TRABAJO DE FIN DE CARRERA

Título del TFC: Learning Maze (Juego de Aprendizaje)

Titulación: Ingeniería Técnica de Telecomunicación, especialidad Telemática

Autor: Mario Viktorov Mechoulam Nikolaeva

Director: Juan Hernández Serrano

Fecha: 22 de julio de 2009

A mi familia, por ayudarme a llegar hasta aquí.
A Zoe, por aguantarme y entenderme.
A mis hermanos, por mostrarme el camino.
A los amigos, por brindarme esta oportunidad.

Título: Learning Maze (Juego de Aprendizaje)

Autor: Mario Viktorov Mechoulam Nikolaeva

Director: Juan Hernández Serrano

Fecha: 22 de julio de 2009

Resumen

Actualmente, el mundo tecnológico avanza a pasos agigantados en comparación frente a cualquier otra industria. Esta situación nos permite innovar y buscar continuamente maneras más eficientes de realizar nuestras tareas, aunque no siempre sea fácil estar al día pese a disponer de los recursos. En este TFC se pretende mostrar un ejemplo de ello, a la vez que se crea algo útil.

El teléfono móvil es más que un dispositivo para llamar y mandar mensajes: es una herramienta con incesantes capacidades emergentes, las cuales se van a aprovechar como gestor de almacenamiento de información, sustituyendo el papel y sus inconvenientes; y como plataforma de juego y aprendizaje. Se ha creado una aplicación que permite al usuario introducir material didáctico de cualquier tipo y posteriormente aprenderlo, mientras está entretenido jugando por unos mapas laberínticos. El tiempo es oro, y precisamente por eso se ha decidido implementar en la aplicación un método de aprendizaje de un conocido científico que ha estudiado el campo de la enseñanza, logrando optimizar el tiempo que pasaremos estudiando delante de la pantalla. Al mismo tiempo, se han utilizado nuevas librerías gráficas y lenguajes de programación portables, dando soporte a ambos a la vez que se trabaja en ampliar el conocimiento.

El resultado, como se podrá ver a continuación, es altamente satisfactorio. Se suministra una herramienta de estudio eficaz a la par que entretenida, la cual, a pesar de sufrir de algunos defectos de implementación, muestra decenas de funcionalidades y presenta incontables posibilidades de expansión. Una aplicación para todas las edades y sexos, que puede permitirse el lujo de subrayar que es capaz de ayudar en todos los campos de formación; y es que, en tiempos de crisis no hay mejor inversión que la educación.

Title: Learning Maze (Juego de Aprendizaje)
Author: Mario Viktorov Mechoulam Nikolaeva
Director: Juan Hernández Serrano
Date: 22nd July, 2009

Overview

Nowadays, technology market grows huge every year in comparison with any other industry. The current situation allows us to innovate and continuously search for more efficient ways of carrying out our work, though it is not always that easy even having the needed resources. This TFC will do its best in order to show an example implementation of the former, while trying to create something useful.

The cell phone is more than a call and send-sms terminal: it is a tool with unending emerging features, which are going to be exploited as an information storage manager, substituting paper and its inconveniences; and as a game and learning platform. An application has been created, such that it allows the user to input any kind of educational information and learn it afterwards, while he is playing through labyrinthine maps. Time is money, and itself a reason for the implementation of a learning method made up by a known scientist who has studied the teaching field, achieving an optimization of the time spent in front of the screen. At the same time, new graphic libraries and portable programming languages have been used, helping support both while working on broaden our knowledge.

The result, as can be seen through this document, is highly satisfactory. An efficient and at the same time entertaining studying tool is being supplied, which, albeit lacking a feature or two and having some flaws, arises providing tens of configurable characteristic and innumerable expansion possibilities. An application for all ages and sexes, capable of helping in every educational purpose; and so it is that, in times of crisis there is no better investment than education.

ÍNDICE

INTRODUCCIÓN

Se ha decidido estructurar el trabajo en tres capítulos principales, más uno de pensamientos al final. En esta introducción se hablará de la idea del proyecto y las causas que nos han llevado a realizarlo. A continuación, en el primer capítulo, trataremos de introducir al lector a los conceptos previos necesarios para la compresión de este TFC. El segundo capítulo intentará ser una especie de manual de usuario, se recorrerán las funcionalidades más importantes y se entrará en los detalles más llamativos. En el tercer capítulo se abordará el tema más técnico, sumergiendo de lleno al lector por las vías cruzadas de multitud de sus componentes, intentando alcanzar la máxima profundidad y comprensión en cuanto al modo de funcionar del programa y su motor de juego. Para acabar, dedicaremos un capítulo a las conclusiones, las líneas futuras, limitaciones técnicas y efectos medioambientales de este proyecto.

Motivación/Objetivo del TFC

Partimos de la idea de desarrollar un software, cuyas premisas son: que sea apto para todas las edades, la posibilidad de utilizarlo en prácticamente cualquier lugar, que ayude al aprendizaje de cualquier temática y, sobre todo, que sea entretenido.

La elección más importante ha sido considerar el dispositivo móvil como soporte de la aplicación. Este aparato ya no es ningún misterio para las personas en la actualidad; unos pocos botones y una cruceta digital hacen de su manejo algo muy intuitivo. Por otra parte, el móvil siempre permanece con nosotros, lo cual nos permite aprovechar todos estos ratos “muertos” de espera (en el metro, el tren, el aeropuerto, en la playa, mientras aguardamos la llegada de alguien...) que de lo contrario, simplemente se desperdician. El móvil sustituye ese libro o revista que llevamos para esos momentos, pero sin tener que cargar ningún peso extra. Este software pretende sustituir nuestros juegos del móvil, que utilizamos en aquellos momentos, por no sólo diversión, sino también aprendizaje.

El programa que presentaremos a continuación, puede utilizarse para estudiar cualquier temática: podemos ayudarnos a aprender un idioma, a repasar para el último examen de la carrera de medicina, para mejorar nuestras matemáticas, conocimientos de música o arte... las posibilidades son infinitas.
Por último, aprender debe ser divertido, así lo hemos comprobado con el tirón actual de los juegos educativos de la marca Nintendo por ejemplo. Esta ventaja permite llegar a un público más amplio, sin hacer diferencia entre sexos ni edades, y desmintiendo el mito de que los juegos educativos son un “rollo”. Por esta razón, se ha decidido afrontar un reto aún mayor y convertir toda la aplicación en un juego.

CAPÍTULO 1. CONCEPTOS FUNDAMENTALES

Este primer comentaremos brevemente las tecnologías utilizadas y sus diferencias a la hora de implementarlas y finalizaremos con una sección muy importante, que sienta las bases de conocimientos necesarios para entender todo lo que viene a continuación.

1.1. Tecnología utilizada [Java, JME, CDC, CLDC]

Java es un lenguaje de alto nivel que fue creado a mediados de los 90 por Sun Microsystems. A pesar de que no fue diseñado para ello, sus usos e implementaciones crecieron rápidamente en diversas áreas, para lo cual fueron creadas distintas ediciones que se amoldaran a las necesidades concretas de las plataformas que las ejecutaban. Para más información sobre Java **[1]**.

Una de estas distribuciones es la edición micro de Java (*Java 2 Micro Edition* – J2ME, ahora conocida simplemente como JME), ideada para dispositivos con capacidades computacionales y gráficas reducida; entre los cuales se encuentran los dispositivos móviles. JME tiene funcionalidades reducidas respecto a sus hermanos mayores, posee una interfaz de programación de aplicaciones (a*pplication program interface* – API) adaptada y una máquina virtual de kilobyte (*kilobyte virtual machine* – KVM) que se encarga de interpretar el código. Para más información sobre todo esto **[2]**.

El perfil de información móvil del dispositivo (*Mobile Information Device Profile* – MIDP) son las librerías y conjunto de herramientas que proporcionan soporte a nuestro código. En 2009 existen dos versiones, la segunda de las cuales incorpora numerosas mejoras utilizadas en este proyecto como la pantalla a color, capacidad de reproducción de sonidos, multitud de mejoras para distintos tipos de comunicación o la API de multimedia para la reproducción de video (MMAPI). Toda aquella aplicación que se ha realizado con este perfil recibe el nombre de MIDlet y distintas aplicaciones pueden coexistir entre ellas; adicionalmente su estado siempre será uno de los tres: activo (ejecutándose), pausado o destruido. Por último, los MIDlets se empaquetan en una *suite* que es formada por dos archivos con extensión .JAD y .JAR; el jar contiene el código compilado a ejecutar, mientras el archivo descriptor Java (*Java Archive Descriptor* - JAD) contiene información sobre la aplicación, rutas, tamaño y decenas de etiquetas opcionales más. Lamentablemente, debido a la extensión de esta memoria, no podemos entretenernos más entrando en detalles en esta sección; para más información se puede consultar **[3]**.

1.2. Bases del programa

En esta sección se explica primero el método de aprendizaje SRS basado en los estudios de Sebastian Leitner y luego enseñamos la implementación más común, las *flashcards*. A continuación presentamos las librerías gráficas utilizadas para el desarrollo de este TFC y para acabar se comentan los conceptos más relevantes para un correcto entendimiento del juego y la generación del contenido.

1.2.1. Sebastian Leitner y el SRS

Está bien, pero ¿cómo es el proceso de aprendizaje? Sin duda muchas personas han repetido hasta la saciedad textos memorizados, han escrito una y otra vez términos en otros idiomas... pero luego transcurre un mes y se olvidan por completo. ¿Cómo es posible? Usaremos el método del científico alemán Sebastian Leitner, detallado en su libro "*How to learn to learn*".

Este método, parcialmente implementado en las *flashcards* (ver siguiente punto), consiste en la repetición de los conceptos cuya respuesta tenemos menos afianzada. Por poner un ejemplo, si respondemos bien a una pregunta, definición o cualquier otro **desafío** que estemos estudiando, decrecerá la frecuencia con la que este aparece durante nuestros periodos de estudio. Futuros aciertos tendrán el mismo resultado, y lo que conseguiremos al final es que, acertando un desafío varias veces, este pase de aparecer a diario, a aparecer cada pocos días, luego semanalmente, y por último, que aparezca cada mes.

En estos momentos, el lector probablemente se esté preguntando de qué manera ayudará este método a su aprendizaje y en qué se diferencia del método tradicional. La respuesta es sencilla y se divide principalmente en dos ventajas: 1) mejoras en el uso de la memoria inmediata, la memoria a corto plazo y la memoria a largo plazo; y 2) la optimización del tiempo dedicado al estudio.

La manera en la que funciona el cerebro humano dicta que, si sabemos la respuesta a un desafío, probablemente recordemos esa respuesta durante los próximos minutos (es lo que se denomina **memoria inmediata**). Entonces, si un usuario sabe la respuesta a una pregunta, no sirve de nada seguir haciéndosela, al menos por el momento. Sin embargo, es conveniente verificar la respuesta un par de días después de ser acertada, es decir, verificar si el mismo usuario ha sido capaz de almacenarla en su memoria (lo que se denomina **memoria a corto plazo**). Si la persona la vuelve a acertar, esto quiere decir que no la olvidará fácilmente durante los próximos días; hemos dado un paso más hacia el aprendizaje permanente, que culmina retando de nuevo al usuario al cabo de unas semanas, o un mes. Si éste continúa sabiendo la respuesta, perfecto: acabamos de almacenar un concepto más en la **memoria a largo plazo**. Por supuesto ahí no queda la cosa, cada cierto tiempo es altamente recomendable hacer revisiones de los conceptos

aprendidos para evitar su olvido o deterioro, algo de lo cuál también se encargará este *software*.

La correcta gestión de los intervalos de aparición de los desafíos, tal y como se ha explicado anteriormente, permite optimizar el **tiempo dedicado al estudio**. Esto quiere decir que el usuario no perderá tiempo, repitiendo una y otra vez conceptos que ya conoce y aprovechará al máximo su tiempo de estudio repasando lo que de verdad necesita aprender.

A todo lo anterior se lo conoce comúnmente como un sistema/*software* de repetición espaciada (*Spaced Repetiotion Software/System* - SRS). Es un gestor de las repeticiones que se irán dando a lo largo de un periodo temporal, para de esta manera, evitar cansar al usuario con repeticiones excesivas de desafíos que ya conoce.

1.2.2. Flashcard

Se conoce como *flashcards* a un conjunto de cartas cuyo contenido es información a estudiar, repartida en ambos lados a modo de pregunta-respuesta/s válidas. Escribimos la pregunta o desafío en una cara y en la otra figurará la o las respuestas correctas. El temario a estudiar puede ser de virtualmente cualquier tipo y existen *flashcards* de hasta tres caras, conteniendo una de ellas ayuda, información adicional o pistas para resolver el desafío. Los inconvenientes de este material son la incomodidad de llevar las cartas encima, que crece a medida que lo hace el temario, la relación precio-cantidad de muchas de las temáticas ofrecidas y el desgaste del material con el uso y el tiempo.

