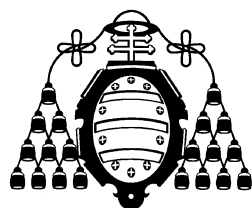


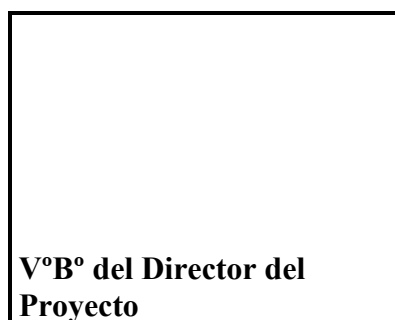
# UNIVERSIDAD DE OVIEDO



ESCUELA UNIVERSITARIA DE INGENIERÍA TÉCNICA  
EN INFORMÁTICA DE OVIEDO

## PROYECTO FIN DE CARRERA

AVENTURA GRÁFICA PARA MÓVILES “ENTRE 2 ALMAS”



**Director:** Iván Fernández Lobo

**Autor:** Pedro Javier Sáez Martínez





## Resumen

Entre 2 Almas es una aventura gráfica en 2D, en la que el jugador tiene como objetivo el participar en una historia interactiva, en la cual él pasa a ser el protagonista, teniendo, que interactuar con personajes y objetos, para resolver por medio de la lógica una serie de interrogantes para así poder seguir avanzando en la historia.

Entre la acciones que un jugador podrá realizar en el juego se encuentran la de hablar con otros personajes, interactuar con objetos, moverse entre distintos escenarios, ver distintos videos que le irán saltando durante el juego, cargar y guardar partidas, o incluso cambiar de tipo de menú.

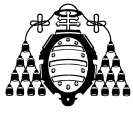
Además aquellos usuarios más avanzados, y que tengan ciertos conocimientos de XML, podrán crear su propia aventura gráfica, simplemente modificando unos archivos .xml, sin necesidad de recompilar ni tocar código. Esto es gracias a que los lugares, objetos, personajes, secuencias de diálogos... están definido por XML, lo que convierte al juego en 100 % configurable y 100 % internacionalizable.

La plataforma donde se podrá ejecutar el juego será el teléfono móvil con tecnología Java. Aunque dentro de estos, como en los ordenadores, siempre hay requerimientos básicos, como el que estos móviles, para que el juego pueda ser ejecutado en ellos, deben tener MIDP 2.0, un determinado tamaño de pantalla, etc...

No se utiliza ningún sdk propietario de ninguna marca, así que el juego puede ser ejecutado por cualquier modelo, de cualquier marca, que cumpla con los requerimientos mínimos. Es decir no está limitado a ninguna marca o modelo en concreto.

## Palabras clave

- Videojuego.
- Aventura gráfica.
- Gráficos 2D.
- Móvil.
- Configurable.
- Internacionalizable.
- J2ME.
- Midlet.







# Índice

<b>RESUMEN .....</b>	<b>3</b>
<b>PALABRAS CLAVE.....</b>	<b>3</b>
<b>ÍNDICE.....</b>	<b>5</b>
<b>TABLA DE ILUSTRACIONES.....</b>	<b>11</b>
<b>1. INTRODUCCIÓN .....</b>	<b>13</b>
1.1. ¿QUÉ ES UN VIDEOJUEGO? .....	13
1.2. ¿POR QUÉ UN JUEGO PARA MÓVILES? .....	14
1.3. AVENTURA GRÁFICA .....	15
<b>2. MEMORIA DEL PROYECTO.....</b>	<b>17</b>
2.1. DESCRIPCIÓN .....	17
2.2. OBJETIVOS.....	18
2.3. REFERENCIAS .....	19
<b>3. DISEÑO DEL JUEGO.....</b>	<b>21</b>
3.1. MECÁNICA DEL JUEGO .....	21
3.1.1. Cámara.....	21
3.1.2. Teclas.....	21
3.1.3. Pantallas.....	22
3.1.3.1. En un escenario.....	22
3.1.3.2. En una conversación.....	23
3.1.3.3. En el mapa .....	24
3.1.3.4. En un video .....	25
3.1.3.5. En el menú .....	25
3.1.4. Sonido.....	25
3.1.5. Acciones disponibles .....	26
3.1.6. Menú.....	27
3.2. ELEMENTOS DEL JUEGO .....	28
3.2.1. Personajes .....	28
3.2.1.1. Personajes principales.....	28
3.2.1.2. Personajes secundarios .....	31
3.2.2. Mapas .....	32
3.2.3. Escenarios .....	33
3.3. HISTORIA.....	34
3.3.1. Video inicial .....	34
3.3.2. Fase 1: ¿Qué hago aquí?.....	35
3.3.3. Video 1 .....	36
3.3.4. Fase 2: Vaya pintas!! .....	36
3.3.5. Video 2 .....	38
3.3.6. Fase 3: Maldito gato .....	38
3.3.7. Video Final .....	39
<b>4. ANÁLISIS .....</b>	<b>41</b>
4.1. ESPECIFICACIÓN DE REQUERIMIENTOS.....	41



4.1.1.	Requisitos generales .....	41
4.1.2.	Requisitos del menú .....	41
4.1.3.	Requisitos de la interfaz .....	42
4.1.4.	Requisitos de la historia.....	42
4.1.5.	Requisitos de jugador .....	42
4.1.6.	Requisitos de desarrollador .....	43
4.2.	IDENTIFICACIÓN DE ACTORES .....	43
4.3.	IDENTIFICACIÓN DE LOS CASOS DE USO .....	43
4.3.1.	Mirar ítem .....	44
4.3.2.	Coger ítem .....	45
4.3.3.	Usar ítem .....	45
4.3.4.	Ir a ítem.....	46
4.3.5.	Combinar ítems .....	47
4.3.6.	Ir al menú.....	47
4.3.7.	Hablar con personas.....	48
4.3.8.	Ver video .....	48
4.3.9.	Moverse por escenario.....	49
4.3.10.	Moverse entre escenarios.....	49
4.3.11.	Moverse entre estancias.....	50
4.3.12.	Cambiar de fase .....	50
4.4.	ESPECIFICACIONES TÉCNICAS .....	51
4.5.	DIAGRAMA DE FLUJO DEL MENÚ .....	51
4.6.	DIAGRAMA DE ESTADOS .....	52
4.7.	PATRONES DE DISEÑO .....	53
4.8.	ESTRUCTURAS DE DATOS.....	53
4.9.	BASE DE DATOS .....	55
4.10.	PROTOTIPOS.....	55
<b>5.</b>	<b>DISEÑO.....</b>	<b>57</b>
5.1.	DISEÑO DEL MENÚ .....	57
5.1.1.	Diagrama de clases .....	57
5.1.2.	Descripción detallada de las clases.....	57
5.1.2.1.	CountDown.....	57
5.1.2.2.	Entre2Almas .....	58
5.1.2.3.	SplashScreen.....	63
5.1.2.4.	UIFactory.....	65
5.1.2.5.	UITerminalST.....	66
5.1.2.6.	UITerminalSTBG .....	69
5.1.2.7.	Utils .....	71
5.2.	DISEÑO DE LA ESTRUCTURA .....	83
5.2.1.	Diagrama de clases .....	83
5.2.2.	Descripción detallada de las clases.....	84
5.2.2.1.	Entre2Almas .....	84
5.2.2.2.	Utils .....	84
5.2.2.3.	AVLTree.....	84
5.2.2.4.	Combinacion.....	87
5.2.2.5.	Comparable.....	88
5.2.2.6.	Conversacion .....	88
5.2.2.7.	Ejecucion .....	91
5.2.2.8.	Escenario .....	107



