<e-Adventure>

Proyecto de Sistemas Informáticos
Facultad de Informática

Universidad Complutense de Madrid

Francisco M. Pérez Padilla
Eduardo Sollet Galeán
Bruno Torijano Bueno

Directores:
Baltasar Fernández Manjón
Pablo Moreno Ger

Curso 2006 / 2007

Se autoriza a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Francisco M. Pérez Padilla Bruno Torijano Bueno Eduardo Sollet Galeán

Lista de palabras clave para búsqueda bibliográfica:

- **eLearning**
- **Educativo**
- **Videojuego**
- **LMS**
- **Adaptación**
- **Evaluación**

Índice general

Capítulo 1

Introducción

1.1. Acerca de este documento

El presente documento tiene como fin explicar y describir la filosofía del proyecto <e-Adventure>, así como su proceso de concepción, diseño e implementación. Este proyecto ha sido realizado como proyecto de Sistemas Informáticos del curso lectivo 2006-2007, bajo la dirección de Baltasar Fernández Manjón y Pablo Moreno Ger, co-director y miembro respectivamente de <e-UCM>, grupo de investigación en el campo del e-Learning en la Universidad Complutense de Madrid. Los integrantes del equipo de desarrollo han sido Francisco M. Pérez Padilla, Eduardo Sollet Galeán y Bruno Torijano Bueno.

El proyecto <e-Adventure> (llamado originalmente <e-Game>) surge como respuesta a la necesidad de incluir nuevas formas de contenido para sistemas de e-Learning. Su filosofía es la de facilitar la integración de videojuegos con contenidos educativos en LMS (Learning Management Systems), permitiendo a los educadores crear contenidos adecuados sin grandes conocimientos en programación o informática. También es importante destacar que el proyecto <e-Adventure> es de libre distribución, y que su código fuente se encuentra disponible en el servidor CVS de SourceForge (http://sourceforge.net/projects/e-game/).

El proyecto <e-Adventure> conforma una solución completa de creación y ejecución de contenidos educativos. Abarca tanto la especificación del formato de los contenidos como la creación de un motor de aventuras gráficas capaz de cargar y representar dichos contenidos. Este motor tiene capacidad de conexión con un servidor LMS para proporcionar apoyo al aprendizaje, llevando a cabo su ejecución procesando aventuras notadas en lenguaje XML.

Además del motor, se ha desarrollado un editor de aventuras que permite a los educadores crear contenidos sin necesidad de tener conocimientos de XML. De esta forma se provee de todo el conjunto de herramientas y especificaciones necesarias para hacer funcionar un sistema completo de aprendizaje basado en el uso de videojuegos.

Como características más relevantes, el proyecto <e-Adventure> cuenta con la capacidad de adaptación y evaluación de los contenidos ejecutados. La adaptación de los contenidos permite que el motor pueda cambiar la experiencia de juego en tiempo real dependiendo de las acciones que lleve a cabo el usuario. Por otro lado, la evaluación permite al motor llevar un seguimiento de los progresos que lleva a cabo el usuario dentro de la experiencia de juego. Con estas capacidades el sistema <e-Adventure> se coloca como un producto realmente innovador dentro del campo del e-Learning, integrando robustamente el uso de videojuegos en dichas tecnologías de aprendizaje.

<e-Adventure> soporta dos modos distintos de ejecución: Por una parte, se puede ejecutar como aplicación independiente, habiendo descargado tanto el motor como una o más aventuras al disco duro. Por otra parte, se puede ejecutar directamente en un navegador web, aprovechando de este modo todos los servicios de aprendizaje que pueda proporcionarnos el servidor e-Learning.

El proyecto en sí ha partido de una especificación provista por el equipo de investigación <e-UCM>, que ha sido ampliada y revisada en numerosas ocasiones a lo largo del proceso de desarrollo. Para el seguimiento del trabajo realizado se han llevado a cabo reuniones frecuentes entre el equipo de desarrollo y el de dirección, que se han usado principalmente para discutir las características a incluir dentro del sistema <e-Adventure>, así como para marcar los distintos plazos e hitos del desarrollo.

En definitiva, se presenta el proyecto <e-Adventure> como un producto innovador dentro del campo del e-Learning, ofreciendo soluciones y ventajas en el campo del aprendizaje tanto para educadores como para estudiantes.

1.2. About this document

The aim of this document is to explain and describe the <e-Adventure> project's philosophy, as well as its conception process, design and implementation. This project has been implemented as a final course project on the year 2006-2007, under the direction of Baltasar Fernández Manjón and Pablo Moreno Ger, co-director and member respectively of <e-UCM>, research group specialized in e-Learning fields in Universidad Complutense of Madrid. The development team members of this project are M. Pérez Padilla, Eduardo Sollet Galeán and Bruno Torijano Bueno.

The <e-Adventure> project (formerly called <e-Game>) comes up as a reply to the necessity of including new contents forms for e-Learning systems. Its philosophy is to integrate computer games containing educative contents with a Learning Management System, allowing teachers to create suitable contents without great knowledge about programming or computing. It's important to emphasize that the <e-Adventure> project is distributed as free software, and its source code can be downloaded from SourceForge's CVS server (http://sourceforge.net/projects/e-game/).

The <e-Adventure> project is a complete solution to create and deliver learning contents. It includes a specification of the learning contents' format, as well the development of a point-and-click adventure games engine, capable of loading and depicting this contents. This engine has the ability to connect with a LMS server for learning-related operations, reading the scripts of the adventures from XML files.

In addition to the engine, an adventure editor has been developed, allowing the teachers to create contents without knowledge in XML technologies. This way, the <e-Adventure>project provides a wide set of tools and specifications used to deploy a complete learning system based on videogames.

As one of the most relevant features, the <e-Adventure> project can adapt and evaluate the executed contents. The adaptation of this contents allows the engine to change the game experience in real time, depending on the actions that the player chooses. On the other hand, the evaluation of the contents allows the engine to keep track of the progress of the player. With this features, the <e-Adventure> project stands as an innovative system in the field of e-Learning, integrating the use of videogames in this learning technologies.

<e-Adventure> allows to separate types of execution. It can be executed as a standalone software, if the user has downloaded the engine along with some adventures. On the other hand, it can be executed on a web browser, making easier the integration with other e-Learning systems and solutions.

The proyect started with a specification provided by the <e-UCM> research group. This specification was extended and checked several times during the development of the proyect. To keep track on the work done, frequent meetings were carried out, to define the features that the proyect had to possess and to establish the different periods and milestones of the development process.

To summarize, the <e-Adventure> proyect is an innovative product in the field of e-Learning, offering solutions and advantages in learning fields for teachers and for students as well.

Capítulo 2

Motivación y descripción del proyecto <e-Adventure>

2.1. e-Learning y videojuegos

Dentro del campo del e-Learning una de las motivaciones más importantes es la facilidad de creación y distribución de contenidos, y la necesidad de que dichos contenidos sean adecuados para las necesidades de sus usuarios. Los contenidos deben adaptarse a los conocimientos de los estudiantes, y los sistemas de e-Learning deben ser capaces de hacer un seguimiento de los avances realizados, para poder realizar calificaciones apropiadas.

En la actualidad, el e-Learning basa sus cimientos mayoritariamente en contenidos de texto o audiovisuales, de baja capacidad interactiva. En términos generales se puede afirmar que estos contenidos, además, son los mismos que en procesos de aprendizaje tradicionales, con la diferencia de que se distribuyen por canales digitales. Se sustituye el papel por la pantalla de ordenador, pero en el fondo los contenidos siguen siendo fundamentalmente los mismos.

Hay que destacar que en cualquier caso las tecnologías de e-Learning van más alla de un cambio en el medio de distribución del material de aprendizaje. Facilita a los educadores la tarea de suministrar dichos contenidos, y apoyan tareas de corrección y seguimiento de los avances de los estudiantes. Estos sistemas otorgan también la capacidad de que los estudiantes se relacionen entre ellos y con los profesores de manera más libre e inmediata, gracias al uso de redes (generalmente internet).

Aún y con todo, es patente que la tarea de aprendizaje no varía significativamente para el usuario, con respecto a modelos más tradicionales. Es en este punto cuando los videojuegos se presentan como una posibilidad real dentro del campo de la educación asistida por tecnolgías de la información.

Si hay una característica auténticamente relevante en los videojuegos, es su interactividad. Disponen de la capacidad de reaccionar a las acciones del jugador, presentando situaciones y posibilidades distintas en cada partida. Un mismo jugador puede desarrollar un centenar de partidas independientes, y encontrarse con multitud de situaciones totalmente distintas (partiendo de la premisa de que la dinámica del juego es común a todas ellas). Es

justo esta interactividad la que otorga a los videojuegos un potencial inexplorado dentro del terreno del e-Learning.

Desgraciadamente, hoy en día hay muy pocos videojuegos integrados en procesos de aprendizaje. En la mayoría de las ocasiones los videojuegos se presentan como aplicaciones totalmente cerradas, sin más flujos de entrada y salida que los necesarios para desarrollar las partidas. Esto hace imposible que los sistemas de e-Learning puedan evaluar las decisiones y avances hechos por el jugador en el videojuego, anulando así toda posibilidad de retroalimentación en el proceso de aprendizaje.

Llegados a este punto, se antoja necesaria la creación de videojuegos preparados para proporcionar retroalimentación de los avances del estudiante o usuario. Dichos juegos han de contar con la capacidad de generar informes y evaluar a los jugadores, en base a parámetros designados por los educadores. Es decir, es necesario que los educadores puedan determinar que aspectos de las partidas desarrolladas por los jugadores son relevantes para su formación, para que el videojuego en sí envíe dichos datos al sistema e-Learning, permitiéndole llevar a cabo un proceso más exhaustivo de evaluación.

2.2. Motivación del proyecto <e-Adventure>

El proyecto <e-Adventure> nace como solución a los impedimentos descritos anteriormente, impulsado por el grupo <e-UCM>, grupo de investigación de la Universidad Complutense de Madrid en tecnologías de e-Learning. Dicho grupo de investigación muestra un gran interés tanto por las tecnologías web orientadas al aprendizaje como por los lenguajes de marcado, hechos que han influido de manera significativa en la especificación del proyecto <e-Adventure>. Cabe destacar además que el proyecto <e-Adventure> se desarrolla como un sistema de libre distribución.

Para ampliar información tanto del grupo de investigación <e-UCM> como del proyecto <e-Adventure>, se ofrecen las direcciones de sus páginas web:

- **Página del grupo <e-UCM>** http://www.e-ucm.com/
- **Página del proyecto <e-Adventure>** http://e-adventure.e-ucm.com/

<e-Adventure> está por lo tanto diseñado para proporcionar una solución adecuada al desarrollo y a la integración de videojuegos en aprendizaje basado en tecnologías de la información. Es importante destacar sin embargo que no todos los géneros de videojuegos tienen el mismo potencial educativo: por ejemplo los juegos de conducción parecen ideales para videojuegos relacionados con educación vial, pero cuesta imaginar un videojuego de este tipo aplicado a otro aspecto, como podría la formación en cuestiones de protocolo u hostelería.

De esta forma surge la duda de qué tipos de juegos son más adecuados para dar apoyo a labores de aprendizaje, a la vez que puedan ser abordables desde el punto de vista de la automatización de su desarrollo. Parece sensato escoger videojuegos que hacen uso de conceptos reales (al contrario que los puzzles, por ejemplo, que suelen disponer de conceptos muy abstractos en su planteamiento), con gran interactividad y con una buena capacidad

de expresar ideas y situaciones. Tratando de cumplir estos tres objetivos, uno de los géneros que encajan satisfactoriamente son las aventuras gráficas.

En términos generales este tipo de juegos representa una serie de localizaciones, en los que se encuentran distintos objetos y personajes. Entre los personajes, se encuentra uno que es controlado por el jugador, que hace el papel de protagonista a lo largo de la aventura. El avance en las partidas se realiza haciendo que el personaje protagonista interaccione con los distintos elementos del juego, conversando con personajes, utilizando y combinando objetos, examinando situaciones, recorriendo localizaciones... En resumen, las aventuras gráficas proporcionan un tipo de juego lo suficientemente flexible y variado como para colocarse como una de las opciones más adecuadas en términos de educación.

Sin embargo, aún habiendo elegido un género adecuado para utilizar en el campo educativo, se plantea otro problema. En la inmensa mayoría de las ocasiones, los videojuegos se distribuyen con un contenido prefijado, contando en el mejor de los casos con algunas posibilidades de modificación en algunos campos. Esto supone un problema pues se pretende que estos videojuegos puedan aplicarse a muchos campos, y desde un punto de vista tradicional esto conllevaría el realizar videojuegos para cada una de las áreas de conocimiento. Para solucionar esto, se puede abordar el problema desde la perspectiva del *motor de videojuegos* en lugar de desde la perspectiva de *videojuego*. De esta manera se concibe el videojuego como una aplicación capaz de ejecutar distintos contenidos, creados de manera externa.

En términos generales, se describe entonces al proyecto <e-Adventure> como un *motor de aventuras gráficas con fines educacionales*, capaz de ejecutar distintos contenidos, siempre y cuando estos respeten el formato soportado por el motor. Con esto, <e-Adventure> pasa a constituir una solución de aplicación real en el uso de videojuegos en campos educativos. Sin embargo, este proyecto no se compone exclusivamente del motor encargado de la ejecución de los contenidos, si no que abarca la definición del formato de dichos contenidos e incluye en su especificación una herramienta de autoría para crear aventuras gráficas de forma sencilla.

Uno de los puntos en los que conviene hacer incapié es en la funcionalidad de motor. Los guiones se crean de manera externa a la aplicación, de manera que se puede concebir una aventura sin necesidad de modificar una sola línea de código de la aplicación principal. De esta forma los educadores no tienen necesidad de tener altos conocimientos en informática para crear y distribuir sus propios contenidos educativos.

Por último, recordamos que una de las necesidades de los videojuegos para su uso en aprendizaje es su capacidad de producir realimentación. La especificación de <e-Adventure> contiene, por lo tanto, un sistema de conexión con servidores LMS (Learning Management System) de intercambio de información en ambos sentidos. De esta forma, el videojuego puede informar al servidor LMS de los avances que el jugador realiza, indicando de forma concreta que objetivos ha cumplido el estudiante; por su parte, el servidor LMS tiene la capacidad de enviar información al videojuego para efectuar cambios en tiempo real sobre su contenido, de forma que la experiencia interactiva varíe en distintas ejecuciones y circunstancias.

2.3. Características del sistema

Una vez descrito el proyecto <e-Adventure> en términos generales, es necesario definir de una manera un poco más exhaustiva las capacidades que debe soportar. Este apartado presenta dichas capacidades, ordenadas en distintos bloques. Es necesario destacar, sin embargo, que <e-Adventure> no es sólo un motor de aventuras gráficas educacionales, si no que proporciona una solución completa de aprendizaje, incluyendo también un formato de descripción de contenidos y un editor visual para sus aventuras.

2.3.1. Funcionamiento del motor

En este apartado se describen algunas características generales del funcionamiento del motor, como sus capacidades de conexión o los tipos de archivo que debe ser capaz de manejar.

- **Ejecución de aventuras externas**. Como ya se ha comentado, el motor ha de ser capaz de cargar aventuras contenidas en archivos externos a la aplicación. De esta forma, las aventuras ejecutadas deben de poder desarrollarse sin necesidad de modificar una sola línea de código del motor.
- **Procesamiento de archivos XML**. Los guiones de las aventuras, así como sus informaciones anexas, han de estar contenidas en archivos con formato XML. Dicho formato tiene la capacidad de ser modificado de manera relativamente sencilla, de manera que los guiones puedan modificarse sin necesidad de herramientas específicas, si así se desea.
- **Soporte de archivos de imagen, audio y vídeo**. El motor ha de soportar distintos formatos de imagen, audio y vídeo para mostrar por pantalla la representación del estado del juego. Los formatos que se han barajado inicialmente para su ejecución son: PNG y JPG para imagenes, MP3 y WAV para audio, y MPG para vídeo.
- **Soporte de contenedores ZIP**. Para facilitar la distribución de los contenidos creados para <e-Adventure>, se ha decidido utilizar archivos con formato ZIP para almacenar los guiones, así como sus archivos de recursos multimedia (imágenes, audio y vídeo). Es decir, el motor debe de ser capaz de ejecutar aventuras empaquetadas en archivos ZIP, encargándose él de acceder a los datos directamente.
- **Conexión con LMS y control de evaluación**. Dentro del apartado educativo, el motor ha de ser capaz de comunicarse con un servidor de LMS (Learning Management System) para intercambiar información, notificando así de los avances del usuario y recibiendo datos que puedan alterar el curso de la partida. El motor ha de implementar además un sistema capaz de evaluar los avances del usuario, de acuerdo con información anexa que puede proporcionarse junto con la aventura.

2.3.2. Capacidades como videojuego

Dado que el motor <e-Adventure> puede definirse como un videojuego, concretamente una aventura gráfica, pasamos a describir aquellas situaciones y capacidades que debe presentar, en su componente interactiva.

- **Escenas interactivas**. Esta es una de las características esenciales de <e-Adventure> como videojuego. En este tipo de escenas, el jugador puede, a través del personaje protagonista, interaccionar con su entorno. De esta forma tiene la posibilidad de recoger o utilizar objetos, así como dialogar con personajes o salir de dichas escenas interactivas para pasar a otras escenas distintas.

- **Escenas no interactivas**. Para potenciar las capacidades narrativas del motor, este ha de ser capaz de mostrar información por pantalla, de una forma no interactiva. Estas escenas se han catalogado en dos tipos inicialmente: escenas de diapositivas, que consisten en una serie de imagenes estáticas que se van sucediendo, y archivos de vídeo, que se muestran para representar información de una forma audiovisual. En algunos puntos de la documentación nos referiremos a este tipo de escenas como *cutscenes*.

- **Soporte para libros**. Dada su vertiente educativa, es probable que en las aventuras creadas para ser ejecutadas por <e-Adventure> haya que mostrar grandes cantidades de texto en determinadas ocasiones. Para paliar este problema, se ha concebido un modo que permite mostrar por pantalla grandes cantidades de texto escrito, representadas en forma de libro. De esta manera el jugador tiene la posibilidad de acceder a grandes cantidades de texto, navegando por las páginas del libro como si el protagonista del juego lo estuviera leyendo. Destacamos que esta característica no estaba incluida en la especificación inicial del proyecto <e-Adventure>, si no que fue propuesta y añadida durante el desarrollo.

- **Comportamientos guionizados**. Como último recurso para potenciar la narratividad del motor, este debe contar con la posibilidad de hacer que los personajes se desplacen por la pantalla y desarrollen conversaciones entre ellos, como si de una obra teatral se tratara. De esta forma se pueden representar eventos que tienen lugar en las partidas, sin necesidad de mostrar diapositivas o vídeos. Esta característica fue objeto de ampliaciones y revisiones a lo largo del desarrollo del proyecto, debido a que su especificación se antojaba incompleta para algunas situaciones.

2.3.3. Interfaces de usuario

Uno de los apartados que más importancia cobran dentro del proyecto <e-Adventure>, más tratándose de un videojuego, es la definición de los interfaces de usuario. A continuación se enumeran los interfaces con los que debe de contar la aplicación, junto con sus particularidades. En este punto es necesario definir el concepto de HUD (Head-Up Display), que es el conjunto de imágenes mostradas en pantalla que proporcionan información y opciones al usuario. Es decir, forman parte del HUD los botones de acciones, el inventario, la información del puntero del ratón, etc...

- **Interfaz de selección de juego**. El motor ha de contar con un pequeño menú de selección de aventura, en caso de haber más de una disponible para su ejecución.

- **HUD de juego *tradicional***. Este interfaz de juego debe contar con un espacio reservado para albergar los botones de acción y el inventario, mostrándose la representación de la partida en el resto de la pantalla. Dicho interfaz debe funcionar por medio de

selección de acciones del menú. Recibe el nombre de *tradicional* debido a su similaritud con las primeras aventuras gráficas aparecidas de mano de LucasArts™. En las figuras 2.1 y 2.2 se pueden ver algunos ejemplos de juegos comerciales que usan dicha interfaz.

Cuadro 2.1: HUD tradicional. Monkey Island 2®- LucasArts™

- **HUD de juego *contextual***. Este interfaz de juego muestra la representación de la partida en la totalidad de la pantalla. El inventario de juego debe aparecer y esconderse con un atajo de ratón o teclado, y los menus de acciones deben mostrarse con otra entrada especifica, al llevarse a cabo sobre un objeto interactivo. Recibe el nombre de *contextual* debido a que las acciones se pueden realizar únicamente sobre un contexto determinado (un objeto o un personaje, en este caso). En las figuras 2.3 y 2.4 se muestran ejemplos de juegos comerciales que implementan este tipo de interfaz.
- **Menú de opciones**. Este interfaz debe proporcionar las opciones necesarias para configurar el motor al gusto del usuario, en la medida de lo posible. También debe incluir opciones para salir del juego, salvar o cargar la partida, etc...
- **Interfaz de generación de informes**. Este interfaz complementario debe permitir al usuario seleccionar opciones para generar informes de los avances realizados durante las partidas. Este interfaz concreto está muy ligado a la parte de retroalimentación en el proceso educativo.

2.3.4. Editor de aventuras

Como parte del proyecto <e-Adventure> se ha planteado además desarrollar un editor gráfico de aventuras. De esta forma se permite que los educadores puedan crear contenidos sin

Cuadro 2.2: HUD tradicional. Day of the Tentacle®- LucasArts™

Cuadro 2.3: HUD contextual. Monkey Island 3®- LucasArts™

Cuadro 2.4: HUD contextual. The Dig®- LucasArts™

necesidad de tener conocimientos en XML; aunque sí es necesario que tengan conocimientos sobre la forma en que se agrupan los datos dentro de los guiones de las aventuras creadas.

A continuación se describen algunas de las características generales con las que debe contar el editor de aventuras. Señalamos, sin embargo, que a lo largo del documento se facilitará información casi exclusivamente sobre el motor de ejecución de las aventuras y el lenguaje utilizado para su marcado. Para más detalles sobre el editor de aventuras se puede consultar el capítulo 6.

- **Soporte total de las capaciades de <e-Adventure>**. En primer lugar, es imprescindible que el editor de aventuras soporte todos los elementos descritos en la especificación del formato de las aventuras. De esta forma los creadores de contenidos pueden disponer de una herramienta potente y completa para llevar a cabo las aventuras que se deseen.

- **Abstracción completa del XML**. Una de las características más importantes que debe tener el editor de aventuras es la facultad de abstraer al educador del formato XML de una forma completa. Es decir, es vital que una persona que no tenga conocimientos de XML pueda crear y editar aventuras utilizando el editor. De este modo, el editor debe presentar formas alternativas de representación y modificación de los datos, de una forma totalmente visual.

- **Facilidad de uso**. Es importante que el interfáz de usuario del editor de aventuras sea lo suficientemente potente e intuitivo como para poder crear contenidos de una manera relativamente sencilla. De esta forma se prevee la inclusión de distintos campos de

previsualizaciones (para escenas o libros), así como posiblidad de colocación de objetos y personajes de manera visual (indicando en qué puntos del escenario han de colocarse).

- **Control y verificación de aventuras**. Por último, y para facilitar las tareas de creación a los educadores, el editor de aventuras debe contar con mecanismos que aseguren que las aventuras generadas sean válidas. Es decir, el editor ha de encargarse de comprobar que todos los elementos estén correctamente construidos, además de asegurarse de que no haya referencias a elementos no existentes, por ejemplo.

2.4. Aspectos destacables del proyecto <e-Adventure>

Si bien hasta ahora se ha presentado el proyecto <e-Adventure> como poco más que un motor de aventuras gráficas con algunas mejoras, es conveniente destacar que algunas de sus características lo posicionan como un proyecto realmente ambicioso, y de una gran calidad innovadora. A continuación se describen aquellas características especialmente destacables dentro de la concepción del proyecto <e-Adventure>.

2.4.1. Solución completa de aprendizaje

En primer lugar, es importante destacar que <e-Adventure> supone una solución completa de aprendizaje con tecnologías de la información. De esta forma, el proyecto <e-Adventure> no sólo proporciona un motor de ejecución de contenidos educativos, si no que define su propio formato de contenidos, facilita herramientas con los que crear dichos contenidos, y proporciona componentes que permiten una integración más que satisfactoria con entornos de aprendizaje como servidores LMS.

A continuación se detallan aquellos elementos que conforman el proyecto <e-Adventure> en su totalidad:

- **Generación de contenidos**. El proyecto <e-Adventure> cuenta con su propio formato de contenidos educativos, ejecutables en forma de aventuras gráficas. Además de dicho formato de aventuras, se provee de un editor de aventuras que facilita la tarea de autoría para los educadores.
- **Distribución de contenidos**. En cuanto a la distribución del motor en sí, cabe destacar que <e-Adventure> puede ser desplegado desde una página web, o ejecutado directamente utilizando un único archivo JAR. Por otro lado, las aventuras se empaquetan individualmente, facilitando así la distribución de dichos contenidos.
- **Ejecución de aventuras**. Como una de las características principales, el motor <e-Adventure> permite ejecutar contenidos educacionales en forma de aventuras gráficas. El motor <e-Adventure> cuenta con todas las funcionalidades necesarias para reproducir dichas aventuras de forma correcta, de acuerdo a la especificación de los contenidos.
- **Adaptación de la experiencia**. Uno de los puntos más importantes de <e-Adventure> es la capacidad de adaptar los contenidos de las aventuras ejecutadas en tiempo real. El motor de adaptación se describe en profundidad más adelante.

- **Evaluación del aprendizaje**. El otro punto relevante de <e-Adventure> en cuanto a capacidades educativas es la posibilidad de evaluar los progresos del jugador en tiempo real, con la posibilidad de generar informes con dichas evaluaciones, o de notificar dichos progresos al servidor LMS. El motor de evaluación se describe en profundidad más adelante.

2.4.2. Guiones notados en XML

Una de las características más importantes del proyecto <e-Adventure> es la de contar con archivos XML para describir las aventuras. En principio puede parecer que esto no supone una ventaja significativa con respecto a otros motores, pero el formato XML presenta numerosas ventajas con respecto a otro tipos de archivos de contenidos (archivos con contenido binario, en la mayoría de las ocasiones).

En primer lugar, el XML facilita la legibilidad inmediata de la aventura por parte de una persona. De esta forma cualquier persona puede echar un vistazo a una aventura notada para <e-Adventure> y ser capaz de reconocer estructuras y elementos de la aventura, si bien hacen falta tener unas nociones en XML.

Gracias a este formato, además, se facilita que los guiones compartan un formato común, fácilmente ampliable y extensible. Por otro lado, hay que tener en cuenta la gran cantidad de aplicaciones (tanto libres como de pago) que permiten editar archivos XML, permitiendo así que los guiones para <e-Adventure> puedan ser modificados por multitud de editores distintos.

Por último, es importante destacar que si bien el formato XML es relativamente sencillo, dispone de la suficiente potencia y flexibilidad como para ser capaz de contener grandes cantidades de información de complejidad elevada; siendo así adecuado para la notación de las aventuras para el proyecto.

2.4.3. Motor de adaptación

El motor de adaptación tiene como fin permitir la inclusión de variaciones en los contenidos reproducidos por el motor <e-Adventure>. De esta forma se puede proveer a distintos jugadores de experiencias de juego distintas sin necesidad de alterar el contenido que se les suministra.

La idea de adaptar el contenido para cada usuario tiene una fuerte relación con el concepto de evaluación de progresos. El motor de adaptación se incluye para permitir que distintos usuarios, con distintos conocimientos, tengan acceso a distintas partes de los contenidos, para que puedan aprovechar al máximo sus capacidades de aprendizaje.

Esta adaptación se puede llevar a cabo de dos maneras, la primera es adaptando los contenidos del juego nada más comenzar las partidas, y la segunda recibiendo información de los servidores de LMS, adaptando el contenido de las aventuras a la vez que se desarrollan.

La adaptación inicial de los contenidos, por ejemplo, permite configurar los puntos de introducción del personaje y su avance en la historia. De esta forma se pueden distribuir

contenidos que están preparados para arrancar desde un punto concreto de la aventura. Por ejemplo, se puede proporcionar a un grupo con conocimientos avanzados una aventura preparada para comenzar en un punto de dificultad intermedia, ahorrando así a los estudiantes el tener que pasar por situaciones o problemas que están de sobra preparados para afrontar.

Un ejemplo de fichero de adaptación para avanzar de nivel a un alumno sería tan sencillo como el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE adaptation SYSTEM "adaptation.dtd">

<adaptation>

  <initial-state>
    <activate flag="go-level-2">
  </initial-state>

</adaptation>
```

Por su parte, la aventura debería utilizar el flag *go-level-2* de algún modo, para que, en caso de estar activado, efectivamente el grado de dificultad cambie, o simplemente el personaje entre en un punto más avanzado de la aventura, pasando una hipotética fase de comprobación de conocimientos muy básicos.

La adaptación en tiempo real de los contenidos tiene un mayor abanico de posibilidades. Con esta característica, el servidor encargado del proceso de aprendizaje puede enviar mensajes al motor <e-Adventure> para que varíe distintas condiciones de la partida. De esta forma, por ejemplo, si el servidor determina que el jugador está avanzando muy rápidamente en la aventura, puede adaptar el contenido para que la dificultad se vea incrementada. De esta forma se establece un mecanismo reactivo en el servidor LMS, que permite tomar decisiones con respecto a las acciones tomadas por el usuario.

En este caso, el fichero de adaptación tendría la siguiente forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE adaptation SYSTEM "adaptation.dtd">

<adaptation>

  <adaptation-rule>
    <description>
      This is an advanced player - make it harder
    </description>
    <uol-state>
      <property id="make-it-harder" value="1">
    </uol-state>
    <game-state>
      <activate flag="MakeItHarder">
    </game-state>
```

```
  </adaptation-rule>

</adaptation>
```

Lo único que estamos estableciendo es que debemos activar el flag *MakeItHarder* cuando el servidor LMS nos mande la propiedad *make-it-harder* con valor a *1*. Una vez más, el diseñador del juego debería hacer la aventura más difícil si el flag *MakeItHarder* está activado.

2.4.4. Motor de evaluación

El motor de evaluación, por su parte, tiene como fin establecer un sistema robusto de evaluación y calificación de los avances de los usuarios. Para poder utilizar el motor de evaluación, se provee a las aventuras con archivos que describen reglas de evaluación, que sirven para describir qué condiciones suponen un hito en el desarrollo de la partida.

Cuando las condiciones de una de las reglas se dispara, el motor de evaluación se encarga de procesar la regla, llevando a cabo las acciones descritas en la misma. Estas acciones tienen dos vertientes concretas: la generación de informes locales, y el envío de informes remotamente.

Para generar informes locales, el motor se encarga de almacenar qué reglas han sido disparadas y en qué momentos. Dicha información permanece en memoria y puede ser empleada para generar informes. Estos informes se presentan en formato XML (fácilmente legibles por máquinas) o en formato HTML (fácilmente legibles por personas). De esta forma, los usuarios pueden generar informes de evaluación que describan que avances se han llevado a cabo durante la partida, y cuánto tiempo ha tomado el cumplir cada uno de estos hitos. Dichos informes, una vez generados, pueden ser procesados por otras máquinas o personas, de cara a llevar a cabo evaluaciones de los avances de los usuarios.

El envío de informes remotos funciona de manera similiar a la generación local de informes. Cuando se dispara una regla de evaluación, esta puede tener definidas un conjunto de variables con valores que han de ser notificados al servidor de gestión de aprendizaje (servidor LMS). Estos valores se envían al servidor con el fín de que almacene dichas valoraciones, y pueda utilizarlas en futuros procesos de evaluación, calificación y adaptación de contenidos. El envío de dichos datos en tiempo real tiene una especial relevancia en el funcionamiento del motor de adaptación. De esta forma, podemos definir en el servidor LMS un comportamiento que modifique el contenido del juego remotamente si se dispara un conjunto de reglas especiales de evaluación.

Vamos a ver un ejemplo concreto del funcionamiento del motor de evaluación. Supongamos una aventura en la que se le presenta al jugador el reto de elaborar tres tipos distintos de chocolate: blanco, negro y con leche. Para ello cuenta con la ayuda de varios libros que contienen la información de qué ingredientes son necesarios para cada tipo de chocolate. Por supuesto, también dispone de los propios ingredientes.

La cuestión consiste en conocer el avance del jugador durante la partida, y no sólo saber si finalmente ha sido capaz o no de resolver el problema. Es decir, cada vez que el jugador pruebe una nueva mezcla, queremos saber si la mezcla era correcta (correspondía a un tipo

concreto de chocolate) o no. También queremos saber en qué momento ha pasado la prueba y ha elaborado correctamente los tres tipos de chocolate.