1.2.3. LWUIT

El *moto* de Java, "*write once, run anywhere*", es difícil de realizar cuando estamos hablando de una aplicación JME para terminales móviles. La razón de ello es la tremenda fragmentación del mercado para estos dispositivos, cada fabricante tiene su propia manera de implementar la máquina virtual de java, las funcionalidades, teclas... eso sin contar con el código nativo de cada compañía.

Por otra parte, la interfaz gráfica de alto nivel que nos provee Sun es poco atractiva y, desde mi punto de vista, inaceptable para una aplicación comercial. Apenas tenemos opción de configuración o diseño de los componentes que queramos usar. El Canvas, es la interfaz de bajo nivel disponible en el JME y con ella sí se pueden conseguir unos resultados realmente profesionales. Sin embargo, no todo son facilidades, lograrlo requiere muchas horas de trabajo, una gran organización, abstracción y paciencia; hay que dibujar todos y cada uno de los componentes, botones, *listeners*, y mil cosas más. La compañía media tarda más de dos meses en preparar sólo la interfaz gráfica para su

aplicación, y si su aplicación no era una interfaz gráfica... es un tiempo invertido que no suele ser remunerado.

La solución viene de la mano de Sun y su, relativamente moderna*, librería "conjunto de herramientas livianas para la interfaz de usuario" (*LightWeight User Interface Toolkit* - LWUIT). LWUIT es una librería gráfica que simula la estructura y el estilo de Swing, es *open source* y permite al programador centrarse en el diseño de su *software* y no en el entorno de usuario. También pone especial atención en solucionar la fragmentación de dispositivos, implementando alternativas para terminales de bajo nivel o que no soporten determinadas funcionalidades, sin que el desarrollador se tenga que preocupar por ello.

A efectos prácticos y a modo de ejemplo, el autor de este documento ha tardado cuatro días en hacer con LWUIT algo que había tardado dos semanas en hacer con el Canvas, y finalizó toda la interfaz gráfica en algo más de treinta días.

1.2.4. Nuestro juego y los RPG

En nuestro juego controlaremos a un personaje que va recorriendo mapas laberínticos buscando la salida, a la par que aprende la temática que él mismo se ha preparado, mediante combates aleatorios que se suceden durante la aventura.

Un juego de rol (*Role-Playing Game* – RPG) en la industria de los videojuegos es generalmente aquel en el que controlamos a uno o más personajes que van completando una serie de requisitos para vencer en la partida. Es extremadamente común en ellos la posibilidad de personalizar el personaje o partes del juego, así como la exploración de mapas laberínticos o el combate táctico. Existen muchas otras características que nuestro programa no tiene, pero sin duda es personalizable (el aprendizaje), contiene infinidad de laberintos a explorar y el combate es, en cierto sentido, táctico. Aunque realmente tratamos de definir un estilo propio, probablemente éste es el estilo al que más se asemeja, para que el lector se haga una idea.

1.2.5. Contenido procedimental

Se conoce como contenido procedimental, aquel que es el fruto de un algoritmo o procedimiento programado. Hemos escogido que todos los niveles de nuestro juego sean aleatorios y ello nos permitirá tres cosas:

* LWUIT aparece en Julio de 2008 como una demo de código cerrado. A finales de 2008 se libera su código.

1. Ahorrar espacio – un terminal móvil es un dispositivo muy delicado en términos de memoria y tamaño de la aplicación (archivo .jar). Si generamos el contenido de manera aleatoria, no necesitamos almacenar todos los datos relacionados con los mapas.

2. Evitar la monotonía – si cada mapa es único y los elementos en él son emplazados de manera completamente diferente, evitaremos que el usuario se aburra o tenga una ventaja al reconocer mapas o patrones repetidos. Sin duda, una ventaja frente a otros productos y una garantía para la competitividad entre usuarios.

3. Número de niveles – el programa puede amoldar el número de niveles al tamaño de los catálogos estudiados por el usuario; mientras siga habiendo fichas sin estudiar, el generador continuará creando mapas aleatorios. Además, nosotros no necesitaremos a un grupo de personas que desarrollen los niveles, ni decidir un número seguro de niveles prefabricados para tamaños grandes de catálogo. Sin duda, otro punto a favor frente a nuestros competidores.

1.2.6. Semilla (*Seed*)

Bien, entonces ¿cómo funciona exactamente? Tenemos un generador de números pseudo-aleatorios a las cuales nosotros llamaremos *seeds* (semillas). Esta semilla se introduce en el punto de entrada del generador de mapas aleatorios y, tras varios cientos de líneas de códigos, reglas y funciones, produce un nivel completo. Como cada ejecución del código generará un *output* diferente, obtenemos casi diez millones de niveles distintos en menos de 90 Kb de memoria y con semillas de 7 cifras de longitud; una verdadera maravilla de la ciencia!

Pero esto, no es ningún descubrimiento, ni mucho menos, que desde hace más de veinte años innumerables cantidades de programadores hayan utilizado esta técnica para desarrollar sus juegos en consolas cuyas limitaciones de memoria los hubieran hecho, de otro modo, imposibles de llevar a cabo. Dos de los más conocidos son *The Sentinel* y *Elite*; este último iba a salir al mercado con más de 2^{48} mundos posibles, aproximadamente 282 billones (trillón americano).

1.2.7. Tile

Un *tile* es una imagen de un tamaño muy pequeño con respecto a la envergadura de la pantalla en la que se muestra y que, de por si sola, representa un elemento o detalle, a modo de una pieza del mosaico. Se usa para recrear fondos o imágenes excesivamente grandes o que en ningún momento se ven enteras y que probablemente causarían problemas en la memoria del sistema. Entre sus funciones, también destaca la creación de escenarios (generalmente con *scroll*) para la industria de los videojuegos, que es precisamente el uso que vamos a darle nosotros: recreamos con tiles los mapas laberínticos que nos produce el generador de mapas.

CAPÍTULO 2. ANÁLISIS FUNCIONAL

Para el segundo capítulo se ha elegido explicar el funcionamiento del software, abordándolo de una manera sencilla y sin excesivos tecnicismos, con la finalidad de que cualquier persona pueda entretenerse y comprenderlo a la vez. La intención es que esta parte del documento pueda servir también como una especia de manual para el usuario. Básicamente, el programa se puede dividir en dos grandes secciones: el **gestor de contenidos**, y el **juego de aprendizaje** en sí.

2.1. Gestor de contenidos

El gestor nos permite generar contenedores (catálogos) en los que creamos y almacenamos los distintos desafíos que queremos aprender. El contenido de estas estructuras puede guardarse en el disco duro del móvil, modificarse, reutilizarse para otros catálogos o intercambiarse entre usuarios.

2.1.1. Catalog

Como bien hemos dicho, un catálogo es una estructura a la cuál podemos añadir elementos de estudio, y posteriormente seleccionarla a la hora de jugar para forzar el aprendizaje. Existen varias maneras de conseguir un catálogo, no todas de las cuales están implementadas en este punto del prototipo (Figura 2.1):

- **Crear un catálogo partiendo de cero** – si seleccionamos esta opción en el menú, un pop-up nos solicitará un nombre de referencia para el catálogo. La única limitación es que el nombre del catálogo no puede estar vacío. En caso de poder crear el catálogo con éxito, seremos llevados a la pantalla de catálogo. Inicialmente, veremos que esta pantalla está vacía, esto es así, porque aún no hemos añadido ninguna ficha de estudio (Figura 2.2).

- **Cargar un catálogo ya existente desde el disco duro del terminal** – si ya habíamos creado y guardado un catálogo o hemos conseguido uno por otros medios, podemos seleccionar y abrirlo nuevamente. Esto se consigue por medio de un gestor de los discos duros del teléfono; analiza las carpetas root, nos muestra sus contenidos, y vuelve a repetir el proceso ante cualquier selección, permitiendo de esta manera, navegar por el sistema de directorios. Los archivos de catálogo tienen la extensión .fic.

- **Importar un catálogo de una aplicación similar para PC*** – importar catálogos tiene dos finalidades: 1) que la aplicación cuente desde el momento de su lanzamiento con un amplio abanico de temáticas a estudiar, y 2) evitar que el usuario se sienta abrumado o sin ganas de crear e introducir grandes cantidades de datos. El programa de compatibilidad elegido en cuestión es Mnemosyne, con ficheros .xml fácilmente parseables. Sin embargo, no hay que olvidar que parte de la "magia" de este software radica precisamente en la posibilidad de crear uno mismo el contenido que quiera estudiar.

- **Compartir catálogos vía Bluetooth o descargarlos de un servidor HTTP*** – integrar un sistema para compartir los catálogos vía Bluetooth quedó descartado prácticamente al principio debido a la dificultad añadida para un aporte muy pequeño. Se decidió que sería muchísimo más sencillo dejar que la tecnología actual actuara de medio para la transferencia externa al programa. Por el contrario, encontramos que un servidor HTTP dedicado a la aplicación sería de gran ayuda. Desde él, los usuarios podrían descargarse y compartir catálogos hechos por ellos mismos, subir y comparar sus puntuaciones y competir en eventos premiados que podrían anunciarse en el mismo (Figura 2.3).

Una vez estemos dentro de un catálogo, echando un rápido vistazo a los comandos de usuario de la pantalla, descubrimos que podemos crear nuevas fichas, borrar o modificar fichas existentes, salvar la información actual del catálogo al disco duro del dispositivo físico y salir del catálogo sin guardar nada. Adicionalmente, se han creado dos botones para navegar por las varias páginas que puede tener un catálogo; para mejor fluidez en el manejo, sólo se muestran un máximo de veinte fichas por página (Figura 2.4).

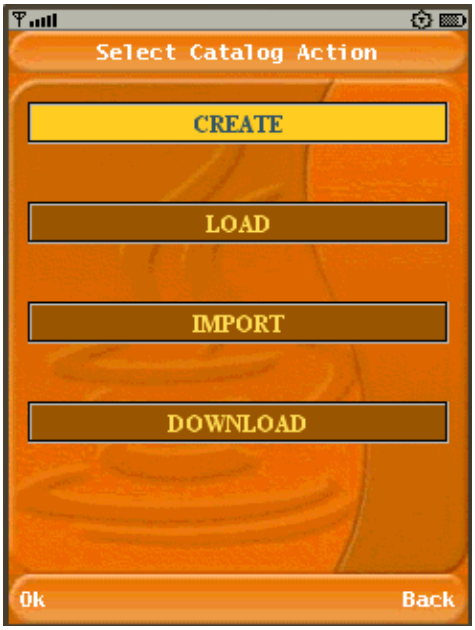

Figure 2.1 – Opciones del catálogo

Figure 2.2 – Nuevo catálogo

* Nota: esta opción no estará implementada hasta la próxima versión del programa.

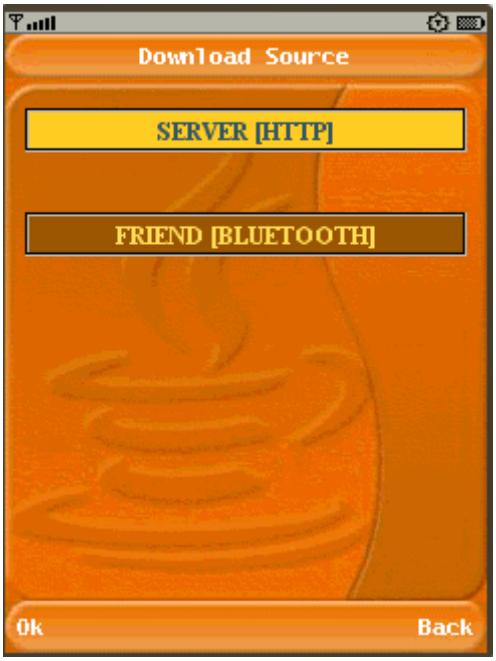

Figure 2.3 – Descargar catálogos

Figure 2.4 – Vista del catálogo

2.1.2. Card

En cuanto seleccionemos el botón “Detail” o apretemos el botón central de la cruceta digital, veremos los detalles de la ficha o card que en ese momento tiene el foco. En la pantalla de ficha aparecen cuatro pestañas que explicaremos a continuación:

- **Basic** – en esta pestaña aparece la información general de la ficha. Podemos ver el campo identificador (id) que es el *string* alfanumérico con el que se registra la ficha en el catálogo; el tipo (*type*) de contenido agregado a la ficha (permite ver una imagen, escuchar una pista de audio o ver un fragmento de vídeo). En la parte inferior, se ha incluido un botón que permite añadir contenido multimedia (en caso de haber seleccionado el tipo correspondiente en el campo anterior) gracias a un gestor similar al que usamos para cargar el catálogo. La ruta del archivo quedará almacenada en el campo “path” de esta misma pestaña (Figura 2.5).