5.2.2.9.	Estancia.....	109
5.2.2.10.	Item.....	112
5.2.2.11.	Mapa.....	117
5.2.2.12.	Node.....	118
5.2.2.13.	Opcion.....	120
5.2.2.14.	Opcionable.....	121
5.2.2.15.	PathFinding.....	123
5.2.2.16.	Resultado.....	125
5.2.2.17.	Video.....	125
5.3.	DIAGRAMAS DE SECUENCIA.....	127
5.3.1.	Mirar ítem.....	127
5.3.2.	Coger ítem.....	128
5.3.3.	Usar ítem.....	129
5.3.4.	Ir a Ítem.....	130
5.3.5.	Combinar ítems.....	131
5.3.6.	Ir al menú.....	132
5.3.7.	Hablar con personas.....	133
5.3.8.	Ver video.....	134
5.3.9.	Moverse por escenario.....	134
5.3.10.	Moverse entre escenarios.....	135
5.3.11.	Moverse entre estancias.....	136
5.3.12.	Cambiar de fase.....	136
5.4.	DESCRIPCIÓN DE LOS FICHEROS XML.....	137
5.4.1.	Estructura y función de basics.xml.....	137
5.4.1.1.	Visión general.....	138
5.4.1.2.	Basicos / menu.....	140
5.4.1.3.	menu / menuPrin / opciones.....	143
5.4.1.4.	menu / ayuda / opciones.....	144
5.4.1.5.	Basicos / apariencias.....	145
5.4.2.	Estructura y función de FaseX.xml.....	146
5.4.2.1.	Visión general.....	146
5.4.2.2.	Fase / Videos.....	147
5.4.2.3.	Fase / Conversaciones.....	148
5.4.2.4.	Conversaciones / Conversación / Opciones.....	148
5.4.2.5.	Conversación / Opciones / Opcion / Consecuencias.....	149
5.4.2.6.	Fase / Mapas.....	149
5.4.2.7.	Mapas / Mapa /Estancias.....	150
5.4.2.8.	Estancias / Estancia / Opciones.....	151
5.4.2.9.	Estancia / Opciones / Opcion / Consecuencias.....	152
5.4.2.10.	Estancias / Estancia / Escenarios.....	152
5.4.2.11.	Escenarios / Escenario / Items.....	154
5.4.2.12.	Items / Item / combinaciones.....	156
5.4.2.13.	Identificadores de resultados.....	156
6.	IMPLEMENTACIÓN.....	161
6.1.	TECNOLOGÍA.....	161
6.1.1.	Lenguaje de programación.....	161
6.1.1.1.	Elección.....	161
6.1.1.2.	Java 2 Platform, Micro Edition (J2ME).....	161
6.1.1.3.	Resumen.....	164



6.1.2.	Entorno de desarrollo.....	164
6.1.3.	Librerías y código externo.....	165
6.1.3.1.	Librería KXML .....	165
6.1.3.2.	Árbol AVL.....	165
6.1.3.3.	Algoritmo A* (A Estrella).....	166
6.2.	GUÍA DE ESTILO DE PROGRAMACIÓN .....	167
6.3.	ESTRUCTURA DE FICHEROS.....	168
<b>7.</b>	<b>MANUAL .....</b>	<b>171</b>
7.1.	REQUISITOS MÍNIMOS .....	172
7.2.	OBJETIVO DEL JUEGO.....	172
7.3.	INSTALACIÓN.....	172
7.4.	EL MENÚ.....	173
7.5.	CONTROLES.....	174
7.6.	PANTALLAS .....	175
7.6.1.	En un escenario.....	175
7.6.2.	En una conversación.....	176
7.6.3.	En el mapa .....	176
7.6.4.	En un video.....	177
7.7.	CRÉDITOS .....	177
7.7.1.	Equipo de desarrollo.....	177
7.7.2.	Agradecimientos.....	177
<b>8.</b>	<b>PRUEBAS .....</b>	<b>179</b>
8.1.	ALFA .....	179
8.1.1.	Objetivos.....	179
8.1.2.	Casos de prueba.....	179
8.1.2.1.	Pruebas de interfaz.....	179
8.1.2.2.	Pruebas del núcleo de ejecución.....	182
8.1.2.3.	Pruebas de lógica.....	183
8.2.	BETA .....	183
8.2.1.	Objetivos.....	183
8.2.2.	Casos de prueba.....	183
8.2.2.1.	Pruebas de interfaz.....	183
8.2.2.2.	Pruebas del núcleo de ejecución.....	185
8.2.2.3.	Pruebas de lógica.....	186
<b>9.</b>	<b>AMPLIACIONES.....</b>	<b>187</b>
9.1.	MINIJUEGOS .....	187
9.2.	SONIDO.....	187
9.3.	VIDEOS.....	187
<b>POSTMORTEN.....</b>		<b>189</b>
<b>GLOSARIO.....</b>		<b>191</b>
<b>BIBLIOGRAFÍA .....</b>		<b>193</b>
<b>APENDICE A: CÓDIGO FUENTE .....</b>		<b>195</b>
AVLTREE.....		195
COMBINACION .....		202
COMPARABLE .....		204



CONVERSACION .....	204
COUNTDOWN .....	207
EJECUCION .....	208
ENTRE2ALMAS .....	280
ESCENARIO .....	297
ESTANCIA .....	300
ITEM .....	304
MAPA .....	309
NODE .....	311
OPCION .....	314
OPCIONABLE .....	315
PATHFINDING .....	317
RESULTADO .....	325
SPLASHSCREEN .....	326
UIFACTORY .....	329
UITERMINALST .....	330
UITERMINALSTBG .....	337
UTILS .....	360
VIDEO .....	374





## Tabla de ilustraciones

IMAGEN 1: VIDEOJUEGOS .....	13
IMAGEN 2: SÚPER NINTENDO .....	13
IMAGEN 3: JUGANDO CON EL MÓVIL .....	14
IMAGEN 4: MÍTICAS AVENTURAS REALIZADAS CON EL MOTOR SCUMM .....	16
IMAGEN 5: ZONA DE JUEGO, TEXTO E INVENTARIO .....	23
IMAGEN 6: OPCIONES Y VENTANA DE MENSAJES .....	23
IMAGEN 7: CONVERSACIÓN .....	24
IMAGEN 8: MAPA .....	25
IMAGEN 9: VIDEO .....	25
IMAGEN 10: OSCAR Y SU VERSIÓN MALVADA ROMO .....	29
IMAGEN 11: IMAGEN DEL INSPECTOR RINGO .....	30
IMAGEN 12: BOCETO DEL INSPECTOR DUTE .....	30
IMAGEN 13: BOCETOS DEL DETECTIVE PRIVADO ALFREDO SALABERRI .....	31
IMAGEN 14: DEPENDIENTE DE UNA TIENDA DE ALIMENTACIÓN CHINA .....	31
IMAGEN 15: PRESO .....	32
IMAGEN 16: GATO DEL DETECTIVE PRIVADO ALFREDO SALABERRI .....	32
IMAGEN 17: MAPA DE ENTRE 2 ALMAS .....	33
IMAGEN 18: ESCENARIOS .....	34
IMAGEN 19: DIAGRAMA DE CASOS DE USO .....	44
IMAGEN 20: DIAGRAMA DE FLUJO DEL MENÚ .....	52
IMAGEN 21: DIAGRAMA DE ESTADOS .....	52
IMAGEN 22: IMÁGENES DE LA MAQUETA .....	56
IMAGEN 23: DIAGRAMA DE CLASES DEL MENÚ .....	57
IMAGEN 24: CLASE COUNTDOWN .....	57
IMAGEN 25: CLASE ENTRE2ALMAS .....	58
IMAGEN 26: CLASE SPLASHSCREEN .....	63
IMAGEN 27: CLASE UIFACTORY .....	65
IMAGEN 28: CLASE UITERMINALST .....	66
IMAGEN 29: CLASE UITERMINALSTBG .....	69
IMAGEN 30: CLASE UTILS .....	72
IMAGEN 31: DIAGRAMA DE CLASES DE LA ESTRUCTURA .....	83
IMAGEN 32: CLASE AVL TREE .....	84
IMAGEN 33: CLASE COMBINACION .....	87
IMAGEN 34: CLASE COMPARABLE .....	88
IMAGEN 35: CLAS CONVERSACION .....	88
IMAGEN 36: CLASE EJECUCION .....	91
IMAGEN 37: CLASE ESCENARIO .....	107
IMAGEN 38: CLASE ESTANCIA .....	109
IMAGEN 39: CLASE ÍTEM .....	112
IMAGEN 40: CLASE EPIGRAFE .....	117
IMAGEN 41: CLASE OPCION .....	120
IMAGEN 42: CLASE OPCIONABLE .....	121
IMAGEN 43: CLASE RESULTADO .....	125
IMAGEN 44: CLASE VIDEO .....	125
IMAGEN 45: DIAGRAMA DE SECUENCIA DE "MIRAR ÍTEM" .....	127
IMAGEN 46: DIGRAMA DE SECUENCIA DE "COGER ÍTEM" .....	128
IMAGEN 47: DIAGRAMA DE SECUENCIA DE "USAR ÍTEM" .....	129