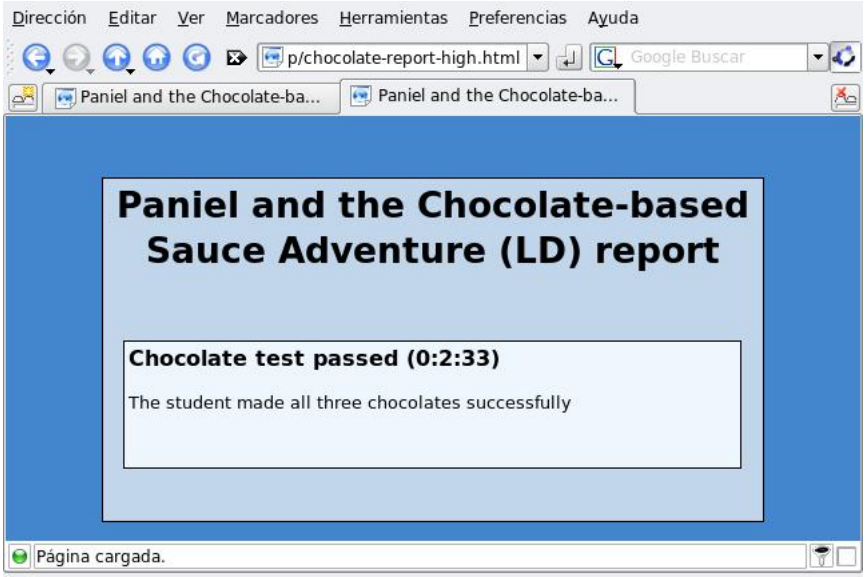
Para esto último, escribimos el fichero de reglas de generación de informes, que quedaría como mostramos a continuación:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE assessment-rules SYSTEM "assessment.dtd">

<assessment-rules>

  <assessment-rule id="MadeWhiteChocolate" importance="normal">
    <concept>New mixture<concept>
    <condition>
      <active flag="HasWhiteChocolate">
    </condition>
    <effect>
      <set-text>
        The student made white chocolate successfully
      </set-text>
    </effect>
  </assessment-rule>

  <assessment-rule id="MadeMilkChocolate" importance="normal">
    <concept>New mixture<concept>
    <condition>
      <active flag="HasMilkChocolate">
    </condition>
    <effect>
      <set-text>
        The student made milk chocolate successfully
      </set-text>
    </effect>
  </assessment-rule>

  <assessment-rule id="MadeDarkChocolate" importance="normal">
    <concept>New mixture<concept>
    <condition>
      <active flag="HasDarkChocolate">
    </condition>
    <effect>
      <set-text>
        The student made dark chocolate successfully
      </set-text>
    </effect>
  </assessment-rule>
```

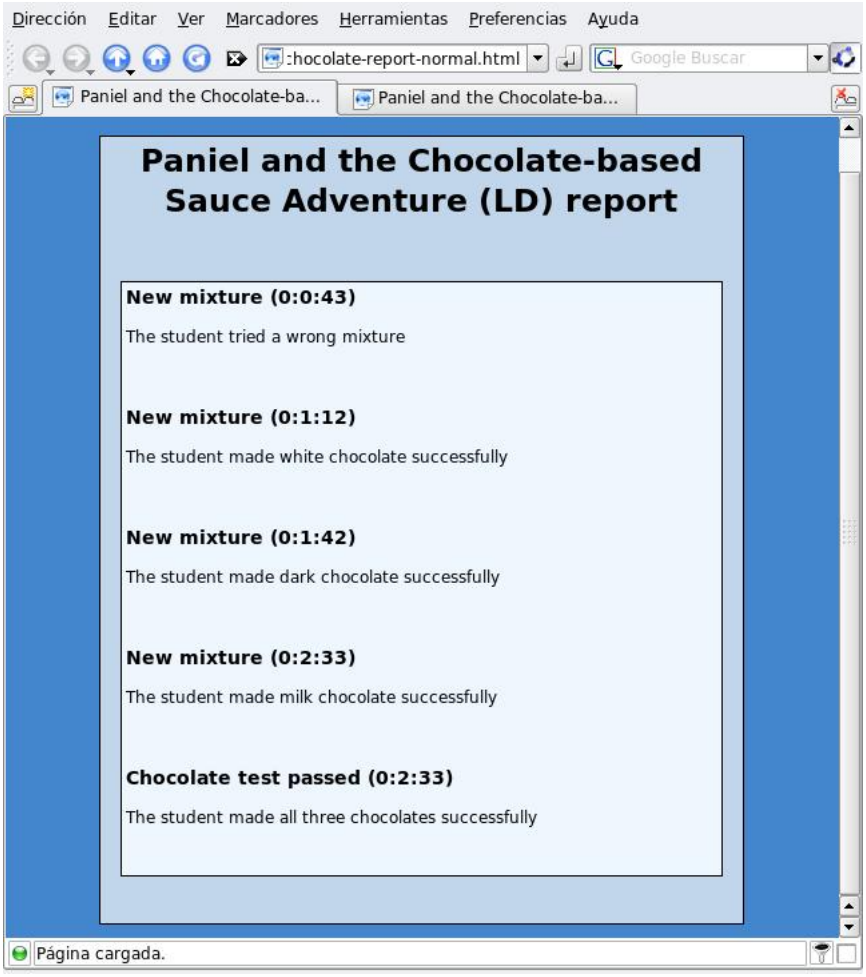
```
<assessment-rule id="MadeWrongChocolate" importance="normal">
  <concept>New mixture<concept>
  <condition>
    <active flag="TriedWrongMixture">
  </condition>
  <effect>
    <set-text>
      The student tried a wrong mixture
    </set-text>
  </effect>
</assessment-rule>

<assessment-rule id="MadeAllChocolates" importance="high">
  <concept>Chocolate test passed<concept>
  <condition>
    <active flag="HasWhiteChocolate">
    <active flag="HasMilkChocolate">
    <active flag="HasDarkChocolate">
  </condition>
  <effect>
    <set-text>
      The student made all three chocolates successfully
    </set-text>
  </effect>
</assessment-rule>