- **Description** – en esta pestaña sólo hay un campo de texto en el que aparecerá la descripción del desafío. El usuario puede introducir cualquier cosa, una ayuda para responder a la pregunta, la explicación de un significado o incluso un razonamiento por puntos para deducir la respuesta (Figura 2.6).

- **Challenge** – en esta pestaña figura la pregunta o desafío en cuestión. Este es exactamente el mismo texto que aparecerá posteriormente en el juego, en el momento de evaluar nuestros conocimientos. También

dispondremos de un botón “Play Media” con el que, en caso de haber seleccionado el tipo de archivo adjunto de la ficha y haber añadido la ruta de este, se podrá reproducir el contenido en un nuevo pop-up (Figura 2.7).

- **Answers** – la última pestaña contiene todas las respuestas que se dan por válidas ante el desafío de la pestaña anterior. Con los comandos de usuario, podemos eliminar respuestas o crear tantas como queramos. Las respuestas aparecen con el contenido “New answer” y, en cuanto las seleccionemos con el botón central de la cruceta, aparecerá un pop-up en el cuál podremos editar la respuesta (Figura 2.8).

Cabe recordar, que cualquier cambio que hagamos se mantendrá al volver a la pantalla del catálogo, pero si no guardamos este al disco duro antes de salir, perderemos definitivamente todas las modificaciones hechas.

Una vez que tenemos un catálogo con un número considerable de desafíos, ya estamos listos para jugar y aprender.

Figure 2.5 – Card, pestaña Basic

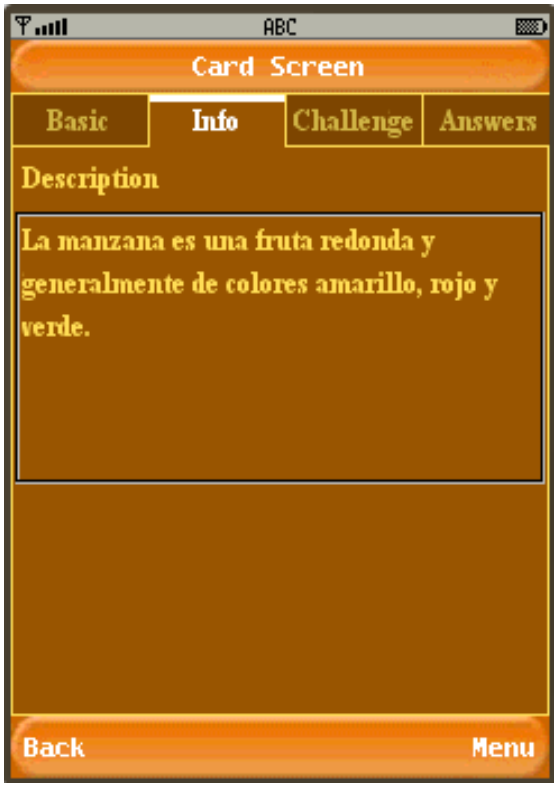

Figure 2.6 – Card, pestaña Info

Figure 2.7 – Card, pestaña Challenge

Figure 2.8 – Card, pestaña Answers

2.2. Juego de aprendizaje

El juego, en cambio, es la sección que complementa al programa, permitiéndonos estudiar y examinarnos de todos los desafíos que incluimos previamente en un catálogo (Figura 2.11).

2.2.1. Objetivo

El juego se desarrolla constantemente en mapas laberínticos (Figura 2.9), sobre los que el usuario controlará un personaje (visualmente se asemeja más a un cuadrado) que deberá recorrer cada rincón para encontrar la salida; este elemento nos llevará al siguiente nivel, en el que el objetivo del usuario será exactamente el mismo, y así sucesivamente. Para poder acceder a la salida (Figura 2.10), el jugador ha de tener en su posesión la llave del nivel, un objeto que estará escondido en uno de los tres cofres aleatoriamente emplazados en cada mapa. Los mapas (en el momento de la demo) serán de 990x990 pixels, presentando una extensión de aproximadamente cien habitaciones conectadas a recorrer y serán completamente diferentes unos de otros gracias a la generación aleatoria. Como no todo iba a ser un paseo, batallas aleatorias sucederán mientras nos desplazamos por el mapa, mostrando el verdadero propósito de este programa, aprender los desafíos que preparamos con antelación. A las respuestas se aplicará la técnica explicada de Leitner, con el

añadido de restar energía al jugador por cada fallo que acumule para retar aún más al usuario y aprovechar todo su potencial.

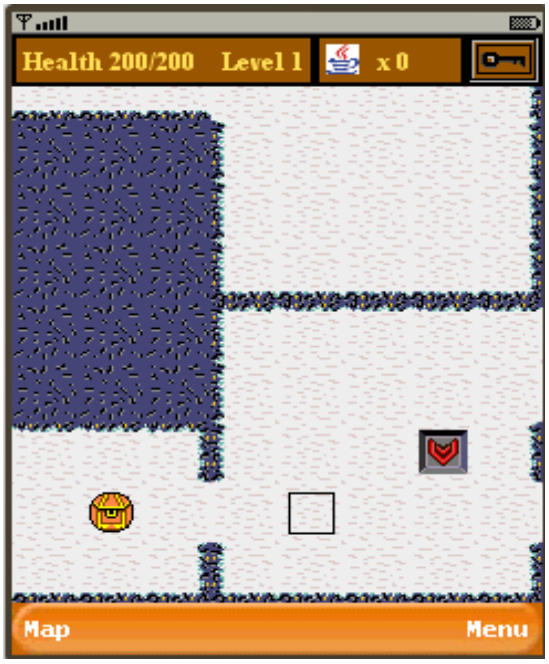

Figure 2.9 – Mapa con entrada del nivel

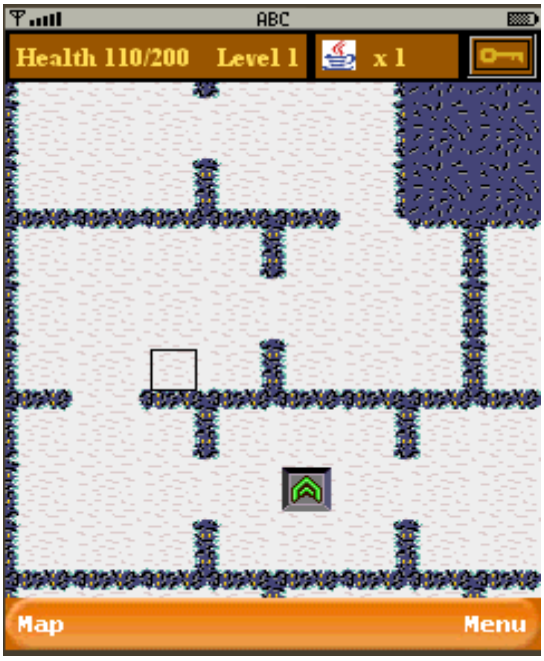

Figure 2.10 – Salida de un nivel

2.2.2. Previo

Antes de lanzarnos a jugar, conviene hacer una visita a la pantalla de opciones (Figura 2.12). La única que nos interesa en este momento es la dificultad que tendrá el juego, ya que todas las demás se pueden seguir modificando una vez estemos jugando. El modo fácil es para personas que nunca han jugado a un juego similar y que quieren una toma de contacto. El modo normal presenta verdaderos retos, pero realmente ayuda más a aprender. Para empezar una partida, deberemos antes escoger el catálogo (*.fic) con un gestor de directorios similar al explicado en el punto previo. Inmediatamente después comienza el juego.

Figure 2.11 – Pantalla principal

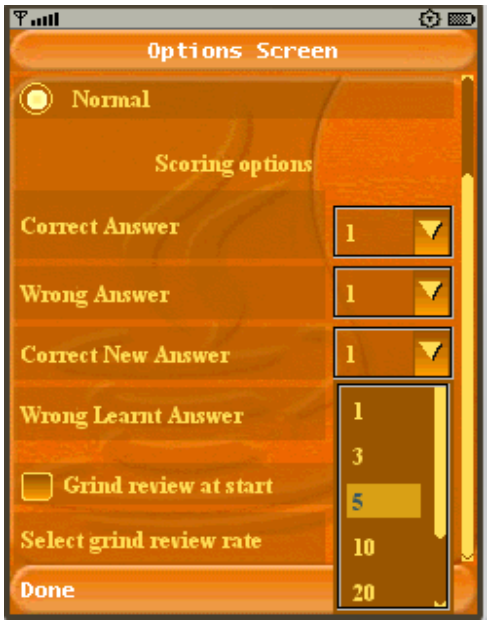

Figure 2.12 – Pantalla de opciones

2.2.3. Elementos

Antes se mencionó que tres cofres serían emplazados en cada nivel, cada uno de ellos podrá contener uno de los siguientes elementos:

- **Llave** – cada nivel tiene una llave y no podremos tomar la salida hacia el siguiente nivel sin haberla encontrado previamente (Figura 2.14).

- **Café** (Java) – estudiar cansa, y cuando estamos muy cansados y nos queda poca energía hay una gran probabilidad de quedarnos dormidos. Si lamentablemente esto ocurriese sería el fin de la partida, por eso si encontramos un café, podemos tomarlo y recuperar nuestras fuerzas.

- **Vacío** – también hay cofres vacíos, nada pasa ni nada cambia al abrirlos (Figura 2.13).

En cada nivel habrá un elemento de cada uno de estos contenidos en los cofres.

Figure 2.13 – Cofre vacío

Figure 2.14 – Cofre con llave

2.2.4. Mapa

Para no perdernos en la aventura, tendremos a nuestra disposición en todo momento un botón que hará aparecer un pop-up con una miniatura del mapa del nivel. El jugador será representado mediante un punto negro en dicho mapa (Figura 2.16).

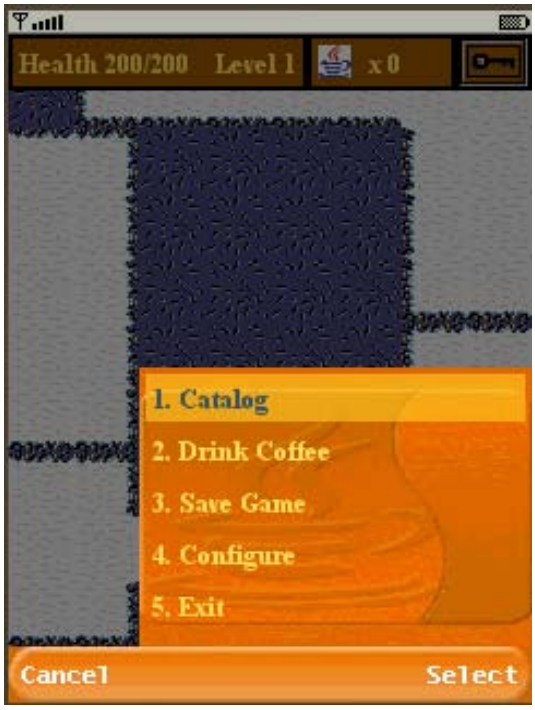

Figure 2.15 – Menú dentro del juego

Figure 2.16 - Minimapa

2.2.5. Pantalla de configuración

El programa no estaría completo, si no pudiésemos configurar nuestra propia manera de estudiar. En este menú (al cual podemos acceder desde las opciones principales, o desde el comando “Configuración” una vez dentro del juego) podremos decidir cómo trata el juego nuestras acciones.

El SRS que hay dentro del programa, trata las fichas dependiendo de la puntuación que tiene cada una de ellas. Lo que se permite, es que el usuario ajuste los puntos que gana por acertar una respuesta, fallarla, acertar un desafío que aparece por primera vez o fallar una pregunta que supuestamente ya tenía aprendida. Además, podemos activar un repaso cada vez que vamos a jugar, indicando el porcentaje de fichas activadas que queremos repasar, antes de continuar con la aventura.

2.2.6. Batallas

Tal como explicamos anteriormente, mientras vamos avanzando por los laberintos se irán sucediendo batallas de manera automática. El número de fichas con cuyo desafío tendremos que lidiar, están entre una y cinco, dependiendo de la dificultad y otros factores. El usuario no tendrá límite de tiempo para contestar cada desafío, pero tendrá varias opciones a la hora de “luchar”: puede intentar responder, pasar al siguiente desafío o huir de la batalla, en cuyo caso se saltan todas las fichas que quedan y se vuelve al mundo laberíntico (Figuras 2.17, 2.18, 2.19).