IMAGEN 48: DIAGRAMA DE SECUENCIA DE "IR A ÍTEM" .....	130
IMAGEN 49: DIAGRAMA DE SECUENCIA DE "COMBINAR ITEMS" .....	131
IMAGEN 50: DIAGRAMA DE SECUENCIA DE "IR AL MENÚ" .....	132
IMAGEN 51: DIAGRAMA DE SECUENCIA DE "HABLAR CON PERSONAS" .....	133
IMAGEN 52: DIAGRAMA DE SECUENCIA DE "VER VIDEO" .....	134
IMAGEN 53: DIAGRAMA DE SECUENCIA DE "MOVERSE POR ESCENARIO" .....	134
IMAGEN 54: DIAGRAMA DE SECUENCIA DE "MOVERSE ENTRE ESCENARIO" .....	135
IMAGEN 55: DIAGRAMA DE SECUENCIA DE "MOVERSE ENTRE ESTANCIAS" .....	136
IMAGEN 56: DIAGRAMA DE SECUENCIA DE "CAMBIAR DE FASE" .....	136
IMAGEN 57: BASICOS.XML (VISIÓN GENERAL).....	138
IMAGEN 58: BASICOS.XML (BASICOS / MENU) .....	141
IMAGEN 59: BASICOS.XML (MENU / MENU PRIN / OPCIONES ) .....	143
IMAGEN 60: BASICOS.XML (MENU / AYUDA / OPCIONES ).....	144
IMAGEN 61: BASICOS.XML (BASICOS / APARIENCIAS).....	145
IMAGEN 62: FASEX.XML (VISIÓN GENERAL).....	146
IMAGEN 63: FASEX.XML (FASE / VIDEOS) .....	147
IMAGEN 64: FASEX.XML (FASE / CONVERSACIONES).....	148
IMAGEN 65: FASEX.XML (CONVERSACIONES / CONVERSACIÓN / OPCIONES).....	148
IMAGEN 66: FASEX.XML (CONVERSACIÓN / OPCIONES / OPCION / CONSECUENCIAS) ..	149
IMAGEN 67: FASEX.XML (FASE / MAPAS) .....	149
IMAGEN 68: FASEX.XML (MAPAS / MAPA / ESTANCIAS) .....	150
IMAGEN 69: FASEX.XML (ESTANCIAS / ESTANCIA / OPCIONES).....	151
IMAGEN 70: FASEX.XML (ESTANCIA / OPCIONES / OPCION / CONSECUENCIAS).....	152
IMAGEN 71: FASEX.XML (ESTANCIAS / ESTANCIA / ESCENARIOS).....	152
IMAGEN 72: FASEX.XML (ESCENARIOS / ESCENARIO / ITEMS).....	154
IMAGEN 73: FASEX.XML (ITEMS / ITEM / COMBINACIONES).....	156
IMAGEN 74: ARQUITECTURA DE LA PLATAFORMA JAVA 2 DE SUN.....	162
IMAGEN 75: ARQUITECTURA DEL ENTORNO DE EJECUCIÓN J2ME.....	164
IMAGEN 76: ESCENARIO CON SU AREA CAMINABLE.....	166
IMAGEN 77: MENU PRINCIPAL (AVANZADO) .....	173
IMAGEN 78: VISTA DEL MENÚ AVANZADO Y MENÚ CLÁSICO .....	180





# 1. Introducción

## 1.1. ¿Qué es un videojuego?

Un videojuego es un programa informático, creado expresamente para divertir, basado en la interacción entre una persona y un aparato electrónico donde se ejecuta el videojuego. Estos recrean entornos virtuales en los cuales el jugador puede controlar a un personaje o cualquier otro elemento de dicho entorno, para conseguir uno o varios objetivos por medio de unas reglas determinadas.

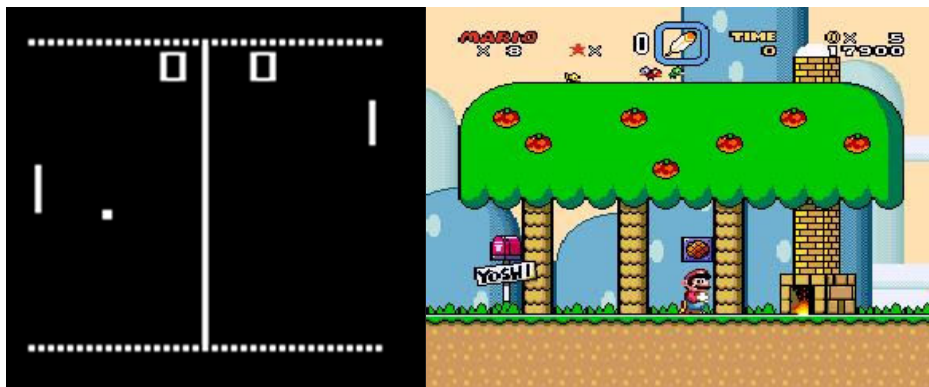


Imagen 1: Videojuegos

Los videojuegos pueden ser grabados en algún medio de almacenamiento (como un CD, un DVD...) para ser ejecutados en un aparato específico. El hardware que ejecuta los videojuegos puede ser desde una computadora a un artefacto especialmente creado para ello, como las videoconsolas o las máquinas arcade, pasando por teléfonos móviles y otros dispositivos electrónicos.



Imagen 2: Súper Nintendo

Los videojuegos, dependiendo de cual sea su finalidad, se dividen en géneros:



- Aventura → Juegos en los que el protagonista debe avanzar en la trama interactuando con diversos personajes y objetos.
- Deportivo → Juegos basados en deportes.
- Shoot'em up → Juego de disparos.
- Educativo → Juegos cuyo objetivo es transmitir al jugador algún tipo de conocimiento.
- Estrategia → Se caracterizan por la necesidad de manipular a un numeroso grupo de personajes, objetos o datos para lograr objetivos varios.
- Lucha → Juegos basados en el combate físico.
- Plataformas → Juegos en los que el protagonista ha de avanzar a través de un mapeado con múltiples alturas.
- Puzzle → Juegos de inteligencia.
- Rol → El protagonista interpreta un papel y ha de mejorar sus habilidades.
- Simulación → Juegos basados en la reproducción, generalmente de forma realista, del funcionamiento de alguna actividad.
- Carreras → Son juegos en los que se pilotan diferentes vehículos, ya sean reales o ficticios, para ganar en diferentes carreras.

Pero los videojuegos no son solo diversión, sino que alrededor de estos se ha creado una industria, que en los últimos años ha estado generando más dinero que la del cine, formada por grandes multinacionales, que invierten enormes sumas de dinero, en esta tarea multidisciplinar, que es desarrollar un juego.

## 1.2. ¿Por qué un juego para móviles?

Son varios los motivos que me han llevado a elegir el móvil como la plataforma para la cual crear el juego.



**Imagen 3: Jugando con el móvil**

Uno de ellos es que actualmente todo o casi todo el mundo tiene uno o incluso varios móviles, es decir, cada persona tiene una pequeña consola de videojuegos en su bolsillo, puede que no jueguen con ella, pero tienen esa posibilidad. Lo cual significa



que si tu desarrollas un juego para móvil, estarás creando un producto para la plataforma de videojuegos más vendida del mundo.