</assessment-rules>
```

Como vemos, hay tres reglas de importancia normal, una para cada tipo de chocolate. Hay una cuarta regla, también de importancia normal, que cubre el caso en que el estudiante pruebe una mezcla incorrecta. Por último, hay una regla de importancia alta, que se dispara cuando se han elaborado correctamente todos los tipos de chocolate. El sistema de prioridades (distintos niveles de importancia) es bastante útil, ya que en ocasiones nos interesará tener una visión global del desarrollo de la partida, mientras que en otras ocasiones necesitaremos un informe más detallado. Con el sistema de prioridades no necesitaremos modificar el fichero de reglas, si no que simplemente elegiremos la importancia adecuada a la hora de generar el informe. Podemos ver dos ejemplos de informe en los cuadros 2.5 y 2.6, cuyo contenido es auto-explicativo.

Podría ser interesante tener un control más minucioso aún del desarrollo de la partida. Por ejemplo, puede que quisiésemos saber qué ingredientes en concreto ha mezclado el jugador en cada momento. Esto sería posible añadiendo una regla para cada ingrediente, que se activase al utilizar dicho ingrediente. Esta regla podría tener importancia baja, de modo que los informes arriba mencionados seguirían intactos, y sólo se añadirían las entradas correspondientes al uso de cada ingrediente individual si generamos un informe de importancia baja.



Cuadro 2.5: Informe de importancia alta



Cuadro 2.6: Informe de importancia normal

2.4.5. Impacto educativo

Una vez más, recalcamos que las características anteriormente expuestas contribuyen a hacer de <e-Adventure> un proyecto especialmente diseñado a mejorar y facilitar las tareas de aprendizaje por medio de tecnologías de la información, tanto para educadores como para usuarios.

Así, los educadores tiene la posibilidad de crear y distribuir contenidos de manera sencilla, pudiendo incluir distintas opciones de adaptación o evaluación dentro de los contenidos. Es importante recordar que todas estas opciones son lo suficientemente flexibles como para permitir llevar a cabo desde controles simples a funcionamientos extremadamente complejos. En la parte de los usuarios, se les provee con una herramienta sencilla y agradable de usar. El enfoque de la aventura gráfica ayuda además a que el proceso de aprendizaje sea menos intrusivo y abstracto, para convertirse en una experiencia más inmersiva.

Capítulo 3

Análisis y especificación del sistema

3.1. Requisitos

En este apartado se enumeran aquellos requisitos definidos para el motor <e-Adventure>. Recordemos que los requisitos son las características que debe cumplir el proyecto para que pueda considerarse finalizado. Cuando hablamos de requisitos, podemos clasificarlos en dos tipos, requisitos funcionales y requisitos no funcionales. Los primeros tienen como fin definir aquellas funcionalidades con las que debe contar el sistema. Representan la especificación del uso que van a dar los usuarios al sistema, describiendo las operaciones que debe ser capaz de realizar el sistema. Los requisitos no funcionales, por otro lado, enumeran aquellas condiciones en las que el sistema presta servicios a los usuarios.

En primer lugar, se muestran los requisitos funcionales para el sistema <e-Adventure>:

- **Sistema independiente de plataforma**. Se espera que el sistema sea utilizado en multitud de entornos, con una amplia variedad de fines; por lo tanto es importante que el sistema sea capaz de ejecutarse en distintas máquinas sin importar el sistema operativo en el que funcionen.
- **Despliegue sencillo**. Por lo comentado en el punto anterior, el sistema debe ser fácilmente desplegable. Es decir, el motor debe poder instalarse y ejecutarse de forma rápida, con la menor cantidad de opciones de instalación y configuración posibles.
- **Capacidad de conexión con LMS**. El sistema debe poder conectarse con servidores de LMS, con el fin de intercambiar información e informar de los avances que el jugador lleva a cabo en la aventura. La conexión con LMS es uno de los puntos más importantes del desarrollo de <e-Adventure>, ya que la evaluación de los avances en el juego es de suma importancia para adaptar el contenido educativo para cada persona.
- **Interfaz sencilla**. El sistema debe contar con una interfaz lo más sencilla posible. Las acciones deben poder realizarse con el menor número de entradas posible. Es conveniente además que sea intuitivo utilizar el sistema, de cara a que el usuario no necesite pasar por un largo periodo de aprendizaje para poder utilizar el sistema correctamente.

- **Capacidad de expansión**. Es importante que el sistema sea lo suficientemente flexible como para realizar cambios sin grandes reestructuraciones del sistema. Dado que es la primera versión que se implementa de <e-Adventure> cabe la posibilidad de que el sistema se amplie en futuras versiones con más funcionalidades, con lo que es imprescindible disponer de una infraestructura lo suficientemente modular como para ampliar el sistema sin problemas graves.
- **Uso de guiones sencillos**. <e-Adventure> se ha creado con la intención de proporcionar contenidos educativos de manera rápida. De esta forma, es importante además que los educadores puedan crear dichos contenidos de forma relativamente rápida, aún cuando hablamos de personas con poca experiencia en el campo de la informática. Por lo tanto, el formato de los guiones que procese el motor ha de ser lo más sencillo posible, de cara a que los educadores puedan crear sus propios contenidos sin necesidad de una gran cantidad de horas de trabajo.

En cuanto a requisitos no funcionales, podemos definir los siguientes:

- **Conexión a internet**. Es necesario que aquellas personas que utilicen el sistema <e-Adventure> conectado con un servidor LMS dispongan de conexión a internet (o a redes locales) para poder llevar a cabo la adaptación y evaluacion de los contenidos de forma correcta.
- **Requisitos de hardware**. Aunque los siguientes datos son una estimación, la ejecución del sistema <e-Adventure> requiere de unas características mínimas de hardware para poder llevar a cabo ejecuciones sin problemas mayores. A continuación se indican algunos datos estimados para el hardware.
  - Procesador: 1GHz
  - Memoria RAM: 64Mb
  - Tarjeta gráfica: Mínimo de 32Mb
  - Disco duro: 50Mb (susceptible de modificación debido al posible contenido de las aventuras)

3.2. Actores y casos de uso

A continuación se describen los casos de uso que comprenden el uso del motor <e-Adventure>, junto con los actores que aparecen en dichos casos de uso. Al final, podemos ver un diagrama sencillo con los principales casos de uso, en el cuadro 3.1.

3.2.1. Actores

Dentro de la ingeniería del software, se llaman actores a aquellos individuos (ya sean humanos o aplicaciones informáticas externos) que interactúan con el sistema que se está desarrollando. Los actores que se han podido diferenciar en el desarrollo de <e-Adventure> son los siguientes:

- Usuario. A lo largo de los casos de uso, se hará referencia al usuario también como *jugador*. Esta es la persona que interactúa con el sistema jugando y resolviendo las aventuras gráficas ejecutadas por el motor. Su papel es el de utilizar el sistema con el fin de aprender de los contenidos que está consumiendo.
- Servidor LMS. El servidor de LMS (Learning Management System) tiene la obligación de proveer al usuario con contenidos educativos al usuario. En nuestro sistema, el LMS es responsable además de hacer un seguimiento de los avances del jugador, para así poder variar la experiencia de juego a medida que ésta se desarrolla. De este modo, el servidor LMS se comunica con el motor <e-Adventure> para intercambiar datos sobre el avance del usuario dentro de las aventuras.

Conviene destacar a un tercer actor que no aparece en los casos de uso, ya que no interactúa directamente con el sistema <e-Adventure>; se trata del educador. Su fin es el de crear y difundir los contenidos educativos a los usuarios, por mediación del servidor LMS u otros medios. De esta forma, el educador crea las aventuras que se van a ejecutar con el motor <e-Adventure> y las distribuye entre los usuarios. Sin embargo, vemos que al no interaccionar directamente con el motor <e-Adventure> no llega a aparecer en los casos de uso del sistema.

Por último se puede mencionar al desarrollador, que al igual que el educador interactua con el sistema <e-Adventure> pero no aparece en los casos de uso. Su labor es la de introducir modificaciones en el motor <e-Adventure> cuando es necesario ampliar la expresividad del lenguaje o las funcionalidades del motor.

3.2.2. Casos de uso generales

Los siguientes dos casos de uso contemplan el uso más general que se puede llevar a cabo con el sistema <e-Adventure>. Concretamente, ambos casos contemplan el ciclo completo de desarrollo de una partida.

Ejecución standalone

CASO DE USO #J1	Jugar partida vía motor standalone	
Objetivo en contexto	El usuario interactúa con el sistema, jugando una partida ejecutada por el motor <e-Adventure>	
Entradas	Nombre de la aventura a ejecutar, entradas por medio de ratón	
Precondiciones		
Salidas	Representación por pantalla mostrando el estado de la partida	
Poscondicion si exito	Se generan las estadísticas de la partida, si se ha elegido así	
Poscondicion si fallo	Se muestra un mensaje de error	
Actores	Usuario	
Secuencia normal	Paso	Acción
	1	El usuario elige una aventura para ejecutar de una lista de aventuras disponibles
	2	El motor carga el primer capítulo de la aventura, junto con los datos de adaptación (configuración inicial del juego) y evaluación (reglas para dictaminar el progreso del usuario en el juego), si los hay
	3	El usuario interactúa con el motor para desarrollar la ejecución del capítulo. Esto incluye tanto escenas interactivas como no interactivas
	4	El motor finaliza la ejecución del capítulo en el punto indicado por el guión
	5	Si la aventura consta de más capítulos se carga el siguiente capítulo y se pasa al paso 3. Si la aventura no consta de más capítulos, se finaliza la ejecución de la aventura

Ejecución vía web

CASO DE USO #J2	Jugar partida vía web (con conexión a LMS)	
Objetivo en contexto	El usuario interactúa con el sistema, jugando una partida ejecutada por el motor <e-Adventure>	
Entradas	Nombre de la aventura a ejecutar, entradas por medio de ratón	
Precondiciones	El motor <e-Adventure> ha de ejecutarse desde una página web con conectividad a un servidor de LMS	
Salidas	Representación por pantalla mostrando el estado de la partida	
Poscondicion si exito	Se generan las estadísticas de la partida, si se ha elegido así	
Poscondicion si fallo	Se muestra un mensaje de error	
Actores	Usuario, servidor LMS	
Secuencia normal	Paso	Acción
	1	El motor carga la aventura que ha sido distribuida junto con el motor <e-Adventure>
	2	El motor carga el primer capítulo de la aventura, junto con los datos de adaptación (configuración inicial del juego) y evaluación (reglas para dictaminar el progreso del usuario en el juego), si los hay
	3	El motor se conecta con el servidor de LMS para pedirle datos de juego (como configuraciones iniciales, que pueden sobreescribir los datos de adaptación)
	4	El usuario interactua con el motor para desarrollar la ejecución del capítulo. Esto incluye tanto escenas interactivas como no interactivas. Durante la ejecución se recibe y envía información al LMS, adaptando el contenido del juego dinámicamente
	5	El motor finaliza la ejecución del capítulo en el punto indicado por el guión
	6	Si la aventura consta de más capítulos se carga el siguiente capítulo y se pasa al paso 3. Si la aventura no consta de más capítulos, se finaliza la ejecución de la aventura

3.2.3. Acciones básicas del juego

A continuación se exponen los casos de uso que son necesarios e indispensables para desarrollar una partida por mediación del sistema <e-Adventure>. Dichos casos de uso contemplan las acciones que puede realizar el usuario con el sistema por medio del personaje que controla. Estas acciones constituyen las piezas elementales sobre las que se construye la experiencia de juego dentro de <e-Adventure>.

Mover personaje

CASO DE USO #ACN1	Mover personaje	
Objetivo en contexto	Hacer que el personaje controlado por el jugador se desplaze por el escenario	
Entradas	Selección de la posición del escenario a la que moverse	
Precondiciones	El motor ha de estar ejecutando una escena interactiva	
Salidas	Cambio en la posición del protagonista. Cambio de escena	
Poscondicion si exito	La posición del protagonista varía. Se carga una nueva escena	
Poscondicion si fallo	El jugador no se mueve	
Actores	Usuario, servidor LMS	
Secuencia inicial	Paso	Acción
	1	El usuario hace click en un punto del escenario
	2	El protagonista se desplaza hasta el punto indicado. Si este punto es una salida de la escena ir al paso S1
Secuencia cambio escena	Paso	Acción
	S1	Se ejecutan los efectos previos al cambio de escena, si los hay
	S2	El motor cambia la escena representada
	S3	Se ejecutan los efectos posteriores al cambio de escena, si los hay
	S4	Si el estado del motor ha variado significativamente, se informa al servidor LMS, si existe comunicación con él

Mirar objeto

CASO DE USO #ACN2	Mirar objeto	
Objetivo en contexto	Hacer que el protagonista de una descripción corta de un objeto	
Entradas	Selección de la operación *Mirar* y del objeto a mirar	
Precondiciones	El motor ha de estar ejecutando una escena interactiva	
Salidas	Representación del estado del juego por pantalla	
Poscondicion si exito	El protagonista emite un mensaje con la descripción corta del objeto	
Poscondicion si fallo	-	
Actores	Usuario, servidor LMS	
Secuencia normal	Paso	Acción
	1	El usuario selecciona la acción *Mirar* y el objeto en la escena a describir
	2	El protagonista emite un mensaje con la descripción corta del objeto

Examinar objeto

CASO DE USO #ACN3	Examinar objeto	
Objetivo en contexto	Hacer que el protagonista de una descripción detallada de un objeto	
Entradas	Selección de la operación *Examinar* y del objeto a examinar	
Precondiciones	El motor ha de estar ejecutando una escena interactiva	
Salidas	Representación del estado del juego por pantalla	
Poscondicion si exito	El protagonista emite un mensaje con la descripción detallada del objeto. Se ejecutan los efectos asociados a la acción, si los hay	
Poscondicion si fallo	-	
Actores	Usuario, servidor LMS	
Secuencia normal	Paso	Acción
	1	El usuario selecciona la acción *Examinar* y el objeto en la escena a describir
	2	Si el objeto no se encuentra en el inventario el protagonista se desplaza para colocarse a su lado
	3	Si la acción *Examinar* del objeto tiene efectos asociados y el estado del motor cumple las condiciones para la ejecución, se ejecutan los efectos asociados a la acción. Si no tiene efectos asociados el protagonista emite un mensaje con la descripción detallada del objeto
	4	Si el estado del motor ha variado significativamente, se informa al servidor LMS, si existe comunicación con él

Coger objeto

CASO DE USO #ACN4	Coger objeto	
Objetivo en contexto	Hacer que el personaje controlado por el jugador recoja un objeto del escenario	
Entradas	Selección de la operacion *Coger* y del objeto a recoger	
Precondiciones	El motor ha de estar ejecutando una escena interactiva	
Salidas	Representación del estado del juego por pantalla	
Poscondicion si exito	Si el guión no indica lo contrario, el objeto desaparece del escenario y aparece en el inventario del jugador. Se ejecutan los efectos asociados a la acción, si los hay	
Poscondicion si fallo	El protagonista muestra un mensaje indicando que no puede recoger el objeto	
Actores	Usuario, servidor LMS	
Secuencia normal	Paso	Acción
	1	El usuario selecciona la acción *Coger* y el objeto a recoger
	2	Si el objeto se encuentra en el inventario, pasar a S1
	3	El protagonista se desplaza hasta donde está el objeto a recoger
	4	El protagonista recoge el objeto si el estado del motor cumple las condiciones establecidas en el guión. Si el objeto no puede ser recogido, pasar a S2
	5	A menos que los efectos asociados indiquen lo contrario, el objeto desaparece del escenario y pasa a colocarse en el inventario
	6	Se ejecutan los efectos asociados a la acción, si los hay
	7	Si el estado del motor ha variado significativamente, se informa al servidor LMS, si existe comunicación con él
Secuencia alternativa	Paso	Acción
	S1	El protagonista emite un mensaje indicando que ya posee el objeto. Fin del caso de uso
	S2	El protagonista emite un mensaje indicando que no puede recoger el objeto. Fin del caso de uso

Usar objeto (de forma independiente)

CASO DE USO #ACN5	Usar objeto independientemente	
Objetivo en contexto	Hacer que el personaje controlado por el jugador use un objeto	
Entradas	Selección de la operacion *Usar* y del objeto a utilizar	
Precondiciones	El motor ha de estar ejecutando una escena interactiva	
Salidas	Representación del estado del juego por pantalla	
Poscondicion si exito	Se ejecutan los efectos asociados a la acción, si los hay	
Poscondicion si fallo	El protagonista muestra un mensaje indicando que no puede utilizar el objeto	
Actores	Usuario, servidor LMS	
Secuencia normal	Paso	Acción
	1	El usuario selecciona la acción *Usar* y el objeto a utilizar
	2	Si el objeto se encuentra en el inventario, pasar a 4
	3	El protagonista se desplaza hasta donde se encuentra el objeto
	4	El protagonista utiliza el objeto si el estado del motor cumple las condiciones establecidas en el guión. Si el objeto no puede ser utilizado, pasar a S1
	5	Se ejecutan los efectos asociados a la acción, si los hay
	6	Si el estado del motor ha variado significativamente, se informa al servidor LMS, si existe comunicación con él
Secuencia alternativa	Paso	Acción
	S1	El protagonista emite un mensaje indicando que es incapaz de usar el objeto

Usar objeto con objeto

CASO DE USO #ACN6	Usar objeto con otro objeto	
Objetivo en contexto	Hacer que el personaje controlado por el jugador use un objeto con otro	
Entradas	Selección de la operacion *Usar* y los objetos a utilizar	
Precondiciones	El motor ha de estar ejecutando una escena interactiva. Al menos uno de los dos objetos debe encontrarse en el inventario. Ninguno de los dos objetos debe permitir ser usado individualmente en el estado actual del motor	
Salidas	Representación del estado del juego por pantalla	
Poscondicion si exito	Se ejecutan los efectos asociados a la acción, si los hay	
Poscondicion si fallo	El protagonista muestra un mensaje indicando que no puede utilizar los objetos	
Actores	Usuario, servidor LMS	
Secuencia normal	Paso	Acción
	1	El usuario selecciona la acción *Usar* y los objetos a utilizar
	2	Si ambos objetos se encuentran en el inventario, pasar a 4
	3	El protagonista se desplaza hasta donde se encuentra el objeto que no está en su inventario
	4	El protagonista utiliza el objeto si el estado del motor cumple las condiciones establecidas en el guión. Si el objeto no puede ser utilizado, pasar a S1
	5	Se ejecutan los efectos asociados a la acción, si los hay
	6	Si el estado del motor ha variado significativamente, se informa al servidor LMS, si existe comunicación con él
Secuencia alternativa	Paso	Acción
	S1	El protagonista emite un mensaje indicando que es incapaz de usar el objeto

Entregar objeto a personaje

CASO DE USO #ACN7	Dar un objeto a un personaje	
Objetivo en contexto	Hacer que el personaje controlado por el jugador entrege un objeto a un personaje	
Entradas	Selección de la operacion *Dar*, el objeto a entregar y el personaje al que dar dicho objeto	
Precondiciones	El motor ha de estar ejecutando una escena interactiva. El objeto a entregar debe encontrarse en el inventario	
Salidas	Representación del estado del juego por pantalla	
Poscondicion si exito	Se ejecutan los efectos asociados a la acción, si los hay	
Poscondicion si fallo	El protagonista muestra un mensaje indicando que no puede utilizar los objetos	
Actores	Usuario, servidor LMS	
Secuencia normal	Paso	Acción
	1	El usuario selecciona la acción *Dar*, un objeto de su inventario y un personaje de la escena
	2	El protagonista se desplaza hasta donde se encuentra el personaje
	3	El protagonista entrega el objeto si el estado del motor cumple las condiciones establecidas en el guión. Si el objeto no puede ser entregado, pasar a S1
	4	Se ejecutan los efectos asociados a la acción, si los hay
	5	A menos que los efectos asociados indiquen lo contrario, el objeto entregado desaparece del inventario
	6	Se ejecutan los efectos asociados a la acción, si los hay
	7	Si el estado del motor ha variado significativamente, se informa al servidor LMS, si existe comunicación con él
Secuencia alternativa	Paso	Acción
	S1	El protagonista emite un mensaje indicando que es incapaz de entregar el objeto

Conversar con personaje

CASO DE USO #ACN8	Conversar con un personaje	
Objetivo en contexto	Desarrollar una conversación entre el protagonista y un número indeterminado de personajes	
Entradas	Selección de la acción *Hablar* y el personaje con el que hablar. Selección de respuestas dentro de la conversación	
Precondiciones	El motor ha de estar ejecutando una escena interactiva	
Salidas	Representación del estado del juego por pantalla. Líneas de diálogo de los personajes	
Poscondicion si exito	Se ejecutan los efectos asociados al fin de la conversación, si los hay	
Poscondicion si fallo	El protagonista muestra un mensaje indicando que no puede hablar con el personaje	
Actores	Usuario, servidor LMS	
Secuencia inicial	Paso	Acción
	1	Si la conversación se ha disparado por medio de un efecto, ir directamente al paso 4
	2	El usuario selecciona la acción *Hablar* y un personaje con el que dialogar
	3	El protagonista se desplaza hasta donde se encuentra el personaje
	4	El motor carga el primer bloque de diálogo lineal
	5	Se ejecuta el bloque de dialogo lineal. Si el dialogo termina con el bloque actual, pasar al paso 8. Si no se carga el siguiente bloque de diálogo, si es lineal se ejecuta de nuevo el paso actual, si es de selección de opción pasar al paso 6
	6	Se ejecuta el bloque de selección de opcion actual mostrando por pantalla las posibles respuestas
	7	El usuario selecciona una de las respuestas. Se carga el bloque de diálogo que corresponde a esa respuesta. Si dicho bloque es de diálogo lineal, ir al paso 5. Si es un bloque de selección de opción, ir al paso 6
	8	Finalizada la conversación, se ejecutan los efectos asociados al nodo terminal ejecutado, si los hay
	9	Si el estado del motor ha variado significativamente, se informa al servidor LMS, si existe comunicación con él

3.2.4. Casos de uso complementarios

Además de los casos de uso descritos en el apartado anterior, hay otros que sirven de apoyo para le ejecución de las partidas. Dichos casos de uso contemplan acciones secundarias, como son la edición de opciones, cargado y guardado de partidas, etc... Estos casos de uso no se consideran como núcleo de la parte jugable, pero son indispensables para el buen funcionamiento del motor.

Modificar opciones

CASO DE USO #OP1	Modificar opciones	
Objetivo en contexto	Modificar las opciones que rigen el comportamiento del juego	
Entradas	Selección de las opciones que se quieren habilitar o deshabilitar	
Precondiciones	El motor ha de estar ejecutando una escena interactiva	
Salidas	Información sobre las opciones del juego	
Poscondicion si exito	Las opciones se guardan y se aplican a lo largo del juego	
Poscondicion si fallo	Se deshechan los cambios efectuados	
Actores	Usuario	
Secuencia normal	Paso	Acción
	1	El usuario despliega el menu de configuración, por medio entradas de ratón o teclado
	2	El usuario modifica los valores de configuración
	3	El usuario sale del menú, volviendo así al juego, con las nuevas opciones de configuración
	4	Se guardan en un archivo los nuevos valores de configuración para el juego

Salvar partida

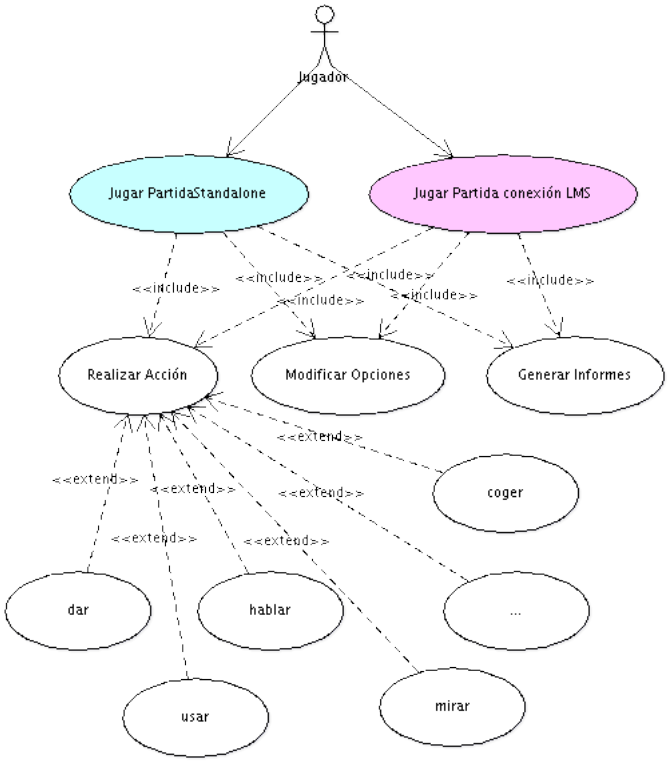
CASO DE USO #OP2	Salvar partida	
Objetivo en contexto	Guardar el estado de una partida	
Entradas	Selección de la opción de guardado, selección de slot de guardado	
Precondiciones	El motor ha de estar ejecutando una escena interactiva. El motor ha de estar ejecutándose en modo standalone (no en un Applet)	
Salidas	Información sobre el guardado de la partida	
Poscondicion si exito	Los datos de la partida se guardan en un archivo	
Poscondicion si fallo	No se guardan los datos de la partida actual	
Actores	Usuario	
Secuencia normal	Paso	Acción
	1	El usuario despliega el menú de salvado, por medio de entradas de ratón o teclado
	2	El usuario selecciona un slot de entre los disponibles para guardar la partida
	3	Se guarda la partida en el slot indicado (escribiendo la información en un archivo)
	4	El usuario sale del menú, volviendo así al juego

Cargar partida

CASO DE USO #OP3	Cargar partida	
Objetivo en contexto	Cargar el estado de una partida	
Entradas	Selección de la opción de cargado, selección de slot de cargado	
Precondiciones	El motor ha de estar ejecutando una escena interactiva. El motor ha de estar ejecutándose en modo standalone (no en un Applet)	
Salidas	Información sobre el cargado de la partida	
Poscondicion si exito	Se restaura el estado del juego, a partir de los datos leídos	
Poscondicion si fallo	La ejecución del motor sigue normalmente	
Actores	Usuario	
Secuencia normal	Paso	Acción
	1	El usuario despliega el menú de cargado, por medio de entradas de ratón o teclado
	2	El usuario selecciona un slot de entre los disponibles para cargar la partida
	3	El motor lee los datos del slot (por mediación de un archivo) y restaura el estado de la partida
	4	El motor reanuda la ejecución de la partida

Generar informes de evaluación

CASO DE USO #OP4	Generar informes de evaluación	
Objetivo en contexto	Generar un informe con los datos de los progresos del usuario en el juego	
Entradas	Selección de la opción de generación de informes, selección de localización para la generación, selección de configuración para los formatos de los informes	
Precondiciones	El motor ha de estar ejecutando una escena interactiva. La aventura cargada actualmente debe disponer de datos de evaluación	
Salidas	Archivo con la evaluación de los progresos del usuario	
Poscondicion si exito	Se genera un archivo con los progresos del usuario	
Poscondicion si fallo	No se genera informe alguno	
Actores	Usuario	
Secuencia normal	Paso	Acción
	1	El usuario despliega el menú de generación de informes, por medio de entradas de ratón o teclado
	2	El usuario selecciona la configuración que quiere para la generación del informe
	3	El motor genera los archivos de evaluación en la ruta indicada, utilizando para ello los datos de evaluación que se han recogido a lo largo de la ejecución
	4	El motor reanuda la partida



Cuadro 3.1: Casos de uso básicos

3.3. Metodología y planificación

En el presente capítulo se describe la metodología que se ha seguido durante el desarrollo en lo que respecta a la organización del grupo y su metodología de trabajo. Además, se muestra información sobre la planificación que se ha llevado a cabo durante el desarrollo del sistema.

3.3.1. Roles de los integrantes

Antes de comenzar, conviene describir la posición y responsabilidades que ha tenido cada persona involucrada en el desarrollo del sistema. A continuación se incluye el nombre y las responsabilidades del grupo de trabajo:

- Baltasar Fernández Manjón. Director de proyecto. Su trabajo ha sido el de coordinar y dirigir el desarrollo del proyecto en terminos generales, controlando los hitos más significativos, así como evaluando la documentación generada durante dichos periodos de desarrollo.
- Pablo Moreno Ger. Director de proyecto. Su trabajo ha sido el de supervisar y dirigir directamente el desarrollo del proyecto, organizando reuniones con el equipo de desarrollo periodicamente, fijando objetivos y proponiendo ideas para su implementación.

- Bruno Torijano Bueno. Coordinador de equipo, desarrollador. Como coordinador de equipo, ha sido su labor la de conseguir que la comunicación entre el área de dirección y la de desarrollo fuera la correcta y deseada, ayudando a la organización de las reuniones y pasando mensajes en ambos sentidos siempre que fuera necesario. Como desarrollador, ha participado activamente en el análisis, diseño e implementación del motor <e-Adventure>, junto con el resto del equipo de desarrollo.
- Francisco M. Pérez Padilla y Eduardo Sollet Galeán. Desarrolladores. Como integrantes del equipo de desarrollo, su trabajo ha sido también el de analizar, diseñar e implementar la totalidad del sistema <e-Adventure>. Cabe destacar que en el equipo de desarrollo no ha habido roles establecidos, si no que todos los tres miembros han participado en la implementación de las distintas partes del motor de manera conjunta y coordinada.

Una vez definidos los roles de cada uno de los integrantes del equipo, podemos pasar a describir la metodología de trabajo que hemos seguido a lo largo del año.

3.3.2. Metodología de trabajo

Para el desarrollo del sistema hemos definido conjuntos de microhitos semanalmente, agrupados en grandes bloques de objetivos finales. De esta forma, cada microhito consta de un número reducido de objetivos a implementar en un periodo de una semana aproximadamente, mientras que los hitos mayores (que hemos llamado iteraciones) son grandes conjuntos de hitos. Concretamente, hemos realizado un total de tres iteraciones, sumando un total de veinte microhitos con una duración de una a dos semanas cada uno.

Antes de comenzar con las iteraciones, llevamos a cabo reuniones preliminares para fijar y estudiar los objetivos del proyecto. Así mismo, desarrollamos algunos prototipos tecnológicos para verificar la viabilidad de las funcionalidades que necesitábamos para el proyecto. En dichas reuniones preliminares se fijó además un esbozo preliminar de los objetivos para cada una de las tres grandes iteraciones; en cualquier caso dichos objetivos fueron revisados y modificados a lo largo del desarrollo del sistema.

El esquema general de desarrollo de las iteraciones ha sido el siguiente, habiendo sido aplicado en las tres ocasiones:

- **Análisis**. En primer lugar se llevaba a cabo una reunión entre el equipo de desarrollo y el supervisor para fijar y refinar los objetivos para el fin de la iteración. De esta forma, se evaluaba y examinaba el trabajo realizado previamente, y se fijaban los objetivos para la iteración que se iba a llevar a cabo, proponiendo mejoras y cambios, tanto para la planificación del trabajo como para los objetivos en sí.
- **Implementación**. Una vez fijados los objetivos de la iteración, el trabajo se divide en microhitos que se van implementando poco a poco a lo largo de la iteración. La información sobre la metodología de trabajo en los microhitos puede consultarse más adelante.
- **Evaluación**. Terminada la parte de implementación, el equipo de desarrollo llevaba a cabo otra reunión con el director supervisor de cara a evaluar los avances conseguidos

en la iteración. En esta reunión se evaluan también las dificultades y problemas surgidos durante la etapa de implementación, para poder prevenirlos y atajarlos de cara a futuras iteraciones.

- **Documentación y cierre**. Una vez evaluados los avances llevados a cabo en la iteración, el equipo de desarrollo procede a realizar la documentación relativa a la implementación. Dicha iteración contempla información sobre la apertura y cierre de la iteración, sus objetivos, la descripción de los microhitos y la información más relevante sobre los cambios y mejoras efectuados en el sistema. Dicha documentación se entrega después a los directores, de cara a ser evaluada.

Los microhitos también han tenido su propia metodología de trabajo, que se describe a continuación:

- **Reunión inicial de microhito**. Al comienzo de cada microhito, el equipo de desarrollo lleva a cabo una reunión con el director supervisor del proyecto, de cara a confirmar los objetivos para el microhito, además de para proponer ideas, mejoras y modificaciones en la funcionalidad del sistema. Estas reuniones se han llevado a cabo casi siempre semanalmente, aún cuando los microhitos han tenido duraciones de más de una semana; en estos casos las reuniones han tenido un marcado carácter de seguimiento, proporcionando información al supervisor del estado del desarrollo, pudiendo asi solventar problemas surgidos en las fases de implementación en sí.
- **Implementación y seguimiento**. A lo largo del microhito, el equipo de desarrollo se encarga de implementar las funcionalidades descritas como objetivos dentro del microhito. Durante estos periodos de implementación se ha mantenido la comunicación entre el supervisor y el equipo de desarrollo, por medio del coordinador del equipo de trabajo, encargándose este de notificar al supervisor las dificultades encontradas, o solicitando información sobre problemas específicos que surgían durante la implementación.

En términos generales, puede verse cómo se ha intentado tener un control permanente sobre el estado del proyecto, llevándose a cabo numerosas reuniones para asegurar que la planificación era cumplida dentro del lo previsto, y para solucionar posibles problemas y dudas que surgieran a lo largo del desarrollo.

En los tres apartados siguientes, se describen las tres grandes iteraciones con las que se ha desarrollado el sistema. Dentro de cada iteración se encuentran los microhitos que la componen, indicando además la cantidad de semanas empleadas para llevar a cabo cada uno de estos microhitos. Además, al final podremos ver unos sencillos diagramas de Gantt, con la divisón del trabajo en iteraciones y en semanas.

3.3.3. Primera iteración

Esta primera iteración tuvo lugar entre el 16 de octubre y el 21 de diciembre de 2006. Sus objetivos básicos fueron realizar un pequeño rediseño del prototipo de partida, así como adaptarlo para cargar partidas dinámicamente desde un fichero XML. Además, se llevó a cabo la inclusión del soporte para conversaciones en el motor. La división del trabajo por semanas de esta primera iteración fue la siguiente.

1^a semana

En esta primera semana, del 2 al 9 de octubre, no hubo ningún desarrollo, ya que la dedicamos a familiarizarnos con la primera versión del lenguaje <e-Game> (que era como se llamaba entonces, antes de pasar a ser <e-Adventure>), además de con las tecnologías que íbamos a usar para el análisis de ficheros XML (SAX), así como con la programación de juegos en Java.

2^a semana

Del 9 al 16 de octubre redactamos una primera versión del documento de requisitos de recursos. En él definimos qué tipos de archivos multimedia íbamos a soportar dentro del motor. Por otra parte, implementamos pequeños prototipos tecnológicos que diesen soporte a cada uno de estos recursos.

3^a semana

Del 16 al 23 de octubre escribimos un juego sencillo consistente en dos escenarios conectados por una salida, más otra salida de fin de juego. Con ello comprobamos que el lenguaje era consistente y que podíamos continuar con su desarrollo.

4^a y 5^a semana

Para este punto de control establecimos un plazo más largo de lo normal (del 23 de octubre al 6 de noviembre), ya que era necesario debido a la envergadura del trabajo realizado. Por una parte, creamos un nuevo diseño para el HUD del juego. Por otro, dimos soporte a la carga de personajes desde el fichero XML (aún sin conversaciones). Al final de este periodo también se soportaba la acción de coger objetos del escenario.

6^a semana

En esta semana, del 6 al 13 de noviembre, dimos soporte al resto de acciones del motor (mirar, examinar, dar y usar), a excepción de hablar, ya que aún no cargábamos las conversaciones. Además, se soportaban ciertos efectos, la generación dinámica de objetos del inventario, así como la capacidad de ser consumidos.

7^a semana

Del 13 al 20 de noviembre se implementó el soporte para cutscenes sencillas de imágenes fíjas, estilo diapositivas. Por otro lado, también comenzamos con el desarrollo de las escenas de libros. Además, implementamos una primera versión de la carga y salvado de juegos a ficheros binarios, que más tarde se convertiría para utilizar ficheros de texto.

8^a semana

En la semana del 20 al 27 de noviembre escribimos el guión de una pequeña aventura sobre seguridad en construcción en el lenguaje <e-Adventure>. Dicha aventura era una demostración tecnológica de las capacidades hasta el momento del motor, que consistía en tres escenas en las que se podían coger y usar unos pocos objetos, así como entablar pequeñas conversaciones. Para esto último, tuvimos que introducir el sistema de conversaciones. Con la creación de este pequeño prototipo tecnológico en forma de aventura pudimos verificar que lo desarrollado hasta el momento era correcto.

9^a semana

En esta penúltima semana de la primera iteración, del 27 de noviembre al 4 de diciembre, terminamos con la implementación, y nos dedicamos a buscar una licencia que se adaptase a las tecnologías utilizadas, preferentemente GPL. Teníamos problemas con las librerías utilizadas para reproducción de MPEGs (que estaban contemplados en el documento de requisitos de recursos), pero finalmente el soporte para dicho formato de vídeo se ha abandonado, aunque puede que se complete más adelante.

10^a semana

Del 4 de al 20 de diciembre de 2006 realizamos la documentación de la primera iteración. Además se escribió un manual de autor, y se añadieron los cambios realizados a la especificación del lenguaje <e-Adventure>. Entonces tuvimos un pequeño tiempo de descanso en vacaciones de navidad, y se continuó con el desarrollo la segunda semana del año 2007.

3.3.4. Segunda iteración

En esta iteración, que ocupó del 8 de enero al 5 de marzo de 2007, <e-Game> pasó oficialmente a llamarse <e-Adventure>. En esta iteración convertimos el sistema en un verdadero motor, con opciones de cargar uno u otro juego tras lanzar la aplicación, con menús para guardar o cargar partidas, y con un menú de opciones para controlar parámetros del juego como el sonido o la velocidad el texto. Por otro lado, realizamos la integración de la aplicación en un navegador web, así como la comunicación con el LMS. Además, desarrollamos un sistema de seguimiento del desarrollo de la partida de cada alumno para su evaluación posterior.

11^a semana

En esta primera semana de la segunda iteración, del 8 al 15 de enero, realizamos la refactorización para reflejar el cambio del nombre del sistema en el código. Actualizamos las escenas de libro para que soporten imágenes y listas. También implementamos el scroll en el inventario. Por último, hicimos que las escenas se actualicen si es necesario tras la ejecución de un efecto, ya que se podría modificar un flag de modo que se cambiase, por ejemplo, la imagen de fondo de la escena en curso, y habría que actualizarla en el momento.

12ª semana

Del 15 al 22 de enero hicimos que <e-Adventure> fuese un motor completo, con un menú de selección de la partida a cargar tras lanzar la aplicación, y con menús para cambiar las opciones de la partida. A partir de entonces cada aventura está contenida en un fichero ZIP, que habrá que cargar desde el motor para poder ejecutarla.

13ª semana

Del 22 al 29 de enero implementamos el menú de las opciones de juego, de modo que se puedan controlar el sonido, la velocidad del texto, etc. desde dentro de la aplicación. También implementamos en esta semana la carga y guardado de partidas. Otro apartado importante implementado esta semana es la lectura del fichero de configuración, que prepara el comienzo de la partida con los valores iniciales que se desee.

14ª y 15ª semana

En esta ocasión volvemos a tener un punto de control más largo, ya que hay bastante trabajo, y comienzan los exámenes de febrero. Del 29 de enero al 12 de febrero implementamos la lectura del fichero de reglas y evaluación, que define bajo qué condiciones se generan informes para la evaluación posterior. También implementamos la generación de dichos informes en formato HTML y XML, así como la comunicación del desarrollo de la partida al servidor LMS.

16ª semana y 17ª semana

Del 12 al 26 de febrero, de nuevo un período largo, preparamos el final de la segunda iteración. Hicimos las pruebas pertinentes y comenzamos la documentación de esta iteración. El único cambio fue la inclusión de un diálogo para la generación de informes.

18ª semana

Del 26 de febrero al 5 de marzo, dando fin a la segunda iteración, finalizamos también la documentación de la misma.

3.3.5. Tercera iteración

El objetivo principal de esta tercera y última iteración es refinar el lenguaje y, por tanto, el motor, para hacer que sea capaz de ejecutar partidas más o menos complejas, a fin de maximizar su capacidad expresiva. Por otro lado, intentamos construir un nuevo tipo de interfaz gráfica más amigable, conservando la interfaz anterior con un estilo más clásico. Para reflejar todos estos cambios, creamos una aventura de una complejidad mucho mayor a las anteriores aventuras de prueba.

19^a semana

En la primera semana de la tercera iteración, del 5 al 12 de marzo, y para facilitar la construcción de aventuras complejas, hicimos que varias aventuras simples formasen una más compleja en forma de capítulos. Para ello, fue necesario adaptar el sistema para que cargase la descripción del juego como una lista de subaventuras a ejecutar por orden. Añadimos efectos consistentes en la reproducción de sonidos, y la posibilidad de personalizar el color del texto de cada personaje independientemente. Además realizamos otros cambios para mejorar la eficiencia y solucionar fallos.

20^a y 21^a semana

Del 12 al 26 de marzo añadimos el soporte para la interfaz gráfica basada en menús contextuales. Por otro lado hicimos posible que se pudiese usar un sólo objeto, ya que antes sólo se soportaba la combinación de dos objetos.

22^a y 23^a semana

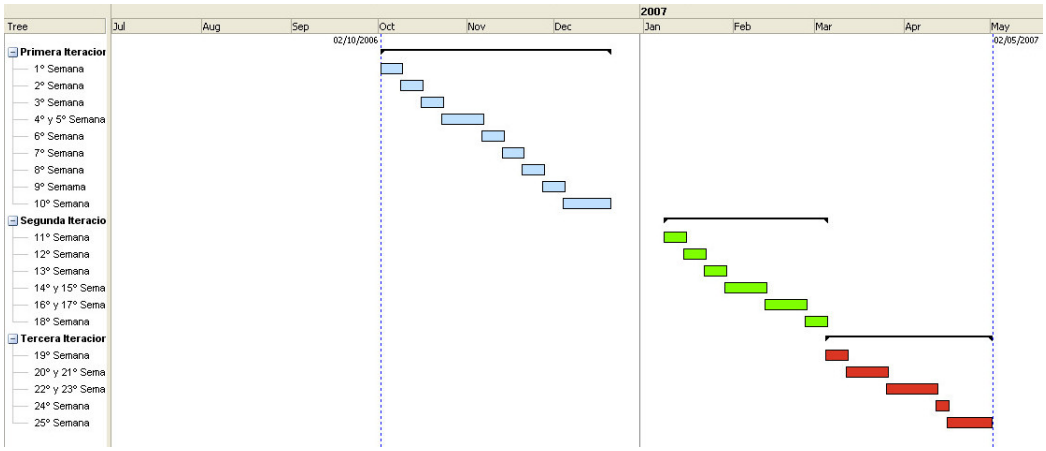
Del 16 de marzo al 12 de abril dimos soporte a sencillas escenas no interactivas, en las que podemos mover a los personajes por el escenario y hacer que hablen entre ellos. Para ello, añadimos varios efectos, como mover personajes y ejecutar animaciones. Por último, dimos soporte al disparo de efectos al entrar a una escena. Anteriormente, sólo se soportaba que se disparasen efectos al salir, pero no al entrar. La ejecución de efectos al entrar en una escena era indispensable para la construcción de escenas no interactivas.

24^a semana

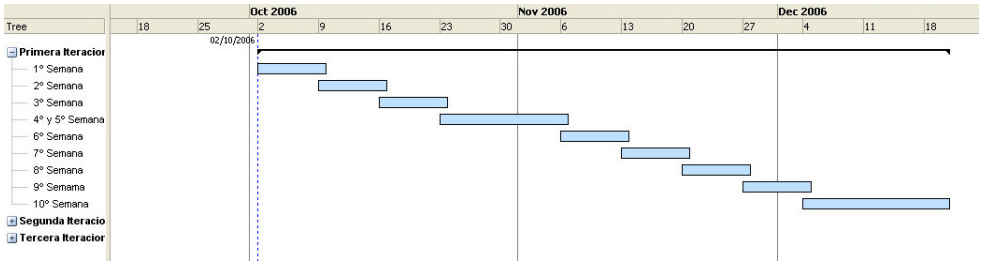
Del 12 al 16 de abril dimos soporte al desplazamiento de escenas, de forma que se pudiese jugar en escenas con fondos más grandes que la pantalla de juego. Para ello, se dibujaba la parte de la escena en la que se encontraba el jugador en cada momento, y se desplazaba la zona a pintar cuando el jugador se acercaba a los márgenes de la pantalla de juego. Además, hicimos posible que se pudiesen cambiar los valores de los flags desde el servidor en tiempo real tal y como se actualizan al realizar una acción, ya que hasta el momento desde el servidor sólo se podían recibir unos flags al comenzar la partida con los que configurar el inicio de la misma.

25^a semana en adelante

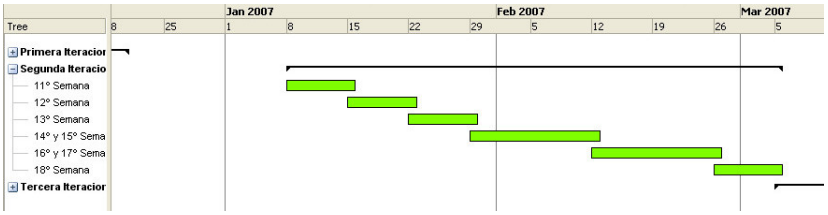
A partir de entonces, el estado del sistema cumplía todos los requisitos iniciales del proyecto. A partir del 16 de abril comenzamos con la documentación del mismo. Algún cambio interesante añadido fue la inclusión de imágenes de primer plano, incluyendo un nuevo sistema de pintado para que se los objetos y personajes se pinten de más a menos profundidad, de modo que los que están delante aparezcan efectivamente en primer plano. Por último, continuamos el desarrollo de la aventura, junto con la corrección de pequeños fallos y la mejora del sistema.



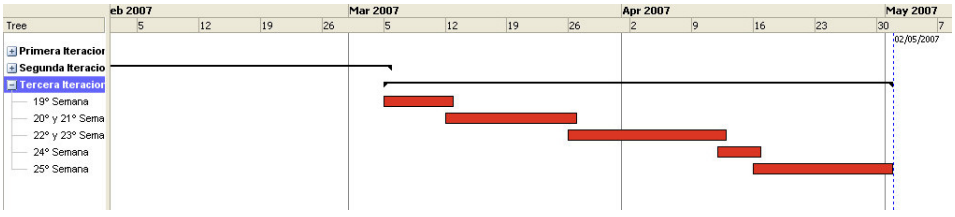
Cuadro 3.2: Diagrama de Gantt - Tres iteraciones



Cuadro 3.3: Diagrama de Gantt - Primera iteración



Cuadro 3.4: Diagrama de Gantt - Segunda iteración



Cuadro 3.5: Diagrama de Gantt - Tercera iteración

3.4. Fundamentos tecnológicos

3.4.1. Java 5.0

Debido al requisito de que el sistema pueda ejecutarse en la web y por lo tanto sea independiente de la plataforma para ejecuarse, la eleccion de Java fue clara.

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 1990, actualmente en su version 6.0 (Java SE 6 (1.6.0) publicada el 11 de diciembre de 2006). Para este proyecto usaremos una versión anterior pero mas común, la versión J2SE 5.0 (1.5.0) del 29 de septiembre de 2004. No utilizamos la versión más popular y/o extendida de Java (J2SE 1.4.2 del 26 de junio de 2003) debido a ciertos componentes nuevos necesarios para nuestra aplicación, por ejemplo, para poder acceder a un nivel muy básico en las imágenes y poder modificar sus colores y transparencia.

Todo el sistema está diseñado en Java tanto para lanzarse desde una web mediante un applet como para ejecuarse independientemente como una aplicación. Usamos el API original de Java 5.0 para todo, desde cargar archivos ZIP, como imágenes como comunicación, etc. La única libreria externa es JLayer para reproducir archivos MP3, todas las demás características y funcionalidades del proyecto han sido implementadas usando el API original.

Aunque JAVA nos ofrece componentes graficos (como los frameworks AWT o Swing) hemos creado nuestro propio control de pintado para la aventura para poder tener mayor control y poder realizar características como la interfaz contextual. Hemos obtenido el componente basico para el pintado de JAVA, es decir, el Graphics del frame y sobre él realizamos todo nuestro trabajo de pintado personalizado en vez de usar los frameworks Awt o Swing. De esta forma no sobrecargamos ni el pintato ni el sistema con componentes que no vamos a utilizar, lo cuales son necesarion en dichos frameworks, y asi poder obtener mejores rendimiento o aumentar la calidad o cantidad de los objetos a pintar.

3.4.2. XML

XML es un metalenguaje extensible de etiquetas que permite definir la grámatica para lenguajes específicos, en este caso, el usado para el especifico de <e-Adventure>.

Mediante el XML se ha podido definir el lenguaje utilizado para la creacion de las aventuras y mediante la DTD, siglas de “Document Type Definition”, hemos definido las restricciones en la estructura y sintaxis del mismo. Aunque la DTD se puede incluir dentro del XML, hemos optado por la opción mas comun que es tenerlo en otro archivo por separado. La ventaja mas clara de esto es que cada aventura no tendra que tener una copia de la DTD dentro de su XML sino que el sistema solo necesita un archivo DTD con la sintaxis del lenguaje de aventuras para cualquier numero de ellas.

3.4.3. SAX

Para la lectura del XML, de las aventuras, dentro de nuestro sistema JAVA se ha utilizado la API de JAVA SAX (siglas de “Simple API for XML”), que se ha convertido en el API

estándar para el uso del XML en JAVA, ademas permite de forma sencilla verificar el formato del XML corresponda con el definido en la DTD.

Se ha elegido SAX en vez de DOM por varios motivos como la cantidad de memoria que requiere para funcionar el parser SAX es mucho mas pequeña que el parser DOM. El parser DOM tiene que contener en memoria el árbol XML entero para poder empezar a procesarlo, por lo que la cantidad de memoria usada por DOM depende del tamaño de dicho fichero XML, pero SAX solo requiere tener en memoria la profundidad máxima del fichero XML y la cantidad máxima de datos almacenadas en los atributos de un único elemento XML, y ambas son siempre mas pequeñas que el propio árbol XML.

Otro beneficio de SAX frente a DOM es que dada la naturaleza de SAX, la lectura "streaming" del disco es posible, por lo que se soporta el procesado de documentos XML que no cabrían directamente en memoria.

También tiene sus inconvenientes. Cierta validación de XML requiere acceso al documento por completo. Por ejemplo, el atributo IDREF de la DTD requiere que haya cierto elemento en el documento que use el texto dado como atributo ID de la DTD. Para poder validar esto mediante SAX se tendría que seguir cada atributo ID previamente encontrado como cada IDREF, para poder ver si se puede realizar un emparejamiento. Ademas, si no se encuentra una ID relacionada con un IDREF, solo se descubre después de haber terminado de procesar el documento y si ese enlace era importante se ha desperdiciado tiempo en procesar el XML para luego darlo como inválido.

3.4.4. AJAX

AJAX, acrónimo de Asynchronous JavaScript And XML (JavaScript y XML asíncronos), es una técnica para el desarrollo y creacion de aplicaciones web interactivas. Estas se ejecutan en el navegador del usuario y mantiene una comunicación asíncrona con el servidor. Mediante esta comunicación nuestra aplicación <e-Adventure> se comunica con el servidor LMS y viceversa. Gracias a esta tecnologia podemos realizar la comunicación de forma sencilla, segura y fiable para el intercambio de informacion con el LMS.

3.4.5. JDOM

El Document Object Model (en español "Modelo de Objetos de Documento"), frecuentemente abreviado DOM, es una forma de representar los elementos de un documento estructurado (tal como una página web HTML o un documento XML) como objetos que tienen sus propios métodos y propiedades. El responsable del DOM es el World Wide Web Consortium (W3C).

JDOM es una librería de código fuente para manipulaciones de datos XML optimizados para Java. A pesar de su similitud con DOM del consorcio World Wide Web (W3C), es una alternativa como documento para modelado de objetos que no está incluido en DOM. La principal diferencia es que mientras que DOM fue creado para ser un lenguaje neutral e inicialmente usado para manipulación de páginas HTML con JavaScript, JDOM se creó específicamente para usarse con Java y por lo tanto beneficiarse de las características de Java,

incluyendo sobrecarga de métodos, colecciones, etc. Para los programadores de Java, JDOM es una extensión más natural y correcta.

Capítulo 4

Diseño y arquitectura del sistema

El motor (o engine) esta dividido en los siguientes paquetes y subpaquetes

- es.eucm.eadventure.engine.adaptation
- es.eucm.eadventure.engine.assessment
- es.eucm.eadventure.engine.comm
- es.eucm.eadventure.engine.gamelauncher
- es.eucm.eadventure.engine.loader
- es.eucm.eadventure.engine.multimedia
- es.eucm.eadventure.engine.core
  - es.eucm.eadventure.engine.core.control
  - es.eucm.eadventure.engine.core.data
  - es.eucm.eadventure.engine.core.gui

A continuación resumimos el cometido de cada uno de dichos paquetes.

4.1. es.eucm.eadventure.engine.adaptation

Este paquete tiene como fin encargarse de la inicialización y modificación de variables de juego obtenidas a través de fuentes externas al juego. Su clase principal recibe el nombre de `AdaptationEngine`, y es la responsable de almacenar los datos de adaptación de contenido, así como de gestionar cuándo se ejecutan dichos datos de adaptación. La lectura de los archivos con información de adaptación se lleva a cabo con el método `loadAdaptationRules()` de la clase `es.eucm.eadventure.engine.loader.Loader`.

Las clases auxiliares empleadas por `AdaptationEngine` son básicamente dos:

`AdaptationRule` y `AdaptedState`.

La primera almacena reglas de adaptación, que determinan que acciones se deben tomar en caso de que el servidor de LMS envíe determinadas señales. La segunda contiene los cambios concretos que hay que efectuar sobre el estado del motor, como parte de la adaptación del contenido; esta clase se utiliza tanto para definir el estado inicial de adaptación del juego como para describir los efectos que producen la ejecución de las reglas contenidas en los objetos de tipo `AdaptationRule`.

Por último, y debido al refresco que debe de efectuar el motor de adaptación, se incluye la clase `AdaptationClock`, encargada de notificar a `AdaptationEngine` en qué momentos debe de consultar al servidor LMS para nuevos datos. Esto se lleva a cabo por mediación de la función `requestNewState()` (ubicada en la clase `AdaptationEngine`) que, a través del paquete `es.eucm.eadventure.engine.comm`, consulta los cambios que se hayan podido producir en el servidor LMS.

4.2. es.eucm.eadventure.engine.assessment

Este paquete tiene la responsabilidad de gestionar el sistema de reglas de seguimiento implantado en la ejecución del juego. La clase principal de este paquete recibe el nombre de `AssessmentEngine`, y se encarga de la carga de las reglas a través de archivos, así como de la evaluación de dichas reglas y la generación de informes. Esta última operación tiene dos funciones específicas: `generateXMLReport()`, que se encarga de generar el informe en un archivo XML, y `generateHTMLReport`, que genera dicho informe en un archivo HTML, visible desde un navegador web.

Para la carga de archivos, la clase principal hace uso del método `loadAssessmentRules()` de la clase `es.eucm.eadventure.engine.loader.Loader`, que contiene la implementación necesaria para procesar las reglas de evaluación. Cada una de estas reglas se almacena en clases del tipo `AssessmentRule`, que contienen toda la información necesaria relativa a la regla (identificador, descripción, prioridad, condiciones de disparo, efectos en caso de ejecución...). La clase `ProcessedRule` por su parte, representa las reglas que han sido disparadas en la ejecución, almacenando así la regla original y el momento en el que se disparó dicha regla.

Por último, este paquete alberga también a la clase `ReportDialog`. Esta clase es una pequeña GUI encargada de proveer al usuario de las opciones pertinentes de generación de informes. Esta clase está conectada a su vez con la clase principal, `AssessmentEngine`, para que efectúe las generaciones de informes que el usuario haya pedido.

4.3. es.eucm.eadventure.engine.comm

Este paquete se encarga de la comunicación del motor y la web en la que se está ejecutando mediante AJAX, para la comunicación entre el cliente y el servidor LMS. Consta principalmente de una interfaz llamada `AsynchronousCommunicationApi`, que define una serie de operaciones de comunicación asíncrona. Dicha interfaz se implementará en una clase encargada de llevar a cabo la comunicación real entre el servidor LMS y la aplicación.

4.4. es.eucm.eadventure.engine.gamelauncher

Este paquete recibe el cometido de mostrar un pequeño menú de selección de aventura, que se muestra cuando la ejecución del motor se lleva a cabo de forma offline. La clase principal de dicho paquete es `GameLauncher`, la cual se encarga de mostrar el interfáz gráfico de selección y de llevar a cabo las operaciones de lectura de aventuras, así como de arrancar el bucle principal de juego, una vez elegida la aventura.

Dentro de este paquete se encuentra otro, llamado `gameentry`, encargado de almacenar las clases relativas a las aventuras que hay disponibles, visibles en el cargador. La clase `GameEntry` se encarga de almacenar una información esquematizada sobre las aventuras que contienen cada uno de los archivos ZIP de la localización seleccionada. La clase `GameEntryHandler` por otro lado se encarga de llevar a cabo la lectura de las cabeceras de las aventuras, para extraer la información antes mencionada. Cabe destacar que `GameEntryHandler` hereda de `DefaultHandler`, con lo que su funcionamiento se basa en el uso de SAX.

4.5. es.eucm.eadventure.engine.loader

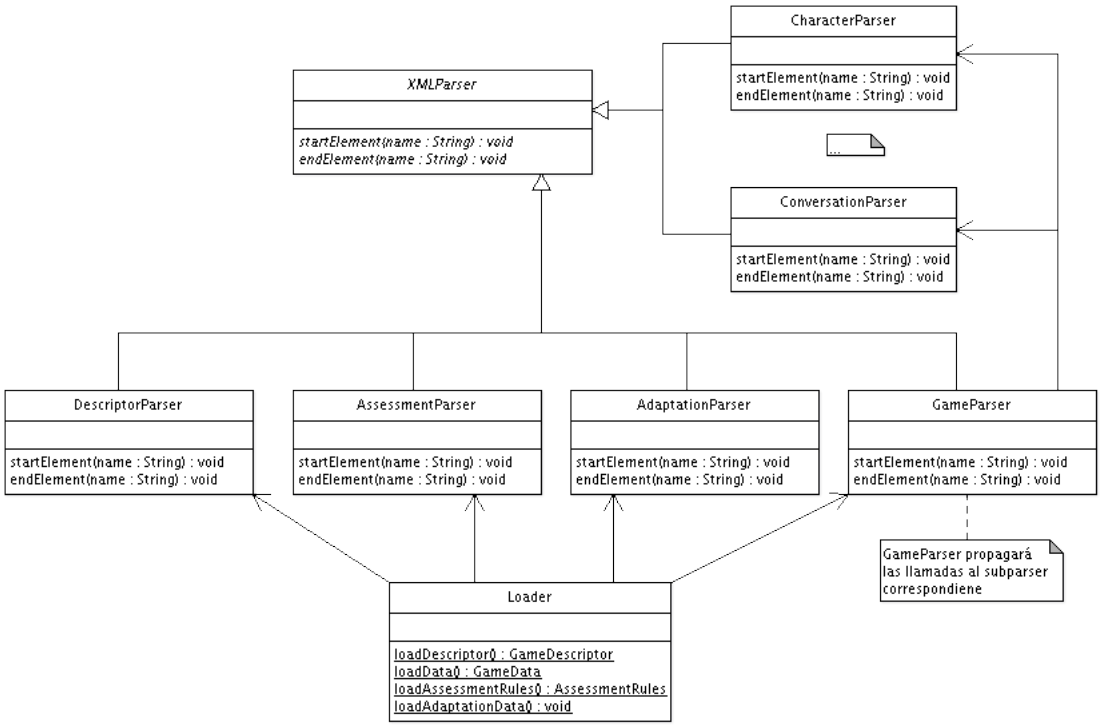
Este paquete se encarga de verificar y leer las aventuras descritas en XML. Una vez verificada la correcta sintaxis del XML que representa la aventura, procede a transformarlo a datos internos que el motor usará para representar dicha aventura.

La clase `Loader` proporciona, para cada tipo de fichero XML a analizar, un método estático que devuelve una clase con todos los datos contenidos en el fichero. El análisis de los ficheros XML se realiza de un modo basado en eventos, notificando al analizador cada vez que se abre o se cierra una nueva marca, de modo que pueda procesar todos sus datos asociados. Si la marca a procesar es demasiado compleja, o si se repite a lo largo del lenguaje <e-Adventure>, el analizador principal puede delegar el procesado de la marca en un subanalizador. Por ejemplo, el analizador principal del fichero correspondiente a un capítulo de la aventura procesará los datos generales para la marca <e-Adventure>, mientras que el análisis de escenas, personajes, conversaciones, etc, será responsabilidad de analizadores específicos. Podemos verlo más claro en el cuadro 4.1

4.6. es.eucm.eadventure.engine.multimedia

Debido a los diferentes métodos que JAVA nos ofrece para leer y cargar archivos multimedia (entre otros imágenes y sonidos), este paquete se encarga de abstraer esas diferentes formas a una única que usa el motor. Se encarga además de su almacenamiento y gestión de tanto imágenes como sonidos mediante una caché, haciendo así que el uso de una misma imagen y/o sonido en múltiples zonas de las aventuras (que es muy común) sea eficiente, sencilla y rápida al evitar el procesamiento y lectura de una nueva carga.

También gracias a esta abstracción de la gestión y carga de archivos multimedia, se nos permite (de forma hipotética y en un futuro) modificar esta carga y/o gestión de dichos

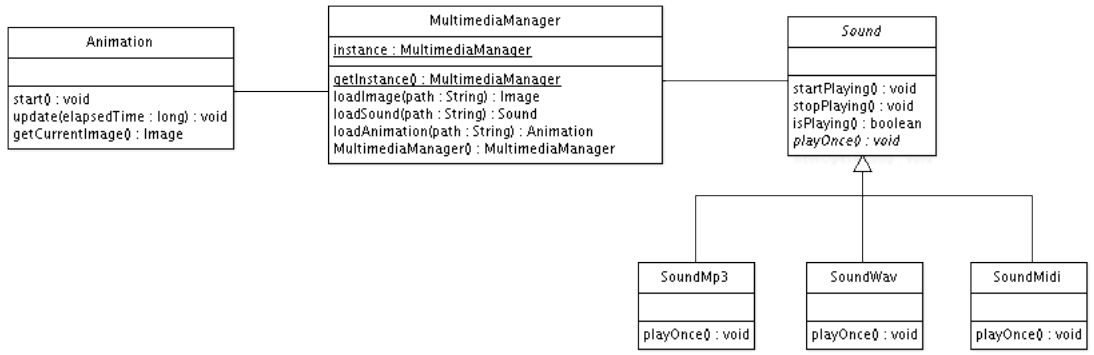


Cuadro 4.1: Paquete es.eucm.eadventure.engine.loader

archivos multimedia por versiones mejoradas o mediante el uso de paquetes de terceros, y que el motor no sufra modificación alguna, permitiendo mantener el núcleo del motor pero estar siempre actualizado en cuando a gestión y reproducción multimedia.

No hace falta decir que una de las partes más delicadas del motor <e-Adventure> es la gestión de recursos multimedia. Necesitamos un medio sencillo para gestionar varios tipos de formatos de imágenes, sonidos y animaciones, de la manera más uniforme posible, y flexible en cuanto a posibles futuros cambios.

Como podemos ver en el cuadro 4.2, hemos elegido centralizar toda la gestión de recursos multimedia a una clase que hace la función de fachada frente a los distintos formatos de recursos soportados. Se puede ver claramente lo relativamente sencillo que sería añadir soporte para, por ejemplo, archivos de sonido en formato ogg. Tendríamos que añadir una clase `SoundOgg`, que extendiese a `Sound`, e implementar el método `playOnce()`. Dicho método es un *Template Method* (método plantilla) con el que `Sound` se encarga de la reproducción de sonidos, ya sea una sóla reproducción (efectos de sonido) o en un bucle, hasta que se solicite explícitamente que finalice la reproducción (música de fondo). Luego, en la llamada a `getInstance` de `MultimediaManager`, crearíamos un sonido de tipo SoundOgg, si la ruta al sonido finalizase en ".ogg". No habría que realizar cambio alguno en el motor, que cargaría la ruta al sonido desde el archivo XML como hasta ahora y lo reproduciría con la misma llamada a `MultimediaManager`. De este modo, desacoplamos totalmente el motor frente a diferentes tecnologías de reproducción de recursos multimedia.



Cuadro 4.2: Paquete es.eucm.eadventure.engine.multimedia

4.7. es.eucm.eadventure.engine.core

El paquete `es.eucm.eadventure.engine.core` está dividido en los siguientes paquetes: `data`, `gui` y `control`. Como sus nombres indican, estos paquetes realizan la función de *modelo*, *vista* y *controlador* del patrón Modelo Vista Controlador (*MVC*).

En primer lugar, el paquete `es.eucm.eadventure.engine.core.data` es básicamente una representación de los datos contenidos en los distintos ficheros XML que forman cada aventura. El paquete `es.eucm.eadventure.engine.core.loader` vuelca en las clases que forman parte de este paquete toda la información necesaria para la ejecución del motor, y a partir de entonces están disponibles en todo momento.

En segundo lugar, el paquete `es.eucm.eadventure.engine.core.gui` contiene las clases necesarias para crear la interfaz de usuario. Se encarga de crear una ventana en la que pintar el estado del juego en cada momento, así como de recibir los eventos del jugador.

Por último, el paquete `es.eucm.eadventure.engine.core.control` se encarga de llevar a cabo el control del motor, tanto de la ejecución del bucle principal de juego como de añadir comportamiento a las clases del paquete `es.eucm.eadventure.engine.core.data`.

4.7.1. es.eucm.eadventure.engine.core.data

Como ya hemos comentado, el paquete `es.eucm.eadventure.engine.core.data` se limita a contener los datos almacenados en los ficheros XML que componen cada aventura, por lo que no es necesario comentarlo con más detenimiento.

4.7.2. es.eucm.eadventure.engine.core.gui

La GUI (Graphical User Interface, Interfaz Gráfica de Usuario) tiene la función de mostrar el estado actual del motor, así como de recibir eventos por parte del jugador y notificarlos al controlador del motor. Ofrece funciones para el pintado de gráficos y texto, y se encarga de registrarse como oyente de eventos externos de teclado y ratón. Además, define el comportamiento para los distintos tipos de HUD (Head-Up Display), es decir, tanto el tradicional como el contextual. Cada tipo de HUD tiene dos elementos diferenciados: los botones para

realizar acciones (hablar, examinar, ...) y el inventario, donde podemos interactuar con los elementos que hemos recogido o nos han dado a lo largo de la partida.

En el HUD tradicional, tanto el inventario como los botones de acción se mostrarán en todo momento en la parte inferior de la pantalla.

En el HUD contextual, el escenario ocupará la totalidad de la pantalla. El inventario sólo aparecerá cuando el usuario sitúe el puntero del ratón en el borde superior o inferior de la pantalla. Los botones de acción se mostrarán al hacer click con el botón derecho del ratón sobre un personaje u objeto con el que podamos interactuar.

4.7.3. es.eucm.eadventure.engine.core.control

Este paquete es, con diferencia, el más complejo de los tres que constituyen el núcleo del motor <e-Adventure>. Las partes que lo constituyen son las siguientes:

es.eucm.eadventure.engine.core.control.Game

La clase `Game` es la clase principal del control del juego. Sus responsabilidades son varias: cargar cada capítulo de la aventura, realizar las acciones del jugador sobre el estado actual del motor, ejecutar el bucle principal del juego, cargar y salvar el estado de la partida, recibir las notificaciones de eventos externos por parte de la GUI, y ejercer de nexo entre distintas partes del motor.

es.eucm.eadventure.engine.core.control.animations

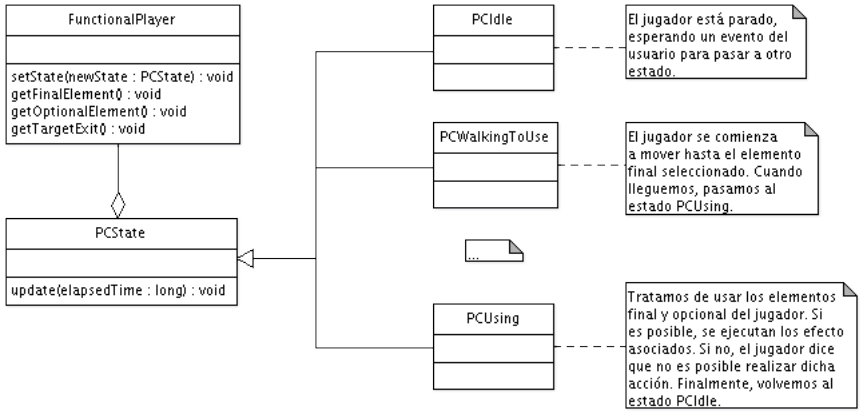
Para controlar el estado actual tanto del jugador principal como del resto de personajes, hemos optado por implementar un patrón *state*, de modo que podamos definir el comportamiento de cada estado (andando, hablando, parado, ...) en una clase distinta e independiente a las demás, así como poder pasar de un estado a otro de un modo sencillo. Por otro lado, desacoplamos al motor de los distintos estados en los que puedan estar los personajes, y facilitamos la tarea de añadir nuevos estados en un futuro.

Para ello, el jugador (y del mismo modo, los personajes) definen su comportamiento en clases que extienden la clase abstracta `PCState`. Esta clase controla la posición, velocidad y dirección de movimiento del jugador, y además mantiene una referencia a la clase del propio jugador, para facilitar la gestión de los efectos asociados a las acciones del usuario. Por ejemplo, necesitamos saber qué dos objetos están siendo combinados a la hora de lanzar el efecto correspondiente.

Para tener una idea más clara de como funciona el control de estados para las animaciones de los personajes, supongamos por ejemplo que el jugador lanza la acción necesaria para hablar con otro personaje (recordemos que la forma de lanzar una acción depende de la interfaz gráfica que estemos utilizando, contextual o tradicional). Tanto el jugador como el personaje se encuentran inicialmente en el estado *Parado* (*PDIdle* - *NPCIdle*). Justo después de lanzar la acción, el jugador pasa al estado *AndandoParaHablar* (*PCWalkingToTalk*). En este estado, el jugador comienza a avanzar hacia el personaje con el que va a entablar

una conversación. Cuando por fin ha llegado a la altura de dicho personaje, se comprueba si hay alguna conversación disponible con el personaje con el que tratamos de hablar. Si la hay, cambiamos el estado de juego al estado *Conversación* (hablaremos más sobre los estados de juego más adelante). Durante la conversación, ambos jugadores irán alternando sus estados *Parado* (*PDIdle* - *NPCIdle*) y Hablando ((*PDTalking* - *NPCTalking*). Al finalizar la conversación, ambos personajes volverán a su estado *Parado*, y podremos seguir esperando nuevos eventos del jugador para continuar la partida.

Podemos ver un diagrama simplificado de este patrón estado en el cuadro 4.3



Cuadro 4.3: Paquete es.eucm.eadventure.engine.core.control.animations.pc

4.7.4. es.eucm.eadventure.engine.core.control.functionaldata

Dentro del paquete

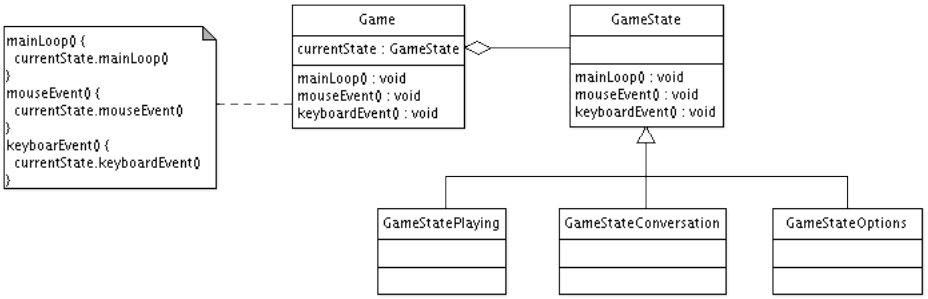
`es.eucm.eadventure.engine.core.control.functionaldata`

se encuentran las clases que dotan de un comportamiento a las clases del paquete `es.eucm.eadventure.engine.core.data`, que recordemos son las clases que contienen los datos cargados desde los ficheros XML. En estas clases definimos, por ejemplo, cómo combinar varios objetos (en `FunctionalItem.useWith( FuntionalItem otherItem )`, cómo cargar y actualizar los recursos gráficos y sonoros de una escena (en `FuncionalScene`), o el comportamiento citado anteriormente de jugador y personajes (`FunctionalPlayer` y `FunctinoalNPC`, respectivamente).

4.7.5. es.eucm.eadventure.engine.core.control.gamestate

Del mismo modo que un personaje pasa por distintos estados, con comportamientos muy dispares, a lo largo del desarrollo de una partida, no más parecidos son los comportamientos de los distintos tipos de escenas. Por lo tanto hemos optado de nuevo por hacer uso del patrón *state* para definir el comportamiento de los diferentes tipos de escenas (*BookScenes*, *SlideScenes*, escenas interactivas, etc.). Las clases relativas al estado del juego se encuentran en el paquete `es.eucm.eadventure.engine.core.control.gamestate`.

Como ya dijimos anteriormente, la clase `Game` es la encargada de ejecutar el bucle principal de juego. En realidad, esta responsabilidad se reduce a una llamada a `mainLoop()` de el estado de juego actual. Del mismo modo, se delega el tratamiento de eventos en dicho estado actual. Podemos ver un pequeño resumen de esto en el cuadro 4.4.



Cuadro 4.4: Paquete es.eucm.eadventure.engine.core.control.gamestate

Capítulo 5

Implementación

En este capítulo describimos cómo hemos pasado del análisis y diseño iniciales a la construcción del sistema. Se incluyen además en esta sección algunos detalles sobre la implementación del sistema, así como los cambios y modificaciones que ha sufrido el motor a lo largo del desarrollo.

Podemos ver varios extractos de código fuente en el apéndice B, en la página 123.

5.1. Metodología de programación

En esta sección se detallan cuestiones relativas a la codificación de la aplicación, incluyendo datos sobre el entorno de desarrollo utilizado, los estándares de código empleados y el uso de repositorios.

5.1.1. Entorno de desarrollo

El entorno de desarrollo elegido ha sido Eclipse. Las razones son varias:

Es plataforma independiente Esto es muy útil ya que no todos los miembros del equipo de desarrollo trabajaban sobre la misma plataforma. También ayuda a asegurar que el sistema construido sea también independiente de la plataforma.

Se integra bien con Java Las herramientas de ejecución, depuración, resaltado de sintaxis, gestión de paquetes, etc., agilizan enormemente el trabajo al manejar un proyecto de la envergadura de <e-Adventure>.

Se integra bien con CVS La herramienta utilizada durante casi todo el desarrollo para el contro de cambios ha sido el CVS de SourceForge. Eclipse proporciona una interfaz para gestionar los cambios desde el CVS, lo cual también ayuda a que el trabajo sea más cómodo y eficaz.

5.1.2. Estilo de código

Para facilitar el trabajo en equipo es necesario usar un estilo de código estándar, ya que alguien que trabaje con un código que no ha escrito y que no está acostumbrado a su notación, puede que no se encuentre a gusto. Tampoco hay que olvidar que <e-Adventure> es un proyecto que acaba de nacer, y que su desarrollo continuará durante varios años por distintos programadores. Mediante el uso de un buen estilo de código, este futuro trabajo será mucho más rápido y eficaz.

El estilo de código elegido, basado en el descrito en http://java.sun.com/docs/codeconv/, se detalla a continuación:

Idioma de codificación y documentación El idioma elegido tanto para la codifificación como para la documentación de código de <e-Adventure> es el inglés.

Nombres de ficheros Los ficheros de código fuente terminan en .java, mientras que los compilados terminan en .class. Por supuesto, el nombre del fichero (sin extensión) es el nombre de la clase pública que definen.

Declaraciones de paquete e importaciones La primera línea que no es un comentario en un fichero de código .java define el paquete al que pertenece dicho fichero. A continuación están las declaraciones de paquetes importados por ser dependencias de la(s) clase(s) definidas en el fichero de código fuente.

Declaraciones de clase e interfaz El orden de declaraciones dentro de un fichero de código fuente es el siguiente:

1. Documentación de la clase o interfaz
2. Declaración de la clase o interfaz
3. Variables estáticas públicas
4. Variables protegidas
5. Variables privadas
6. Constructores
7. Métodos
8. Clases auxiliares

Comentarios En el encabezado de cada clase y método púbico (y algunos privados, si es necesario) se escribirá *qué* hace, mientras que durante su implementación, se escribirán comentarios que hagan posible entender *cómo* lo hace. Los primeros se harán en un formato compatible de javadoc (es decir, del tipo `/** documentación **/`, para generar posteriormente la documentación del código en dicho formato. Los últimos se harán usando `// comentario` para comentarios de una sóla línea, y `/* comentario */` para comentarios multilínea.

Implementación de clases y métodos La llave de apertura ({) de un bloque de código debe ir a continuación de su declaración. La llave de cierre (}) irá en una nueva línea, alineado con la declaración, a no ser que el bloque sea vacío, en cuyo caso irá justo después de la llave de apertura. Ejemplo:

```
public class MyClass extends MyClassParent implements MyClassInterface {
  private int myValue;
  public MyClass( int myValue ) {
    this.myValue = myValue;
  }
  public int myMethod() {
    return myValue;
  }
  public void myVoidMethod() { }
}
```

Estructuras de control La codificación de los bloques de estructuras de control, en caso de componerse éstos de más de una instrucción, será como la implementación de métodos anteriormente descrita. En el caso de las estructuras `if then else`, `do while` y `try catch`, tanto los `else` como los `while` y los `catch` irán en la misma línea que la llave de cierre (}) del bloque anterior. Ejemplos:

```
if( condition ) {
    statements;
} else if( condition ) {
    statements;
} else {
    statements;
}

for( initialization; condition; update ) {
    statements;
}

while( condition ) {
    statements;
}

do {
    statements;
} while( condition );

switch( condition ) {
    case A:
        statements;
        break;
    case B:
    case C:
        statements;
        break;
    default:
        statements;
```

```
        break;
}

try {
    statements;
} catch( Exception e ) {
    statements;
}
```

Convenciones de nombres A continuación se explican unas sencillas reglas para nombrar distintos elementos del lenguaje Java durante la implementación:

- Paquetes: Los nombres de todos los paquetes irán en minúscula. Los paquetes de <e-Adventure> irán dentro de la jerarquía
  `es.eucm.eadventure.*`.
- Clases e interfaces: Los nombres de las clases y las interfaces serán una serie de palabras, con la primera letra de cada palabra en mayúsculas. Ejemplo:
  `EAdventureApplet`.
- Métodos: Los métodos serán una serie de palabras, con la primera letra de cada palabra a partir de la segunda en mayúscula. La primera palabra irá toda en minúscula. Ejemplos:
  - `getHeight()`
  - `setName( String name )`
  - `mainLoop()`
- Variables: Los nombres de las variables serán como los nombres de los métodos, es decir, la primera palabra en minúscula, el resto con la primera letra de cada palabra en mayúsculas. Ejemplo:
  `walkingAnimation`
- Constantes: Los nombres de las constantes serán una serie de palabras en mayúscula, separadas por guiones bajos (_). Ejemplo:
  `public static final int WINDOW_WIDTH = 800`

5.1.3. Repositorios

Para el desarrollo del proyecto <e-Adventure>, se decidió hacer uso de repositorios desde un primer momento. Dichos repositorios consisten en localizaciones, accedidas normalmente a través de internet, que permiten el almacenamiento y catalogación de archivos. Gracias al uso de un repositorio es posible conseguir que el equipo pueda trabajar desde distintas localizaciones a la vez, además de permitir que todos los integrantes dispongan de la última versión de los archivos en cada momento. También el repositorio permirte almacenar todos los cambios realizados y todas la versiónes realizadas sobre el código fuente, permitiendo volver a ellas en cualquier momento y recuperarlas. Todos estos cambios se pueden seguir mediante los historiales asociados a cada archivo.

Aunque la principal utilidad del repositorio ha sido el código fuente del proyecto, cabe destacar que también se ha utilizado para el almacenamiento de otros archivos, como ha sido la información anexa al proyecto (aventuras, recursos gráficos...) y la documentación.

En cuanto al tipo de repositorio escogido, se bajaron dos posibilidades: CVS y SVN. La primera había sido utilizada anteriormente por el equipo de desarrollo, pero requería de servidores de CVS externos. Por otro lado, SVN había dado buenos resultados a los directores del proyecto, y podía ser desplegado facilmente en uno de los ordenadores de la Facultad de Informática, disponiendo de acceso físico al equipo.

Finalmente se optó por utilizar SVN, desplegando el servidor y facilitando cuentas al equipo de desarrollo. Sin embargo, problemas con los *plugins* de Eclipse encargados de gestionar el uso del repositorio hicieron que finalmente se cambiara el repositorio SVN por uno de tipo CVS. Dicho repositorio fue facilitado finalmente por SourceForge, central de desarrollos de software libre; la dirección de la página del proyecto en SourceForge y la del repositorio se dan a continuación:

- **Página del proyecto** http://sourceforge.net/projects/e-game/
- **Repositorio del proyecto** e-game.cvs.sourceforge.net

5.1.4. Página web del proyecto y wiki de desarrollo

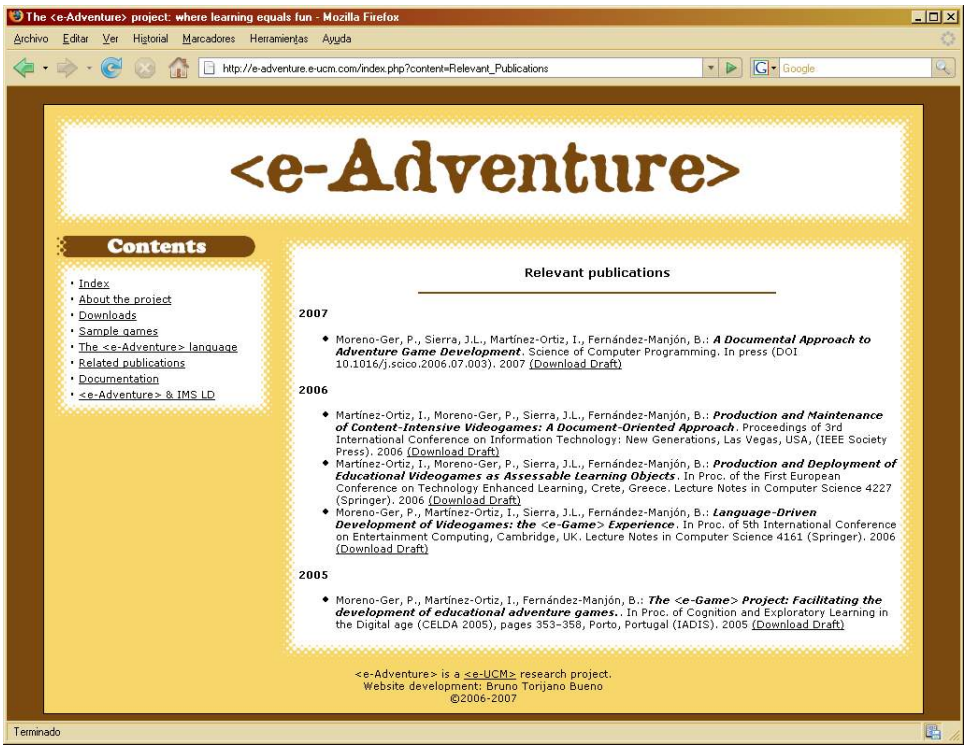
Además de lo mencionado anteriormente se utilizaron distintas tecnologías de soluciones web como apoyo al proyecto. Entre dichos usos destacan dos: la página web oficial del proyecto y el wiki de desarrollo. A continuación se ofrecen las direcciones de ambas páginas.

- **Página del proyecto** http://e-adventure.e-ucm.com/
- **Wiki de desarrollo** http://e-adventure.e-ucm.com/wiki/

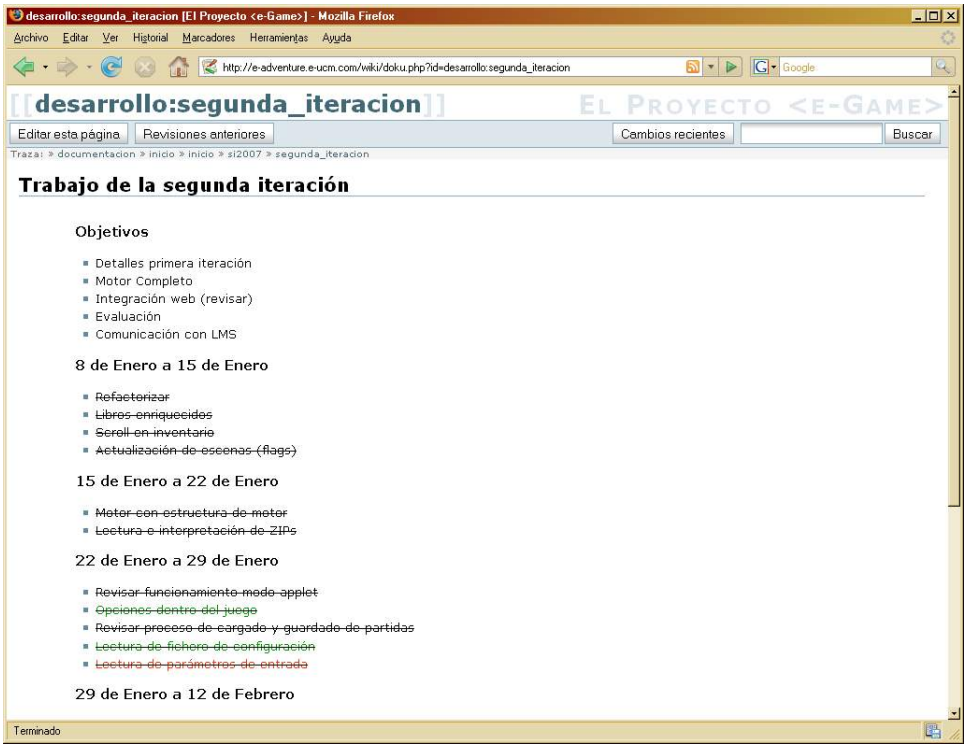
La página web del proyecto se llevó a cabo por el equipo de desarrollo con el fín de promocionar el proyecto <e-Adventure>. Dicha página cuenta con información sobre el proyecto y la base sobre la que se sustenta. Dispone además de apartados con referencias de documentación sobre el proyecto, así como enlaces a descargas del motor y diversas aventuras desarrolladas. En la figura 5.1 se puede ver una captura de la página del proyecto.

El wiki de desarrollo tuvo el fín de facilitar la comunicación y el seguimiento de la planificación. Puesta en marcha por Pablo Moreno Ger, el wiki se concibió con el fín de almacenar las fechas y fines de cada una de las planificaciones y microhitos. El equipo de desarrollo tenía la responsabilidad de marcar como implementadas aquellas tareas que ya habían sido realizadas, además de actualizar el wiki con la planificación que se acordaba en cada una de las reuniones. En la figura 5.2 se puede ver una captura del wiki de desarrollo.

Hay que destacar que el alojamiento para ambas páginas ha sido posible gracias al amparo del grupo de investigación <e-UCM>, dedicado a la investigación de tecnologías web y de marcado con fines educativos. Para más información sobre este grupo de investigación se recomienda visitar su página web, ubicada en **http://www.e-ucm.com/**.



Cuadro 5.1: Página web del proyecto



Cuadro 5.2: Página web del wiki de desarrollo

5.2. Evolución del motor <e-Adventure> y su lenguaje

En este apartado se detallan los cambios que ha sufrido el motor durante su desarrollo. Es importante destacar que las especificaciónes iniciales del motor y su lenguaje se antojaron insuficientes en algunos puntos del desarrollo, con lo que se acordaron cambios y nuevas funcionalidades a añadir.

A continuación se detallan las características que se propusieron a mejora, y que finalmente fueron llevadas a cabo. Dichos cambios se estructuran en dos apartados, uno detallando los elementos que sufrieron modificaciones a lo largo del desarrollo, y otro detallando los elementos que se incorporaron partiendo de cero.

5.2.1. Elementos y comportamientos añadidos

En esta sección se contemplan las funcionalidades y elementos que se añadieron a la especificación de <e-Adventure>. Se indica en qué iteración se produjo cada uno de los cambios.

Slidescenes

Elemento añadido en la **Primera iteración**.

A lo largo del desarrollo, y en vista de los problemas que estábamos encontrando con la inclusión de videos se llegó a la conclusión de que era necesario añadir algún otro tipo de elemento narrativo sin interactividad que sirviera para narrar acontecimientos en la historia. Tras discutir el asunto, surgió la idea de incluir escenas de diapositivas (como en una presentación) que el jugador pudiese visualizar como si de un video se tratase, usando el ratón para pasar de una diapositiva a la siguiente.

La estructura de las cutscenes por diapositivas se copió en su totalidad de las escenas de vídeo, y el desarrollo de ambas a posteriori fue simétrico. Por ello, de ahora en adelante, para los cambios menores, nos referiremos a ambas secuencias como *cutscenes* en general.

Libros

Elemento añadido en la **Primera iteración**.

Por sugerencia de Pablo Moreno Ger acabamos incluyendo un tipo especial de representación de información para el jugador, los libros. Estos consisten en una estructura sencilla notada en XML, con un núcleo importante de texto escrito, que se pinta por pantalla para que el jugador pueda leerlo. Dentro de la estructura se define una imagen de fondo que se coloca para simular la apariencia de un libro, y el texto se corta y renderiza en pantalla a medida que el jugador avanza o retrocede por las páginas.

La inclusión de los libros propició además la adición de una nueva marca de efecto, que se pasó a llamar <`trigger-book`>. De esta forma es posible disparar la aparición de libros al mantener conversaciones o interaccionar con objetos.

Descriptores de aventuras

Elemento añadido en la **Tercera iteración**.

Se ha incluido como modificación del cargador un sistema de capítulos para las aventuras. De esta forma se han suprimido las marcas <title> y <story> de la raíz de la aventura, para colocarse en un nuevo archivo llamado *descriptor.xml* que debe contener toda aventura.

En este archivo también se añade información sobre el tipo de GUI de la aventura, asi como los archivos de adaptación y evaluación para cada capítulo de la aventura.

Posición de pintado y ordenación de elementos

Comportamiento añadido en la **Tercera iteración**.

Se ha variado la posición de pintado para los elementos de las escenas. Ahora, las coordenadas de posición describen el pie de los personajes u objetos.

Adicionalmente, en el pintado los elementos se ordenan por profundidad del pie, de forma que los objetos que más altos se encuentran en la escena son los que estarían mas al fondo.

Post efectos en <next-scene>

Comportamiento añadido en la **Tercera iteración**.

Como complemento al conjunto de efectos <effect> que se ejecuta en la marca <next-scene> antes de cambiar de escena, se ha incluido otra estructura llamada <post-effect> que se ejecuta una vez se ha cambiado la escena.

De esta forma permitimos incluir conjuntos de condiciones en la entrada de escenas, potenciando las secuencias no interactivas en <e-Adventure>.

Nuevos tipos de efectos

Elemento añadido en la **Tercera iteración**.

Se han incluido un total de cinco nuevos efectos, que potencian principalmente las secuencias no interactivas. A continuación se enumeran dichos efectos, junto con su forma de uso:

- <play-sound> permite la reproducción de un sonido. Incluye un atributo que indica si el sonido bloquea el flujo de eventos en ejecución o no.
- <play-animation> permite la reproducción de una animación en un punto dado del escenario.
- <move-player> hace que el jugador se desplace a la posición del escenario indicada por esta marca.

- <move-npc> hace que el personaje seleccionado se desplace a la posición del escenario indicada por esta marca.
- <trigger-scene> ejecuta un cambio de escena. Esto transporta al jugador a la nueva escena en el punto indicado, si se ha especificado uno en la marca.

5.2.2. Elementos y comportamientos modificados

En esta sección se contemplan las funcionalidades y elementos que estaban contenidas en la especificación de <e-Adventure> pero fueron objeto de modificación. Se indica en qué iteración se produjo cada uno de los cambios.

Puntos de inserción del personaje

Comportamiento modificado en la **Primera iteración**.

Uno de los problemas que vimos al mover al personaje protagonista entre escenas era la posición en la que este aparecía. De esta forma dictaminamos dos mecanismos para ayudar a especifica la posición en la que debía de aparecer el protagonista al llegar a una escena determinada.

En primer lugar, contemplamos el caso de que el personaje pasara de una escena a otra. De esta forma es lógico pensar que el personaje aparecería en la nueva escena lo más próximo a la salida que le condujese a la escena de la que había venido. De esta manera respetábamos la continuidad de la posición del personaje. Tras algunas discusiones, ampliamos la funcionalidad de la marca <next-scene> con dos atributos opcionales llamados x e y. Estos dos atributos se pueden colocar en una marca <next-scene> para indicar la posición que debería ocupar el personaje protagonista en la nueva escena.

En segundo lugar, cabía la posibilidad de que quisiésemos colocar un punto de inserción por defecto en los escenarios, de forma que si no se especificaba una posición concreta de transición, el personaje apareciese en un lugar razonable. Dado que no podemos preveer las zonas que serán transitables y las que no en un escenario cualquiera, lo más sensato era añadir un punto de inserción por defecto en las escenas interactivas. De esta forma se añadirá la marca opcional <default-initial-position>, con atributos x e y obligatorios que indican el punto concreto en el que debe colocarse el personaje en caso de que no se haye información en la marca <next-scene> que ha conducido al personaje a esta nueva ubicación.

Nombre en escenas y cutscenes

Elemento modificado en la **Primera iteración**.

Otro problema que detectamos al poco de comenzar a trabajar con el prototipo, fue la ausencia de nombre para las distintas escenas y cutscenes. Esto se antojaba un problema al colocar el cursor del ratón sobre una salida. En cualquier aventura gráfica se suele colocar el nombre de la estancia a la que conduce dicha salida, pero no disponíamos de nombre para mostrar.

Decidimos entonces añadir una marca <name> a las escenas y a las cutscenes de forma obligatoria, de modo que toda escena tiene un nombre que puede visualizarse en caso de que se coloque el ratón sobre una salida que conduzca a ella.

Comportamiento del fin de las cutscenes

Comportamiento modificado en la **Primera iteración**.

Una vez implementadas las cutscenes, pudimos ver que el sistema que se utilizaba para decidir si se finalizaba el juego no resultaba del todo correcto. En la primera versión del lenguaje la ausencia de marcas <next-scene> indicaba fin de juego. Tras algunas discusiones llegamos a la conclusión de que era mas provechoso contemplar tres posibilidades para la finalización de la ejecución de una cutscene:

- Vuelta al juego: en algunas condiciones nos interesaba poder volver de nuevo a la partida tras mostrar una pequeña cutscene. Supongamos que al examinar un objeto este objeto se rompe, mostrando dicho efecto con un video. Al terminar el video es necesario que la acción siga transcurriendo en el mismo sitio en el que se llevó a cabo la acción, con lo que es necesario no cambiar de escenario (y por lo tanto no incluir marca <next-scene>). Se terminó decidiendo que la vuelta al juego se produciría en ausencia de marca al finalizar la estructura de la cutscene.
- Paso a otra escena: en otras ocasiones, o por regla general, nos interesa que despues de mostrar una cutscene se nos conduzca a una nueva área de juego. Para ello se decidió que ha de incluirse una o varias marcas <next-scene> para garantizar el salto a otro escenario.
- Fin de juego: por último, otro efecto deseado es el fin de juego. Hay que tener en cuenta además que la única forma de terminar una aventura <e-Adventure>es por medio de una cutscene. De esta forma creamos una nueva marca vacía llamada <end-game> que indica al motor que el juego debe finalizar una vez ha terminado la cutscene.

Orden de ejecución en caso de conflicto de condiciones

Comportamiento modificado en la **Primera iteración**.

Tras algunas pruebas, pudimos ver que no había una política a seguir en caso de que varias acciones o saltos de escena cumpliesen sus condiciones para dispararse. Tras comentar el asunto, llegamos a la conclusión de que un sistema ordenado era lo más conveniente. De esta forma, si en algún momento el motor se encuentra con varias estructuras condicionadas, se elegirá para ejecutar la primera de ellas que cumpla sus condiciones, en orden de lectura.

Acción examinar

Comportamiento modificado en la **Primera iteración**.

Tras analizar el funcionamiento de varias aventuras gráficas, llegamos a la conclusión de que era provechoso añadir una nueva acción a los objetos en el juego, la de examinar. Si bien

antes de este cambio la acción examinar existía, provocaba en todo caso que el protagonista recitase la descripción detallada con que contaba el objeto.

En cambio, con la nueva marca, es posible desencadenar efectos al examinar un objeto concreto; por el contrario si el objeto no dispone de acciones de examinar, o ninguna de ellas cumple con sus condiciones, el protagonista recitará la descripción del objeto. Esta nueva marca se llamó <examine> y tiene la misma estructura que las otras tres marcas de accion de un objeto.

Desactivación de flags

Elemento modificado en la **Primera iteración**.

Aunque en un principio no estaba contemplado como efecto desactivar un flag, vimos que en algunos casos concretos era conveniente el poder desactivarlos para poder llevar un mejor control del flujo del juego. Por ejemplo, en un puzzle que se deba construir siguiendo unos pasos en un orden concreto, resulta más facil ir activando flags si la secuencia es adecuada, para desactivarlos todos si alguno de los pasos es erróneo (con lo que habría que volver a comenzar desde el principio). Por lo tanto acabamos incluyendo una nueva marca de efecto llamada <deactivate>, análoga en estructura a <activate>, pero con un resultado en ejecución opuesto.

Marca obligatoria de <resources>

Elemento modificado en la **Segunda iteración**.

Durante la primera iteración, todos los elementos con recursos multimedia podían contar o no con un bloque <resources> que recogiese dichos recursos. En esta iteración hemos modificado la DTD para obligar a que estos elementos deban contener al menos UN bloque <resources>, de forma que siempre tengan definidos recursos multimedia para su ejecución.

Eliminación de <instance> en los personajes

Elemento modificado en la **Segunda iteración**.

Aunque la marca <instance> no está realmente soportada de momento en el lenguaje, hemos optado por retirarla de los personajes (marca <character>) dado que su funcionamiento no lo exigía. De esta forma si queremos poner un personaje en varias escenas, no tenemos más que añadir las referencias de manera normal, y el motor se encargará de instanciar los personajes que deba.

Libros enriquecidos

Elemento modificado en la **Segunda iteración**.

Como ampliación a los libros implementados en la primera iteración, convenimos en mejorarlos añadiendo nuevas marcas para describir elementos especiales, como enumeraciones

e imágenes. Dichos cambios han sido finalmente implementados como nuevas marcas que se pueden insertar dentro de la marca <text> en los libros. De esta manera se pueden redactar libros más vistosos gráficamente.

Marca <exits> de carácter opcional

Elemento modificado en la **Tercera iteración**.

La marca <exits>, perteneciente a las escenas, se ha notado como opcional. Esto es debido a que con los nuevos efectos es posible cambiar de escena sin necesidad de una salida. En cualquier caso, la inclusión de la marca <exits> implica siempre que la escena debe tener al menos una salida.

5.3. Del diseño a la implementación

En este apartado vamos a resumir el proceso de transformación del diseño estudiado anteriormente a la implementación final.

En general, podemos destacar que la implementación se ha desarrollado íntegramente como un proyecto Java (dos proyectos, si contamos el editor de aventuras) usando eclipse como entorno de desarrollo y CVS como herramienta de control de cambios.

5.3.1. es.eucm.eadventure.engine.resourcehandler

Para comenzar, vamos a comentar la función del paquete
`es.eucm.eadventure.engine.resourcehandler`,
que no apareció en el apartado de diseño pero fue necesario de cara a la implementación. Dicho paquete se ha creado debido a las restricciones y diferencias para leer archivos cuando el motor se está ejecutando en forma de applet y cuando lo está haciendo de forma independiente. Al igual que con el paquete Multimedia, se ha abstraído esta funcionalidad para poder gestionar de forma diferente la lectura de ficheros según la forma de ejecución pero haciéndolo de forma transparente al motor.

5.3.2. es.eucm.eadventure.engine.loader

Como comentamos en el apartado de diseño, el cargador de ficheros XML basaba su funcionamiento en evenentos, tanto al comienzo como al final de cada marca procesada. Con esto en mente, hemos optado por utilizar SAX como herramienta de apoyo para analizar los distintos ficheros XML necesarios para el funcionamiento del motor <e-Adventure>. De este modo, cada uno de nuestros analizadores extienden de la clase
`org.xml.sax.helpers.DefaultHandler`,
de modo que son notificados automáticamente de las aperturas y cierres de cada marca contenida en el fichero XML, con los métodos