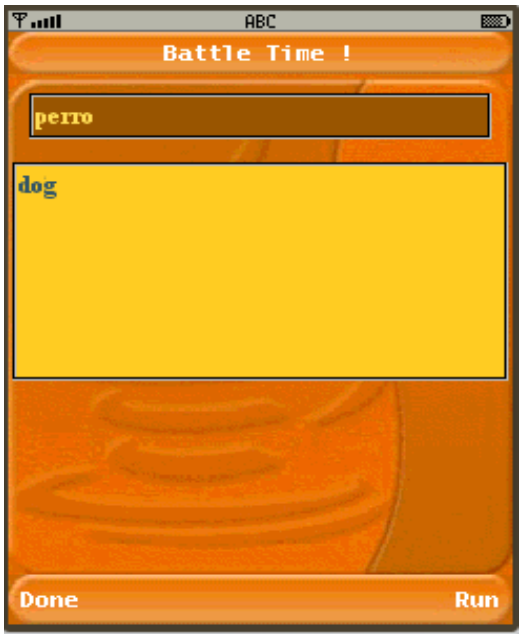

Figure 2.17 – Desafío y respuesta

Figure 2.18 – Resultado del desafío

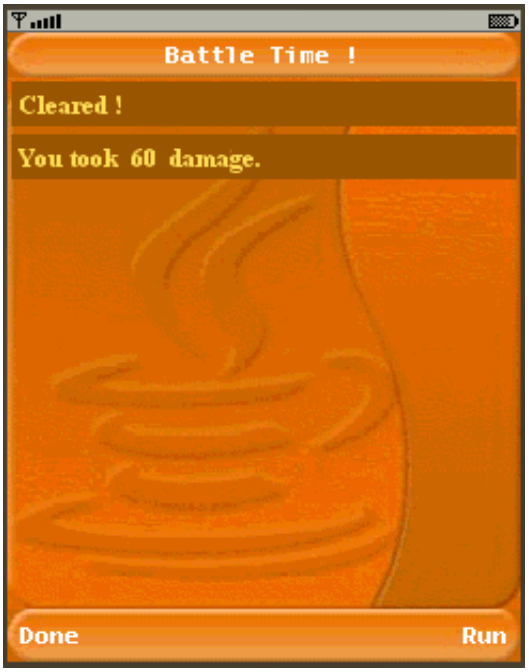

Figure 2.19 – Resultado de la batalla

2.2.7. GUI (graphic user interface)

La interfaz gráfica de usuario (graphic user interface – GUI) es el modo a través del cual el juego es capaz de comunicarle cosas al jugador de una manera visual, utilizando la pantalla como medio. Nuestro personaje tendrá un menú en la barra superior, en el que figurarán varios datos (nivel, salud, llave del nivel...) que en todo momento estarán actualizados. Otro menú interno, en el que podrá consultar y disponer de todos los desafíos que se le han asignado hasta el momento (pregunta, respuesta, descripción...) será accesible a través de los botones de esta misma pantalla. Al comenzar cada uno de los niveles se actualizará dicha lista con las nuevas fichas pertenecientes a ese nivel. Adicionalmente, será aquí el sitio para acceder al menú de configuración de partida que explicaremos más adelante (Figura 2.15).

2.2.8. Game Over

El juego terminará cuando la salud de nuestro personaje sea reducida a cero. Si esto ocurriese, siempre podemos cargar la última partida guardada que tengamos.

CAPÍTULO 3. ESTRUCTURA Y DETALLES TÉCNICOS

A partir de este punto, trataré de detallar cómo esta compuesta cada clase y la manera en la cuál está interconectada con el resto. Se hará especial énfasis en las transiciones, métodos y elementos que forman el programa. Hemos escogido fraccionar la estructura del programa en varios módulos. Iremos explicándolos uno por uno y trataremos de presentar algunos conceptos entre medio, para mayor comprensión del lector. Primero se hablará del módulo gráfico, que está presente en prácticamente todas las pantallas. A continuación trataremos por fin el módulo de datos, detallando en profundidad la estructura y formación de los catálogos y fichas, y cómo los gestores se relacionan con estas clases. Daremos una breve explicación del módulo de aprendizaje y finalmente descubriremos el motor del juego junto con todos sus componentes, lo que forma el módulo del mundo.

3.1. Screen Module

Este módulo está compuesto por las pantallas gráficas y la interfície que comunica el usuario con el programa. Recordemos que para conseguir un acabado profesional a tiempo, hemos usado las librerías gráficas de Sun LWUIT. Para comenzar, explicar que prácticamente todas las pantallas de la aplicación se componen de tres elementos principales.

- **Title bar** – es un componente rectangular, que permanece estático en todo momento en la parte superior de la pantalla. Normalmente, en este elemento tenemos el título de la pantalla o el nombre del catálogo o ficha.

- **Softbutton bar o Menu bar** – es un componente similar al anterior, pero cuyo emplazamiento se encuentra en la parte inferior de la pantalla. En él, es habitual colocar los comandos que el usuario puede activar por medio de las *soft keys* del teléfono móvil.

- **ContentPane** – abarca todo el espacio que hay entre los dos componentes anteriores. Es el cuerpo de la pantalla (análogo al *body* de una página HTML) y en él se colocan el resto de componentes, que formarán la vista de la cual dispone el usuario. Este elemento es extensible; mientras las dos barras anteriores permanecen estáticas siempre en sus posiciones, el ContentPane irá creciendo a medida que se coloquen otros componentes en él, permitiendo efectivamente, un *scroll* vertical por la pantalla.

Además de esto, será conveniente explicar al lector los componentes o ***widgets*** **[4]** más importantes que vamos a utilizar a partir de aquí:

- **Form** – el Form es uno de los componentes de más alto nivel; se compone de los tres elementos principales previamente explicados. Necesitamos un Form para construir una pantalla para el usuario.

- **Dialog** – el Dialog es similar al Form en todo, salvo que a la hora de mostrarlo, este se superpone a modo de pop-up encima de la pantalla anterior. Dependiendo del tamaño del Dialog, es posible que este no cubra toda la superficie visible; en este caso, veremos como se guarda una imagen de la anterior pantalla y se pinta todo el fondo sombreado con ella, a modo de GlassPane, mientras el Dialog permanece activo en un primer plano.

- **Layout** – un Layout es un modo de colocar todos los elementos que incrustamos en el ContentPane de un Form o Dialog. De manera muy similar al Swing, existe el GridLayout (los elementos se colocan en celdas, dado un número de filas y columnas) el BorderLayout (en el cual hay una zona central para el elemento más importante y cuatro zonas más pequeñas en los puntos cardinales para elementos que necesitan ocupar menos espacio) y el BoxLayout, con el cual podemos colocar los elementos uno detrás de otro sobre el eje X o Y, ocupando todo el ancho posible (este Layout es muy útil para hacer listas por ejemplo). También tenemos el FlowLayout, el cual no hemos usado en este proyecto, que simplemente va colocando los elementos uno detrás de otro, siempre que quede espacio en la línea horizontal en la que está.

- **Label** – el Label o etiqueta, es un componente básico que utilizamos para representar una cadena de texto, una imagen o ambas a la vez. El Label permite varias opciones de alineación de su contenido, pero no diferentes estados del componente.

- **Button** – un Button, a diferencia del Label es un componente que sí puede tener diferentes estados, conservando todas las propiedades del anterior. Este botón tiene tres estados básicos: presionado, seleccionado o por defecto. Debido a la naturaleza de este elemento, se le pueden asignar Listeners, los cuales detectarán si presionamos el botón, si este tiene el foco o activarán cualquier otra acción que se haya programado ante un cambio de estado de este tipo.

- **TextArea** – el TextArea presenta un texto que puede ser editado y modificado por medio del editor nativo del dispositivo físico. Hacerlo de esta manera, permite al usuario poder usar otros widgets ya desarrollados como el T9 (texto predictivo).

- **TextBox** – por el contrario, el TextBox es un campo que permite la introducción y adición de texto en el momento (en la misma pantalla). Provee un comando para borrar texto (para evitar incompatibilidades con el mapeo de teclas para borrar de determinados fabricantes y dispositivos) y otro para cambiar el modo de input actual del teclado. Sin embargo, también presenta otro comando más, que permitirá al usuario

pasar a la pantalla de edición T9 (la misma que la del TextArea) en caso de que lo necesite.

- **ComboBox** – el ComboBox es un elemento que hace aparecer una lista de elementos cuando lo seleccionamos. Hace la función del clásico desplegable con opciones, de las cuales sólo se puede escoger una.

- **CheckBox** – la representación de este componente es una casilla, que podemos marcar o desmarcar. Si la seleccionamos estando enfocada, sufrirá un cambio de estado y una pequeña "v" se dibujará o eliminará de su recuadro, haciendo visible su estado.

- **RadioButton** – es un botón similar al componente Button, pero que mantiene su estado único dentro de un grupo de botones similares. El siguiente elemento clarificará esta explicación.

- **ButtonGroup** – es el encargado de gestionar varios RadioButtons, asegurando que únicamente uno de ellos puede estar seleccionado dentro del mismo grupo.

- **List** – una lista es a la vez uno de los componentes más complejos de implementar, pero también uno de los más útiles de cara a prácticamente cualquier diseño. Una lista se encarga de representar y actualizar gráficamente una serie de elementos siguiendo el modelo vista controlador (MVC). Sin embargo, para que esto funcione, antes debemos cargar los elementos dentro de lo que se conoce como un ListModel; podemos cargar un *array* de cadenas de texto, un vector de clases creadas por nosotros mismo... no hay limitaciones en cuanto a este aspecto. El componente List, usa una interfaz llamada ListCellRenderer, que es la que se encarga de colocar y dibujar cada uno de los elementos que componen la lista. Aunque se puede usar el renderizador por defecto, lo mejor es las posibilidades de configuración que presenta crear el nuestro propio; manera de presentar los objetos, colocación, aspecto que tienen al poseer o no el foco...

- **TabbedPane** – este componente es similar al Form, pero permite añadir lo que se conoce comúnmente como Tabs; pestañas con distinta representación dentro de una misma pantalla. Cada una de las Tabs es enteramente configurable por su cuenta; la única pega es que los comandos del MenuBar han de ser comunes para todas las pestañas.

- **Listener** – esta interfaz es necesaria para todos los componentes que puedan cambiar de estado o sean susceptibles a inputs realizados por el usuario, debiendo responder con cambios gráficos ante ello. Utilizamos Listeners para conocer qué botones aprieta el jugador y para detectar si esto provocará la navegación por pantallas o el desplazamiento de componentes en la pantalla actual.

Con esto finalizamos nuestro pequeño glosario de componentes y conceptos. A continuación, procederemos a detallar los módulos, tal y como habíamos

prometido. Se da por hecho que el lector está familiarizado con el método de crear y destruir MIDlets y se omitirán detalles referentes a tales funcionalidades.

3.1.1. Main

Main es nuestra clase cuya ejecución se automatiza al comenzar el programa; posee las funciones obligatorias de comenzar, pausar y destruir un MIDlet. Esta clase no posee parte gráfica visible para el usuario, pero hemos creído oportuno incluirla aquí porque tiene un papel importante en la navegación entre pantallas. Se trata de un método showScreen, al que se le pasa un valor numérico, que no es otra cosa que el identificador que tiene asignado esa pantalla. La función se encarga de construir y mostrar la nueva pantalla al usuario.

3.1.2. MainScreen

La pantalla principal nos da acceso a las dos grandes secciones del programa (gestionar catálogos y jugar) además de ofrecer un nexo hacia el menú de opciones. Utilizamos componentes Button para representar estos vínculos y un Listener para las determinar las acciones del usuario (Tabla 3.1 y 3.2).

Comando	**Efecto**
Ok	Confirma la selección que tiene el foco en ese momento y nos lleva a la consiguiente pantalla.
Exit	Abre un Dialog, preguntándonos si queremos salir.

Table 3.1 – Comandos de la pantalla principal

Componente	**Efecto**
Game [Button]	PreGameScreen
Catalog [Button]	PreCatalogScreen
Options [Button]	OptionsScreen
Exit [Button]	ExitDialog pop-up

Table 3.2 – Componentes de la pantalla principal

3.1.3. PreCatalogScreen

Esta pantalla es muy similar a la anterior. Su objetivo es permitir que el usuario elija la vía por la cual quiere obtener un catálogo (Tabla 3.3 y 3.4). Utilizamos componentes Button para representar los vínculos y de nuevo un Listener.