Otro de los motivos es que, desde hace unos años, los móviles incluyen una maquina virtual de Java limitada, y partiendo de que mi lenguaje de programación favorito es Java, hace que esta plataforma me atraiga aun más.

Además me atrae también la posibilidad de aprender J2ME, que es el lenguaje utilizado en la programación Java para móviles, y que actualmente es muy demandado por un sector en constante crecimiento.

Pero la principal motivación, es el poder sentirme como unos años atrás, cuando se creaban sencillos pero entretenidos juegos para PC que podían ser realizados completamente por uno o pocos programadores. Esa posibilidad de crear un juego completo, que tenga una calidad parecida a los que hay en el mercado y el poder verlo en una plataforma, pudiendo jugarlo en su totalidad o que otros lo jueguen, es una idea que es difícil que no atraiga.

### **1.3. Aventura gráfica**

Ante la pregunta de que es una aventura grafica, cuyo antecedente son las aventuras conversacionales, yo creo que se podría describir como un genero de los juegos que se caracteriza por que el usuario del producto participa en una historia interactiva, en la cual él pasa a ser el protagonista, teniendo, que interactuar con personajes y objetos, para resolver por medio de la lógica una serie de interrogantes para así poder seguir avanzando en la historia. En este tipo de juegos lo normal es que la visión del protagonista sea en 3º persona, pero siempre hay excepciones.

Los pioneros en este tipo de aventuras fueron Sierra (con Leisure Suit Larry, King's Quest o Police Quest)y LucasFilm Games, la actual LucasArts, (Maniac Mansión, The Secret of Monkey Island o Indiana Jones and the Fate of Atlantis).

Uno de las claves del éxito de LucasArts fue su motor SCUMM, que fue creado para Maniac Mansión en 1988, y permitía a los diseñadores de juegos crear lugares, objetos y secuencias de diálogos sin necesidad de escribir en el mismo lenguaje con el que se había escrito el código fuente del juego. SCUMM fue utilizado en algunos juegos míticos como Day of the Tentacle o Sam & Max Hit the Road, entre otros.



Imagen 4: Míticas aventuras realizadas con el motor SCUMM

Viendo la importante función que tuvo SCUMM en el progreso de LucasArts, en mi proyecto haré algo parecido, a un nivel obviamente más sencillo, usando para ello XML.

El porque me decidí por este genero, es por que a mí siempre me ha interesado el tema de los juegos y sobre todo siempre he tenido la ilusión de desarrollar mi propio juego, el que una idea tuya se plasme en un juego y que esta la disfrute otra gente me parece una forma, por lo menos, gustosa de dedicar tu tiempo, por eso creo, que esta es una buena oportunidad de realizarlo a mi manera y gusto sin tener demasiadas presiones externas.

Además la lectura es algo que me gusta y recomiendo, y también me atrae la idea de escribir un libro, por eso me incline por el genero de las aventuras graficas, ya que creo que es una manera de aunar programación de juegos y en menor medida la creación de un libro. Además de esto las aventuras graficas es un genero con el que crecí y que creo que abre mucho la mente, ya que te hacen esforzarte en conseguir unos objetivos y si además esta tarea se hace divertida o entretenida, pues tenemos un cóctel que a mucha gente le encanta, entre ellos a mi.



## 2. Memoria del proyecto

### 2.1. Descripción

La aplicación es un juego, una aventura gráfica, genero poco explotado todavía en los teléfonos móviles, ya que por ejemplo a mi me encantan, y no tengo muchas alternativas a las que poder jugar.

El mundo móvil es complejo, ya que cada marca y modelo tiene sus propias peculiaridades, pero la idea es no utilizar ningún sdk propietario de ninguna marca, para así intentar que el juego sea lo más portable, entre móviles, posible.

Para que la aventura gráfica guste a la gente, se cuenta con unos buenos gráficos, adaptados a las características de los móviles actuales, y una buena historia que capte a la gente. Aunque la calidad de ambos campos la tendrá que fijar el usuario final, el jugador.

Una vez iniciemos la aplicación, entraremos al menú principal, donde podremos escoger entre 2 tipos de menús, pudiendo cambiar de uno a otro en cualquier momento, a través de la pantalla de opciones. Al cerrar la aplicación se guardará el último tipo de menú utilizado, para que la próxima vez que inicies la aplicación vuelva a mostrarse ese mismo menú.

Cuando empecemos a jugar podremos volver al menú en cualquier momento, simplemente pulsando sobre un acceso directo. Así durante la ejecución del juego podremos cargar o guardar la partida. No será necesario llegar a un determinado lugar para poder guardar la partida, sino que en cualquier momento, excepto cuando estés en un video o en una conversación, podrás guardar tu estado.

Durante el juego, deberemos realizar una serie de acciones, resolver unos determinados puzzles, y tener una conversaciones con algunos personaje, para así poder seguir avanzando en la historia. Entre las posibilidades que tendremos están la de coger objetos, combinarlos, hablar con gente, viajar entre distintos mapas, estancias y escenarios, ver videos...

Nuestro personaje principal además cambiará de apariencia de vez en cuando, a veces llevará gorra, otras irá de verde, otras se le verá pequeño, todas ellas acordes con su situación actual.

Y por último mencionar una característica que al jugador común no le interesará, pero para aquellos que estén interesados en desarrollar su propia aventura gráfica, seguro que es algo muy interesante. Y esta es la posibilidad de crear su propia aventura gráfica para móviles, sin necesidad de tocar una sola línea de código, simplemente modificando unos archivos XML, y por su puesto poner tus propias imágenes. Así este



usuario-desarrollador podrá crear su aventura gráfica, en el idioma que quiera, con las imágenes que quiera, y con la historia que quiera. Para lo cual sólo necesita conocimiento en XML.

## 2.2. Objetivos

**Desarrollar aventura grafica:** Y esto ya es algo complicado, ya que a mi entender, este es uno de los géneros más difíciles de programar, ya que tienes que tener en cuenta multitud de posibilidades, y no te limitas sólo a poner 4 escenarios y 3 bichos a los que matar.

**Ejecutable en teléfonos móviles:** El juego deberá poder ser ejecutado en al menos un móvil, aunque se espera que pueda ser ejecutado en muchos más, ya que se ha hecho una programación, que pese a tener unos requerimientos altos, no se ha limitado a una marca en concreto.

**Historia entretenida:** Esto es algo subjetivo, ya que lo que para mi es algo divertido para el resto puede ser un aburrimiento, pero creo que la historia tiene un nivel aceptable, que entretiene, con variados puzzles y situaciones. Espero que los posibles jugadores piensen como yo.

**Dos tipos de menú:** Desde un principio tuve en mente el que el usuario pudiese elegir entre dos menús. Ya que aunque habrá un menú bastante colorido y agradable a la vista, este puede que en algunos casos no sirva en determinados móviles. Para que los usuarios de esos móviles no se vean limitados a la hora de ejecutar el juego, también se dispone de un menú adaptado a las características del móvil, cuya apariencia es similar a la del menú nativo del móvil en el que se ejecuta el juego.

**Cargar y guardar en cualquier momento:** El jugador podrá cargar su estado de una partida anterior en cualquier momento. Del mismo modo, también podrá guardar su partida sin necesidad de llegar a puntos de salvar. Esto es muy positivo ya que en los móviles las partidas o el tiempo de juego que se dedica es corto, por lo tanto no se le puede exigir al jugador que llegue a un determinado punto para poder guardar.

**Sencillez de uso:** Y es que, para que un juego no cause frustración en el jugador, esta tiene que tener un interfaz agradable y fácil de manejar. Para ello, y debido a que la ayuda del juego no se suele leer (aunque exista), durante el juego saltan tutoriales que te ayudan a controlar el interfaz.