```
public void startElement(
```

```
      String namespaceURI, String sName, String qName, Attributes attrs
   ) y
  public void endElement(
      String namespaceURI, String localName, String qName
   ),
```

respectimente. De este modo, la función de cada analizador se limita a extraer los datos sobre la marca que acaba de comenzar o finalizar, que se reciben como parámetro en cada uno de los métodos, así como de crear los subanalizadores correspondientes y propagar a ellos las llamadas cuando fuese necesario.

5.3.3. es.eucm.eadventure.engine.multimedia

Para reproducir ficheros de sonido en formato MP3 y WAV, necesitamos una librería externa, llamada JLayer, que nos proporciona el paquete `javax.sound.sampled`, que contiene varias utilidades para tratar dichos formatos de sonido. Por otra parte, para reproducir MIDIs utilizamos librerías estándar de Java, `javax.sound.midi`.

En cuanto a imágenes, hemos usado las librerías estándar de `awt`.

5.3.4. es.eucm.eadventure.engine.core

Conociendo el requisito de que el motor debe ser capaz de ejecutarse tanto de forma independiente como embebido en un navegador web con conexión a un servidor LMS, hemos optado por utilizar un `java.awt.Frame` para el pintado y la recepción de eventos, de modo que no tengamos que modificar la interfaz de usuario tanto si está ejecutándose como aplicación independiente o conectado con el servidor en forma de applet.

5.4. Estructura de una aventura

Cada aventura <e-Adventure>consiste de un único fichero ZIP, que contendrá los ficheros XML y multimedia necesarios para describir totalmente dicha aventura. A continuación detallamos el contenido de un fichero ZIP válido con una aventura <e-Adventure>.

Descriptor de aventura Este fichero, que debe llamarse invariablemente *descriptor.xml*, contiene la información general de la aventura. Por una parte, especifica el título y la descripción de la aventura, así como el tipo de GUI (contextual o tradicional) que usará. Además, enumera los capítulos que forman parte de la aventura, estableciendo para cada uno de ellos un título, una descripción, y tres ficheros XML: uno con la descripción del capítulo, y otro dos (opcionales) con las reglas de adaptación y de evaluación, respectivamente.

Capítulos Para cada capítulo que forme parte de la aventura existirá un fichero XML con la descripcion de la misma. Su contenido será todo el conjunto de escenas, objetos, personajes, conversaciones, acciones, efectos, etc., que describen el contenido jugable de

la aventura. No existe una notación específica para los nombres de dichos archivos, pues las referencias están contenidas en el descriptor.

Reglas de adaptación y evaluación Opcionalmente se pueden definir reglas de adaptación y evaluación para cada capítulo. Dichos archivos se guardan también en formato XML.

Recursos Por último, se deben incluir los recursos multimedia que se referencian dentro de los capítulos de la aventura. Si bien no es necesario colocarlos en un lugar concreto, se recomienda utilizar una carpeta llamada *assets-<Nombre aventura>* para contener los assets, de forma que la estructura del contenedor quede más organizada.

GUI personalizada En el caso de que el descriptor indique que los gráficos para la GUI son personalizados, el contenedor de la aventura deberá contener una carpeta llamada `gui/hud` que contendrá los recursos gráficos necesarios para definir la interfaz de usuario completa.

Podremos ver algunos ejemplos de aventuras completas en los apéndices B.2.2 y B.2.3, a partir de la página 132.

Capítulo 6

Editor de aventuras

En este capítulo se incluye la documentación relativa a la concepción y realización el editor de aventuras para el motor <e-Adventure>. Dicho editor se ha desarrollado con la finalidad de servir de herramienta de apoyo al sistema <e-Adventure>, proporcionando una aplicación capaz de editar la totalidad de una aventura compatible con el motor sin necesidad de tener conocimientos de tecnologías XML.

6.1. Motivación y características

Una de las principales motivaciones del proyecto <e-Adventure> es la de proporcionar a los educadores formas sencillas de desarrollar aventuras gráficas educativas. De esta forma, es primordial que el proceso de creación y desarrollo de dichas aventuras sea lo más intuitivo posible.

La primera aproximación a este problema se llevó a cabo a través del lenguaje XML. Dicho lenguaje requiere de conocimientos bajos en programación, y es relativamente legible para una persona poco familiarizada con tecnologías de marcado. Usando XML, se intenta proporcionar al educador un lenguaje de descripción que tanto él como el motor <e-Adventure> puedan entender.

Por desgracia, aún con esta aproximación, a una persona con conocimientos bajos en informática puede resultarle difícil crear un guión en XML. A raíz de ahí surgió la idea de desarrollar un editor de aventuras, que permitiese al usuario diseñar la totalidad del guión de forma más sencilla, utilizando dicha herramienta propietaria asociada a <e-Adventure>.

La característica principal del editor debe ser la de ser capaz de leer, editar y escribir guiones completos sin necesidad de herramientas externas (en lo que al guión concierne, suponemos los recursos gráficos como ya creados). De esta manera la herramienta debe de ser completa, soportar todas las marcas y situaciones que es capaz de procesar el motor <e-Adventure>. Además, se ha de procurar que la interfaz sea intuitiva y agradable para el usuario, todos los elementos de edición tienen que ser facilmente accesibles por el usuario, para facilitar la creación de los guiones.

6.2. Análisis

A continuación se describe el análisis relativo al editor de aventuras. Este análisis comprende tanto interfaz como tecnologías para el desarrollo.

6.2.1. Interfaz

Para la concepción del editor de aventuras, se ha optado por una herramienta con una ventana principal, constando de una estructura de árbol que esquematiza los elementos más importantes de la aventura, y un panel principal para poder editar dichos elementos. La estructura de árbol para la vista general proviene del formato natural del archivo XML; de esta forma se puede mostrar toda la información de la aventura de manera esquematizada, para que el usuario pueda seleccionar que elementos quiere editar.

En cuanto a los paneles de edición, se despliegan dependiendo del elemento del árbol que el usuario tenga seleccionado. Dichos paneles incluyen siempre que es posible apartados de previsualización de los elementos (ya sean escenas, personajes, items...) para que el usuario pueda ver el estado del guión sin necesidad de ejecutarlo con el motor. En la figura 6.1 se puede ver un ejemplo de previsualización.

Cuadro 6.1: Previsualización en el editor de aventuras. En la esquina inferior derecha

6.2.2. Tecnología

Dado que el editor es completamente independiente del motor <e-Adventure> se barajaron distintas posibilidades para la implementación de la herramienta. Las dos opciones

principales fueron C++ (trabajando bajo Borland C++ Builder para desarrollar la GUI) y Java, ya que ambas alternativas proporcionan soluciones sencillas para la creación de interfaces de usuario y potencia de cara al desarrollo.

Tras el estudio de ambas tecnologías, se eligió realizar el desarrollo en Java por las siguientes razones:

- Ejecución multiplataforma. Dado que el mismo motor <e-Adventure> es multiplataforma, parece más sensato que el editor de conversaciones comparta esta característica. Eligiendo Java nos aseguramos que el editor pueda ejecutarse en cualquier maquina que soporte el estandar J2SE.
- Reaprovechamiento de cógido. Muchas de las clases, especialmente de datos, podían ser reaprovechadas del desarrollo del motor <e-Adventure>, con la consiguiente reducción de tiempo de desarrollo.
- Reaprovechamiento del editor de conversaciones. Antes de las primeras versiones implementadas de <e-Adventure>, el grupo de investigación <e-UCM> encargó la realización de un editor de conversaciones para los guiones de aventures <e-Adventure>. Dicho editor se realizó en Java, con lo que toda la parte de edición de escenas podía ser reaprovechada de dicho editor.

6.3. Diseño

A continuación se describe el diseño del editor de aventuras, comprendiendo la estructura de paquetes y clases, así como el uso de patrones de diseño.

6.3.1. Estructura del diseño

Para la estructura general del editor de aventuras, se optó por el modelo vista controlador. Esto es debido a que nuevas versiones del motor <e-Adventure> podrían incluir nuevas marcas que deberían agregarse al editor. Con el modelo vista controlador es más sencillo recabar que partes del código hay que modificar. Además, el modelo vista controlador favorece una mejor estructuración del codigo fuente en responsabilidades definidas, aumentando la legibilidad del código y la facilidad para ampliarlo.

La estructura esquematizada de los guiones ha sido altamente relevante en el proceso del diseño. La gran diversidad de elementos presentes en una aventura de <e-Adventure> sugería la creación de un controlador capaz de regular todas las acciones de edición de datos. Esto hubiera acarreado un controlador demasiado pesado e ilegible, al ser responsable del control de la totalidad del guión. Sin embargo, con la previsualizacion esquematizada del guión, surgió la idea de aislar los controladores dependiendo de sus responsabilidades. De esta forma se ha creado un conjunto de microcontroladores especializados en distintos elementos, repartiendo así la carga que iba a recaer sobre el controlador principal.

Esta estructura de microcontroladores encaja perfectamente con el conjunto de datos del guión, asi como con la representación en nodos de la vista esquematizada. Cada nodo

del árbol de previsualización dispone de un microcontrolador, que se encarga de regular las acciones que se pueden realizar con ese nodo. Cada microcontrolador esta asociado a un nodo del árbol, a un tipo de dato (cuya edición regula) y a un panel encargado de representar los elementos editables del nodo en cuestión.

6.3.2. Estructura de paquetes y clases

Dentro del editor, se distinguen tres paquetes básicos que conforman la aplicación. A continuación se describen sus clases más importantes y sus responsabilidades. Cabe destacar que todas las rutas de paquetes se escribiran relativas al paquete base que alberga el editor de conversaciones, llamado `es.eucm.eadventure.adventureeditor`.

Paquete `control`

Este paquete alberga todos los elementos de control de la aplicación.

El controlador principal recibe el nombre de `Controller`, y es el encargado de proporcionar las tareas más generales de control sobre el guión. Así, tiene como responsabilidades comunicarse con el interfaz gráfico para mostrar distintos dialogos de información o petición de datos, proporcionar las llamadas necesarias para la carga y el guardado de aventuras, almacenar la información completa del guión en edición y proporcionar funciones auxiliares para los microcontroladores.

El paquete `control.auxiliar` contiene distintas clases auxiliares al controlador. La mayoria de estas clases son filtros de archivos (que heredan de `FileFilter`) para la selección de distintos tipos de recursos y archivos en el editor.

El paquete `control.controllers` contiene la totalidad de los microcontroladores del editor de aventuras. Cada uno de estos microcontroladores se encarga de gestionar la edición de un elemento concreto (como puede ser una acción de un objeto, una conversación, un bloque de efectos...). Dentro de estos microcontroladores, cabe destacar la familia que hereda de `DataControl`, esta clase define las llamadas necesarias para trabajar con los datos como si fueran nodos. De esta forma, `DataControl` permite añadir nuevos datos al guión (que aparecerán como nuevos hijos en el árbol), mover dichos elementos, eliminarlos, etc... Cada clase que hereda de `DataControl` implementa las operaciones para editar los nodos que les representan, así como métodos especificos para sus tipos de datos, para que estos puedan ser editados.

El paquete `control.loader` es el contenedor las clases encargadas de leer los guiones a partir de archivos XML. Se compone de una clase principal que devuelve un conjunto de datos completo, y una serie de clases encargadas de leer distintos elementos de los guiones.

El paquete `control.writer` contiene las clases encargadas de la escritura del guión eun archivo XML. Dicha escritura se lleva a cabo por medio del DOM de Java. Este paquete contiene distintas subclases encargadas de generar árboles DOM concretos (para una escena, un personaje...). Aunque todas las llamadas dentro de este paquete se ejecutan de manera estática, se decidió separar las llamadas en distintas clases para facilitar la búsqueda y edición de código de esta funcionalidad.

Paquete `data`

Este paquete alberga la totalidad de los datos relativos a un guión de <e-Adventure>. Se compone básicamente de dos paquetes.

El primero, llamado `data.scriptdata` contiene los datos propios del guión en si. Su clase más general es la de `ScriptData`, que alberga todos los elementos que conforman el guión.

El segundo, llamado `data.supportdata` contiene distintas clases con información de apoyo al guión. Dichos datos de apoyo son en su mayoría sumarios de elementos, de identificadores, de variables (flags), etc...

Paquete `gui`

Este paquete contiene todas las clases encargadas de la interacción con el usuario (la interfaz).

La clase principal de este paquete es `MainWindow`, que hereda de `JFrame`. Esta clase constituye la vista global del editor de aventuras, la ventana principal que muestra los menus, la estructura de árbol y el panel de edición de elementos.

El paquete `gui.sound` contiene las clases necesarias para reproducir sonidos. La totalidad de estas clases ha sido reaprovechado del motor de <e-Adventure>.

El paquete `gui.auxiliar` contiene distintas clases auxiliares para la creación de la GUI.

Los paquetes `gui.displaydialogs` y `gui.displaypanels` contienen distintas clases (diálogos y paneles respectivamente) encargadas de la representación de datos por pantalla. En su mayoria se utilizan para representar imágenes, animaciones y sonidos para las vistas previas.

Los paquetes `gui.editdialogs` y `gui.editpanels` contienen distintas clases (diálogos y paneles respectivamente) encargadas de la edición de los distintos elementos del guión. Gracias a estas clases se hace posible la edición de los contenidos del guión, como son los objetos, los personajes, los escenarios, las condiciones y efectos, etc...

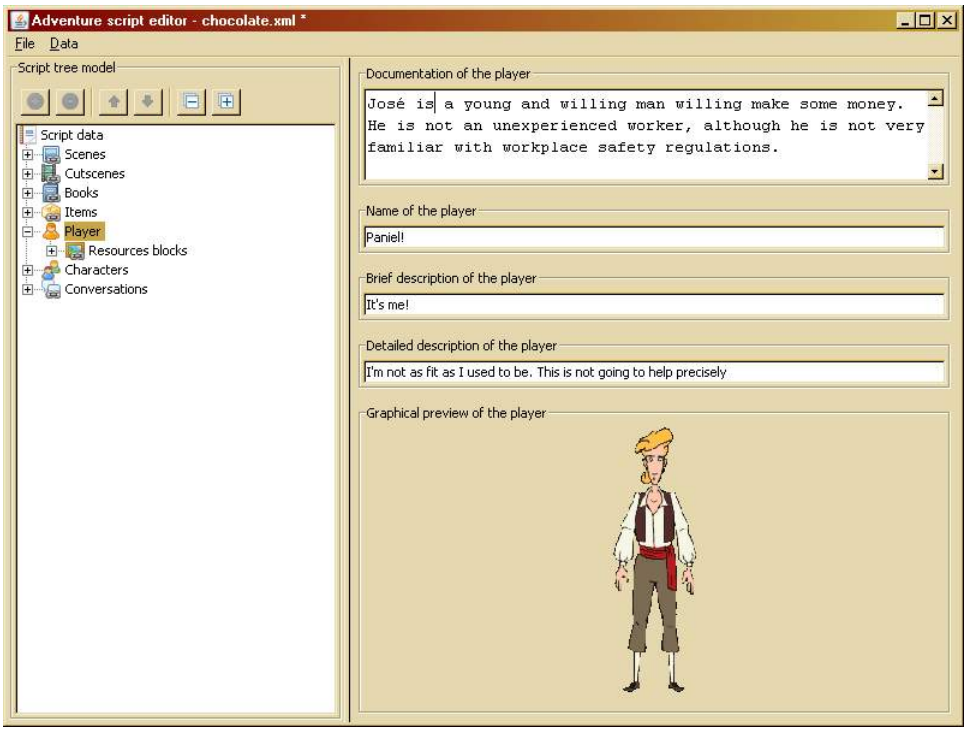
El paquete `gui.treepanel` recoge la clase encargada de representar por pantalla el árbol esquematizado de la aventura, así como las clases utilizadas para representar los nodos de los árboles. La clase principal de la que heredan todos los nodos es `TreeNode`, y contiene los métodos necesarios para llevar a cabo la edición de los nodos.

6.4. Conclusiones

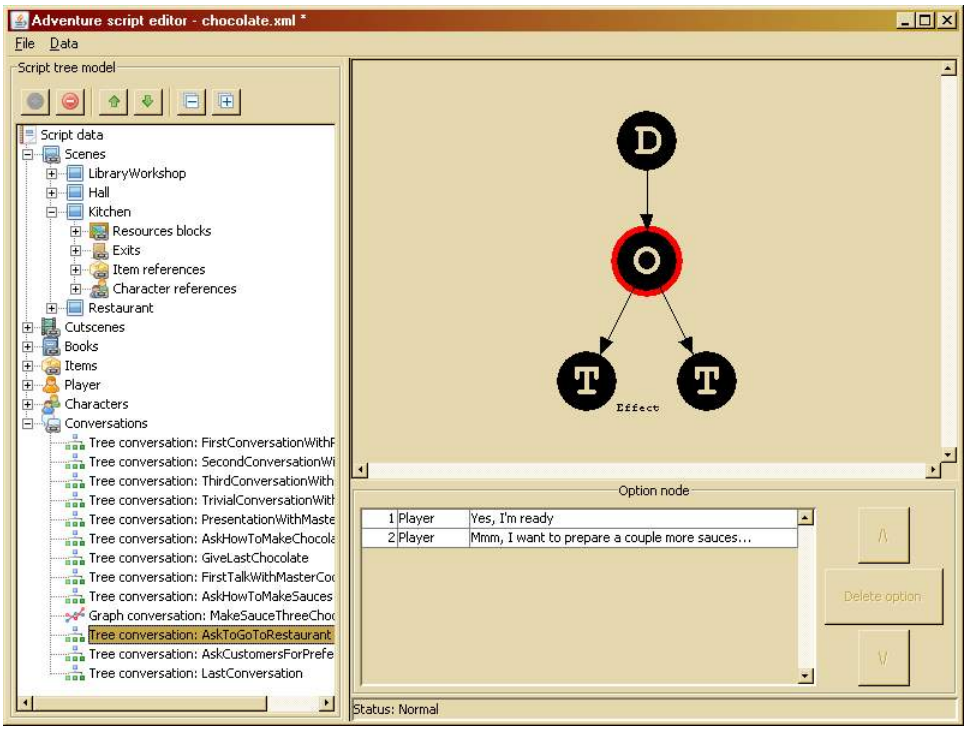
En el momento de la realización de esta memoria, el editor de aventuras consta de una primera versión preliminar. Esta primera versión es capaz de editar la totalidad de los elementos que componen un guión. En distintos periodos de prueba se dictaminará que elementos hay que mejorar, aunque se prevee que la mayoría de las características a mejorar comprendan la interfaz de usuario.

Tras unos primeros usos, se ha podido comprobar como la dificultad para crear guiones para el motor <e-Adventure> se ve rebajada gracias al editor. Su capacidad para mostrar vistas previas de los elementos, y el rápido acceso de los distintos elementos de la aventura acelera el proceso de creación de aventuras, y resulta mucho más intuitivo que trabajar con herramientas de edición de archivos XML. Cabe destacar además que la aplicación se encarga de controlar la coherencia de los datos que constan en el guión, con lo que se libra al usuario de la responsabilidad de verificar que el guión que ha escrito es correcto (es decir, el editor se encarga de comprobar que el guión sea válido para ser leído por el motor de <e-Adventure>).

Por último, se muestran además algunas capturas efectuadas sobre el prototipo desarrollado del editor de aventuras. Dichas imágenes están presentes en las figuras 6.1, 6.2 y 6.3.



Cuadro 6.2: Captura del editor de aventuras. Panel de edición del protagonista



Cuadro 6.3: Captura del editor de aventuras. Panel de edición de conversaciones de árbol

Capítulo 7

Conclusiones

A continuación se describen las conclusiones extraídas del desarrollo del proyecto <e-Adventure>. Dichas conclusiones se componen de dos apartados principalmente: los objetivos que se han cumplido durante el desarrollo, y las características que no se han podido implementar por distintos motivos, y que se dejan como trabajo futuro. A continuación se detallan ambos apartados.

7.1. Objetivos alcanzados

Tras el trabajo realizado, podemos afirmar que el proyecto <e-Adventure> ha quedado desarrollado completamente bajo una especificación básica y completa. Las capacidades del motor al cierre del desarrollo del mismo han sido las previstas en el comienzo del trabajo, proporcionando una aplicación completa y perfectamente funcional. Dichas características se pueden resumir de la siguiente manera:

- **Interactividad**. El jugador o usuario es capaz de interactuar totalmente con el sistema. De esta forma, puede guiar a su personaje, haciendo que se desplace por distintas localizaciones, interactuando mientras tanto con objetos y personajes. El motor dispone además de las capacidades de control de flujo de juego necesarias para permitir que las historias en las que se desenvuelve el jugador tengan la coherencia adecuada. Todas estas características dotan al motor <e-Adventure> de la interactividad suficiente como para poder considerarse un videojuego.
- **Comunicación**. El motor <e-Adventure> puede comunicarse con servidores de LMS, preparados para recibir informes sobre los avances del jugador en el juego. Esta característica es indispensable en la vertiente educativa del proyecto, ya que permite que el entorno global de aprendizaje tenga conocimiento de las aptitudes y destrezas que posee y adquiere el usuario, pudiendo llevar a cabo un seguimiento completo de las mismas.
- **Interfaz sencilla y accesible**. La interfaz en la totalidad del proyecto <e-Adventure> está cuidada de forma que resulte sencillo para cualquier neófito desenvolverse con la aplicación. Concretamente dispone de los siguientes elementos de interfaz para facilitar su uso.

- *Diálogo de selección de juego.* Mostrado sólo en determinadas ejecuciones, permite al usuario seleccionar de manera rápida y sencilla que aventura quiere ejecutar. Esta acción se puede llevar a cabo con un par de clicks de ratón, en el mejor de los casos.
- *Interfaz de juego.* Se han llevado a cabo dos interfaces de juego totalmente distintas, permitiendo a los creadores de contenidos elegir cuál de las dos utilizar. Ambas están diseñadas para permitir al usuario llevar el control de su personaje de manera sencilla. En las figuras 7.1 y 7.2 se pueden ver las interfaces de juego tradicional y contextual, respectivamente. Además, en la figura 7.3 se muestra una captura de los libros que pueden ser incluidos en los juegos para facilitar información a los jugadores de forma sencilla.

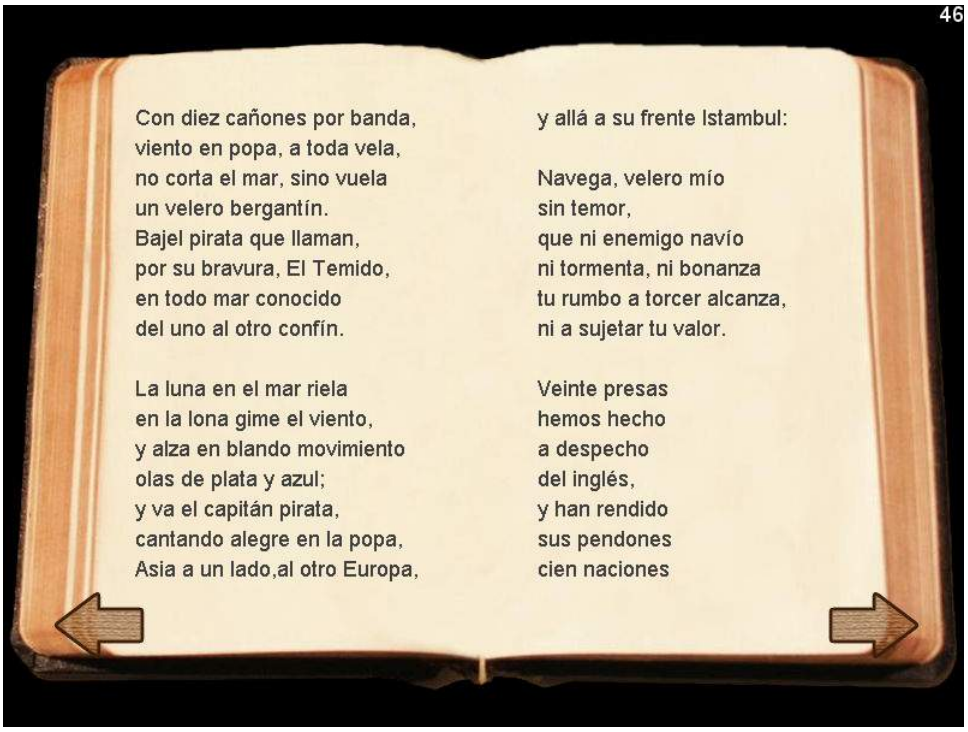
Cuadro 7.1: GUI tracicional

- *Interfaz de opciones.* Además de la interfaz de juego en sí, se ha dotado al motor de una serie de menus de configuración que permiten al usuario personalizar algunas características del motor, como son la aparición de sonidos o la velocidad del texto mostrado. Este interfaz se muestra en la figura 7.4.

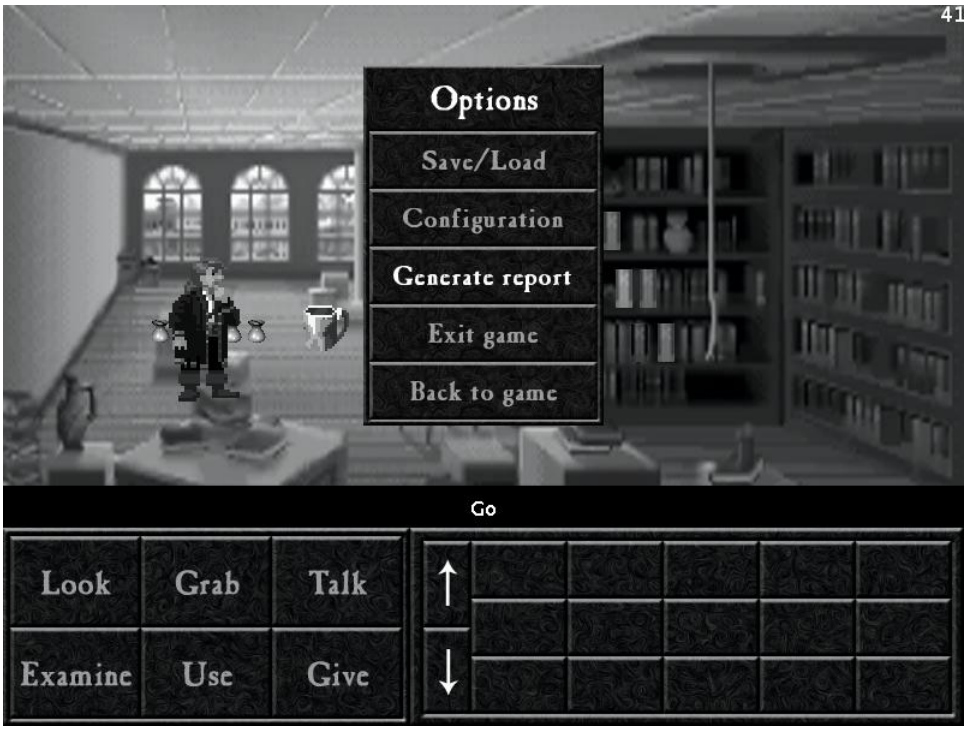
- **Implementación de editor de aventuras**. Dentro del plazo de trabajo del proyecto, se ha podido desarrollar finalmente un primer prototipo del editor de aventuras. Aunque algo precario, soporta totalmente los elementos definidos en el lenguaje de <e-Adventure>, siendo capaz de crear, cargar, guardar y modificar guiones de aventuras compatibles con el motor.

En cuanto a los requisitos fijados al comienzo del proyecto, podemos decir que todos ellos se han cumplido satisfactoriamente. A continuación se detallan de que forma se ha abordado

Cuadro 7.2: GUI contextual



Cuadro 7.3: Ejemplo de libro



Cuadro 7.4: Menú de opciones en el juego

la especificación de cada requisito para poder ofrecer soluciones a los mismos:

- **Sistema independiente de plataforma**. La implementación del sistema en Java ha permitido que el motor <e-Adventure> pueda ser ejecutado en una multitud de máquinas y sistemas operativos distintos.
- **Despliegue sencillo**. El motor se ha concebido para ser ejecutado de dos formas distintas. La primera es una ejecución independiente, que se lleva a cabo ejecutando un archivo JAR; dicho proceso lleva apenas unos segundos siempre y cuando la configuración e instalación de la máquina virtual de Java sea la correcta. La segunda es una ejecución vía web, con un applet; una vez más, si la configuración de Java es adecuada, el motor se puede ejecutar con un simple click de ratón.
- **Capacidad de conexión con LMS**. Como ya se ha comentado anteriormente, el motor dispone de herramientas de comunicación (por medio de la tecnología AJAX) que le permiten comunicarse con un servidor de LMS.
- **Interfaz sencilla**. Las interfaces de juego se han diseñado de forma que las acciones que pueda llevar a cabo el jugador se puedan efectuar únicamente con el ratón, con no más de tres clicks en la mayoría de las ocasiones.
- **Capacidad de expansión**. El diseño del motor <e-Adventure> asegura su alta capacidad de crecimiento. De hecho el lenguaje de descripción de las aventuras ha sido sometido en distintos momentos del desarrollo a cambios, que se han podido llevar a cabo sin grandes modificaciones en la estructura del motor.

- **Uso de guiones sencillos**. Con el uso de guiones notados en lenguaje XML, se da los educadores un lenguaje relativamente sencillo, y a la vez potente, para desarrollar las aventuras. Cabe destacar además que el desarrollo del editor de conversaciones tiene como fin prioritario abstraer a estos educadores de todo conocimiento de la estructura interna de las aventuras, permitiéndoles crear aventuras por medio de aplicaciones diseñadas específicamente para tal fin.

7.2. Futuras ampliaciones

Si bien la inmensa mayoría de los objetivos fijados al principio del proyecto se han completado éxito, hay algunas características que no se han podido implementar, debido a distintas complicaciones. En algunas casos se han propuesto cambios que han sido desestimados debido a su dificultad y la falta de tiempo para realizarlos. A continuación se detallan dos listas de objetivos, constando la primera de aquellas características que no han podido implementarse, y la segunda con distintas ampliaciones y mejoras que podrían realizarse al proyecto en futuros nuevos desarrollos.

A continuación la lista de objetivos desestimados, que se encontraban en la planificación inicial del proyecto:

- **Escenas de video**. Al comienzo del desarrollo del motor <e-Adventure> se propuso en una de las reuniones ampliar el lenguaje por medio de escenas no interactivas, para poder mostrar información al usuario sin necesidad de utilizar conversaciones. Las escenas no interactivas que se propusieron fueron de diapositivas y de video. Para las escenas de video se acordó utilizar el formato de video MPG, y se comenzó a trabajar con Java Media Framework, un API para la reproducción de distintos formatos de video y audio. Desgraciadamente, dicho API requería de un proceso de instalación y configuración que interfería con el requisito de que el sistema fuera fácilmente desplegable. Por lo tanto, y dado que no se pudo encontrar otro API para la reproducción de video, se descartó el uso de secuencias de video dentro del motor <e-Adventure>.
- **Movimiento completo**. En el comienzo del desarrollo, se optó por hacer que el personaje se moviera en una línea horizontal a lo largo del escenario, para simplificar las tareas de movimiento. Sin embargo, una de las características deseadas en la versión final de la aplicación era que el protagonista pudiera moverse libremente por el escenario, de acuerdo a una información adjunta de áreas transitables e intransitables de la escena (llamados mapa de durezas). Sin embargo, dado que dicha característica no era vital para el funcionamiento del motor, y debido a falta de tiempo, acabó por excluirse de la planificación.

Por último, la lista de objetivos propuestos como ampliación, de cara a futuros desarrollos del motor <e-Adventure>.

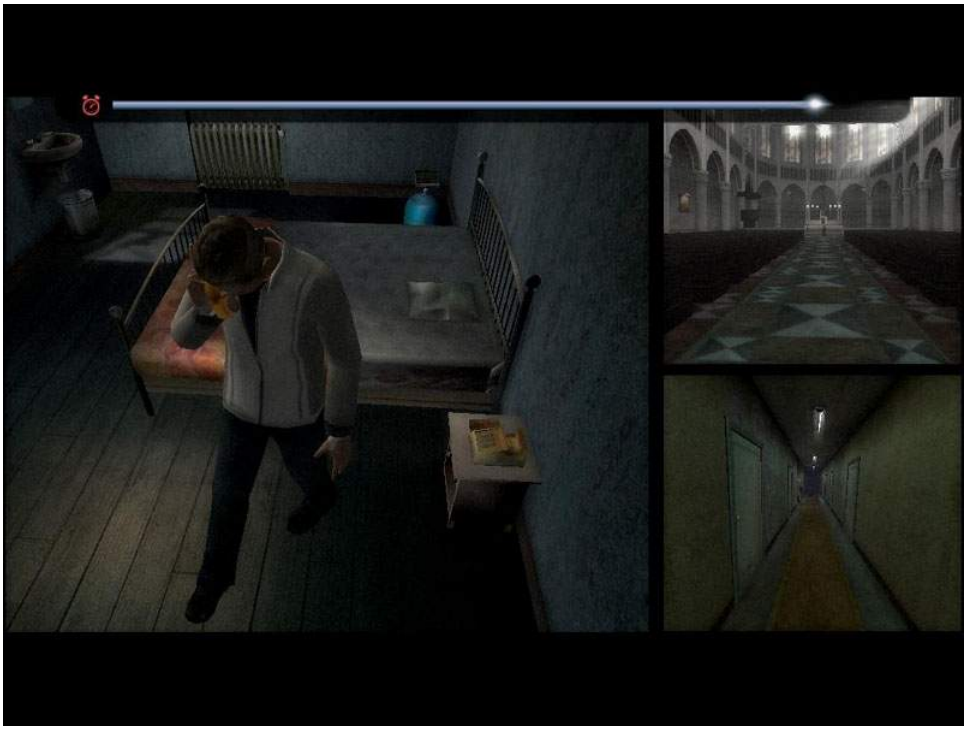
- **Paso a 3D**. Una de las posibilidades de ampliación para el motor es el paso a gráficos en tres dimensiones. Durante el presente año lectivo ha habido importantes avances en la compatibilidad de OpenGL con Java, con lo que parece factible el paso a las tres

dimensiones, ya que se puede preservar la mayoría de la lógica interna del motor. En las figuras 7.5 y 7.6 se muestran ejemplos de aventuras gráficas comerciales con entornos en 3D, en lugar de en 2D.

- **Interfaz de usuario mejorada**. Si bien la versión actual del motor cuenta con dos interfaces de usuario, se ha estudiado la posibilidad de ampliarla con otro nuevo tipo de interfaz. Esta no contaría con acciones diferenciadas que el jugador puede realizar (mirar, usar, examinar, coger...), si no que cada objeto tendría un conjunto de acciones contextuales que se pueden llevar a cabo, sin estar clasificadas de una forma concreta. Cada una de estas acciones se dispararía con una entrada específica, como un patrón de movimiento concreto del ratón, por ejemplo. Esta idea de interfaz toma como base el control implementado en el videojuego Fahrenheit, desarrollado por Quantic Dream y publicado en el año 2005. En las figura 7.6 y 7.7 se pueden ver capturas de dicho juego.

Cuadro 7.5: Aventura en 3D. Escape from Monkey Island®- LucasArts™

Cuadro 7.6: Aventura en 3D. Fahrenheit®- Quantic Dream™



Cuadro 7.7: Interfaz de usuario contextual pura. Fahrenheit®- Quantic Dream™

Apéndice A

Manual de autor

A.1. Introducción

A.1.1. Antes de comenzar

Este documento, el manual de autor, tiene como fin explicar los distintos tipos de marcas y su funcionamiento, dentro del marcado de aventuras gráficas para el motor <e-Adventure>.

Antes de comenzar, es importante señalar que son necesarios conocimientos en el marcado en formato XML para escribir las aventuras gráficas. Además, se recomienda el uso de un editor interactivo, usando la DTD para asegurar que la estructura de archivo es respetada en todo momento.

A.1.2. Contenedor de una aventura

Cada aventura <e-Adventure>consiste de un único fichero ZIP, que contendrá los ficheros XML y multimedia necesarios para describir totalmente dicha aventura. A continuación detallamos el contenido de un fichero ZIP válido con una aventura <e-Adventure>.

Descriptor de aventura Este fichero, que debe llamarse invariablemente *descriptor.xml*, contiene la información general de la aventura. Por una parte, especifica el título y la descripción de la aventura, así como el tipo de GUI (contextual o tradicional) que usará. Además, enumera los capítulos que forman parte de la aventura, estableciendo para cada uno de ellos un título, una descripción, y tres ficheros XML: uno con la descripción del capítulo, y otro dos (opcionales) con las reglas de adaptación y de evaluación, respectivamente.

Capítulos Para cada capítulo que forme parte de la aventura existirá un fichero XML con la descripcion de la misma. Su contenido será todo el conjunto de escenas, objetos, personajes, conversaciones, acciones, efectos, etc., que describen el contenido jugable de la aventura. No existe una notación específica para los nombres de dichos archivos, pues las referencias están contenidas en el descriptor. Este manual de autor describe cómo se deben de crear y modificar los guiones para las aventuras.

Reglas de adaptación y evaluación Opcionalmente se pueden definir reglas de adaptación y evaluación para cada capítulo. Dichos archivos se guardan también en formato XML.

Recursos Por último, se deben incluir los recursos multimedia que se referencian dentro de los capítulos de la aventura. Si bien no es necesario colocarlos en un lugar concreto, se recomienda utilizar una carpeta llamada *assets-<Nombre aventura>* para contener los assets, de forma que la estructura del contenedor quede más organizada.

GUI personalizada En el caso de que el descriptor indique que los gráficos para la GUI son personalizados, el contenedor de la aventura deberá contener una carpeta llamada `gui/hud` que contendrá los recursos gráficos necesarios para definir la interfaz de usuario completa.

A.1.3. Elementos básicos de una aventura

El marcado de aventuras gráficas para <e-Adventure> se compone de una serie de elementos predefinidos. Cada uno de estos elementos tiene un capítulo específico dentro del manual, detallando su funcionamiento y marcado.

Estos elementos son los siguientes:

- Escenas: Encuadran escenas de juego estrictamente interactivas.
- Cutscenes: Encuadran las escenas no interactivas, ya sean de vídeo o de diapositivas.
- Libros: Son un tipo de escena especial, que permiten la aparición de texto en pantalla, como si se estuviese leyendo un libro.
- Objetos: Representan los distintos objetos que aparecen a lo largo de la aventura. Son objetos son colocados en los escenarios y en el inventario del jugador, en momentos concretos del juego.
- Personajes: Es el conjunto de personajes (incluyendo al jugador) que aparecen a lo largo del juego, y con los que el protagonista puede interactuar.
- Conversaciones: Las distintas conversaciones que tienen lugar al hablar con los personajes que se encuentran en el juego. También pueden representar monólogos.

A.1.4. Funcionamiento general del motor <e-Adventure>

Antes de comenzar a describir las características de cada elemento de la aventura y cómo marcarlos en un archivo XML, conviene conocer algunos detalles importantes en cuanto al funcionamiento del motor de <e-Adventure>.

El motor de <e-Adventure> consiste básicamente en una pantalla dividida entre vista de juego e interfaz de control. En la vista de juego se disponen los personajes y objetos con los que el protagonista puede interactuar. En el interfaz de control se encuentran los botones de las distintas acciones, así como el inventario del protagonista.

Además de la vista normal de juego el motor puede adoptar otros estados, como la visualización de diapositivas, vídeos o libros, o la reproducción de conversaciones entre personajes.

El transcurrir del juego está regulado en el motor de <e-Adventure> por un sistema de flags. Estos flags son variables con nombre que pueden estar activados o desactivados. Se encargan de regular la aparición de objetos y personajes en el juego, así como la ejecución de distintos eventos o circunstancias durante la aventura. Para controlar el estado del juego el guión debe ser construido de forma que determinadas acciones activen y desactiven distintos flags, dando lugar de este modo a los resultados esperados. El funcionamiento de dichos flags se describe en el el capítulo A.11.

A.1.5. Formato del descriptor

Al igual que los archivos de los capítulos propiamente dichos, los descriptores de las aventuras también están notadas en formato XML. A continuación se presenta un breve ejemplo de descriptor, y se comentan el uso y la funcionalidad de cada una de las marcas mostradas. Dicho fragmento de código se ha extraído del apéndice B.2.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE game-descriptor SYSTEM "descriptor.dtd">

<game-descriptor>

  <title>Título de la aventura (A mostrar en el selector)</title>
  <description>
    Descripción de la aventura
  </description>

  <configuration>
    <gui type="contextual"/>
  </configuration>

  <contents>
    <chapter path="capitulo1.xml">
      <title>Primer capítulo</title>
      <description>Primer capítulo de la aventura</description>
      <adaptation-configuration path="adapt-cap1.xml" />
      <assessment-configuration path="assess-cap1.xml" />
    </chapter>

    <chapter path="capitulo2.xml">
      <title>Segundo capítulo</title>
      <description>Segundo capítulo de la aventura</description>
    </chapter>
  </contents>
</game-descriptor>
```