Comando	**Efecto**
Ok	Confirma la selección que tiene el foco en ese momento y nos lleva a la consiguiente pantalla.
Back	Nos lleva a la pantalla anterior, en este caso MainScreen.

Table 3.3 – Comandos de la pantalla de elección de la fuente del catálogo

Componente	**Efecto**
Create [Button]	Abre un Dialog para que el usuario introduzca el nombre del catálogo. Si es un nombre válido, el usuario será llevado a la CatalogScreen.
Load [Button]	DirectoryScreen
Import [Button]	DirectoryScreen
Download [Button]	DownloadScreen

Table 3.4 – Componentes de la pantalla de elección de la fuente del catálogo

3.1.4. DownloadScreen

A través de esta pantalla, el usuario podrá descargarse un catálogo desde un amigo (vía Bluetooth) o desde el servidor dedicado para la aplicación (vía HTTP). El modo de forjarla es idéntico al de las pantallas anteriores.

Una de estas funcionalidades se ha descartado y la otra aún no se ha implementado en esta versión (Tabla 3.5 y 3.6).

Comando	**Efecto**
Ok	Confirma la selección que tiene el foco en ese momento y nos lleva a la consiguiente pantalla.
Back	Nos lleva a la pantalla anterior, en este caso PreCatalogScreen.

Table 3.5 – Comandos de la pantalla de descarga de catálogo

Componente	Efecto
Friend – Bluetooth [Button]	-
Server - HTTP [Button]	Inicia una conexión con el servidor y muestra una lista de los catálogos disponibles.

Table 3.6 – Componentes de la pantalla de descarga de catálogo

3.1.5. DirectoryScreen

Ésta es una de las pantallas más importantes. Su función es encontrar los directorios root que tenga nuestro dispositivo físico y listarlos en pantalla mediante un componente List. El usuario podrá moverse por todos los objetos de la lista (renderizados como botones en este caso) y, si selecciona uno de ellos, el programa tratará de acceder a ese directorio y de nuevo listara todas las carpetas y ficheros que contenga. El efecto de listado lo conseguimos mediante un BoxLayout sobre el eje Y (Figura 3.1).

Cada vez que el usuario realiza una de estas acciones (Tabla 3.7 y 3.8), se llama primero a la función que descubre y anota el *path* de los elementos, y a continuación se borra y recrea el ListModel con los nuevos datos. Podemos refrescar el Form con el método revalidate. Volviendo al manejo, solamente están permitidos los archivos .fic (extensión de los catálogos propios) y los .xml (archivos a parsear de Mnemosyne). Una vez seleccionemos un archivo válido, seremos llevados a la pantalla del propio catálogo. Hay un caso especial, que permite seleccionar varios formatos de archivos multimedia, para adjuntar a una ficha. Esto se explicará más adelante.

Comando	Efecto
Ok	Confirma la selección que tiene el foco en ese momento, tratará de cargar el catálogo y nos llevara a la CatalogScreen.
Back	Nos lleva a la pantalla anterior, en este caso PreCatalogScreen.

Table 3.7 – Comandos de la pantalla de directorios

Componente	Efecto
Archivo/Carpeta X [Button] (List)	Abre la carpeta y actualiza el contenido en la siguiente vista./Abre el archivo; dependiendo de dónde venimos hay tres posibilidades: - Carga el catálogo en la CatalogScreen. - Carga la ruta del archivo multimedia para la ficha dada y nos devuelve a su CardScreen. - Carga el catálogo en memoria e inicia el juego.

Table 3.8 – Componentes de la pantalla de directorios

3.1.6. CatalogScreen

La pantalla del catálogo nos mostrará las fichas que contiene este y nos permitirá navegar por ellas. Asimismo, pone al alcance del usuario una cantidad de comandos más que suficiente para que este pueda gestionar las fichas (Tabla 3.9 y 3.10).

Utilizamos una clase List para representar los elementos en la pantalla, para no sobrecargar la aplicación, se muestran solamente 20 de ellos por página. Tenemos un algoritmo que calcula el número máximo de páginas que podemos tener, dependiendo del número de fichas, y muestra en la barra del título esta información junto con la página actual en la que nos encontramos. Para la estructura, de nuevo escogemos el BoxLayout sobre el eje de las ordenadas y nos ceñimos a los procedimientos creados para la clase DirectoryScreen (modificar el modelo y refrescar la pantalla) para formar las respuestas al usuario.

Por otro lado, hemos sobrescrito uno de los métodos propios de la clase Form, concretamente la representación de los comandos. Tal y como estaba programada, se mostraba un comando abajo a la izquierda y un menú con el resto de comandos abajo a la derecha de la MenuBar, al estilo del software nativo de todos los fabricantes. Sin embargo, el comando que estaba a la izquierda siempre se incluía en el menú de la derecha también, faceta que queríamos evitar, ya que ese menú ya tenía demasiados comandos de por sí solo.

Comando	Efecto
Detail	Confirma la ficha que tiene el foco en ese momento y nos lleva a la CardScreen del objeto.
Next	Cicla a través de la lista de fichas, mostrándonos las siguientes fichas, hasta un máximo de 20.
Previous	Cicla a través de la lista de fichas, mostrándonos las fichas anteriores, hasta un máximo de 20.
New Card	Crea una nueva ficha y nos lleva automáticamente a la CardScreen.
Delete	Elimina del catálogo la ficha que en ese momento tiene el foco.
Save	Guarda al disco duro del móvil todos los cambios (fichas nuevas, borradas, modificadas) que hayamos realizado durante la sesión.
Exit	Sale del catálogo actual sin guardar los datos en el disco duro. Si queremos conservar las modificaciones tenemos que guardar los cambios. Esto sirve a modo de deshacer acciones incorrectas y volver al último catálogo guardado.

Table 3.9 – Comandos de la pantalla de vista del catálogo

Componente	Efecto
Ficha X [Label] (List)	CardScreen de la Ficha X

Table 3.10 – Componentes de la pantalla de vista del catálogo

3.1.7. CardScreen

La pantalla de las fichas tiene una estructura genérica, que se rellena con la información de la Card seleccionada del catálogo o se deja vacía si es nueva. Implementamos la clase TabbedPane con **cuatro pestañas** (clase Tab) sobre el contenedor base que se nos entrega.

Para la **primera pestaña** creamos una función que nos permite emparejar Labels de texto con TextAreas, de manera que aparezcan en una misma línea horizontal. Fijando el ancho del Label más extenso, también hemos conseguido alinear verticalmente el comienzo de las áreas de texto, mostrando en conjunto un resultado más profesional. Este es el primer punto en el que vemos un ComboBox, que nos permite seleccionar el contenido que puede tener adjuntado la ficha, si es que queremos tal. Para tal objetivo fijamos un botón en esta misma pestaña, que lleva al usuario a la DirectoryScreen. En este caso, deberá elegir un formato soportado por la aplicación, el caso especial del que se habló antes. El sistema tiene tres vectores con formatos de imagen, audio y vídeo que posiblemente pueda soportar un terminal (no todos) y comprueba si la selección es válida. La ruta del fichero quedará almacenada en el TextArea

“Multimedia Path” en esa misma pestaña, con el fin de que el usuario pueda ver o modificar lo que se vaya a reproducir.

En esta **segunda pestaña**, el usuario podrá introducir cualquier clase de información relacionada con la ficha en un TextArea destinado a ese fin. Para estos componentes (en todas las Tabs y no sólo en esta) se ha puesto especial cuidado; se han programado de tal manera que disponen en un tamaño fijo en pantalla, suficiente para la introducción de una cantidad de información razonable. Sin embargo, se ha limitado el tamaño de la mayoría de los TextArea de esta pantalla a 300 o 400 caracteres. Cuando superemos el tamaño visible en pantalla, se activará un *scroll* interno dentro del mismo campo de texto, que permitirá al usuario ver todos los datos sin descuadrar con un tamaño excesivo la pantalla.

La **tercera pestaña** es similar a la anterior: tiene un TextArea para introducir el desafío y un botón justo debajo. Este componente sirve para mostrar el contenido multimedia (si lo hay) en un nuevo Dialog (Figura 3.2) que aparece en medio de la pantalla. Se ha tratado de cerrar automáticamente el pop-up cuando finaliza la reproducción, con los métodos ofrecidos por Sun para tal fin, pero no ha habido manera de que funcionara. El usuario podrá interrumpir la reproducción en cualquier momento por medio de un comando de la MenuBar.

En la última y **cuarta pestaña** figuran todas las respuestas que el sistema dará por válidas ante el desafío de la Tab anterior. De nuevo se trata de un componente List, con su BoxLayout vertical. En este caso, la diferencia (y la dificultad) radicaba en la agregación en tiempo real de nuevos elementos al modelo de la lista. Si nos fijamos, los comandos presentados para la CardScreen son comunes a todas las pestañas (Tabla 3.11 y 3.12). Uno de ellos sirve para añadir respuestas, mientras que el otro las elimina. Para eliminar una respuesta, tenemos que estar en esa pantalla y posicionados con el foco sobre uno de los elementos de la lista. Para crear uno nuevo sin embargo, podemos estar situados en cualquier lugar; el programa nos llevará a la última pestaña y añadirá un elemento con el texto “New answer” a la lista. Si seleccionamos el elemento nuevo (o cualquier otro de esa lista) aparecerá un pequeño Dialog con un TextBox, en el cual podremos modificar el valor almacenado. En ambos casos modificamos el ListModel, pero en esta ocasión no podíamos refrescar la pantalla debido al peligro de perder datos almacenados en otras Tabs o de cambiar la vista actual del usuario. Para ello, se ha hecho un poco de ingeniería inversa, recorriendo el camino de los padres de cada componente hasta llegar al que nos interesaba para vaciarlo y reconstruirlo de nuevo.

Para finalizar, los cambios que se hagan en una ficha se quedan guardados automáticamente al volver atrás al catálogo con el comando “Back”. Esto es así porque todas y cada una de las clases Card están contenidas dentro de su padre, la clase Catalog, y como en todo momento guardamos su referencia no hay necesidad de preocuparse por la información de las fichas. Eso sí, antes de salir de un catálogo sí hay que seleccionar la opción para guardar los datos al disco duro del terminal.

Comando	Efecto
New answer	Este comando nos llevará al Tab “Answers” y añadirá un nuevo elemento a la lista de respuestas, con el texto “New answer”.
Delete answer	Siempre y cuando estamos en el Tab “Answers” y tenemos enfocado un elemento de la lista, este comando lo eliminará. En caso contrario no hará absolutamente nada.
Back	Nos lleva a la pantalla anterior, en este caso CatalogScreen.

Table 3.11 – Comandos de la pantalla de vista de ficha

Componente	Efecto
Id [TextArea]	Abre la pantalla de texto del terminal para introducir o modificar el Id de la ficha.
Type [ComboBox]	Abre el ComboBox para permitir la selección de un tipo de datos multimedia.
Multimedia Path [TextArea]	Pantalla de texto del terminal para introducir o modificar la ruta del contenido multimedia.
Set Media [Button]	DirectoryScreen
Description [TextArea]	Abre la pantalla de texto del terminal para introducir o modificar información adicional sobre la ficha.
Challenge [TextArea]	Abre la pantalla de texto del terminal para introducir o modificar el desafío de la ficha.
Play Media [Button]	Dialog pop-up que muestra el contenido multimedia.
Answer X [Label] (List)	Dialog pop-up que muestra un TextBox con el texto de la respuesta y permite modificarlo.

Table 3.12 – Componentes de la pantalla de vista de ficha

Figure 3.1 – Pantalla de directorios

Figure 3.2 – Diálogo de reproducción multimedia

3.1.8. OptionsScreen

Cuando el usuario accede a esta pantalla tendrá a su disposición una serie de opciones que podrá modificar para ajustar el estilo del juego. La más importante de ellas es la dificultad, incluyendo otras opciones más finas como el sistema de puntuaciones o el repaso al retomar la partida. Este punto está completo y mejor explicado en la sección 3.3 – Learning Module.

3.1.9. InGameCatalogScreen

Ésta es la representación del catálogo una vez estamos dentro del juego. Las diferencias obvias son la imposibilidad de crear fichas nuevas o borrar las existentes y la desaparición de algunos botones ahora innecesarios, como por ejemplo el de guardar el catálogo al disco duro. Las diferencia no tan evidentes son dos: por un lado sólo podremos ver y acceder a las fichas que tengamos activadas en un momento dado (aunque el catálogo sea más amplio) y por otra parte el número de fichas listadas por página dependerá del nivel de dificultad (ver sección 3.3.1).