**Totalmente internacionalizable:** Cualquier persona podrá traducir la aventura a cualquier idioma o dialecto, simplemente cambiando el contenido de los archivos XML. Sin necesidad de recompilar ni de tocar código.

**Totalmente configurable:** Con conocimientos de XML, manteniendo el esquema de los archivos XSD y leyendo para que sirve cada sentencia resultado (*explicadas en el punto 3.8.2.13*) podrá crear su propia aventura gráfica.



## 2.3. Referencias

Los siguientes juegos, películas y libros me han servido como inspiración para el genero del juego, los gráficos, el sonido, la ambientación... e incluso la historia.

***Saga Indiana Jones:*** Increíble saga del genero de la aventura en la que se nos relatan las peripecias de Indiana Jones.

***Saga Monkey Island:*** Quizás sea la saga o los juegos que más me han influido. Esta famosísima saga contiene un humor y una historia que la hace irresistible.

***Runaway:*** Juego bastante actual del que me ha gustado mucho su interfaz y las opciones que te pone a disposición. Además tiene una historia muy entretenida con algunos toques de humor, pero en menor medida que las anteriores.

***El club de la lucha:*** Película en la que el protagonista es esquizofrénico y vive una doble vida sin, en principio, darse cuenta.

***Jekyll & Hyde:*** Personaje que nos muestra dos lados muy diferenciados, en uno se muestra lo mejor del protagonista y en el otro lo peor de él.







## 3. Diseño del juego

### 3.1. Mecánica del juego

#### 3.1.1.Cámara

Se usará una perspectiva en 2 dimensiones( 2D), ya creo que un juego en 3 dimensiones en móviles no es lo mas adecuado, además, aunque en un juego de este genero los gráficos son un buen acompañante, no son lo principal.

La vista sería lateral, viendo siempre al personaje en (Tipo Monkey Island I).

Cuando el personaje tenga que viajar entre 2 zonas que, se supone, están alejadas, entonces aparecerá en pantalla un mapa en 2D visto desde arriba y en el que aparecerán todos los puntos a los que podemos ir. El boceto del mapa y de los distintos tipos de pantallas se puede ver en la sección 4.1.2.2 pantalla.

#### 3.1.2.Teclas

**Botones de dirección.** Se usará el propio joystick del móvil o las teclas: ‘2’ (arriba), ‘8’ (abajo), ‘4’ (izquierda), ‘6’ (derecha). Estos botones no moverían al personaje, sino que moverían un puntero en torno a la pantalla (como el ratón hace en las aventuras graficas del PC). Cuando estemos dentro del inventario, usaremos izquierda y derecha para movernos por los distintos objetos.

**Botón de acción ‘5’.** Serviría para numerosas acciones, como elegir que acción quieres realizar sobre un objeto o seleccionar un objeto del inventario o elegir que frase quieres decirle a un NPC...

**Botón de inventario ‘\*’.** Sirve para entrar o salir del inventario. En dicho inventario estarán todos los objetos de los que disponen nuestro personaje. Para movernos a través de los distintos objetos utilizaríamos los botones de dirección izquierda y derecha, y para seleccionar uno usaríamos la tecla de acción.

**Botón para entrar en el menú ‘#’ o ‘Acceso directo derecho’<sup>1</sup>.** Entraríamos en el menú. Nos moveríamos a través de sus opciones con los botones de dirección ‘2’ (arriba) o ‘8’ (abajo) y para seleccionar utilizaríamos el botón de acción.

---

<sup>1</sup> Acceso directo → Son los botones situados en las esquinas superiores del móvil, en este caso sería el botón situado en la esquina superior derecha del móvil .



### 3.1.3. Pantallas

En el juego podremos encontrarnos 5 tipos de pantallas básicas, en las cuales, la forma de interactuar con la maquina cambian, al igual que también cambia la información que esta nos envía.

#### 3.1.3.1. En un escenario

Como pantalla del juego entendemos aquella en la que se nos está mostrando un escenario, en el cual hay una serie de personajes y objetos con los cuales podemos interactuar. En ese momento en la pantalla nos encontraremos la siguiente información:

**Zona de juego:** (Ver imagen 5) Se muestra el escenario, en el que esta nuestro personaje, y todo lo que en él hay. También se mostrará el puntero (el cual manejamos con los botones de dirección), en el caso de que nuestro puntero este señalando algún objeto o personaje del juego sobre los que se puede realizar alguna acción, este cambiará de color y en la zona de texto de la pantalla, aparecerá el nombre del objeto o del personaje. Si no estamos señalando a nada importante, el puntero retomará su color original y en la zona de texto aparecerá “ Ir a...” queriendo representar que si pulsas el botón acción iras a este lugar.

**Zona de texto:** (Ver imagen 5) Una línea de texto donde aparecerá información sobre objetos o personajes que estén siendo señalados por el puntero. Si el puntero no señala nada en la zona de texto aparecerá “Ir a...”.

**Zona de inventario:** (Ver imagen 5) En caso de que pulses el botón ‘\*’, entras en el inventario. El inventario se muestra justo encima de la zona de texto, en la parte izquierda de la pantalla, mostrará solo un objeto de los disponibles y en caso de que no haya ninguno aparecerá el texto “NADA”. Por el inventario nos moveremos con las teclas de dirección izquierda y derecha y seleccionaremos el objeto deseado con el botón de acción (lo cual implica que estas teclas ya no manejaran el puntero de la zona de juego hasta que volvamos a pulsar ‘\*’). Una vez seleccionado, realizada la acción que queríamos hacer sobre los objetos que teníamos en el inventario o si simplemente queremos volver a manejar el puntero de la zona de juego, pulsaremos de nuevo la tecla ‘\*’.



Imagen 5: Zona de juego, texto e inventario

**Opciones de un objeto:** (Ver imagen 6) Cuando tenemos el puntero situado sobre un Ítem, o cuando estamos mirando un objeto del inventario, el nombre del ítem aparecerá en la zona de texto, pero si además, pulsamos el botón de acción, nos aparecerá una lista de opciones para dicho ítem. Estas opciones ocupan la esquina inferior derecha de la pantalla, justo encima de la zona de texto. Una vez ahí usaremos las teclas arriba y abajo para movernos entre las opciones de ese objeto y elegiremos una con el botón de acción. En caso de que no queramos realizar ninguna acción, usaremos la opción ‘Cancel’.

**Ventana de mensajes:** (Ver imagen 6) Ventana que servirá para que el protagonista de Entre 2 Almas, Oscar Romero, se comunique con nosotros. Así cuando este nos quiera decir algo, esta ventana se abrirá y en ella aparecerá el texto que se nos quiera decir. Para deslizarnos a través del texto, usaremos las teclas de dirección arriba y abajo, y para cerrar la venta de mensajes usaremos el botón de acción.

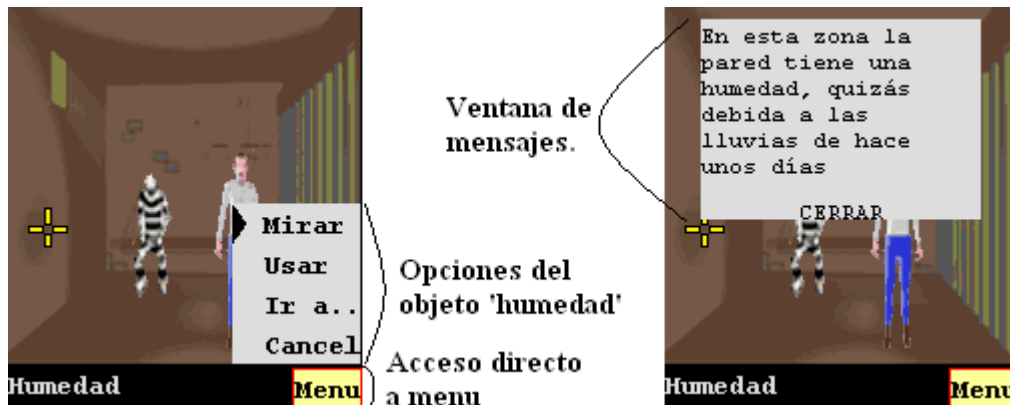


Imagen 6: Opciones y ventana de mensajes

### 3.1.3.2. En una conversación

Durante el juego, tendremos la oportunidad de hablar con algunos personajes, en ese momento la pantalla se dividiría en 3 partes, la zona de imagen (que a su vez se dividiría en 2 partes y en ella irían las fotos de los que hablan,) y la de texto (Ver imagen 7).