Este descriptor de aventura consta de dos capítulos. El funcionamiento de estos es sencillo, ya que se ejecutan de manera secuencial, comenzando por el primer capítulo de la lista, y finalizando la aventura cuando se termina de ejecutar el último de los capítulos definidos.

Como se puede apreciar, la totalidad de los elementos que forman parte del descriptor están dentro del ámbito de la marca <game-descriptor>. Al comienzo se pueden ver dos marcas importantes, que son <title> y <story>. La marca <title> tiene como fin otorgar a la aventura un título para identificarla, mientras que <story> es una breve descripción del contenido de la aventura en sí.

Después de estas dos marcas nos encontramos con un bloque que compone la marca <configuration>. Esta marca proporciona distintas opciones de configuración de las aventuras, pero en este manual sólo veremos la posibilidad de usar distintos tipos de interfaces de usuario. De esta forma, vemos dentro de esta marca otra llamada <gui>, que tiene un atributo llamado *type*, que se encarga de especificar de que tipo es la interfaz usada en el juego. Se pueden dar dos valores a este atributo: *traditional* o *contextual*. Cada uno de estos interfaces tiene sus peculiaridades, como ya se ha explicado anteriormente.

A continuación, se aprecian dos bloques delimitados por la marca <chapter>, siendo cada uno de estos un capítulo de la aventura. Esta marca tiene un atributo llamado *path* que especifica la ruta de acceso al archivo XML que contiene la información sobre el capítulo. En el interior, nos encontramos con marcas <title> y <description>, que tienen usos análogos a las marcas <title> y <story> vistas anteriormente.

Por último, se señalan las marcas utilizadas en el primer capítulo, que son <adaptation-configurat y <assessment-configuration>. Estas dos marcas permiten definir que reglas de adaptación y evaluación han de ser utilizadas en la ejecución del capítulo dado. Ambas poseen un atributo *path*, que sirve para identificar cuáles son los archivos que se utilizarán como datos de adaptación y evaluación.

A.2. Primeros elementos

A.2.1. Estructura básica

Para comenzar, el elemento raíz del archivo XML recibe el nombre de <eAdventure>.

A continuación se presenta un esqueleto básico de una aventura notada, con los elementos básicos (sin incluir escenas, objetos, personajes...):

```
<eAdventure>
  <!--Escenas, objetos, personajes, conversaciones...-->
</eAdventure>
```

Como se puede ver, la totalidad del guión se engloba dentro de la marca <eAdventure>. Dentro de este ámbito se colocarán la totalidad de los elementos que formarán parte de la aventura. En los siguientes capítulos se irán añadiendo estos elementos al esqueleto mostrado. El resultado final se puede encontrar en el apéndice B.2.2.

A.2.2. Documentación

La documentación a lo largo del guión suele ser importante, en muchas circunstancias es difícil saber que hace una determinada acción solo viendo sus condiciones y efectos, y conviene documentar la información, esto se hace con unas marcas colocadas en determinados puntos del guión, con la siguiente estructura:

```
<documentation>Texto de la documentación</documentation>
```

La colocación de estas marcas se detalla dentro de cada uno de los apartados, pues los distintos elementos de la aventura colocan de distinta forma la documentación dentro de sus elementos.

A.3. Escenas interactivas

En este capítulo se describe la manera de crear escenas interactivas, que son el soporte fundamental en el que se basa la experiencia jugable en <e-Adventure>. A continuación se detalla una muestra de código, perteneciente a una escena de la aventura de prueba, a medida que avance el capítulo de irán explicando las distintas marcas.

```
<scene id="Escena">
  <documentation>
    Documentación de la escena
  </documentation>

  <name>Escena</name>

  <exits>
    <exit x="750" y="0" width="50" height="400">
      <documentation>
        Salida por la derecha
      </documentation>
      <next-scene idTarget="Fin"/>
    </exit>
  </exits>

  <objects>
    <object-ref idTarget="Objeto" x="185" y="220">
      <documentation>
        Objeto en el escenario
      </documentation>
    </object-ref>
  </objects>

  <characters>
    <character-ref idTarget="Personaje" x="100" y="200">
```

```
      <documentation>
        Personaje en el escenario
      </documentation>
    </character-ref>
  </characters>
</scene>
```

A.3.1. Elementos generales

Como puede comprobarse, el elemento raiz de las escenas interactivas es <scene>, dicha marca engloba a todas las demas. Podemos ver además que esta marca tiene un atributo de caracter obligatorio llamado *id*, el valor de este atributo se toma como identificador de la escena. Más adelante se podrá ver que casi todos los elementos que componen los guiones de <e-Adventure> constan de su propio identificador.

Otro atributo no presente en el ejemplo es el atributo *start*, que puede tomar los valores "yes" y "no". Este atributo sirve para que el motor <e-Adventure> determine si la aventura debe empezar en esta localización o no. En caso de no incluirse el atributo, el motor presupondrá que el valor esta fijado a "no". En cualquier caso, es muy importante que al menos una de las escenas (ya sean interactivas o cutscenes) tengan el valor *start* fijado a "yes".

El primer elemento interior de la escena es <documentation>. Ya hemos hablado de esta marca, que sirve principalmente para contener información importante para el escritor del guión, ya que es una marca que <e-Adventure> no procesa. Esta marca no es obligatoria.

La siguiente marca es <name>, que contiene el nombre de la escena (esto no es lo mismo que el identificador, que sirve para referirse a la escena internamente, en el XML). Este nombre aparecerá por ejemplo al posicionar el ratón sobre una salida que conecte con la escena.

A.3.2. Salidas

La siguiente marca dentro de la estructura de <scene> es el conjunto <exits>. Este elemento agrupa todas las posibles salidas de las que dispone una escena, cada una de estas marcada con <exit>. Es decir, el conjunto <exits> se compone de una o mas marcas <exit>, siendo cada una de ellas una salida distinta. Cabe destacar que es posible crear una escena sin salidas (sin estructura <exits>), aunque a primera vista carece de sentido.

Tomemos la marca <exit> presente en el ejemplo para analizarla:

```
<exit x="750" y="0" width="50" height="400">
  <documentation>
    Salida por la derecha
  </documentation>
  <next-scene idTarget="Fin"/>
</exit>
```

Podemos ver como la marca <exit> dispone de cuatro atributos: *x*, *y*, *width* y *height*. Estos cuatro atributos sirven para acotar el rectángulo que representa el área de salida. *x* e *y* representan la esquina superior izquierda del rectángulo de la salida, mientras que *width* y *height* representan el ancho y alto del área de salida.

En el interior de la estructura podemos ver una marca de documentación (de carácter opcional), seguida por una marca desconocida: <next-scene>. Esta marca debe estar presente dentro de toda estructura <exit> y sirve para identificar la escena a la que el personaje se desplazará al entrar en el área de salida. <next-exit> consta de una tributo *idTarget* que contiene el identificador de la escena a la que queremos pasar. Es conveniente destacar que esta escena puede ser bien una escena interactiva, o una cutscene de video o diapositivas.

A.3.3. Colocando objetos y personajes

Una de las utilidades principales de estas escenas es la interactividad en si, que se realiza con objetos y personajes. Estos se colocan por un sistema de referencias que se explicará a continuación. Cabe destacar que la estructura de referencias para objetos y personajes es la misma, solo que se agrupan en dos marcas distintas para una separación más clara.

Conviene destacar que al hablar de sistema de referencias, se quiere decir que los elementos que se colocan en el escenario se definen fuera de la estructura del escenario. Estos elementos, objetos y personajes, tienen sus propios capítulos con explicaciones sobre cómo crearlos.

A.3.4. Referencias a objetos

Todos los objetos colocados en una escena se colocan por medio de referencias a su identificador, la marca que agrupa dichas referencias es <objects>, y tiene una estructura como la que sigue:

```
<objects>
  <object-ref idTarget="Objeto" x="185" y="220">
    <documentation>
      Objeto en el escenario
    </documentation>
  </object-ref>
</objects>
```

Cada marca <objects> puede contener una o más referencias a objetos, marcadas con <object-ref>. Las marcas <object-ref> constan de tres atributos obligatorios: *idTarget*, *x* e *y*. *idTarget* contiene el identificador del objeto que queremos colocar en la escena, y *x* e *y* contienen la posición en la que se colocará dicho objeto en la escena (siendo este punto el pie del objeto en cuestión, es decir, el punto centrado en X y el más bajo en Y). Además, si así lo deseamos, podemos colocar en el interior de la referencia a objeto una marca de documentación, para explicar el posicionamiento del elemento en la escena.

A.3.5. Referencias a personajes

Como ya se ha comentado anteriormente, la estructura de referencias a personajes es análoga a la de referencias de objeto. El elemento padre es <characters>, y toma la siguiente forma:

```
<characters>
  <character-ref idTarget="Personaje" x="100" y="200">
    <documentation>
      Personaje en el escenario
    </documentation>
  </character-ref>
</characters>
```

Cada marca <characters> contiene una o más marcas <character-ref>, que se comportan exactamente igual a <object-ref>. Con la única diferencia que el atributo *idTarget* debe contener el identificador de un personaje.

A.4. Cutscenes

Las cutscenes son escenas no interactivas de juego, que sirven para presentar al jugador fragmentos de la aventura, como si de una película se tratase. Hay dos tipos de cutscenes: de *vídeo* y de *diapositivas*. Aunque cada una de ellas se nombra con su propia marca, su estructura interna es la misma. Por esto mismo se describirá el funcionamiento de ambas, para al final especificar cuál es la diferencia entre ambos tipos. A continuación se detallan las dos cutscenes de la aventura de prueba:

```
<cutscene id="Comienzo" start="yes">
  <documentation>
    Escena de diapositivas con el comienzo de la aventura
  </documentation>

  <name>Comienzo</name>

  <next-scene idTarget="Escena"/>
</cutscene>

<cutscene id="Fin">
  <documentation>
    Escena de diapositivas con el fin de la aventura
  </documentation>

  <name>Fin</name>

  <end-game/>
```

```
</cutscene>
```

A.4.1. Elementos generales

En ambos casos, se puede apreciar que el elemento base se llama <cutscene>. Es importante señalar que esta no es una marca real, si no que tiene como fin describir el funcionamiento general de las cutscenes. En el apartado A.4.3 se especifican cuales son las marcas reales a usar para cada tipo de cutscene.

En ambas estructuras, se aprecia como los elementos raiz contienen atributos, ambos conocidos. El primero es *id*, que almacena el identificador de la cutscene (un nombre único para hacer las referencias a la cutscene). El segundo es *start*, que indica si la escena ha de usarse como primera escena en el juego. Podemos ver como en este caso la cutscene identificada como "Comienzo" tiene el valor de *start* a "yes", con lo que se utilizará como comienzo de juego.

La primera marca visible es <documentation>, que almacena la documentación de la cutscene. Destacar que esta marca no es obligatoria.

La última marca general es <name>, que almacena el nombre de la cutscene, y que será mostrado más adelante en pantalla al seleccionar una salida que enlace a la cutscene. Esta marca es de caracter obligatorio.

A.4.2. Fin de cutscene

Para finalizar la cutscene, hay tres comportamientos básicos, dos de los cuales están mostrados en los ejemplos presentados:

- Vuelta a la escena: Cuando se acabe de mostrar la cutscene, se volverá a la última escena interactiva que se ejecutó. Esto se utiliza si queremos mostrar una pequeña secuencia en un momento determinado del juego, para luego volver al punto en el que estabamos. La forma de provocar este comportamiento es dejando el final de la estructura <cutscene> sin ninguna marca.
- Salto a otra escena: Cuando se acabe de mostrar la cutscene, se saltará automáticamente a otra escena determinada. Esto se consigue añadiendo la marca <next-scene> al final de la estructura <cutscene>. Dicha marca tiene un atributo *idTarget* que contiene el identificador de la escena (o cutscene) a la que se debe pasar. Se puede ver esta marca en la cutscene "Comienzo".
- Fin de juego: Cuando se acabe de mostrar la cutscene, el juego finalizará. Esto es conveniente si queremos mostrar una pequeña pantalla de despedida. Conviene destacar que es la única forma reconocible por el motor para terminar el juego, con lo que en ocasiones es conveniente realizar una pequeña cutscene únicamente para finalizar el juego. Se nota con una marca <end-game> ubicada al final de la estructura <cutscene>. Se puede ver esta marca en la cutscene "Fin".

A.4.3. Escenas de vídeo y diapositivas

Las diferencias entre las cutscenes de vídeo y diapositivas es únicamente en su representación por pantalla. Cada una de ellas muestra la secuencia de una forma distinta, mientras que su comportamiento general es exactamente igual.

Hasta ahora hemos usado la marca <`cutscene`> como marca general para las cutscenes, pero esta no es una marca válida, si no que hay que usar de las definidas como cutscenes válidas. Esto es:

- Cutscene de vídeo: se ha de usar la marca <`videoscene`> en lugar de <`cutscene`>.
- Cutscene de diapositiva: se ha de usar la marca <`slidescene`> en lugar de <`cutscene`>.

La otra diferencia importante radica en la forma de cargar los recursos, en cualquier caso estos detalles se explicarán más adelante, en el capítulo A.10.

A.5. Objetos

Los objetos son uno de los elementos más importantes dentro de las aventuras gráficas. En este capítulo se explicará como definir los objetos y su comportamiento, de cara a la ejecución en el juego. A continuación se detalla un ejemplo de objeto, presente en la aventura de prueba:

```
<object id="Objeto">
  <documentation>
    Objeto presente en la aventura
  </documentation>

  <description>
    <name>
      Objeto
    </name>
    <brief>
      Descripcion simple
    </brief>
    <detailed>
      Descripcion detallada
    </detailed>
  </description>

  <actions>
    <grab>
      <documentation>
        El objeto se puede coger
      </documentation>
    </grab>
```

```
    <give-to idTarget="Personaje">
      <documentation>
        El objeto se puede entregar a Personaje
      </documentation>
    </give-to>
  </actions>
</object>
```

A.5.1. Elementos generales

En primer lugar, vemos que la marca general de la estructura es <object>. Esta marca dispone de un atributo *id* que contiene el identificador asociado al objeto a lo largo del archivo XML.

La siguiente marca visible es de documentación, de caracter opcional, como de costumbre. La siguiente estructura es imprescindible para los objetos, pues es el conjunto de descripciones, por esto mismo es un conjunto de marcas obligatorio para todo objeto. Esta estructura tiene como elemento raiz <description>, y contiene otras tres, que son <name>, <brief> y <detailed>. La marca <name> almacena el nombre del objeto, <brief> contiene una descripción simple del objeto (que normalmente aparece en pantalla al mirar el objeto), y <detailed> es una descripción detallada del objeto (que se muestra en pantalla al examinar el objeto).

A.5.2. Acciones

Las acciones es una de las partes más importantes de los objetos. En total hay cuatro tipos de acciones diferentes para realizar con los objetos. Estas acciones son *coger*, *dar*, *usar* y *examinar*. En el ejemplo mostrado se usan únicamente dos, que son *coger* y *dar*, pero se explicará el funcionamiento de las cuatro acciones. Veamos el código que contiene las acciones:

```
<actions>
  <grab>
    <documentation>
      El objeto se puede coger
    </documentation>
  </grab>

  <give-to idTarget="Personaje">
    <documentation>
      El objeto se puede entregar a Personaje
    </documentation>
  </give-to>
```

```
</actions>
```

Podemos ver que todas las acciones están agrupadas dentro de una estructura con elemento raiz <actions>. Este objeto en concreto tiene dos acciones, que son <grab> (*coger*) y <give-to> (*dar*). Podemos ver ambas acciones disponen de su propia documentacion (que es opcional). A continuación se describe el uso y los efectos de cada acción:

- Coger: Se nota con la marca <grab>. Cuando se coge un objeto, este desaparece del escenario y aparece en el inventario del jugador. No requiere atributos.
- Dar: Se nota con la marca <give-to>. Cuando se da un objeto a un personaje, este desaparece automáticamente del inventario del jugador. Requiere un atributo *idTarget* que contiene el identificador del personaje al que podemos entregar el objeto.
- Usar: Se nota con la marca <use>. Esta acción se dispara cuando el protagonista intenta utilizar el objeto; es una marca util cuando queremos interaccionar con los objetos que se encuentran en el juego. Conviene destacar que el objeto no desaparece por defecto una vez utilizado.
- Usar con: Se nota con la marca <use-with>. La diferencia de esta marca de uso con la anterior es que en este caso el objeto debe de ser usado con otro objeto, y no de forma individual. Se puede utilizar cualquier par de objetos seleccionados, si bien uno de ellos debe de estar en el inventario; conviene destacar que los objetos no desaparecen por defecto una vez utilizados. Requiere un atributo *idTarget* que contiene el identificador del objeto con el que se puede usar. Es importante destacar que al intentar usar un objeto con marcas <use> y <use-with>, siempre tendrá prioridad de ejecución la marca <use>, siempre que sus condiciones se cumplan.
- Examinar: Se nota con la marca <examine>. Esta acción se dispara cuando el usuario examina el objeto. En condiciones normales se muestra la descripción detallada del mismo, pero en ocasiones nos convendrá dar a esta acción comportamientos específicos.

Por último, cabe destacar que un objeto puede tener cuantas acciones sean precisas (incluso del mismo tipo). Para más información sobre las acciones y los efectos que estas pueden desencadenar, es recomendable leer los capítulos A.11 y A.12.

A.6. Protagonista

Otro de los elementos fundamentales en cualquier aventura es el protagonista, el avatar que utiliza el jugador para interactuar con el mundo del juego. A continuación se detalla el código que describiría al protagonista del guión de prueba:

```
<player>
  <documentation>
    Documentacion del jugador
  </documentation>
```

```
    <description>
      <name>
        Jugador
      </name>
      <brief>
        Descripcion breve
      </brief>
      <detailed>
        Descripcion detallada
      </detailed>
    </description>
  </player>
```

Se puede apreciar que la marca raiz de la estructura del protagonista es <player>. En primer lugar, vemos como se incluye documentación, de forma optativa, para describir al protagonista de forma interna en el guión.

A continuación viene una estructura conocida, la de <description>. Esta estructura es análoga a la explicada para los objetos (capítulo A.5), contando con nombre, descripción breve y descripción detallada.

A.7. Personajes

Otro elemento indispensable en el motor de <e-Adventure> son los personajes con los que el protagonista puede interactuar de una forma u otra. A continuación se detalla el código del personaje presente en la aventura de prueba:

```
  <character id="Personaje">
    <documentation>
      Documentacion del personaje
    </documentation>

    <resources>
      <asset type="standup" uri="prueba/StanStandingFront"/>
      <asset type="standdown" uri="prueba/StanStandingFront"/>
      <asset type="standright" uri="prueba/StanStandingFront"/>
      <asset type="speakup" uri="prueba/StanStandingFront"/>
      <asset type="speakdown" uri="prueba/StanStandingFront"/>
      <asset type="speakright" uri="prueba/StanStandingFront"/>
      <asset type="useright" uri="prueba/StanStandingFront"/>
    </resources>

    <description>
      <name>
        Personaje
      </name>
```

```
      <brief>
        Descripcion breve
      </brief>
      <detailed>
        Descripcion detallada
      </detailed>
    </description>

    <conversations>
      <conversation-ref idTarget="Conversacion">
        <documentation>
          Podemos tener una conversacion con el personaje
        </documentation>
      </conversation-ref>
    </conversations>
  </character>
```

A.7.1. Elementos generales

Vemos como el elemento raiz de un personaje es la marca <`character`>. Esta tiene un atributo llamado *id* que almacena su identificador de cara a ser referenciado en el resto de la aventura.

A continuación encontramos estructuras ya conocidas, como son la de documentación y la de descripción, análogas a las de los objetos (capítulo A.5) y el protagonista (capítulo A.6).

A.7.2. Referencias a conversaciones

Una de las principales características de los personajes es su capacidad de entablar conversaciones con el jugador. De esta forma tenemos que definir las conversaciones que el personaje puede llevar a cabo a lo largo de la aventura. Dicho conjunto de referencias a conversaciones se encuadran dentro de la marca <`conversations`>. Cada referencia individual se marca con <`conversation-ref`>.

Podemos ver que la marca <`conversation-ref`> dispone de un atributo llamado *idTarget* que contiene el identificador de la conversación que queremos que el personaje sea capaz de reproducir. Además, como elemento interno, podemos añadir documentación de manera opcional.

Un personaje puede tener un número ilimitado de conversaciones para ejecutar, si bien es necesario el uso de condiciones para que se ejecuten unas u otras dependiendo de las exigencias del guión. Para mas información sobre condiciones, visitar el capítulo A.11.