3.1.10. InGameCardScreen

De manera similar al punto anterior, en la pantalla de ficha durante el juego no podremos añadir, eliminar o alterar ninguno de los valores presentes. Sin embargo, se ha añadido a la pestaña “Basic” la puntuación actual de la ficha, valor que oscila entre cero y cien (Figura 3.3). Además, en caso de que la ficha sea nueva o esté aprendida, una etiqueta extra en la misma pestaña nos lo indicará también (Figura 3.4).

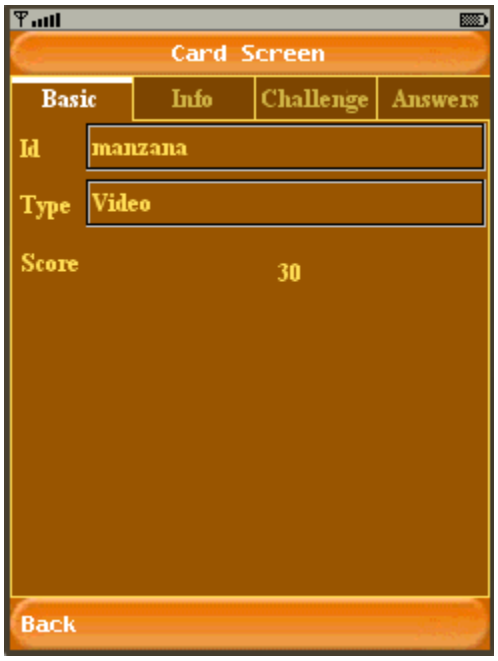

Figure 3.3 – Puntuación de una ficha

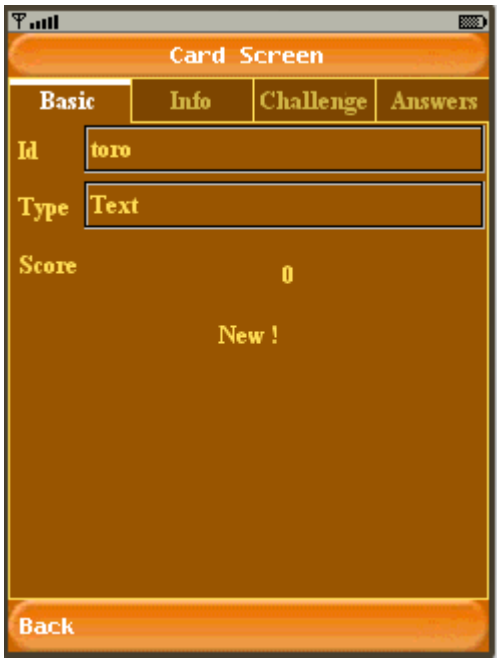

Figure 3.4 – Ficha con etiqueta “New”

3.1.11. InfiniteProgressIndicator

Éste es un pequeño Dialog que muestra el texto “Please wait...” seguido de la imagen de una rueda de progreso girando continuamente. El objetivo es hacer entender al jugador que el programa está trabajando y cargando elementos en *background* y que, tras una breve espera, el juego continuará. No era absolutamente imprescindible, pero aportaba un toque de profesionalidad y evitaba la sensación de “bloqueo” (durante aproximadamente dos segundos, dependiendo del móvil) del terminal al pasar de un nivel a otro (Figura 3.5).

Figure 3.5 – Indicador de progreso

3.2. Data Module

El Data Module está formado por un conjunto de clases y estructuras en las cuales almacenamos la información de los catálogos y fichas. Tiene otras misiones, no menos importantes, como cargar y guardar los archivos, parsearlos y transformarlos a otros formatos.

3.2.1. Catalog

El catálogo es el contenedor de las fichas que vamos a estudiar. Si el catálogo al que queremos acceder es completamente nuevo, se creará su estructura vacía, introduciendo el título que le ha puesto el usuario. Si por el contrario estamos cargando un catálogo, se pasará su ruta al DefaultParser con el fin de que transforme y adapte los datos a nuestra estructura (Tabla 3.13).

Campo	Función
CatalogPath [String]	Contiene la ruta del catálogo en el disco duro.
CatalogName [String]	Contiene el nombre del catálogo.
CatalogDescription [String]	Información adicional sobre el catálogo.
CardList [Vector]	En él se almacenan todos los elementos Card.
CatalogCards [int]	Número de cards pertenecientes a este catálogo.
Active [int]	Páginas activas. Este valor cambia dependiendo del nivel en el que nos encontramos [Se usa sólo durante el juego].

Table 3.13 – Campos de una clase Catalog

3.2.2. Card

Es una de las piezas más pequeñas, sin la cual no sería posible efectuar un aprendizaje. Análogamente a los catálogos, las fichas pueden crearse vacías o cargarlas de un fichero. Aquí vemos su estructura (Tabla 14).

Campo	Función
CardCatalog [String] *	Catálogo al que pertenece la ficha.
CardId [String]	Identificador alfanumérico con el que representamos la ficha en la vista del catálogo.
CardType [int]	Indica si tiene contenido multimedia y cuál es su tipo.
CardMultimediaPath [String]	Ruta del contenido multimedia en el disco duro.
CardDescription [String]	Descripción, información o ayuda sobre la ficha o el desafío.
CardChallenge [String]	El desafío.
CardAnswers [int]	Número de respuestas válidas que tiene la ficha.
CardAnswer [Vector]	Array con las respuestas válidas.
CardNew [boolean] **	Indica si tratamos la ficha como nueva o no.
CardLearnt [boolean] **	Indica si tratamos la ficha como aprendida o no.
CardScore [int] **	Puntuación de la ficha dentro de la partida actual.

Table 3.14 – Campos de una clase Card

* Indica un parámetro no visible para el usuario.
** Estos parámetros sólo son visibles y utilizados durante una partida.

3.2.3. Formato del archivo .FIC

Se trata de un archivo de texto, en el cual separamos los distintos campos por medio de un pipe ' | '. La primera línea contendrá la información del catálogo, mientras que todas las restantes serán dedicadas a las fichas, a razón de una línea por cada Card. Es importante notar que no todos los campos descritos anteriormente aparecerán almacenados en el fichero .fic, muchos de ellos son internos para el programa o para la hora del juego y serán almacenados en RMS.

Ejemplo:

Inglés|Este es un catálogo que permitirá al estudiante aprender algo de vocabulario del idioma|30

Inglés|manzana|0||La manzana es una fruta redonda y generalmente de colores amarillo, rojo y verde.|manzana|3|apple|apple2|apple3correcto

Inglés|melocotón|0||El melocotón es una fruta redonda muy jugosa|melocotón|1|peach

Inglés|plátano|1|file:\\\e:/images/platano.gif|Una fruta amarilla y alargada. No tiene pérdida!|plátano|1|banana

3.2.4. Default Parser

El parseador por defecto es una clase que se encarga de transformar la información contenida en los archivos .fic en datos válidos para la aplicación. Utilizamos la FileConnection API (JSR -75) la cual no esta disponible en todos los móviles, pero sí en la gran mayoría de los modelos que han ido saliendo desde hace 3 años.

Programamos una función que se encarga de realizar lecturas del fichero línea por línea. Acto seguido pasamos las líneas a la función *split* (clase Utils) que es equivalente a la método *split* de Java que trocea una cadena de caracteres en un vector de Strings, dada una expresión regular (en nuestro caso el pipe). Dado que la versión de Java para dispositivos móviles es la 1.3, además se han recortado muchísimas cosas, este método no estaba presente y hemos tenido que programarlo. Se han establecido las condiciones de tal manera, que una vez que el parseador ha finalizado su trabajo, cada uno de los campos de las múltiples estructuras generadas (Catalog y Cards) contiene almacenado el valor que le corresponde.

3.2.5. XML Parser

Este parseador será similar al anterior, con la diferencia notable de que será exclusivo para el formato de ficheros .xml. Esta clase no está disponible en la versión de la aplicación presentada en este TFC y, por tanto, se presenta como una posible línea futura.

3.2.6. Save Manager

La clase Save Manager hace de gestor para guardar todo el contenido de un catálogo y sus cards al disco duro.

Si el catálogo ya existía en el disco duro, el gestor obtendrá su ruta y sobrescribirá el archivo anteriormente existente por el nuevo contenido. Si por el contrario, es un catálogo nuevo de lo que disponemos, la clase tratará de hacerse con los discos duros locales del teléfono, e intentará guardar el archivo en el primero de ellos (file:\\\c:/nombre_del_catalogo.fic – en un móvil, file:\\\root1/nombre_del_catalogo.fic – en el PC).

Sin embargo, existe un problema más que molesto, culpa de la seguridad que se ha querido implementar en el MIDP2.0 y del control que quieren imponer gran parte de los fabricantes, en especial Nokia con sus modelos S60 FP3 [Esto se explica en el apartado de Limitaciones Técnicas]. La “solución” que se ha podido encontrar para el modelo N95 con el que realizamos la demo es forzar la ruta (*hardcoded*) para guardar los catálogos a file:\\\e:/images/nombre_del_catalogo.fic.

3.2.7. Save Game Manager

Para toda la información que está presente mientras jugamos, se ha creado esta clase que la guardará en RMS (Record Management System). El RMS es una memoria persistente que pertenece a cada MIDlet y que sólo es accesible por él mismo. No podemos encontrarla en el disco duro, ya que realmente está guardada dentro del mismo programa, en espacios de 64Kb aproximadamente (dependiendo del dispositivo) dedicados a ese propósito. Creamos un Store con el nombre del catálogo que estamos estudiando, serializamos los datos, los transformamos a bytes y los introducimos en las múltiples entradas que vamos creando sobre la marcha. En caso de existir un catálogo con el mismo nombre, el programa interpreta que es la misma partida y sobrescribe la información. Esto indica que si queremos evitar la pérdida de datos no debemos jugar (y guardar la partida) más de una vez con un mismo catálogo, así como no es recomendable nombrar dos catálogos de la misma manera, porque tendría el mismo efecto.

Volviendo al tema principal, guardamos los siguientes datos: datos del usuario, de sus opciones de juego y aprendizaje, semillas y condición de los niveles visitados y puntuación y estado de cada una de las fichas.

3.2.8. Load Game Manager

Si queremos cargar una partida previa, podemos acceder a una pequeña pantalla que tendrá una entrada (con el título del catálogo) por cada una de las partidas que hayamos guardado. Leeremos y cargaremos los datos en el siguiente orden (Tabla 3.15):

Grupo	**Valor**
PlayerData	- Nivel en el que se encuentra el jugador. - Salud máxima del jugador. - Salud actual del jugador. - Número de cafés. - Número de llaves. - Posición X en el mapa. - Posición Y en el mapa.
LearningOptionsData	- Dificultad. - Fichas por nivel. - Modificador de pasos dados. - Tamaño de batalla. - Daño en batalla. - Puntos por respuesta correcta estándar. - Puntos por respuesta correcta nueva. - Puntos negativos por respuesta errónea estándar. - Puntos negativos por respuesta errónea

	aprendida. - Repaso activado. - Porcentaje de repaso al retomar el juego.
WorldLevelsData	Por cada uno de los niveles almacenados, se carga la semilla y el estado de los tres cofres correspondientes.
CatalogData	- Nombre del catálogo.
CardData	Por cada una de las fichas del catálogo: - Puntuación. - Nueva. - Aprendida.

Table 3.15 – Elementos a cargar

Al terminar, cerramos la conexión con el RMS y cargamos el nivel con la información obtenida.

3.3. Learning Module

El módulo de aprendizaje es corto, pero su función es importante, ya que permite adaptar el juego a todos los gustos (Tabla 3.16). La primera vez que podemos acceder a él es en la pantalla principal; si escogemos el botón “Opciones” seremos llevados a una pantalla que contendrá todo lo que hay por configurar.

La dificultad es el parámetro más notable, siendo este el único punto en el que se puede configurar. Las demás opciones permiten escoger el número de puntos que se sumará o restará de la puntuación de la ficha en caso de aciertos o fallos. Por último podemos activar la opción “Grind” y fijar un porcentaje para el repaso inicial.

Opción	**Función**
Dificultad	Fijar el nivel de dificultad. Con él cambian varias cosas, como la salud máxima del personaje, el número de pasos que podemos dar antes de tener que enfrentarnos en una batalla, el tamaño de esta, el daño que recibimos o las fichas que se activan por cada nivel.
SuccessStandart	Puntos que recibimos por una respuesta correcta.
SuccessNew	Puntos que recibimos por una respuesta correcta para una ficha que sale por primera vez.
MissStandart	Puntos que nos restan por una respuesta incorrecta.
MissLearnt	Puntos que nos restan por una respuesta incorrecta

	para una ficha que ya estaba aprendida.
GridEnabled	Activa o desactiva un repaso inicial de las fichas hasta ese momento, cada vez que reanudamos el juego.
GrindRate	Fija el porcentaje de las fichas que tendremos que repasar si está activa la opción anterior.