En la parte izquierda-arriba de la pantalla aparecería el busto de nuestro personaje y un color de fondo, en la parte derecha-arriba de la pantalla aparecería el busto del NPC con el que hablamos y otro color de fondo. Cuando un personaje habla se podrá ver las distintas expresiones de su rostro, y los comentarios de cada personaje irían acompañados de su correspondiente color de fondo.

En la parte de abajo se iría mostrando lo que el NPC nos dice y cuando nos toque hablar a nosotros, desaparecerá lo que él nos dijo y en su lugar se nos mostrara las posibles respuestas, con los botones de dirección izquierda o derecha nos movemos entre ellas y con el botón de acción seleccionamos la respuesta que queremos decir, entonces aparecerá esa respuesta sola indicando que se la esta diciendo.

Para salir de la conversación habrá que selecciona una respuesta que tenga connotación de despedida o que indique que la conversación ha terminado, Ej. : “ Hasta luego”.



Imagen 7: Conversación

### 3.1.3.3. En el mapa

Cuando vayamos a desplazarnos de una zona a otra, entre las cuales se supone hay mucha distancia, aparecerá en pantalla y a tamaño completo, un mapa con todos los puntos a los que podemos ir y el nombre de esos puntos. Para movernos entre los distintos puntos moveremos el puntero, y al situarnos sobre una localización a la que podemos desplazarnos, el puntero cambia de color, si pulsamos entonces el botón de acción, nos aparecerán una serie de opciones, entre las que estará Ir a... que sirve para ir a ese escenario.



Imagen 8: Mapa

### 3.1.3.4. En un video

En determinados momentos de la partida, saltarán videos para representar una situación. Pero estos no serán videos realmente, sino que serán una sucesión de viñetas con un texto explicatorio. Para deslizarlos, a través del texto explicatorio, utilizaremos las teclas arriba y abajo, y para continuar con la siguiente viñeta, pulsaremos el botón de acción.



Imagen 9: Video

### 3.1.3.5. En el menú

Si pulsamos el botón de acceso directo o ‘#’ accederemos al menú, este tendrá 6 opciones básicas (Seguir jugando, Nueva Partida, Memoria, Opciones, Ayuda y Acerca de...). Para movernos entre las distintas opciones usaríamos los botones de dirección ‘2’ (arriba) u ‘8’ (abajo) y elegiríamos opción con el botón ‘5’.

### 3.1.4.Sonido

El juego no usará sonido, ya que aunque la utilización de sonidos aporta ambientación al juego y hace que el jugador se meta más en el desarrollo, creo que no es algo fundamental en un juego para móvil. Sin embargo el tamaño que tenga el juego si



es algo fundamental, así que en pos de reducir el tamaño del juego de forma considerable, se decidió suprimir cualquier tipo de sonido en el juego.

### 3.1.5. Acciones disponibles

Siempre que tengamos el puntero sobre un objeto o personaje sobre el que se puede realizar alguna acción, el puntero cambiará de color indicándonos que si pulsamos el botón de acción se nos mostrará un listado de las acciones que sobre ese objeto o personaje se pueden hacer.

El numero y tipo de opciones son infinitas (se especifican por XML), así puede ir desde Usar (muy típica) a afeitarse (no típica). Aun así, las posibles acciones más comunes, son las siguientes:

- *Mirar* → Sirve para observar las características o datos a reseñar sobre un objeto o personaje. El objeto puede formar parte del escenario en el que este nuestro protagonista o estar ya recogido en el inventario:

Ej. : Hay una ducha, la señalas con el puntero pulsas acción y aparecen las opciones “mirar” y “usar”, si eliges “mirar” el personaje podría decir “El grifo está roto, debería arreglarlo” (eso dependiendo de cómo lo implementemos en el juego).

- *Usar* → Esta acción indica que quieres usar un determinado objeto Ej.: Hay una ducha, la señalas con el puntero pulsas acción y aparecen las opciones “mirar” y “usar”, si eliges “usar” el personaje podría ducharse o quizás te diga “Ahora no necesito ducharme” (eso dependiendo de cómo lo implementemos en el juego).

Otra posibilidad es que estemos en la zona de inventario, entonces al ponernos encima de un objeto y dar al botón de acción a veces aparecerá entre las opciones disponibles la de “Usar”. En caso de que lo quieras usar con otro objeto del inventario, tendrás que pulsar sobre el primero de los objetos con lo que en la parte de debajo de la pantalla de inventario aparecerá “Usar objetoInventario con...” y luego volver a pulsar con el botón de acción sobre el 2º objeto y si tiene sentido, ocurrirán unas determinadas acciones. Si no tuviera sentido, por pantalla aparecería algo como “Eso no sirve para nada”.

En el caso de que quieras utilizar algo del inventario con algo situado en la pantalla del juego tendrás que: Entrar en el inventario (botón ‘\*’) ir hasta el objeto que quieres usar dar al botón de acción y entre las opciones disponibles elegir “usar”, una vez echo esto vuelves a pulsar ‘\*’ para salir del inventario, entonces en la parte de debajo de la pantalla de juego aparecerá “Usar objetoInventario con...” y al pulsar con el botón de acción sobre un objeto o personaje de la pantalla de juego te dirá si eso tiene sentido o no y si lo tiene realizara una acción.



- *Coger* → Acción que sirve para coger objetos. Ej.: Hay un lápiz en el suelo la señalas con el puntero pulsas acción y aparecen las opciones “mirar” y “coger”, si eliges coger quizás lo coja (entonces el lápiz aparecerá en el inventario) o quizás te diga “No necesito eso para nada”.
- *Hablar* → Esta acción solo podrá ser llevada a cabo sobre personajes (NPC's) y su resultado será tener una conversación con dicho personaje (apareciendo para ello la pantalla de conversación explicada en el punto 2.3).
- *Abrir* → Sirve para abrir objetos, puertas u otro tipo de aberturas. Quizás halla veces que para abrir un determinado objeto o puerta sea necesario un objeto del inventario, en dicho caso tendríamos que utilizar la acción usar con ese objeto y luego señalar el elemento a abrir (usar una llave para abrir una puerta).
- *Ir a...* → Esta es la acción predeterminada, siempre que el puntero este señalando a una zona no activa (es decir sobre la que no se puede hacer nada) en la parte de debajo de la pantalla de juego aparecerá “Ir a...”, queriendo indicar que si pulsas el botón de acción, no te saldrá un menú para elegir opción, sino que el personaje se desplazará hasta el lugar más cercano que pueda de la zona a donde señala el puntero.

### 3.1.6.Menú

Con la idea de que el jugador pueda elegir el tipo de menú que más se adapte a su gusto, se dispone de 2 menús. El cambio entre uno u otro menú se realiza fácilmente a través de la pantalla de opciones, y una vez que el usuario selecciona uno, su elección se queda en la memoria del teléfono para que así, la próxima vez que vuelva a entrar al juego, no tenga que volver a tomar esta decisión. Si quisiese cambiar el tipo de menú, simplemente tendría que pasar de nuevo por las opciones. Los tipos de menús son:

**Clásico.** Este tipo de menú mantendrá el formato y colores de los menús del propio móvil, es decir, será un poco distinto para cada tipo de móvil. El realizar interfaces parecidos a los nativos del propio móvil es una práctica recomendable.

**Avanzado.** Menú adaptado para el juego, con imágenes creadas para este. Este tipo de menú será igual para todos los móviles.