A.8. Conversaciones

Otro de los elementos más relevantes dentro de <e-Adventure> son las conversaciones con los personajes. Dichas conversaciones tienen un carácter interactivo, permitiendo al jugador elegir respuesta en determinados momentos de la conversación, decidiendo así el curso que tomará la misma (o los eventos que desencadenará). El ejemplo de conversación presente en la aventura de prueba es el siguiente:

```
<tree-conversation id="Conversacion">
  <speak-player>Hola</speak-player>
  <speak-char idTarget="Personaje">Hola, ¿que tal?</speak-char>
  <response>
    <option>
      <speak-player>Bien</speak-player>
      <speak-char idTarget="Personaje">Me alegro</speak-char>
      <end-conversation/>
    </option>
    <option>
      <speak-player>Mal</speak-player>
      <speak-char idTarget="Personaje">Lo siento</speak-char>
      <end-conversation/>
    </option>
  </response>
</tree-conversation>
```

Antes que nada, hay que destacar que <e-Adventure> soporta dos tipos de estructura conversacionales. Una de ellas con estructura de árbol, y otra con estructura de grafo. En el presente manual sólo se explicará la estructura de las conversaciones en árbol. En cualquier caso para la notación de las conversaciones es recomendable el uso de la herramienta específica diseñada a tal efecto.

Como elemento raiz, podemos ver la marca <`tree-conversation`>, dicha marca tiene un atributo *id* que almacena el identificador con el que nos referiremos a dicha conversación en el resto del guión.

Dada su estructura de árbol, una conversación dispone de tramos lineales y bifurcaciones en las conversaciones. Los elementos básicos de la conversacion se marcan con <`speak-player`> y <`speak-char`>. La primera marca sirve para que el protagonista pronuncie una frase, y la segunda tiene el mismo efecto sobre un personaje en juego. <`speak-char`> dispone de un atributo llamado *idTarget* que almacena el identificador del personaje que pronunciará la frase. Si bien no es un atributo obligatorio, es altamente recomendable incluirlo en toda ocasión para evitar problemas durante la ejecución de los guiones de <e-Adventure>.

En cuanto a los tramos lineales de conversación, se notan con una secuencia de marcas <`speak-player`> y <`speak-char`>, en cualquier número y orden.

Para las bifurcaciones, se requiere de una estructura más compleja. Para notar una bifurcación se usa la marca <`response`>, que indica que vamos a presentar varias opciones

para que el jugador pueda elegir una de ellas. Cabe recordar que la estructura de árbol no permite que varias ramas converjan, de manera que despues de la marca <response> no puede haber mas marcas de ningun tipo (exceptuando el cierre del elemento padre).

Dentro de la estructura de <response> se han de colocar tantas estructuras <option> como opciones queramos que tenga la bifurcación. Cada una de estas nuevas estructuras ha de contar con una primera marca de tipo <speak-player>, representando una de las opciones de dialogo para el jugador. A partir de esta primera marca, el patrón se repite pudiendo colocar de nuevo tramos lineales y bifurcaciones de forma anidada.

Para el fin de las conversaciones, se utilizan dos marcas especiales. La primera es <end-conversatio
que representa el final de una rama de la conversación (o el fin de la misma); esta marca se puede colocar en cualquier rama, incluso si la conversación tiene un caracter estríctamente lineal. La segunda marca es <go-back>, que sólo puede usarse en ramas de una bifurcación, esta marca concreta permite que al finalizar la rama actual, se vuelve a la última intersección, de forma que el jugador puede elegir de nuevo una respuesta a elegir.

A.9. Libros

Los libros se presentan en el motor <e-Adventure> como una manera más de mostrar información al jugador. Cuando se ejecuta una secuencia con un libro se muestra por pantalla el libro con el texto que contiene, el jugador puede pasar las páginas adelante y atras para leer el contenido del libro, y salir del modo de lectura para volver al juego en si.

Una estructura de libro se notaría de la siguiente manera:

```
<book id="Libro">
  <documentation>
    Ejemplo de libro
  </documentation>

  <text>
    Texto del libro
  </text>
</book>
```

Como se puede apreciar, la marca base de esta estructura es <book>. En ella se anidan el resto de marcas.

La primera marca visible es <documentation>. Esta marca es opcional y sirve para almacenar la documentación del libro. La siguiente marca presente es <text>, en su interior se debe añadir todo el texto que queremos que aparezca en el interior del libro. Es importante señalar que en el procesado del texto, se tendrán en cuenta los saltos de linea, y se ignorarán los caracteres de tabulación.

A.10. Recursos multimedia

Los recursos multimedia sirven para asociar recursos multimedia, como gráficos, animaciones o música, a los escenarios, objetos, personajes, etc... Si bien la construcción de estas marcas difiere para cada elemento, todas siguen un patrón común, que se muestra a continuación:

```
<resources>
  <!--Una o ninguna estructura <condition>-->

  <asset type="Tipo" uri="Direccion"/>
  <!--Más marcas de tipo <asset>-->
</resources>
```

Hay que tener en cuenta la importancia de la inclusión de las condiciones en los recursos multimedia. De esta manera podemos definir que recursos ha de utilizar un elemento para distintas condiciones del juego. En general, todos los elementos pueden tener más de un conjunto de recursos multimedia definido, cada uno con sus propias condiciones. Es conveniente saber que siempre se utilizará el **primer conjunto que satisfaga sus condiciones**; es decir, si tuviesemos dos conjuntos <resources> se comprobarían sus condiciones en orden, y se escogería el primero en satisfacerlas. Si un conjunto <resources> no tiene condiciones, se escoge incondicionalmente (siempre respetando el orden de procesamiento). La información de las condiciones está presente en el capítulo A.11.

A.10.1. Incluyendo recursos

En cuanto a cómo incluir los recursos independientes, se hace con la marca <asset>, como se ha mostrado arriba. Dicha marca dispone de dos atributos: *type* y *uri*. La primera es una cadena especial que sirve para identificar el tipo de recurso (en el ejemplo seria *Tipo*); los valores para los distintos elementos están descritos en sus capítulos correspondientes.

La dirección de los archivos y carpetas (en el parametro *Direccion*) ha de escribirse de forma *case sensitive*, y de forma relativa a la ubicación del archivo XML; además ha de usarse la barra normal (/) para la separación de los directorios. En cuanto a la notación, es distinta para dos conjuntos de recursos distintos:

- Archivos únicos: Este conjunto abarca iconos, archivos de musica, imágenes de fondo, etc... En este caso se debera escribir el nombre completo del archivo, con su extension. Un ejemplo sería “assets/Escenario1/TemaPrincipal.mp3”.
- Animaciones: Las animaciones se definen como un conjunto de archivos en formato PNG, que se reproducen secuencialmente para simular la animación. El formato de nombre para una animación seria *Frame_01.png*, *Frame_02.png*, ... , *Frame_xx.png*. De esta forma la definición de la ruta de los archivos no debe incluir número o extensión, ya que el motor <e-Adventure> se encarga de añadir la extensión “_XX.png” a la ruta que le indicamos, de forma que la ruta para esta animación (de encontrarse en la carpeta

"assets/Personaje") sería "assets/Personaje/Frame". Es conveniente saber además que el propio motor se encarga de encontrar el número de frames de una animación y que el máximo número de frames por animación es de 99 frames, por restricción del motor.

A.10.2. Identificadores para los recursos

En esta sección se detallan los identificadores para cada tipo de elemento (los valores del atributo *type* en la marca <`asset`>), de cara a añadir los recursos necesarios al guión que se esté escribiendo. Tanto en la primera versión de la aventura (apéndice B.2.2) como en la segunda versión (apéndice B.2.3) se incluyen muestras de uso de <`resources`>.

Escenas interactivas

Los tipos usados para las escenas interactivas son los siguientes:

- **background** - Imagen simple (con extensión). Es la imagen de fondo del escenario.
- **foreground** - Imagen simple (con extensión). Esta imagen ha de ser un archivo PNG monocromo (con dos colores, blanco y negro) del mismo tamaño que la imagen de fondo del escenario. Sirve para discernir qué partes del escenario se deben de pintar en primer plano (color negro en el archivo) y cuáles forman parte del fondo (color blanco en el archivo).
- **bgmusic** - Musica (con extensión). Es la música que suena de fondo en el escenario.

A continuación se muestra un ejemplo extraído del apéndice B.2.3:

```
<resources>
  <asset type="background" uri="prueba/Escena1.jpg"/>
  <asset type="bgmusic" uri="prueba/Escena1.mid"/>
</resources>
```

Cutscenes

Los tipos usados para las cutscenes son los siguientes:

- **slides** - Sólo se usa en <`slidescene`>. Conjunto de imagenes (sin extensión). Es la serie de diapositivas a mostrar.
- **video** - Sólo se usa en <`videoscene`>. Archivo de video (con extensión). Es el vídeo que se reproducirá.

A continuación se muestra un ejemplo (de tipo <`slidescene`>) extraído del apéndice B.2.3:

```
<resources>
  <asset type="slides" uri="prueba/Comienzo"/>
</resources>
```

Objetos

Los tipos usados para los objetos son los siguientes:

- **image** - Imagen simple (con extensión). Es la imagen del objeto dentro del escenario.
- **icon** - Imagen simple (con extensión). Es la imagen del objeto en el inventario. Su tamaño ha de ser de 80x48 pixels.

A continuación se muestra un ejemplo extraído del apéndice B.2.3:

```
<resources>
  <asset type="image" uri="prueba/Objeto1.png"/>
  <asset type="icon" uri="prueba/Objeto1.png"/>
</resources>
```

Protagonista

Los tipos usados para el protagonista son los siguientes:

- **standup** - Animación, conjunto de imágenes (sin extensión). Es la animación del protagonista parado, orientado hacia arriba.
- **standdown** - Animación, conjunto de imágenes (sin extensión). Es la animación del protagonista parado, orientado hacia abajo.
- **standright** - Animación, conjunto de imágenes (sin extensión). Es la animación del protagonista parado, orientado hacia la derecha (esta animación se espeja para su análoga hacia la izquierda).
- **speakup** - Animación, conjunto de imágenes (sin extensión). Es la animación del protagonista hablando, orientado hacia arriba.
- **speakdown** - Animación, conjunto de imágenes (sin extensión). Es la animación del protagonista hablando, orientado hacia abajo.
- **speakright** - Animación, conjunto de imágenes (sin extensión). Es la animación del protagonista hablando, orientado hacia la derecha (esta animación se espeja para su análoga hacia la izquierda).
- **useright** - Animación, conjunto de imágenes (sin extensión). Es la animación del protagonista usando o dando un objeto, orientado hacia la derecha (esta animación se espeja para su análoga hacia la izquierda).
- **walkup** - Animación, conjunto de imágenes (sin extensión). Es la animación del protagonista andando, orientado hacia arriba.
- **walkdown** - Animación, conjunto de imágenes (sin extensión). Es la animación del protagonista andando, orientado hacia abajo.

- **walkright** - Animación, conjunto de imágenes (sin extensión). Es la animación del protagonista andando, orientado hacia la derecha (esta animación se espeja para su análoga hacia la izquierda).

A continuación se muestra un ejemplo extraído del apéndice B.2.3:

```
<resources>
  <asset type="standup" uri="prueba/PlayerStandBack"/>
  <asset type="standdown" uri="prueba/PlayerStandFront"/>
  <asset type="standright" uri="prueba/PlayerStandFront"/>
  <asset type="speakup" uri="prueba/PlayerStandFront"/>
  <asset type="speakdown" uri="prueba/PlayerStandFront"/>
  <asset type="speakright" uri="prueba/PlayerStandFront"/>
  <asset type="useright" uri="prueba/PlayerStandFront"/>
  <asset type="walkup" uri="prueba/PlayerStandFront"/>
  <asset type="walkdown" uri="prueba/PlayerStandFront"/>
  <asset type="walkright" uri="prueba/PlayerWalkRight"/>
</resources>
```

Personajes

Los tipos usados para los personajes son los siguientes:

- **standup** - Animación, conjunto de imágenes (sin extensión). Es la animación del personaje parado, orientado hacia arriba.
- **standdown** - Animación, conjunto de imágenes (sin extensión). Es la animación del personaje parado, orientado hacia abajo.
- **standright** - Animación, conjunto de imágenes (sin extensión). Es la animación del personaje parado, orientado hacia la derecha (esta animación se espeja para su análoga hacia la izquierda).
- **speakup** - Animación, conjunto de imágenes (sin extensión). Es la animación del personaje hablando, orientado hacia arriba.
- **speakdown** - Animación, conjunto de imágenes (sin extensión). Es la animación del personaje hablando, orientado hacia abajo.
- **speakright** - Animación, conjunto de imágenes (sin extensión). Es la animación del personaje hablando, orientado hacia la derecha (esta animación se espeja para su análoga hacia la izquierda).
- **useright** - Animación, conjunto de imágenes (sin extensión). Es la animación del personaje recibiendo un objeto, orientado hacia la derecha (esta animación se espeja para su análoga hacia la izquierda).
- **walkup** - Animación, conjunto de imágenes (sin extensión). Es la animación del personaje andando, orientado hacia arriba.

- **walkdown** - Animación, conjunto de imágenes (sin extensión). Es la animación del personaje andando, orientado hacia abajo.
- **walkright** - Animación, conjunto de imágenes (sin extensión). Es la animación del personaje andando, orientado hacia la derecha (esta animación se espeja para su análoga hacia la izquierda).

A continuación se muestra un ejemplo extraído del apéndice B.2.3:

```
<resources>
  <asset type="standup" uri="prueba/StanStandingFront"/>
  <asset type="standdown" uri="prueba/StanStandingFront"/>
  <asset type="standright" uri="prueba/StanStandingFront"/>
  <asset type="speakup" uri="prueba/StanStandingFront"/>
  <asset type="speakdown" uri="prueba/StanStandingFront"/>
  <asset type="speakright" uri="prueba/StanStandingFront"/>
  <asset type="useright" uri="prueba/StanStandingFront"/>
  <asset type="walkup" uri="prueba/StanStandingFront"/>
  <asset type="walkdown" uri="prueba/StanStandingFront"/>
  <asset type="walkright" uri="prueba/StanStandingFront"/>
</resources>
```

Libros

Los tipos usados para los libros son los siguientes:

- **background** - Imagen simple (con extensión). Es la imagen de fondo del libro.

A continuación se muestra un ejemplo extraído del apéndice B.2.3:

```
<resources>
  <asset type="background" uri="prueba/Libro.jpg"/>
</resources>
```

A.11. Condiciones

Antes de comenzar a hablar de condiciones y efectos, conviene destacar que para ilustrar el uso de estas nuevas marcas se ha incluido un nuevo guión de ejemplo (en el apéndice B.2.3) con condiciones y efectos, para demostrar el uso de los mismos. Cabe destacar además que en el primer guión de aventura presente en el apéndice se incluye un libro, pero que no es ejecutable en ningún momento, dado que no se habían explicado la inclusión de efectos.

Las condiciones es uno de las características más potentes del motor <e-Adventure>, siempre y cuando sean bien utilizadas. Permiten incluir condiciones precisas que determinan

bajo qué circunstancias aparecen los objetos o personajes, o cuando deben ejecutarse determinadas acciones. Gracias a las condiciones se puede establecer un flujo de funcionamiento dentro de la aventura de una manera relativamente sencilla. Cabe destacar además, que por regla general, al arrancar el motor <e-Adventure>, todos los flags están desactivados.

Hay dos marcas indispensables para poder notar las condiciones, dichas marcas son:

```
<active flag="Flag"/>
<inactive flag="Flag"/>
```

Ambas contienen un atributo *flag* que debe contener el nombre de la variable que queremos comprobar. En el caso de <active> se evalúa a cierto cuando el flag especificado está activado. Por el contrario, la marca <inactive> se evalúa a cierto cuando el flag esta desactivado. Con estas dos evaluaciones podemos construir condiciones más complejas, como se muestra a continuación

A.11.1. Condición simple

Las condiciones más sencillas se construyen con una o más marcas <active> o <inactive>, tal y como se ha descrito antes. Estas marcas se evalúan con un operador lógico AND, y su construcción es de la siguiente forma:

```
<condition>
  <(in)active flag="Flag">
  <!--Más marcas active o inactive-->
</condition>
```

Se ve que cada marca <condition> puede contener una o más marcas <active> o <inactive>. De esta forma la siguiente condición

```
<condition>
  <active flag="Flag1">
  <inactive flag="Flag2">
  <inactive flag="Flag3">
</condition>
```

se evaluaría como cierta si *Flag1* estuviera activada, y *Flag2* y *Flag3* estuvieran desactivadas. De esta forma se construyen las condiciones simples.

A.11.2. Condición en FCN

Una forma más compleja de condición, es por medio de la Forma Conjuntiva Normal. Esta forma consiste en la conjunción global de una serie de disyunciones. Para expresarlo con un ejemplo, sería de las formas siguientes:

$a \wedge b \wedge c$
$a \wedge (b \vee c)$

$(a \vee b) \wedge (c \vee d)$

La forma de incluir un conjunto disyuntivo es por medio de la marca <either>, que se usa del siguiente modo:

```
<either>
  <(in)active flag="Flag">
  <!--Más marcas active o inactive-->
</either>
```

Se compone de una o mas marcas <active> o <inactive>, que se evalúan por medio del operador lógico OR. De esta forma la construcción siguiente

```
<either>
  <active flag="Flag1">
  <inactive flag="Flag2">
  <inactive flag="Flag3">
</either>
```

se evaluaría como cierta si Flag1 estuviese activada, o si Flag2 o Flag3 estuvieran desactivadas.

Las marcas <either> se introducen dentro de la marca <condition> de la siguiente manera:

```
<condition>
  <(in)active flag="Flag">
  <either>
    <!--Marcas active o inactive de either-->
  </either>
  <!--Más marcas active, inactive o either-->
</condition>
```

Cabe destacar que una construcción <condition> debe tener uno o más elementos, ya sean de tipo simple (<active> o <inactive>) o de tipo compuesto (<either>). De esta forma las construcciones <condition> respetan la estructura de forma conjuntiva normal.

A.11.3. Uso y comportamiento en elementos

A continuación se describen las marcas en las que pueden utilizarse las condiciones.

En cuanto al funcionamiento, hay un patrón que repite a lo largo de las estructuras en <e-Adventure>. Cuando el motor se encuentra varias marcas con el mismo comportamiento (varias acciones del mismo tipo en un objeto, por ejemplo), pero con distintas condiciones, siempre se escogerá la primera que satisfaga la condición que lleve implicita. En caso de que algun elemento no lleve condicion, es evaluado incondicionalmente como cierto.

De esta forma, en la definición de varios casos que no son disjuntos, es recomendable establecer una prioridad en la colocación de los mismos, para permitir evaluar antes los que son más restrictivos.

Marca <resources>

Las estructuras <resources> pueden contener una condición nada mas abrir la estructura. Además todos los elementos soportan más de una estructura <resources>, de forma que se puedan condicionar los recursos de los elementos dependiendo del estado del juego.

Marca <next-scene>

Las estructuras <next-scene> permiten incluir una condición justo despues de la documentación. Algunas estructuras, como <exit>, pueden contener mas de una <next-scene>, de forma que se tenga que elegir una de ellas en funcion de las condiciones impuestas (esto permite desactivar una salida en un momento dado, o que conduzca a otra escena distinta).

Las cutscenes (tanto <videoscene> como <slidescene>) permiten tener mas de una <next-scene> al final de la estructura, permitiendo el salto a distintas escenas dependiendo de las condiciones.

A continuación se muestra un ejemplo extraído del apéndice B.2.3:

```
<exit x="750" y="0" width="50" height="400">
  <documentation>
    Salida por la derecha, fin de juego
    Se activa cuando se ha hablado con el personaje
  </documentation>
  <next-scene idTarget="Fin">
    <condition>
      <active flag="HabladoConPersonaje"/>
    </condition>
  </next-scene>
</exit>
```

Marcas <object-ref> y <character-ref>

Las dos estructuras de referencia de elementos, para objeto y personaje, permiten incluir una condición justo despues de la documentación. Esto permite controlar cuando queremos que determinados objetos y personajes aparezcan en las escenas.

A continuación se muestra un ejemplo extraído del apéndice B.2.3:

```
<character-ref idTarget="Personaje" x="100" y="200">
  <documentation>
    Personaje en el escenario
    Aparece solo al haber recogido el Objeto1
```

```
  </documentation>

  <condition>
    <active flag="CogidoObjeto1"/>
  </condition>
</character-ref>
```

Marcas de acción en objetos

Las cuatro marcas de acción permiten incluir condiciones, justo despues de la documentación. Sumado al hecho de poder definir varias acciones iguales, nos permite especificar en que circunstancias se nos permite ejecutar dichas acciones. Añadido a los efectos (capítulo A.12) podemos establecer condiciones específicas, en las cuales tienen lugar eventos especiales (como que el protagonista pronuncie una frase, se dispare una cutscene, etc...).

A continuación se muestra un ejemplo extraído del apéndice B.2.3:

```
<grab>
  <documentation>
    El objeto se puede coger
    si se ha cogido Objeto1
  </documentation>
  <condition>
    <active flag="CogidoObjeto1"/>
  </condition>
</grab>
```

Marca <conversation-ref>

Las referencias a conversaciones también cuentan con condiciones, que se pueden incluir justo después de la documentación. De esta forma podemos enlazar distintas conversaciones a un mismo personaje, que elegirá una u otra dependiendo del estado del juego.

A continuación se muestra un ejemplo extraído del apéndice B.2.3:

```
<conversation-ref idTarget="Conversacion">
  <documentation>
    Podemos tener una conversacion con el personaje
    si se han recogido los dos objetos
  </documentation>
  <condition>
    <active flag="CogidoObjeto1"/>
    <active flag="CogidoObjeto2"/>
  </condition>
</conversation-ref>
```

A.12. Efectos

La utilidad de los efectos en el motor <e-Adventure> consiste en disparar una serie de eventos en un momento determinado del juego, para dar lugar a nuevas circunstancias y cambios dentro del juego.

La estructura general de un conjunto de efectos es la siguiente:

```
<effect>
  <!--Una o más marcas independientes de efecto-->
</effect>
```

En el interior de la marca <effect> se colocan todos los efectos que queremos que se ejecuten. Es importante recordar que los efectos se ejecutan siempre en orden. El motor <e-Adventure> está diseñado de forma que no se ejecuta un efecto hasta que el efecto anterior ha sido llevado a cabo.

Las marcas que se contemplan como efectos individuales son las siguientes:

- <activate>
- <deactivate>
- <consume-object>
- <generate-object>
- <cancel-action>
- <speak-player>
- <speak-char>
- <play-sound>
- <play-animation>
- <move-player>
- <move-npc>
- <trigger-book>
- <trigger-conversation>
- <trigger-cutscene>
- <trigger-scene>

Todos estas marcas individuales de efecto se pueden combinar en cualquier orden dentro del bloque <effect>, para que el motor lleve a cabo las acciones que deseemos. De esta forma la estructura del bloque <effect> sería como sigue:

```
<effect>
  <!--Cero o más marcas con orden arbitrario:-->
  <!--activate, deactivate, consume-object, generate-object-->
  <!--cancel-action, speak-player, speak-char, play-sound-->
  <!--play-animation, move-player, move-npc, trigger-book-->
  <!--trigger-conversation, trigger-cutscene, trigger-scene-->
</effect>
```

A.12.1. Marcas de efecto

A continuación se describe el comportamiento de cada una de las marcas de efecto.

Marca <activate>

La marca <activate> tiene como fin activar un flag, su formato de uso es el siguiente:

```
<activate flag="Flag"/>
```

Donde *Flag* es el nombre de la variable que queremos activar.

Marca <deactivate>

La marca <deactivate> tiene como fin desactivar un flag, su formato de uso es el siguiente:

```
<deactivate flag="Flag"/>
```

Donde *Flag* es el nombre de la variable que queremos desactivar.

Marca <consume-object>

La marca <consume-object> tiene como fin eliminar un objeto presente en el inventario del jugador. Si se intenta disparar un efecto de este tipo, y el objeto referido no está en el inventario o ha sido ya consumido, el motor <e-Adventure> no hará nada. Su formato de uso es el siguiente:

```
<consume-object idTarget="IdObjeto"/>
```

Donde *IdObjeto* es el identificador del objeto que queremos eliminar del inventario.

Marca <generate-object>

La marca <generate-object> tiene como fin agregar un objeto al inventario del jugador. Si se intenta disparar un efecto de este tipo, y el objeto referido está ya en el inventario o ha sido consumido anteriormente, el motor <e-Adventure> no hará nada. Su formato de uso es el siguiente:

```
<generate-object idTarget="IdObjeto"/>
```

Donde *IdObjeto* es el identificador del objeto que queremos agregar al inventario.

Marca <cancel-action>

La marca <cancel-action> es una marca de efecto que se usa específicamente en algunas acciones de uso en los objetos. Tiene como fin no ejecutar la acción por defecto en una acción. Su formato de uso es el siguiente:

```
<cancel-action/>
```

La inclusión de esta marca anula el comportamiento normal en dos de las acciones que se pueden llevar a cabo con un objeto. Esto es:

- Si se incluye como efecto a una acción <grab>, el objeto no desaparecerá del escenario ni se colocará en el inventario. Si no que seguirá en el sitio que estaba antes de llevar a cabo la acción.
- Si se incluye como efecto a una acción <give-to>, el objeto no desaparecerá del inventario del protagonista, permaneciendo allí.

Marca <speak-player>

La marca <speak-player> tiene como fin que el protagonista recite una linea de diálogo. Si se colocan varias marcas <speak-player> seguidas (o intercaladas con <speak-char>) se recitarán las lineas en orden, como si de una conversación se tratara. Su formato de uso es el siguiente:

```
<speak-player>Frase a pronunciar</speak-player>
```

Marca <speak-char>

La marca <speak-char> tiene como fin que un personaje no jugador recite una linea de diálogo. Si se colocan varias marcas <speak-char> seguidas (o intercaladas con <speak-player>) se recitarán las lineas en orden, como si de una conversación se tratara. Su formato de uso es el siguiente:

```
<speak-char idTarget="IdPersonaje">Frase a pronunciar</speak-char>
```

Donde *IdPersonaje* es el identificador del personaje no jugador que pronunciará la frase.

Marca <play-sound>

La marca <play-sound> tiene como fin que el motor reproduzca un efecto sonoro determinado. Esta marca tiene dos modos de funcionamiento: el primero hace que no se ejecuten más efectos hasta que el efecto sonoro haya terminado de reproducirse, mientras que el segundo hace que los efectos siguientes se vayan ejecutando, a la vez que se reproduce el sonido. Su formato de uso es el siguiente:

```
<play-sound background="yes" uri="Ruta del archivo de audio"/>
```

Donde *background* determina si el sonido se ejecuta a la vez que otros efectos (valor *yes*) o si bloquea el flujo de efectos (valor *no*). La ruta del archivo de audio debe contener la extensión del mismo, y ser relativa a la posición del archivo que contiene la información de la aventura, para más información acerca de rutas de recursos multimedia consultar el capítulo A.10.

Marca <play-animation>

La marca <play-animation> se encarga de representar una animación en un punto de la escena interactiva que se está ejecutando actualmente. Dicha animación se ejecuta una única vez y el efecto termina, siguiendo con la ejecución del resto de los efectos en cola. Su formato de uso es el siguiente:

```
<play-animation uri="Ruta de la animacion" x="Coordenada X" y="Coordenada Y"/>
```

La ruta de archivo debe escribirse de la misma manera que las animaciones para los personajes (consultar el capítulo A.10 para más información). Los atributos x e y son la posición en la que se debe reproducir la animación, representando el punto medio en X de la imagen, y el más bajo en Y.

Marca <move-player>

La marca <move-player> tiene como fin desplazar al personaje protagonista a un punto concreto de la escena que se está ejecutando. Su utilidad es la de poder guionizar escenas no interactivas, junto con las marcas que posibilitan el lanzamiento de conversaciones. Su formato de uso es el siguiente:

```
<move-player x="Coordenada X" y="Coordenada Y"/>
```

Las coordenadas representan la posición a la que queremos trasladar al personaje protagonista. Este efecto termina de ejecutarse cuando el protagonista se encuentra en el punto indicado.

Marca <move-npc>

La marca <move-npc> tiene como fin desplazar a un personaje no jugador a un punto concreto de la escena que se está ejecutando. Este efecto complementa al efecto <move-player> para describir secuencias guionizadas. Su formato de uso es el siguiente:

```
<move-npc idTarget="idPersonaje" x="Coordenada X" y="Coordenada Y"/>
```

Donde *idPersonaje* es el personaje no jugador que queremos desplazar. Al igual que en el efecto <move-player>, las coordenadas indican el punto de destino para el personaje, y el efecto termina de ejecutarse cuando este ha llegado a la localización especificada.

Marca <trigger-book>

La marca <trigger-book> tiene como fin mostrar un libro con información escrita por pantalla. El libro se mostrará por pantalla hasta que el jugador salga del libro, o pase la última página del mismo. Su formato de uso es el siguiente:

```
<trigger-book idTarget="IdLibro"/>
```

Donde *idTarget* es el identificador del libro que se quiere mostrar en pantalla.

Marca <trigger-conversation>

La marca <trigger-conversation> tiene como fin poner en marcha una secuencia de conversación entre varios personajes. Su formato de uso es el siguiente:

```
<trigger-conversation idTarget="IdConversacion"/>
```

Donde *IdConversacion* es el identificador de la conversación que queremos ejecutar. Para más información sobre conversaciones consultar el capítulo A.8.

Marca <trigger-cutscene>

La marca <trigger-cutscene> tiene como fin disparar una secuencia de diapositivas o vídeo. Su formato de uso es el siguiente:

```
<trigger-cutscene idTarget="IdCutscene"/>
```

Donde *IdCutscene* es el identificador de la escena de diapositivas o vídeo que queremos ejecutar. Para más información sobre este tipo de escenas consultar el capítulo A.4.

Marca <trigger-scene>

La marca <trigger-scnee> tiene como fin establecer una escena interactiva para que se ejecute. En el momento de ejecución del efecto, se cambiará automáticamente la escena que se este ejecutando actualmente (ya sea interactiva o de diapositivas o video) a la escena interactiva indicada por el efecto. Su uso es el siguiente:

```
<trigger-scene idTarget="idScene" x="Coordenada X" y="Coordenada Y"/>
```

Donde *idScene* representa la escena interactiva que debe ejecutarse. Las coordenadas indican el punto en el que se colocará el protagonista cuando se realice el cambio de escena. Es importante destacar que el cambio de escena es inmediato, y que todos los efectos que vengan a continuación de una marca <trigger-scene> se ejecutarán en el contexto de la nueva escena interactiva.

A.12.2. Uso y comportamiento en elementos

A continuación se describen las distintas ubicaciones en las que pueden ser colocados conjuntos de efectos.

Marca <next-scene>

Las estructuras <next-scene> permiten incluir cadenas de efectos, justo despues de las condiciones. Los efectos incluidos se ejecutarán **antes del cambio de escena**, de forma que podemos incluir conversaciones de despedida entre cambios de escenas interactivas y parecidos. No se recomienda el uso de efectos en las marcas <next-scene> ubicadas dentro de cutscenes, ya que pueden dar problemas con el funcionamiento normal del motor <e-Adventure>.

Además del bloque normal de efectos, la estructura <next-scene> consta de otro bloque (optativo también) de efectos, que se ejecutarán **después del cambio de escena**. Esta marca tiene la misma composición que los bloques de efectos normales, pero se engloban dentro de la marca <post-effect>. Este bloque debe colocarse después del bloque de efectos normal (o dicho de otro modo, justo antes del cierre de <next-scene>).

A continuación se muestra un ejemplo extraído del capítulo B.2.3:

```
<next-scene idTarget="Escena2">
  <!--Efectos ejecutados antes del cambio de escena-->
  <effect>
    <speak-player>Me voy de la primera escena</speak-player>
  </effect>

  <!--Efectos ejecutados después del cambio de escena-->
  <post-effect>
    <speak-player>He llegado a la segunda escena</speak-player>
  </post-effect>
```

```
</next-scene>
```

Marcas de acción en objetos

Las cinco marcas de acción permiten incluir efectos, justo despues de las condiciones (o dicho de otro modo, justo antes del cierre de la marca de acción). Estos efectos se disparan cuando la acción ha sido ejecutada y cumple con sus requisitos dictaminados por medio de condiciones. Es conveniente recordar que solo se ejecutará una única estructura por cada acción realizada. En estos casos los efectos son útiles para activar flags indicando si se ha recogido algun objeto, si se ha examinado, entregado a alguien, etc. . . .

A continuación se muestran dos ejemplos extraídos del apéndice B.2.3:

```
<grab>
  <documentation>
    El objeto se puede coger
  </documentation>
  <effect>
    <activate flag="CogidoObjeto1"/>
  </effect>
</grab>

<examine>
  <documentation>
    Al examinar el objeto se
    puede leer el libro
  </documentation>
  <effect>
    <trigger-book idTarget="Libro"/>
  </effect>
</examine>
```

Marca <end-conversation>

Las marcas <end-conversation>, que cierran las conversaciones, también pueden contener efectos; estos se colocan como único elemento anidado en <end-conversation>. Suelen ser útiles para determinar cuando se ha seguido una determinada rama de la conversación, para generar y destruir objetos durante los dialogos, y otra serie de funciones.

A continuación se muestra un ejemplo extraído del apéndice B.2.3:

```
<end-conversation>
  <effect>
    <activate flag="HabladoConPersonaje"/>
  </effect>
```

```
</end-conversation>
```

Apéndice B

Código fuente

En este apéndice se recogen distintas muestras de código pertenecientes al proyecto <e-Adventure>. El código reproducido tiene dos fuentes principales: en primer lugar los archivos DTD que se encargan de regular la estructura de los ficheros XML que describen el comportamiento de las aventuras, y en segundo lugar una colección de archivos XML de demostración con información sobre descriptores, capítulos y archivos de adaptación y reglas.