Table 3.16 – Opciones de configuración del aprendizaje

Desde el primer momento que aparece, toda esta información se guarda en una clase singleton (LearningOptions) que permite acceder a ella en cualquier punto del programa sin peligro, condiciones o bucles extra para tomar las decisiones.

3.3.1. Dificultad

Según la dificultad en la que juguemos una partida variarán una serie de cosas (Tabla 3.17):

Fácil	**Normal**
Fichas activadas por nivel: 10	Fichas activadas por nivel: 20
Fichas listadas por página: 10	Fichas listadas por página: 20
Energía máxima: 200	Energía máxima: 100
Pasos base: 15	Pasos base: 5
Tamaño de batalla máximo: 3	Tamaño de batalla máximo: 5
Energía perdida por errar: 10	Energía perdida por errar: 15

Table 3.17 – Factores variantes con la dificultad

3.4. World Module

Éste es el segundo módulo en importancia y el primero en cuestión de tamaño. Es el encargado de construir los mapas, colocar y mover al jugador, y sobre todo representar gráficamente todo el conjunto.

3.4.1. MapGenerator

El súmmum de este TFC, la obra maestra de la aplicación presentada, la clase más diseñada y trabajada de todo el proyecto es la encargada de producir un número niveles aleatorios prácticamente infinito, incluso para el jugador más exigente. La “magia” es generada por el contenido procedimental, una idea

olvidada muy a menudo por las grandes compañías que desarrollan juegos, pero casi un requisito obligatorio para el programador *stand-alone*. El contenido procedimental, es básicamente contenido creado por la máquina, a diferencia del contenido generado por los artistas o diseñadores gráficos. He aquí una clase a la que simplemente se le pasa una semilla, aleatoriamente generada por la máquina virtual de Java, el alto y el ancho del mapa que queremos poblar. En unos pocos segundos (generalmente 2 ó 3) produce el mundo deseado y lo presenta ante nosotros.

A grandes rasgos, se crean las habitaciones y estas calculan su posición y la de sus vecinos lindantes. Se toman decisiones sobre qué habitaciones estarán activas y cuáles no; al acabar algunas de ellas pueden o no unirse con otras para crear habitaciones nuevas. A continuación, se aplican algoritmos transversales para asegurar la conectividad y “transitabilidad”, y para finalizar, se calculan las posiciones de los varios elementos que existirán. Pero, ¿qué ocurre realmente dentro de la clase?. No voy a desvelar este misterio, ni tampoco la lógica aplicada a las miles de líneas de esta clase y las demás de su módulo; dejaré eso a la imaginación del lector. Sin embargo, lo que sí quiero es mostrar la conectividad y los elementos necesarios para la obtención de lo que posteriormente llamaremos “Mundo”. Al cabo de todo el proceso algorítmico, obtenemos dos objetos: una matriz de bytes y una imagen del tamaño indicado al principio, nada más.

3.4.2. Map

El objeto Map es el que contendrá los dos elementos anteriores (la matriz y la imagen). Cada una de las casillas de la matriz contiene un byte con la información del elemento que figurará en ese lugar del mapa, mientras que la imagen es la representación gráfica de todo el nivel.

La transformación de matriz a imagen se realiza por medio de tiles (ver 1.3.7). Un tile es una imagen de un tamaño mucho menor al del nivel, que representa una parte del fondo o decorado (un muro, una puerta...). A través de la concatenación y ordenación de muchos tiles, se pueden formar imágenes, fondos o terrenos completos, que dan la sensación de estar en un laberinto en nuestro caso.

3.4.3. MapInfo

Éste es un objeto necesario si queremos controlar el progreso y la navegación del usuario entre los varios niveles por los que puede estar compuesta una partida. Cada elemento de este tipo contendrá un entero con la semilla necesaria para generar el nivel (y con él su correspondiente matriz e imagen) y un vector de bytes de tres posiciones.

El byte almacena el estado de cada uno de los tres cofres que habrá por nivel; un byte a 0 indicará que el cofre no ha sido abierto por el jugador, mientras que si está a 1 quiere decir lo contrario. Dada una semilla, los cofres de ese mapa siempre se crearán en los mismos puntos y en el mismo orden, por lo tanto podemos estar tranquilos usando ese mismo orden para almacenar los estados. Guardaremos la referencia a otro vector que contendrá objetos de este tipo, MapInfo, durante toda la partida del usuario. Esto nos permitirá la navegación por los niveles en ambos sentidos por parte del usuario, sin que ello nos ocupe memoria; tan solo son necesarios 7 (siete) bytes por guardar un mapa.

3.4.4. Player

Esta estructura es indispensable para mantener un seguimiento de los datos del jugador. Aquí guardamos la posición XY del personaje en el mapa (que realmente apunta a un tile dentro de la matriz de la que se habló anteriormente), la salud máxima y actual que tiene, el número de llaves o de cafés Java de los que dispone, entre otras cosas.

3.4.5. World

El mundo es la clase principal que hará la función de controlador durante el juego. Esta clase es también un Singleton y se inicializa pasándole el tamaño (alto y ancho) del mapa en unidades de tile. World guarda una referencia a los objetos Player, LearningOptions y el vector de niveles (MapInfo) entre otras muchas cosas. A grosso modo, este es el orden de ejecución:

Initialize

Según de dónde provenga el usuario, ya sea que inicia una nueva partida o reanuda una existente, los datos se inicializan de distinta manera.

Si el usuario ha escogido jugar una partida nueva, se tratará de obtener los valores de opciones y opciones de aprendizaje. Si no fuera posible, porque el usuario no ha escogido nada por ejemplo, se cogerán los datos por defecto. Con esta información se crearán los objetos Player y LearningOptions si fuese necesario. También es el momento idóneo para obtener la clase que generará números aleatorios, instanciar el MapGenerator y crear un vector vacío para almacenar la información de los niveles que vayamos visitando. Por último, se guardará la estructura de catálogo que se haya escogido para jugar.

En caso de estar cargando una partida la cosa cambia. Los datos del usuario (Player), las opciones (LearningOptions) y el vector de niveles, provendrán de lo que hayamos guardado previamente en RMS. La clase Random y el

MapGenerator se inician siempre como en el caso anterior. De la misma manera, no habrá hecho falta elegir un catálogo, porque el sistema cargará el utilizado y parseará todos los datos y puntuaciones de las fichas que hubiese en RMS. Adicionalmente, en ambos casos guardaremos una referencia al catálogo elegido para jugar, cargando toda la información de este del mismo archivo .fic que esté en el disco duro. Si reanudamos nuestra partida, este será el momento en el cual se cargarán los últimos datos del RMS y se asociarán con el catálogo de la partida.

Todo este código se ejecuta en un Thread separado, con la finalidad de poder mostrar mientras un diálogo indicando que el programa está en proceso de carga. Este diálogo es el InfiniteProgressIndicator que se explicó en el módulo Screens. La razón de ejecutar el diálogo en el Thread principal (EventDispatcherThread) y los procesos de entrada/salida de información en un Thread separado es sencillamente para no bloquear el programa mientras se realizan estas tareas, o evitar que se quede colgado si hubiera un error no controlado.

BuildNextLevel / BuildPreviousLevel / BuildLoadedLevel

Ya está todo dispuesto, ahora toca obtener el nivel en sí. En este punto existen varias posibilidades que se detallan a continuación.

Si estamos avanzando por los niveles, o se está creando el primer nivel absoluto, ejecutaremos el método buildNextLevel; si es un mapa que visitamos por primera vez, el generador de mapas se encargará de construirlo y se nos asignará un número nuevo de fichas activas según el nivel de dificultad en el que jugamos. Por el contrario, si se trata de un nivel ya visitado será construido con una semilla sacada del vector de niveles – no se asignarán fichas en esta ocasión.

En caso de retroceder, usaremos buildPreviousLevel, como la lógica dicta que cualquier nivel generado con esta función debe ser un mapa ya visitado, siempre usaremos una semilla del vector de niveles para generarlo; el generador de mapas devolverá la matriz y la imagen correspondiente.

Por último, en el caso de ser una partida cargada, el nivel será construido con una semilla sacada del vector de niveles utilizando para la ocasión el método buildLoadedLevel, para esta circunstancia especial, deberemos cargar también los datos de localización y demás del RMS. Para cualquier caso en el que se creen el mapa y la matriz a partir de una semilla, también se deberá recorrer el vector de bytes que contiene el estado de los cofres y actualizarlos sobre ambos elementos devueltos, demostrando una sólida estructura de juego que recuerda las modificaciones efectuadas por el jugador.

La ejecución de estos tres métodos, sigue el mismo mecanismo que la inicialización de los datos del mundo: el indicador de carga del nivel lo hace en el Thread principal, y todas las operaciones se realizan bajo la supervisión de un segundo Thread.

Create MiniMap

El siguiente paso es paso es construir el minimapa, una representación a escala de todo el nivel laberíntico completamente visible, para que el usuario lo pueda consultar y no se desespere buscando la salida. En este mapa sin embargo, no aparecerán los cofres ni la salida del nivel, a no ser que el jugador los haya abierto o haya usado la salida para pasar al siguiente nivel, evitando así que el minimapa sea un arma de doble filo. La imagen del mapa se podrá visualizar y esconder en cualquier momento (salvo durante las batallas) mediante el *soft button* izquierdo del MenuBar.

Form

Para ultimar los preparativos previos al juego, deberemos asegurarnos de que toda esta información creada es visible y, en parte, controlada por el usuario. En la pantalla, usaremos lo que hasta el momento conocíamos como barra del título para construir una barra de estado para el usuario, sobrescribiendo la función que la controla. La información, por este orden es:

- Salud actual y máxima de la que dispone el jugador.
- Nivel en el que se encuentra actualmente el jugador.
- Número de cafés que tenemos en nuestro poder para curarnos.
- Llave de nivel: si está negro, significa que no la tenemos; si está dorado indica que ya la hemos encontrado.

Por otro lado, en la barra de menú añadiremos todos los comandos de control y navegación (Tabla 3.18):

Comando	Efecto
Map	Muestra el mapa completo a escala en un Dialog.
Catalog	Permite acceder al catálogo en la pantalla InGameCatalogScreen.
Take Coffee	El personaje toma un café para restaurar una parte de su energía, siempre y cuando el contador de estos sea positivo.
Configure	Abre un menú gráfico que permite configurar las opciones de aprendizaje de la partida.
Save	Guarda la partida en RMS.
Exit	Sale al menú principal (sin guardar).

Table 3.18 – Comandos del menú durante el juego

Cualquier cambio quedará inmediatamente reflejado en esta interfaz de usuario, mediante las llamadas a unas funciones que actualizan los valores que se muestran en pantalla.

Ya sólo queda el último paso gráfico: mostrar el mundo y los movimientos del usuario por él. Para ello tenemos un método (updateOnScreenImage) que obtiene la posición del jugador en la matriz, el tamaño de la pantalla y el de un tile del mapa; todos ellos almacenados con antelación para precisamente este fin. Con unos cálculos decide qué porción del mapa y en qué medida se debe ver en la pantalla, copia esa misma sección de la imagen que nos devolvía el generador de mapas, almacenada en la clase Map de ese nivel, y la utiliza como imagen de fondo para la pantalla actual. No realizamos esto hasta que la nueva imagen no esté completamente construida, lo que se conoce como técnica del *double buffering* o renderización (del inglés *render*).

Esta vez no podemos dejar que las librerías gráficas actúen de controlador y manejen los eventos, así que sobrescribimos un método para detectar *key releases*, que no es otra cosa que la detección de cuándo dejamos de apretar una tecla del terminal. Este listener convierte la información para asegurar qué tecla ha activado el evento y acto seguido circula por un *switch* con distintas opciones. En el caso de las teclas de dirección, servirán para desplazarnos por el mapa y nada más; tal y como están construidos los tiles, la posición actual del jugador en la matriz nos permite saber en qué direcciones le es permitido moverse y actuar en consecuencia. El único caso en el que realizamos comprobaciones con el tile destino es en el caso de los cofres, al ser estos posicionados después de la construcción de la matriz. Tras cada acción del usuario volvemos a ejecutar el método updateOnScreenImage para actualizar los cambios en la imagen del mundo que ve el usuario, efectivamente, dando una sensación de movilidad y libertad de movimiento por el mapa.

