componente del paquete Level Configuration GUI



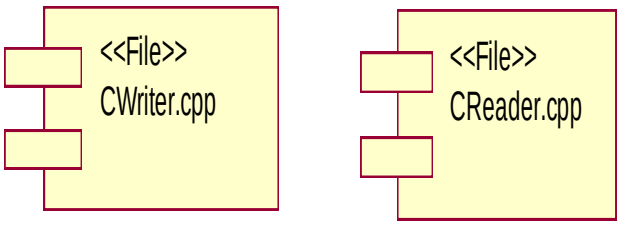
ANEXO 20.

Figura 40. Diagrama de componentes del paquete Level Configuration Core.



ANEXO 21.

Figura 41. Diagrama de componentes del paquete Level Configuration File.



GLOSARIO

GLOSARIO DE TÉRMINOS

Entidad: Datos que se utilizan en los motores gráfico y sonoro; estos datos pueden incluir música, efectos o modelos tridimensionales, texturas o animaciones.

Editor de niveles: Aplicación que permite trabajar con los objetos y los niveles de forma independiente del código del programa; conocido como editor de mapas, campañas y escenarios.

Motor de juego: Rutinas de programación que permiten el diseño, creación y representación de un videojuego; muy conocido por el término anglosajón “game engine”.

Motor lógico: Parte donde se controla el ciclo principal del juego. Incluye la descripción de los atributos de todos los elementos que participan, y de todas las reglas y condiciones que se sitúan en el juego.

Nivel: Conjunto de datos que recopila toda la información de todos los elementos que intervienen en esta sección del juego.

Objeto: Datos que describe todos los parámetros de los elementos que intervienen en el juego.

GLOSARIO DE ABREVIATURAS

API: Application Programming Interface; interfaz de programación de aplicaciones.

GIMP: GNU Image Manipulation Program; programa de edición de imágenes.

GPL: GNU General Public License; Licencia Pública General de GNU. Licencia creada por “Free Software Foundation”.

GUI: Graphical User Interface; interfaz gráfica de usuario, utiliza imágenes y objetos gráficos para representar la información y acciones que puede realizar el usuario.

IDE: Integrated Development Environment; entorno de desarrollo integrado, programa compuesto por un conjunto de de herramientas para un programador, provee un marco de trabajo.

INI: Fichero más común en la configuración de aplicaciones para Windows, el término proviene de "Windows **Ini**tialization File".

LGPL: GNU Lesser General Public License; Licencia Pública General para Bibliotecas de GNU. Licencia de software creada por "Free Software Foundation".

RUP: Rational Unified Process; Proceso Unificado de Rational.

SGML: Standard Generalized Markup Language; lenguaje de marcado generalizado, sistema para la organización y etiquetado de documentos.

UML: Unified Model Language (Lenguaje Unificado de Modelado). Es una notación estándar para modelar objetos del mundo real como primer paso en el desarrollo de programas orientados a objetos. Es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema de software.

XML: Extensible Markup Language; lenguaje de marcado extensible, una forma condensada de SGML.