Las opciones de menú para los 2 tipos, serán las misma, lo único que cambiará será la forma y el diseño a la hora de mostrarlas. Las opciones serán:

- *Nueva partida* → Comienzas una partida nueva.
- *Seguir jugando* → Sigues jugando la partida en el mismo punto en el que lo dejaste cuando entraste al menú.
- *Memoria* → En él podrás guardar, cargar y borrar tus partidas.



- o *Grabar* → Grabas tu estado actual y tu situación en el mapa, para así poder reanudar mas tarde en el mismo sitio en el que lo dejaste. Una vez guardada, sigues jugando la partida en el mismo punto en el que lo dejaste cuando entraste al menú.
- o *Cargar* → Cargas un estado y tu situación en el mapa de un tiempo pasado, para así poder reanudar en el mismo sitio en el que lo dejaste. Una vez cargada la partida, aparecerás en la pantalla de juego (no en el menú).
- *Opciones* → Podrás seleccionar el tipo de menú que quieres (clásico o Avanzado).
- *Ayuda* → La ayuda del juego.
- *Acerca de...* → Información sobre los creadores y participantes en el proyecto.
- *Salir* → Sales del juego.

## 3.2. Elementos del juego

### 3.2.1. Personajes

#### 3.2.1.1. Personajes principales

Como personajes principales entendemos aquellos personajes cuya importancia en la historia es constante y que aparecerán en varias fases del juego.

##### **Oscar Romero:**

- **Edad:** 32 años
- **Color Pelo:** Moreno.
- **Peinado:** Corto, debido a que ya le empiezan a aparecer entradas y el pelo largo le queda ridículo.
- **Altura:** 1,74
- **Peso:** 82 Kg. → No esta gordo, pero esta apunto, no es demasiado ágil, pero es fuerte.
- **Trabajo:** Albañil, empezó la carrera de arquitectura, pero “no era lo suyo” y prefirió seguir los pasos de su padre
- **Descripción:** Oscar es un tipo seguro de si mismo, siempre dice que no hay nada que el no pueda hacer si le dan el suficiente tiempo, aunque esto muchas veces no lo lleva a la platica.
- **Boceto:** En el boceto se puede ver la cara de Oscar Romero cuando es dominado por el alma buena y cuando es dominado por el alma mala (que es cuando se hace llamar Romo)





Imagen 10: Oscar y su versión malvada Romo

**Inspector Ringo:**

- **Edad:** 30 años
- **Color Pelo:** Rubio
- **Peinado:** Hacia atrás como si le hubiese lamido una vaca
- **Altura:** 1,88
- **Peso:** 90 Kg. → Todo fibra y músculo
- **Trabajo:** Inspector de policía.
- **Descripción:** Es un portento físico fue el primero de su promoción en las pruebas físicas, pero no así en las de inteligencia “las cuales paso justito”, aunque otros dicen que eso fue gracias al dinero e influencia de su padre, gracias al cual dicen que también fue capaz de sacar la oposición de inspector. Le llaman Ringo por que de pequeño en vez de Rinoceronte decía “Ringoceronte”.
- **Relación con Oscar:** Es junto a Dute el inspector que lleva el caso de Oscar, creen que es culpable y ya se lo han tomado como un tema personal
- **Boceto:**





**Imagen 11: Imagen del inspector Ringo**

**Inspector Dute:**

- **Edad:** 45 años
- **Color Pelo:** Castaño
- **Peinado:** Calvo, solo pelo por los lados y bigote
- **Altura:** 1,68
- **Peso:** 85 Kg. → Rechoncho
- **Trabajo:** Inspector de policía - veterano
- **Descripción:** Es ya un veterano en el cuerpo que se ha quedado estancado en este puesto y que es conocido por sus salidas de tono, no quiere disciplina para el pero si para los demás. Es inteligente pero esto le lleva a creerse superior a los demás. A Ringo lo tiene dominado totalmente, así que podríamos decir que Ringo es el músculo y Dute es el cerebro. Dute viene de Canduterio, pero bajo ningún contexto quiere que le llamen así.
- **Relación con Oscar:** Es junto a Ringo el inspector que lleva el caso de Oscar, creen que es culpable y ya se lo han tomado como un tema personal.
- **Boceto:**



**Imagen 12: Boceto del inspector Dute**

**Detective privado Alfredo Salaberri**

- **Edad:** 35
- **Color Pelo:** Moreno
- **Peinado:** Corto con raya a un lado
- **Altura:** 1,78
- **Peso:** 67 Kg
- **Trabajo:** Detective privado.
- **Descripción:** Es un personaje serio, pero franco, a él no le importa lo que piensen los demás, a él le importa lo que digan las pruebas y no tiene inconveniente en trabajar para cualquier persona... cualquier persona que a él le caiga bien. El siempre quiso ser policía pero, el tener pies planos se lo impidió, a raíz de ahí trabajo como guarda de seguridad hasta tener 28,



cuando decidió ir por su cuenta. Pero el trabajo no esta resultando como él esperaba y no hace mas que investigar casos de adulterios, quizás le haga falta un caso más emocionante...

- **Relación con Oscar:** Oscar se entera de su existencia gracias a una tarjeta que le proporciona un compañero de celda, a este compañero suyo Alfredo Salaberri le había ayudado a librarse de un caso de asesinato, aunque luego resulto culpable de otro. Oscar decide ir a él porque sabe por lo que le contó su compañero de celda que es un tipo que aceptara llevar el caso.. siempre que le caiga bien.
- **Boceto:** Alfredo de varias formas.

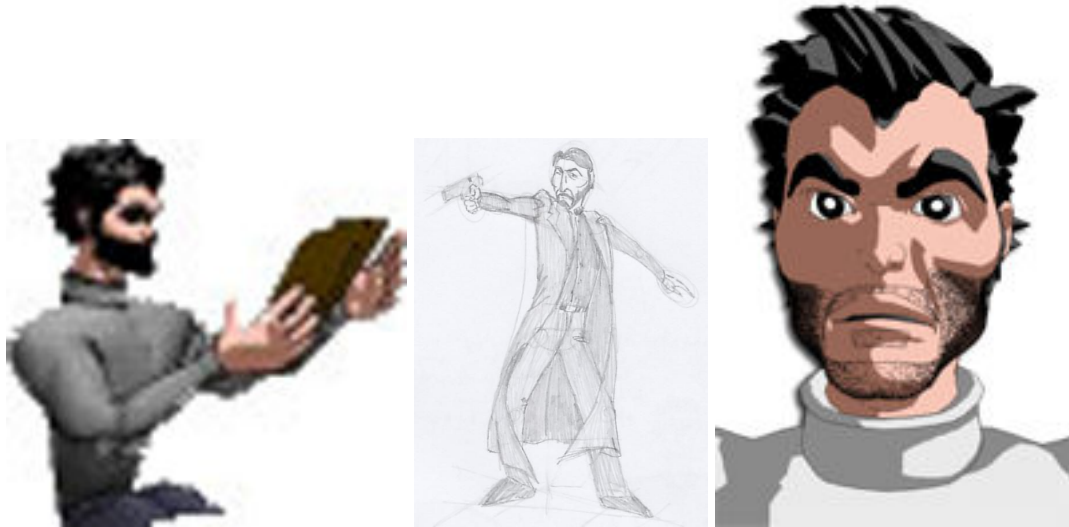


Imagen 13: Bocetos del detective privado Alfredo Salaberri

### 3.2.1.2. Personajes secundarios

Son personajes que aparecen en situaciones muy concretas del juego y para una función determinada, o simplemente para dar mas ambientación en la escena.

Estos personajes pueden ser tanto humanos como animales siempre que cumplan lo dicho anteriormente. A continuación se muestran algunos bocetos de este tipo de personajes.



Imagen 14: Dependiente de una tienda de alimentación china



**Imagen 15: Preso**



**Imagen 16: Gato del detective privado Alfredo Salaberri**

### **3.2.2.Mapas**

La función de los mapas es la de permitir al jugador moverse entre estancias que se suponen están lejos en el espacio. Así se consigue dar la sensación de lejanía, y dejamos claro el cambio de estancia.