En los mapas podremos encontrarnos dos tipos de elementos con los que interaccionar es algo diferente: los cofres y los mecanismos de desplazamiento entre niveles (*warpers* – del inglés *warp*). Por un lado, los cofres se abren con el botón central de la cruceta digital del móvil, pero sólo puede hacerse desde debajo de ellos directamente. Se ha elegido hacerlo de este modo para ahorrar capacidad computacional y también porque el cierre del cofre está en ese punto, es lógico que solamente pueda abrirse desde ese lado. Por otro lado, los *warpers* aparecen en dos tipos: los que contienen una flecha roja indican el lugar de partida de un mapa y también un punto de retorno, a través del cual se puede volver a mapas ya visitados (excepto en el caso del primer mapa absoluto); los que contengan una figura similar a una flecha verde indican la salida del nivel y un punto de acceso para avanzar al siguiente mapa. Únicamente recordar que no será posible avanzar al siguiente nivel si no se ha encontrado la llave correspondiente al mapa actual antes de ello.

Antes de pasar a la siguiente sección importante, merece la pena dar unos detalles y explicar el modo a través del cual el sistema se comunica con el usuario. Utilizamos unos pequeños Dialog que ocupan una pequeña parte de la pantalla, anclándose en el centro de esta. El contenido de estos pop-ups es solamente una cadena de texto con diversa información que tenemos la

necesidad de comunicar al usuario. El mismo componente tiene un temporizador (*timeout*) tras el cual se cierra solo, no permitiendo al usuario hacerlo manualmente, para evitar que cierre por error y no pueda volver a consultar un mensaje relevante para su progreso. Entre este tipo de mensajes podemos encontrar los que aparecen al abrir un cofre, informándonos de su contenido, o los que nos avisan de que no hemos encontrado la llave y no podemos abandonar el nivel, al pasar por el *warp* de salida.

Batallas

Las batallas representan la parte en la que se produce el aprendizaje realmente, se trata de eventos pseudo-aleatorios que nos llevan a una nueva pantalla. En ella se presentarán desafíos del catálogo al jugador y éste tendrá que escribir una de las respuestas válidas para salir victorioso de la situación.

En todo momento el sistema controla el número de pasos dados por el usuario y la decisión de activar un evento de batalla se toma en función de una constante y una variable: la constante es el número de pasos mínimos menos uno, y depende del nivel de dificultad; por su parte la variable es una cifra aleatoria que cambia después de cada batalla, la cual añadimos a la constante para obtener el número de pasos final tras los cuales el jugador será forzado a enfrentarse en una batalla. Inmediatamente tras la activación de este evento, un efecto muy *cool* de *fade-out* nos llevará a la pantalla de desafíos. Aunque solamente veamos una pregunta a la vez, una misma batalla puede estar formada por hasta cinco desafíos consecutivos diferentes, que de nuevo se calculan con una constante y una variable, de manera análoga al caso anterior. Pasado el cálculo cuantitativo nos centramos en la selección de las fichas; un proceso recorrerá las cards que tenemos activas para comprobar si hay alguna marcada como “nueva”, ya que estas tienen preferencia. En caso contrario, y para seguir ciñéndonos al método Leitner, utilizaremos el algoritmo de ordenación QuickSort **[5]** para obtener las fichas del jugador que tienen la puntuación más baja (las que ha fallado más a menudo) y por lo tanto, son las que se tienen que trabajar más.

Una vez en medio de la batalla, el jugador tendrá dos opciones en todo momento.

Por un lado, puede optar por introducir la respuesta al desafío y apretar el comando “Done”. El sistema procederá a analizar el texto introducido con todas las respuestas válidas y mostrará un pop-up con el texto “Ok / Wrong” según la calidad del resultado, asimismo modificará la puntuación de la ficha y sus *tags* especiales (*new*, *learnt*) en función de nuestro desempeño. De cualquier manera se nos llevará a la siguiente pregunta y el proceso continuará hasta que no queden más fichas, momento en el cual pasaremos a la pantalla final.

Por otro lado, puede escoger la opción de “Run” (correr), permitiendo que se salte todas las fichas que pueda haber para trasladarse directamente a la pantalla final. Una vez aquí hay que “pasar factura”: por cada respuesta que hayamos errado se nos restará una parte de nuestra salud, dependiendo del

nivel dificultad en el que juguemos (sólo aparecerá el valor total de salud perdida). Para compensar, también podemos ganar un objeto de curación (*coffee*) en aproximadamente una de cada veinte fichas respondidas correctamente (las respuestas falladas no sirven). Por último, si huimos de una batalla tendremos un tercio de probabilidad de recibir un daño equivalente a errar un desafío; si no sabemos la respuesta es mejor huir en términos de salud perdida, sin embargo no tendremos oportunidad de ganar objetos de curación ni de mejorar la puntuación de nuestras fichas.

Game Over y fin del juego

El único momento en el que se controlará la vida es al salir de una batalla, es en ese instante cuando se actualizará en la barra de información del usuario y cuando se comprueba si es menor o igual a cero; en este caso el juego termina con la pantalla de Game Over. Realmente, mientras esto no ocurra el juego no finalizará nunca y eso es algo muy positivo desde prácticamente cualquier punto de vista. Si en algún momento el usuario completara el juego y éste se acabara, ni siquiera sería capaz de aprender fichas que se han quedado atrás o repasar su catálogo pasado un tiempo, de esta manera el juego siempre estará disponible para cuando el usuario lo quiera retomar. Por otro lado, es raro crearse un catálogo de mil fichas para empezar, la mayoría de los usuarios comenzarán con catálogos más pequeños a los que les gustaría ir añadiendo fichas. Esto será posible gracias a la implementación que se ha escogido para los catálogos y el RMS; podemos aprender todo un catálogo de cien fichas y, en un momento dado, añadir otras cien de la misma temática. Podremos continuar el juego desde el mismo punto donde lo dejamos, pero con un catálogo más amplio. Por último, es más divertido seguir explorando y recorriendo laberintos nuevos mientras aprendemos, que estar vagando por mapas ya conocidos y completados.

CAPÍTULO 4. EPÍLOGO

El capítulo final complementará el trabajo mediante la inclusión de unas conclusiones sobre el mismo, un apartado sobre las limitaciones técnicas que han perjudicado su implementación y una sección con líneas futuras, las cuales se plantean para mejorar el proyecto y ampliar su abanico de posibilidades. Se cierra este capítulo comentando brevemente los efectos medioambientales que puedan surgir.

4.1. Conclusiones

Llegados a este punto, efectuar el proyecto me ha producido una sensación de satisfacción y realización personal imposible de explicar con palabras. Mirando atrás, esta idea surgió hace casi dos años como un proyecto personal por un lado, y por otro, atrajo la posibilidad de codificar programas o juegos completos en JME a nivel comercial, sin contar con ningún equipo o personal extra.

Llevar a cabo el TFC a supuesto alrededor de un año y medio de aprendizaje de JME mediante varias aplicaciones previas y sus respectivas interficies gráficas de usuario. Ha ayudado a conocer el proceso completo de desarrollo de un videojuego, recorriendo todos los pasos y adoptando los roles correspondientes a cada sector; se ha conocido en profundidad la API de multimedia para teléfonos móviles, así como las reveladoras librerías LWUIT. Todo ello siendo nada más que el preámbulo del contenido procedimental, que se ha implementado y explotado en el flexible generador de mapas aleatorios, produciendo frutos útiles y posiblemente aplicables no sólo a este área.

En una línea paralela, se ha obtenido un producto acabado y funcional con dos características principales, tal y como se ha ido relatando a lo largo de este documento: un gestor de información textual y/o audiovisual y un entretenido juego de aprendizaje que recorre laberintos. Por una parte, se ha logrado el objetivo inicial con creces: el usuario dispone de un proceso completo a través del cual es capaz de forjar su material de estudio y además añadir contenido audiovisual, guardarlo, intercambiarlo o expandirlo en cualquier momento. Por otro lado, las modificaciones decididas poco antes de comenzar el proyecto (la más importante de las cuales era hacer un videojuego de la aplicación) han demostrado ser factibles y se han logrado con bastante variedad y satisfacción.

En definitiva, a partir de este punto de inflexión, los usuarios dispondrán en todo momento de un software de aprendizaje personal y que se adapta a las necesidades de cada cliente.

4.2. Limitaciones técnicas

A pesar de que la tecnología de los teléfonos móviles ha experimentado un inmenso crecimiento durante los últimos años y que existen terminales cuya potencia equivale a la de un Pentium de 2.0 Ghz, la parte correspondiente a la máquina virtual Java se ha quedado estancada. Java necesita un espacio de memoria para tratar y almacenar sus clases y objetos, espacio que irá en aumento según el tamaño y la extensión de la aplicación que se quiere ejecutar. En el ejemplo anterior (Nokia N95), el terminal tiene 128 Mb de memoria para la ejecución de sus programas nativos o de su sistema operativo (Symbian OS), sin embargo a la máquina virtual de Java sólo se le permite tomar 1,5 Mb! – con este panorama, se han tenido que hacer malabarismos para evitar la aparición constante del terrorífico *OutOfMemoryException*.

Aun así, la aplicación sufre de un defecto importante: se tiene en memoria el mapa al completo para evitar un procesado excesivo y "tirones" durante el juego; lo que ha causado el uso de la mayor parte de la memoria disponible, a pesar de que se ha optimizado al límite. Lamentablemente, esta situación impide que se pueda reproducir el contenido multimedia de los desafíos durante el juego, debido a la falta de memoria para ello. Esto se debe a un mal diseño de la sección, que a estas alturas es difícil arreglar o modificar, por cuestiones de tiempo y porque haría peligrar la estabilidad del resto de la aplicación.

Por otro lado, el perfil MIDP 2.0 ha supuesto un incremento notable en la seguridad de los programas que se pueden ejecutar en el terminal (seguridad inexistente anteriormente). A pesar de la bondad de Sun en este aspecto, ahora se necesita un certificado para acceder a determinadas funcionalidades del teléfono con nuestra aplicación, certificado de Verisign o Thawte por valor de 4000$, que permite firmar productos durante un año. La cosa va a más, dado que a partir del Symbian S60 tercera versión (la del Nokia N95 de pruebas) es imposible "trucar" la lista de Autoridades Certificadoras para incluir nuestro propio certificado. El resultado es una serie de molestos *prompts*, que nos piden confirmación para leer y escribir en el disco duro o utilizar nuestros datos personales del teléfono, impidiendo automatizar el proceso.

4.3. Líneas futuras

Se han alcanzado muchas de las metas propuestas al inicio de este TFC, algunas se han descartado, otras están por hacer y también han surgido muchas otras durante la realización.

Entre los aspectos más importantes figuran el arreglo de la utilización de memoria para los mapas, y así permitir la reproducción de contenido multimedia; la implementación de un servidor y una comunidad mediante la cual surjan nuevos desafíos y más funcionalidades; la compatibilidad con numeroso programas de la misma índole, más detalles y opciones para el repaso y el aprendizaje… las posibilidades son infinitas.

4.4. Efectos medioambientales

Actualmente la industria de los teléfonos móviles produce toneladas de basura, pero poco a poco comienzan a darse cuenta y a reciclar terminales o a reutilizar componentes. Este TFC no va a cambiar la situación actual, pero al menos intenta no contribuir a la contaminación acústica o vibratoria, excluyendo estas características de la mayor parte del proyecto (a excepción de la situación en la cual se ha de reproducir audio o video concretamente).

Se trata de aprovechar y utilizar los canales y medios que ya existen para la transmisión de datos, evitando crear nuevos protocolos o inundar el espacio con información excesiva a ser transmitida. Por todo ello, se puede decir que la aplicación aquí presentada no producirá efectos medioambientales negativos.

BIBLIOGRAFÍA

1. J. WELLSH, Martin. *J2ME Game Programming*, USA: Premier Press, 2004.
2. KEOGH, James. *J2ME The Complete Reference*, USA: McGraw-Hill, 2003.
3. SUN Microsystems, *LWUIT – Developer Guide*, Worldwide: Sun Microsystems Inc., 2008.
4. LWUIT Forum, http://forums.java.net/jive/forum.jspa?forumID=139 , 2009.

REFERENCIAS

[1] - http://java.sun.com/javase/reference/index.jsp
[2] - http://java.sun.com/javame/index.jsp
[3] - http://developers.sun.com/mobility/midp/articles/midp20/
[4] - http://en.wikipedia.org/wiki/Widget_engine
[5] - http://www.cs.ubc.ca/~harrison/Java/sorting-demo.html