B.1. Archivos DTD

En este apartado se incluyen las DTDs encargadas de validar la estructura de los archivos XML que se utilizan en las ejecuciones. Se presentan las DTDs de los descriptores, capítulos, archivos de adaptación y archivos de evaluación.

B.1.1. Descriptor de aventura

A continuación se reproduce el contenido del archivo *descriptor.dtd*.

```
<!ELEMENT
    game-descriptor (
        title, description, metadata?, configuration, contents
    )
>

<!ELEMENT title (#PCDATA)>
<!ELEMENT description ANY>
<!ELEMENT metadata (#PCDATA)>

<!ELEMENT configuration (gui)>
<!ELEMENT gui EMPTY>
<!ATTLIST gui
    type (traditional | contextual) "traditional"
```

```
    customized (yes | no) "no"
>

<!ELEMENT contents (chapter+)>
<!ELEMENT chapter (
    title, description, metadata?,
    adaptation-configuration?, assessment-configuration?
    )
>
<!ATTLIST chapter
    path CDATA #REQUIRED
>
<!ELEMENT adaptation-configuration EMPTY>
<!ATTLIST adaptation-configuration
    path CDATA #REQUIRED
>
<!ELEMENT assessment-configuration EMPTY>
<!ATTLIST assessment-configuration
    path CDATA #REQUIRED
>
```

B.1.2. Capítulo de una aventura

A continuación se reproduce el contenido del archivo *eadventure.dtd*. Esta DTD contiene la información de la estructura de las aventuras propiamente dichas.

```
<!ENTITY % conversation "tree-conversation | graph-conversation">
<!ENTITY % cutscene "videoscene | slidescene">

<!ELEMENT
    eAdventure (
        (scene | %cutscene;)+, book*, object*,
        player, character*, (%conversation;)*
    )
>
<!ELEMENT
    scene (
        documentation?, resources+, name, default-initial-position?,
        exits?, objects?, characters?
    )
>
<!ATTLIST scene
    id ID #REQUIRED
    start (yes | no) "no"
>
<!ELEMENT documentation ANY>
```

```
<!ELEMENT resources (condition?, asset+)>
<!ATTLIST resources
    id ID #IMPLIED
>
<!ELEMENT asset EMPTY>
<!ATTLIST asset
    type CDATA #REQUIRED
    uri CDATA #REQUIRED
>
<!ENTITY % position "x NMTOKEN #REQUIRED y NMTOKEN #REQUIRED">
<!ENTITY % rectangle
    "%position; width NMTOKEN #REQUIRED height NMTOKEN #REQUIRED"
>
<!ELEMENT default-initial-position EMPTY>
<!ATTLIST default-initial-position
    %position;
>
<!ELEMENT exits (exit+)>
<!ELEMENT exit (documentation?, next-scene+)>
<!ATTLIST exit
    %rectangle;
>
<!ELEMENT next-scene (condition?, effect?, post-effect?)>
<!ATTLIST next-scene
    idTarget IDREF #REQUIRED
    x NMTOKEN #IMPLIED
    y NMTOKEN #IMPLIED
>
<!ELEMENT objects (object-ref+)>
<!ELEMENT object-ref (documentation?, condition?)>
<!ATTLIST object-ref
    idTarget IDREF #REQUIRED
    %position;
>
<!ELEMENT characters (character-ref+)>
<!ELEMENT character-ref (documentation?, condition?)>
<!ATTLIST character-ref
    idTarget IDREF #REQUIRED
    %position;
>
<!ELEMENT
    videoscene (
        documentation?, resources+, name, ( end-game? | next-scene* )
    )
>
<!ATTLIST videoscene
```

```
    id ID #REQUIRED
    start (yes | no) "no"
>
<!ELEMENT slidescene (
        documentation?, resources+, name, ( end-game? | next-scene* )
    )
>
<!ATTLIST slidescene
    id ID #REQUIRED
    start (yes | no) "no"
>
<!ELEMENT end-game EMPTY>
<!ELEMENT book (documentation?, resources+, text)>
<!ATTLIST book
    id ID #REQUIRED
>
<!ELEMENT text (#PCDATA | bullet | img)*>
<!ELEMENT bullet (#PCDATA)>
<!ELEMENT img EMPTY>
<!ATTLIST img
    src CDATA #REQUIRED
>
<!ELEMENT object (
    documentation?, instance*, resources+, description, actions?)
>
<!ATTLIST object
    id ID #IMPLIED
>
<!ELEMENT instance EMPTY>
<!ATTLIST instance
    id ID #REQUIRED
>
<!ELEMENT description (name, brief, detailed)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT brief (#PCDATA)>
<!ELEMENT detailed (#PCDATA)>
<!ELEMENT actions (examine | grab | use | use-with | give-to )+>
<!ELEMENT examine (documentation?, condition?, effect?)>
<!ELEMENT grab (documentation?, condition?, effect?)>
<!ELEMENT use (documentation?, condition?, effect?)>
<!ELEMENT use-with (documentation?, condition?, effect?)>
<!ATTLIST use-with
    idTarget IDREF #REQUIRED
>
<!ELEMENT give-to (documentation?, condition?, effect?)>
<!ATTLIST give-to
```

```
    idTarget IDREF #REQUIRED
>
<!ELEMENT player (documentation?, resources+, textcolor?, description)>
<!ELEMENT character (
    documentation?, resources+, textcolor?, description, conversations?)
>
<!ATTLIST character
    id ID #IMPLIED
>
<!ELEMENT textcolor (frontcolor, bordercolor)>
<!ELEMENT frontcolor EMPTY>
<!ATTLIST frontcolor
    color CDATA #REQUIRED
>
<!ELEMENT bordercolor EMPTY>
<!ATTLIST bordercolor
    color CDATA #REQUIRED
>
<!ELEMENT conversations (conversation-ref+)>
<!ELEMENT conversation-ref (documentation?, condition?)>
<!ATTLIST conversation-ref
    idTarget IDREF #REQUIRED
>
<!ENTITY % dialogue "(speak-char|speak-player)*">
<!ENTITY % continuation "(response|end-conversation)">
<!ELEMENT tree-conversation (%dialogue;, %continuation;)>
<!ATTLIST tree-conversation
    id ID #REQUIRED
>
<!ELEMENT graph-conversation (dialogue-node | option-node)+>
<!ATTLIST graph-conversation
    id ID #REQUIRED
>
<!ELEMENT dialogue-node (
    (speak-player | speak-char)*, (child | end-conversation))
>
<!ATTLIST dialogue-node
    nodeindex CDATA #REQUIRED
>
<!ELEMENT option-node (speak-player, child)+>
<!ATTLIST option-node
    nodeindex CDATA #REQUIRED
>
<!ELEMENT child EMPTY>
<!ATTLIST child
    nodeindex CDATA #REQUIRED
```

```
>
<!ELEMENT speak-char (#PCDATA)>
<!ATTLIST speak-char
    idTarget IDREF #IMPLIED
>
<!ELEMENT speak-player (#PCDATA)>
<!ELEMENT response (option)+>
<!ELEMENT option (speak-player, %dialogue;, (%continuation; | go-back))>
<!ELEMENT go-back EMPTY>
<!ELEMENT end-conversation (effect?)>
<!ENTITY % basic-condition "(active|inactive)">
<!ELEMENT condition (%basic-condition; | either)+>
<!ELEMENT active EMPTY>
<!ATTLIST active
    flag NMTOKEN #REQUIRED
>
<!ELEMENT inactive EMPTY>
<!ATTLIST inactive
    flag NMTOKEN #REQUIRED
>
<!ELEMENT either (%basic-condition;)+>
<!ENTITY % effects "(
    activate | deactivate | consume-object | generate-object |
    cancel-action | speak-player | speak-char | play-sound |
    play-animation | move-player | move-npc | trigger-book |
    trigger-conversation | trigger-cutscene | trigger-scene)*"
>
<!ELEMENT effect (%effects;)>
<!ELEMENT post-effect (%effects;)>

<!ELEMENT activate EMPTY>
<!ATTLIST activate
    flag NMTOKEN #REQUIRED
>
<!ELEMENT deactivate EMPTY>
<!ATTLIST deactivate
    flag NMTOKEN #REQUIRED
>
<!ELEMENT consume-object EMPTY>
<!ATTLIST consume-object
    idTarget IDREF #REQUIRED
>
<!ELEMENT generate-object EMPTY>
<!ATTLIST generate-object
    idTarget IDREF #REQUIRED
>
```

```
<!ELEMENT cancel-action EMPTY>
<!ELEMENT trigger-conversation EMPTY>
<!ATTLIST trigger-conversation
    idTarget IDREF #REQUIRED
>
<!ELEMENT trigger-book EMPTY>
<!ATTLIST trigger-book
    idTarget IDREF #REQUIRED
>
<!ELEMENT play-sound EMPTY>
<!ATTLIST play-sound
    background (yes | no) "yes"
    uri CDATA #REQUIRED
>
<!ELEMENT play-animation EMPTY>
<!ATTLIST play-animation
    uri CDATA #REQUIRED
    x NMTOKEN #REQUIRED
    y NMTOKEN #REQUIRED
>
<!ELEMENT move-player EMPTY>
<!ATTLIST move-player
    x NMTOKEN #REQUIRED
    y NMTOKEN #REQUIRED
>
<!ELEMENT move-npc EMPTY>
<!ATTLIST move-npc
    idTarget IDREF #REQUIRED
    x NMTOKEN #REQUIRED
    y NMTOKEN #REQUIRED
>
<!ELEMENT trigger-cutscene EMPTY>
<!ATTLIST trigger-cutscene
    idTarget IDREF #REQUIRED
>
<!ELEMENT trigger-scene EMPTY>
<!ATTLIST trigger-scene
    idTarget IDREF #REQUIRED
    x NMTOKEN #REQUIRED
    y NMTOKEN #REQUIRED
>
```

B.1.3. Archivo de adaptación

A continuación se reproduce el contenido del archivo *adaptation.dtd*.

```
<!ELEMENT adaptation (initial-state?, adaptation-rule*)>
<!ELEMENT initial-state (initial-scene?, (activate | deactivate)*)>
<!ELEMENT adaptation-rule (description, uol-state, game-state)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT uol-state (property*)>
<!ELEMENT property EMPTY>
<!ATTLIST property
    id NMTOKEN #REQUIRED
    value NMTOKEN #REQUIRED
>
<!ELEMENT game-state (initial-scene?, (activate | deactivate)*)>
<!ELEMENT initial-scene EMPTY>
<!ATTLIST initial-scene
    idTarget NMTOKEN #REQUIRED
>
<!ELEMENT activate EMPTY>
<!ATTLIST activate
    flag NMTOKEN #REQUIRED
>
<!ELEMENT deactivate EMPTY>
<!ATTLIST deactivate
    flag NMTOKEN #REQUIRED
>
```

B.1.4. Archivo de evaluación

A continuación se reproduce el contenido del archivo *assessment.dtd*.

```
<!ELEMENT assessment-rules (assessment-rule*)>
<!ELEMENT assessment-rule (concept, condition, effect)>
<!ATTLIST assessment-rule
    id ID #REQUIRED
    importance ( verylow | low | normal | high | veryhigh ) #REQUIRED
>

<!ELEMENT concept (#PCDATA)>

<!ENTITY % basic-condition "(active|inactive)">
<!ELEMENT condition (%basic-condition; | either)+>
<!ELEMENT active EMPTY>
<!ATTLIST active
    flag NMTOKEN #REQUIRED
>
<!ELEMENT inactive EMPTY>
<!ATTLIST inactive
    flag NMTOKEN #REQUIRED
```

```
>
<!ELEMENT either (%basic-condition;)+>

<!ELEMENT effect (set-text?, set-property*)>
<!ELEMENT set-text (#PCDATA)>
<!ELEMENT set-property EMPTY>
<!ATTLIST set-property
    id NMTOKEN #REQUIRED
    value NMTOKEN #REQUIRED
>
```

B.2. Archivos XML

En este apartado se incluyen archivos XML de demostración con ejemplos de descriptores, capítulos y archivos de configuración.

B.2.1. Descriptor de aventura

A continuación se reproduce el código de un descriptor de aventura de ejemplo. Dicho descriptor recibiría el nombre de *descriptor.xml*, y almacenaría referencias a dos capítulos que conformarían la aventura. Además, el primer capítulo tendrá especificados archivos de adaptacion y evaluación:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE game-descriptor SYSTEM "descriptor.dtd">

<game-descriptor>

  <title>Título de la aventura (A mostrar en el selector)</title>
  <description>
    Descripción de la aventura
  </description>

  <configuration>
    <gui type="contextual"/>
  </configuration>

  <contents>
    <chapter path="capitulo1.xml">
      <title>Primer capítulo</title>
      <description>Primer capítulo de la aventura</description>
      <adaptation-configuration path="adapt-cap1.xml" />
      <assessment-configuration path="assess-cap1.xml" />
    </chapter>
```

```
    <chapter path="capitulo2.xml">
      <title>Segundo capítulo</title>
      <description>Segundo capítulo de la aventura</description>
    </chapter>
  </contents>
</game-descriptor>
```

B.2.2. Aventura simple

A continuación se detalla el código completo de una aventura de prueba, en su forma más sencilla. Esta aventura ha sido utilizada en la primera parte del manual de autor, para ilustrar el uso de los elementos de forma básica. Se han añadido las estructuras <resources> para que el conjunto sea funcional, siendo esta la única diferencia con el código mostrado en los extractos.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE eAdventure SYSTEM "eAdventure.dtd">
<eAdventure>

  <scene id="Escena">
    <documentation>
      Documentación de la escena
    </documentation>

    <resources>
      <asset type="background" uri="prueba/Escena.jpg"/>
      <asset type="bgmusic" uri="prueba/Escena.mid"/>
    </resources>

    <name>Escena</name>

    <exits>
      <exit x="750" y="0" width="50" height="400">
        <documentation>
          Salida por la derecha
        </documentation>
        <next-scene idTarget="Fin"/>
      </exit>
    </exits>

    <objects>
      <object-ref idTarget="Objeto" x="185" y="220">
        <documentation>
          Objeto en el escenario
        </documentation>
```

```
      </object-ref>
    </objects>

    <characters>
      <character-ref idTarget="Personaje" x="100" y="200">
        <documentation>
          Personaje en el escenario
        </documentation>
      </character-ref>
    </characters>
  </scene>

  <slidescene id="Comienzo" start="yes">
    <documentation>
      Escena de diapositivas con el comienzo de la aventura
    </documentation>

    <resources>
      <asset type="slides" uri="prueba/Comienzo"/>
    </resources>

    <name>Comienzo</name>

    <next-scene idTarget="Escena"/>
  </slidescene>

  <slidescene id="Fin">
    <documentation>
      Escena de diapositivas con el fin de la aventura
    </documentation>

    <resources>
      <asset type="slides" uri="prueba/Fin"/>
    </resources>

    <name>Fin</name>

    <end-game/>
  </slidescene>

  <book id="Libro">
    <documentation>
      Ejemplo de libro
    </documentation>

    <resources>
```

```
      <asset type="background" uri="prueba/Libro.jpg"/>
    </resources>

    <text>
      Texto del libro
    </text>
  </book>

    <object id="Objeto">
    <documentation>
      Objeto presente en la aventure
    </documentation>

    <resources>
      <asset type="image" uri="prueba/Objeto.png"/>
      <asset type="icon" uri="prueba/Objeto.png"/>
    </resources>

    <description>
      <name>
        Objeto
      </name>
      <brief>
        Descripcion simple
      </brief>
      <detailed>
        Descripcion detallada
      </detailed>
    </description>

    <actions>
        <grab>
        <documentation>
          El objeto se puede coger
        </documentation>
      </grab>

      <give-to idTarget="Personaje">
        <documentation>
          El objeto se puede entregar a Personaje
        </documentation>
      </give-to>
    </actions>
  </object>

  <player>
```

```
    <documentation>
      Documentacion del jugador
    </documentation>

    <resources>
      <asset type="standup" uri="prueba/PlayerStandBack"/>
      <asset type="standdown" uri="prueba/PlayerStandFront"/>
      <asset type="standright" uri="prueba/PlayerStandFront"/>
      <asset type="speakup" uri="prueba/PlayerStandFront"/>
      <asset type="speakdown" uri="prueba/PlayerStandFront"/>
      <asset type="speakright" uri="prueba/PlayerStandFront"/>
      <asset type="useright" uri="prueba/PlayerStandFront"/>
      <asset type="walkup" uri="prueba/PlayerStandFront"/>
      <asset type="walkdown" uri="prueba/PlayerStandFront"/>
      <asset type="walkright" uri="prueba/PlayerWalkRight"/>
    </resources>

    <description>
      <name>
        Jugador
      </name>
      <brief>
        Descripcion breve
      </brief>
      <detailed>
        Descripcion detallada
      </detailed>
    </description>
  </player>

  <character id="Personaje">
    <documentation>
      Documentacion del personaje
    </documentation>

    <resources>
      <asset type="standup" uri="prueba/StanStandingFront"/>
      <asset type="standdown" uri="prueba/StanStandingFront"/>
      <asset type="standright" uri="prueba/StanStandingFront"/>
      <asset type="speakup" uri="prueba/StanStandingFront"/>
      <asset type="speakdown" uri="prueba/StanStandingFront"/>
      <asset type="speakright" uri="prueba/StanStandingFront"/>
      <asset type="useright" uri="prueba/StanStandingFront"/>
      <asset type="walkup" uri="prueba/StanStandingFront"/>
      <asset type="walkdown" uri="prueba/StanStandingFront"/>
      <asset type="walkright" uri="prueba/StanStandingFront"/>
```

```
    </resources>

    <description>
      <name>
        Personaje
      </name>
      <brief>
        Descripcion breve
      </brief>
      <detailed>
        Descripcion detallada
      </detailed>
    </description>

    <conversations>
      <conversation-ref idTarget="Conversacion">
        <documentation>
          Podemos tener una conversacion con el personaje
        </documentation>
      </conversation-ref>
    </conversations>
  </character>

  <tree-conversation id="Conversacion">
    <speak-player>Hola</speak-player>
    <speak-char>Hola, ¿que tal?</speak-char>
    <response>
      <option>
        <speak-player>Bien</speak-player>
        <speak-char>Me alegro</speak-char>
        <end-conversation/>
      </option>
      <option>
        <speak-player>Mal</speak-player>
        <speak-char>Lo siento</speak-char>
        <end-conversation/>
      </option>
    </response>
  </tree-conversation>
</eAdventure>
```

B.2.3. Aventura compleja

A continuación se detalla una ampliación de la aventura expuesta anteriormente. Esta nueva aventura cuenta con dos escenas interactivas, dos cutscenes, dos objetos, un libro,

un personaje y una conversación. Se han establecido varias condiciones y efectos. Las flags utilizadas son las siguientes:

- *CogidoObjeto1* - Se activa cuando el Objeto1 ha sido recogido.
- *CogidoObjeto2* - Se activa cuando el Objeto2 ha sido recogido.
- *HabladoConPersonaje* - Se activa cuando se ha hablado con Personaje.

Se han establecido las siguientes condiciones:

- No se puede recoger Objeto2 hasta que no se ha recogido Objeto1.
- Personaje no aparece en Escena2 si Objeto1 no ha sido recogido.
- Se puede hablar con Personaje tras recoger Objeto1 y Objeto2.

En cuanto a los efectos, se han definido los siguientes:

- Se muestra el libro por pantalla al examinar Objeto1.
- Se activa *CogidoObjeto1* al recoger Objeto1.
- Se activa *CogidoObjeto2* al recoger Objeto2.
- Se activa *HabladoConPersonaje* tras hablar con Personaje y contestar "Si".

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE eAdventure SYSTEM "eadventure.dtd">
<eAdventure>

  <scene id="Escena1">
    <documentation>
      Documentación de la escena
    </documentation>

    <resources>
      <asset type="background" uri="prueba/Escena1.jpg"/>
      <asset type="bgmusic" uri="prueba/Escena1.mid"/>
    </resources>

    <name>Escena 1</name>

    <exits>
      <exit x="750" y="0" width="50" height="400">
        <documentation>
          Salida por la derecha, conduce a la segunda escena
        </documentation>
        <next-scene idTarget="Escena2">
```

```
        <!--Efectos ejecutados antes del cambio de escena-->
        <effect>
          <speak-player>Me voy de la primera escena</speak-player>
        </effect>

        <!--Efectos ejecutados después del cambio de escena-->
        <post-effect>
          <speak-player>He llegado a la segunda escena</speak-player>
        </post-effect>
      </next-scene>
    </exit>
  </exits>

  <objects>
    <object-ref idTarget="Objeto1" x="185" y="220">
      <documentation>
        Objeto1 en el escenario
      </documentation>
    </object-ref>
  </objects>
</scene>


<scene id="Escena2">
  <documentation>
    Documentación de la escena
  </documentation>

  <resources>
    <asset type="background" uri="prueba/Escena2.jpg"/>
    <asset type="bgmusic" uri="prueba/Escena2.mid"/>
  </resources>

  <name>Escena 2</name>

  <exits>
    <exit x="0" y="0" width="50" height="400">
      <documentation>
        Salida por la izquierda, conduce a la primera escena
      </documentation>
      <next-scene idTarget="Escena1"/>
    </exit>

    <exit x="750" y="0" width="50" height="400">
      <documentation>
        Salida por la derecha, fin de juego
```

```
        Se activa cuando se ha hablado con el personaje
      </documentation>
      <next-scene idTarget="Fin">
        <condition>
          <active flag="HabladoConPersonaje"/>
        </condition>
      </next-scene>
    </exit>
  </exits>

  <objects>
    <object-ref idTarget="Objeto2" x="185" y="220">
      <documentation>
        Objeto2 en el escenario
      </documentation>
    </object-ref>
  </objects>

  <characters>
    <character-ref idTarget="Personaje" x="100" y="200">
      <documentation>
        Personaje en el escenario
        Aparece solo al haber recogido el Objeto1
      </documentation>

      <condition>
        <active flag="CogidoObjeto1"/>
      </condition>
    </character-ref>
  </characters>
</scene>


<slidescene id="Comienzo" start="yes">
  <documentation>
    Escena de diapositivas con el comienzo de la aventura
  </documentation>

  <resources>
    <asset type="slides" uri="prueba/Comienzo"/>
  </resources>

  <name>Comienzo</name>

  <next-scene idTarget="Escena1"/>
</slidescene>
```

```
<slidescene id="Fin">
  <documentation>
    Escena de diapositivas con el fin de la aventura
  </documentation>

  <resources>
    <asset type="slides" uri="prueba/Fin"/>
  </resources>

  <name>Fin</name>

  <end-game/>
</slidescene>


<book id="Libro">
  <documentation>
    Ejemplo de libro
  </documentation>

  <resources>
    <asset type="background" uri="prueba/Libro.jpg"/>
  </resources>

  <text>
    Texto del libro
  </text>
</book>


<object id="Objeto1">
  <documentation>
    Objeto presente en la aventura
  </documentation>

  <resources>
    <asset type="image" uri="prueba/Objeto1.png"/>
    <asset type="icon" uri="prueba/Objeto1.png"/>
  </resources>

  <description>
    <name>
      Objeto 1
    </name>
    <brief>
```

```
        Descripcion simple
      </brief>
      <detailed>
        Descripcion detallada
      </detailed>
    </description>

    <actions>
      <grab>
        <documentation>
          El objeto se puede coger
        </documentation>
        <effect>
          <activate flag="CogidoObjeto1"/>
        </effect>
      </grab>


      <examine>
        <documentation>
          Al examinar el objeto se
          puede leer el libro
        </documentation>
        <effect>
          <trigger-book idTarget="Libro"/>
        </effect>
      </examine>
    </actions>
  </object>


  <object id="Objeto2">
    <documentation>
      Objeto presente en la aventura
    </documentation>

    <resources>
      <asset type="image" uri="prueba/Objeto2.png"/>
      <asset type="icon" uri="prueba/Objeto2.png"/>
    </resources>

    <description>
      <name>
        Objeto 2
      </name>
      <brief>
```

```
      Descripcion simple
    </brief>
    <detailed>
      Descripcion detallada
    </detailed>
  </description>

  <actions>
    <grab>
      <documentation>
        El objeto se puede coger
        si se ha cogido Objeto1
      </documentation>
      <condition>
        <active flag="CogidoObjeto1"/>
      </condition>
      <effect>
        <activate flag="CogidoObjeto2"/>
      </effect>
    </grab>
  </actions>
</object>


<player>
  <documentation>
    Documentacion del jugador
  </documentation>

  <resources>
    <asset type="standup" uri="prueba/PlayerStandBack"/>
    <asset type="standdown" uri="prueba/PlayerStandFront"/>
    <asset type="standright" uri="prueba/PlayerStandFront"/>
    <asset type="speakup" uri="prueba/PlayerStandFront"/>
    <asset type="speakdown" uri="prueba/PlayerStandFront"/>
    <asset type="speakright" uri="prueba/PlayerStandFront"/>
    <asset type="useright" uri="prueba/PlayerStandFront"/>
    <asset type="walkup" uri="prueba/PlayerStandFront"/>
    <asset type="walkdown" uri="prueba/PlayerStandFront"/>
    <asset type="walkright" uri="prueba/PlayerWalkRight"/>
  </resources>

  <description>
    <name>
      Jugador
    </name>
```

```
      <brief>
        Descripcion breve
      </brief>
      <detailed>
        Descripcion detallada
      </detailed>
    </description>
  </player>

  <character id="Personaje">
    <documentation>
      Documentación del personaje
    </documentation>

    <resources>
      <asset type="standup" uri="prueba/StanStandingFront"/>
      <asset type="standdown" uri="prueba/StanStandingFront"/>
      <asset type="standright" uri="prueba/StanStandingFront"/>
      <asset type="speakup" uri="prueba/StanStandingFront"/>
      <asset type="speakdown" uri="prueba/StanStandingFront"/>
      <asset type="speakright" uri="prueba/StanStandingFront"/>
      <asset type="useright" uri="prueba/StanStandingFront"/>
      <asset type="walkup" uri="prueba/StanStandingFront"/>
      <asset type="walkdown" uri="prueba/StanStandingFront"/>
      <asset type="walkright" uri="prueba/StanStandingFront"/>
    </resources>

    <description>
      <name>
        Personaje
      </name>
      <brief>
        Descripcion breve
      </brief>
      <detailed>
        Descripcion detallada
      </detailed>
    </description>

    <conversations>
      <conversation-ref idTarget="Conversacion">
        <documentation>
          Podemos tener una conversacion con el personaje
          si se han recogido los dos objetos
        </documentation>
        <condition>
```

```
          <active flag="CogidoObjeto1"/>
          <active flag="CogidoObjeto2"/>
        </condition>
      </conversation-ref>
    </conversations>
  </character>


  <tree-conversation id="Conversacion">
    <speak-player>Hola</speak-player>
    <speak-char idTarget="Personaje">Hola, ¿quieres salir?</speak-char>
    <response>
      <option>
        <speak-player>Si</speak-player>
        <speak-char idTarget="Personaje">Puedes marcharte</speak-char>
        <end-conversation>
          <effect>
            <activate flag="HabladoConPersonaje"/>
          </effect>
        </end-conversation>
      </option>
      <option>
        <speak-player>No</speak-player>
        <speak-char idTarget="Personaje">
          Cuando quieras poder
          salir habla conmigo
        </speak-char>
        <end-conversation/>
      </option>
    </response>
  </tree-conversation>
</eAdventure>
```

B.2.4. Archivos de adaptación

A continuación se reproduce el código de dos archivos de adaptación de ejemplo. El primero lleva a cabo una adaptación inicial, sin necesidad de conexión con un servidor LMS. El segundo define una serie de reglas de adaptación, para cuando reciba datos del servidor LMS poder traducirlos en cambios en el comportamiento del juego.

El primer archivo define únicamente una adaptación inicial, estableciendo el valor del flag *go-level-2* a true.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE adaptation SYSTEM "adaptation.dtd">
```

```
<adaptation>

  <initial-state>
    <activate flag="go-level-2">
  </initial-state>

</adaptation>
```

El segundo archivo define una regla de adaptación, por la cual se establece el flag *MakeItHarder* a true en el caso que el servidor LMS envíe información indicando que el valor de la propiedad *make-it-harder* es igual a 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE adaptation SYSTEM "adaptation.dtd">

<adaptation>

  <adaptation-rule>
    <description>
      This is an advanced player - make it harder
    </description>
    <uol-state>
      <property id="make-it-harder" value="1">
    </uol-state>
    <game-state>
      <activate flag="MakeItHarder">
    </game-state>
  </adaptation-rule>

</adaptation>
```

B.2.5. Archivo de evaluación

A continuación se reproduce el contenido de un archivo de evaluación de prueba. Dicho archivo define un conjunto de cinco reglas, cuatro con importancia normal y una con importancia alta. Las tres primeras reflejan en qué momento el protagonista ha sido capaz de realizar tres acciones concretas (mezclar chocolate blanco, con leche y negro respectivamente); la cuarta regla indica si el jugador ha llegado a producir alguna mezcla con los ingredientes que no de lugar a ningún tipo de chocolate, y la quinta indica en qué momento ha conseguido el jugador terminar de elaborar los tres tipos de chocolate.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE assessment-rules SYSTEM "assessment.dtd">

<assessment-rules>
```

```
<assessment-rule id="MadeWhiteChocolate" importance="normal">
  <concept>New mixture<concept>
  <condition>
    <active flag="HasWhiteChocolate">
  </condition>
  <effect>
    <set-text>
      The student made white chocolate successfully
    </set-text>
  </effect>
</assessment-rule>

<assessment-rule id="MadeMilkChocolate" importance="normal">
  <concept>New mixture<concept>
  <condition>
    <active flag="HasMilkChocolate">
  </condition>
  <effect>
    <set-text>
      The student made milk chocolate successfully
    </set-text>
  </effect>
</assessment-rule>

<assessment-rule id="MadeDarkChocolate" importance="normal">
  <concept>New mixture<concept>
  <condition>
    <active flag="HasDarkChocolate">
  </condition>
  <effect>
    <set-text>
      The student made dark chocolate successfully
    </set-text>
  </effect>
</assessment-rule>

<assessment-rule id="MadeWrongChocolate" importance="normal">
  <concept>New mixture<concept>
  <condition>
    <active flag="TriedWrongMixture">
  </condition>
  <effect>
    <set-text>
      The student tried a wrong mixture
    </set-text>
```

```
    </effect>
  </assessment-rule>

  <assessment-rule id="MadeAllChocolates" importance="high">
    <concept>Chocolate test passed<concept>
    <condition>
      <active flag="HasWhiteChocolate">
      <active flag="HasMilkChocolate">
      <active flag="HasDarkChocolate">
    </condition>
    <effect>
      <set-text>
        The student made all three chocolates successfully
      </set-text>
    </effect>
  </assessment-rule>

</assessment-rules>
```

Apéndice C

Bibliografía

- **http://java.sun.com/docs/codeconv/**
  Estándar de codificación en Java.
- **Developing Games in Java**
  David Brackeen, Bret Barker, Laurence Vanhelsuwé.
  New Riders Publishing.
  ISBN 1-5927-3005-1
- **A Documental Approach to Adventure Game Development**
  Moreno-Ger, P., Sierra, J.L., Martínez-Ortiz, I., Fernández-Manjón, B.
  Science of Computer Programming.
  In press (DOI 10.1016/j.scico.2006.07.003). 2007
- **Production and Maintenance of Content-Intensive Videogames: A Document-Oriented Approach**
  Martínez-Ortiz, I., Moreno-Ger, P., Sierra, J.L., Fernández-Manjón, B.
  Proceedings of 3rd International Conference on Information Technology: New Generations, Las Vegas, USA, (IEEE Society Press). 2006
- **Production and Deployment of Educational Videogames as Assessable Learning Objects**
  Martínez-Ortiz, I., Moreno-Ger, P., Sierra, J.L., Fernández-Manjón, B.
  In Proc. of the First European Conference on Technology Enhanced Learning, Crete, Greece.
  Lecture Notes in Computer Science 4227 (Springer). 2006
- **Language-Driven Development of Videogames: the <e-Game> Experience**
  Moreno-Ger, P., Martínez-Ortiz, I., Sierra, J.L., Fernández-Manjón, B.
  In Proc. of 5th International Conference on Entertainment Computing, Cambridge, UK.
  Lecture Notes in Computer Science 4161 (Springer). 2006

- **The <e-Game> Project: Facilitating the development of educational adventure games**

  Moreno-Ger, P., Martínez-Ortiz, I., Fernández-Manjón, B.

  In Proc. of Cognition and Exploratory Learning in the Digital age (CELDA 2005), pages 353-358, Porto, Portugal (IADIS). 2005

Apéndice D

Glosario

- **e-Learning**. Aprendizaje asistido por tecnologías de la información. El e-Learning facilita la creación, adopción y distribución de contenidos, así como la adaptación del ritmo de aprendizaje y la disponibilidad de las herramientas de aprendizaje independientemente de límites horarios o geográficos.
- **LMS (Learning Management System)**. Es un software que, instalado en un servidor, sirve para administrar, distribuir y controlar las actividades de formación presencial o e-Learning de una organización.
- **Aventura gráfica**. Género de videojuegos basado en la resolución de puzzles y situaciones por medio de la interacción con personajes y objetos. El control en este tipo de juegos se lleva a cabo con ratón de forma casi exclusiva.
- **HUD (Head-Up Display)**. Información que se muestra por pantalla en todo momento en la ejecución de un videojuego, suele constar de una serie de iconos y números que indican el estado de la partida.
- **Contextual**. En términos de interfaces, un interfaz contextual se define como aquel que permite llevar a cabo unas u otras opciones dependiendo del tipo de elemento sobre el que se quieren realizar.