Pese a que el juego está programado y preparado para disponer de múltiples mapas, en esta historia sólo era necesario un mapa, el de la ciudad de Albora.

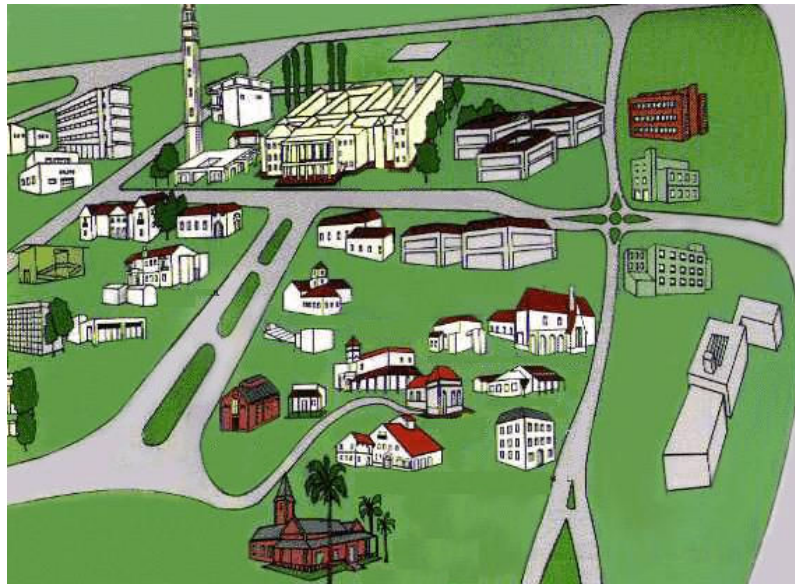


Imagen 17: Mapa de Entre 2 Almas

### 3.2.3.Escenarios

El mapa nos permitirá movernos entre estancias, las cuales a su vez estarán formadas por varios escenarios. En esto es donde realmente se desarrolla el juego, ya que es donde podremos encontrar los objetos y personajes con los que interactuar.

Cada escenario tendrá un area caminable por donde nuestro personaje podrá moverse, y fuera de cuyos limites no podrá salirse. A continuación se muestran algunos ejemplos, en los cuales todavía faltaban por insertar algunos objetos y personajes.

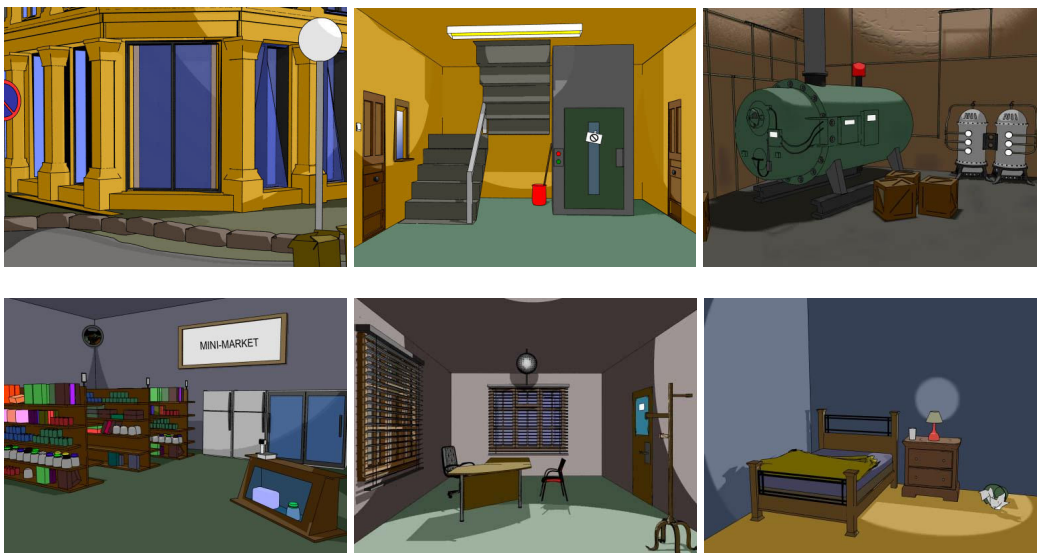




Imagen 18: Escenarios

### 3.3. Historia

A continuación se expone toda la historia del juego, en ella se explica como pasarse el juego de principio a fin.

De los siguientes puntos hay que aclarar que todo lo que esta escrito en cursiva, se supone que no es visto en ese mismo momento por el usuario, sino que se pone como aclaración de lo que pasa a espaldas de nuestro protagonista.

#### 3.3.1.Video inicial

Oscar Romero, como ponía en el mono de trabajo que llevaba puesto, estaba preparándose para ir a su trabajo. Oscar trabajaba de albañil para una contrata llamada CDM (construcción, destrucción y mantenimiento). Esta vez había tenido suerte y le había tocado una obra cerca de su casa, situada a las afueras de la ciudad de Albora, así que ese día decidió ir andando hasta el trabajo, “así estiro las piernas” pensó Oscar.

- Cuando estaba cogiendo las botas sonó el teléfono:
- ¿Si?
- Buenos días, mi nombre es Ana,- dijo una telefonista - le llamo de Seguros de vida Derticia, ¿es usted Oscar Romero?
- No perdone, el señorito no esta – contesto Oscar a sabiendas que ella quería venderle un seguro de vida – yo soy la asistente.
- Si, ya había notado algo en la voz. – comentario que no gusta a Oscar- ¿Cuando podría hablar con él? – replicó la telefonista.
- Oscar es mudo, lo siento. Adiós – y Oscar colgó antes de que ella pudiese decir más.

Él siempre decía que los seguros de vida solo eran para los que tenían miedo a la muerte, y que era una perdida de dinero. Él confiaba en que le quedaba mucha vida, porque sobre todo confiaba en si mismo.





Por fin, sale de casa y se dirige a su trabajo en lo que parece que será un día mas en su rutinaria vida, pero entonces ocurre lo que nadie puede esperarse... Oscar es atropellado por un coche que al no respetar el paso de peatones, lo golpea y lo lanza a mas de 20 metros de distancia. Fue un golpe durísimo y todo aquel que lo vio dio, al que será nuestro protagonista, por muerto.

Así lo parece en los primeros minutos, no responde a los estímulos y no tiene pulso, pero 1 o 2 minutos antes de que llegase la ambulancia, Oscar abrió los ojos y como si nada hubiera pasado se levantó y comprobó que no tenia ninguna herida grave, la gente se quedó sin habla, Romero había sobrevivido aun golpe brutal y ni si quiera tenia un rasguño. Aun así deciden trasladarlo al hospital para dejarlo 24 horas en observación y ver su evolución.

Una vez echo el traslado, se procede a darle la ropa del hospital a nuestro protagonista y se le emplaza en una habitación en la que pasar las 24 horas.

Pero al anochecer la cosa se complica, Oscar ha desaparecido y lo mismo ha ocurrido con una ambulancia.

A la mañana siguiente nuestro protagonista se despierta en el calabozo de la comisaría, magullado, aturdido, con la ropa del hospital y no recuerda nada de lo que ocurrió esa noche. Que raro...

### **3.3.2.Fase 1: ¿Qué hago aquí?**

Oscar necesita saber que ha hecho para estar allí, pero no hay ningún policía, así que decide hablar con un compañero de celda, un delincuente... como el mismo. Este le dirá que Oscar se fue por la noche se fue del hospital robando una ambulancia, para mas tarde atracar una tienda y escapar por las calles de la ciudad en esa misma ambulancia mientras la policía le perseguía quienes finalmente le reducirían. Lo más seguro es que sean varios años de cárcel.

No se lo puede creer, alguien le ha tenido que tender una trampa, el no recuerda nada de eso y ¿por qué se iba a escapar del hospital? ¡¡1 a 5 años en la cárcel y por algo que el no ha hecho!! ¡¡NO!! Decide que no piensa cumplir una condena que a él no le corresponde, pero ¿qué puede hacer?.

Su compañero, un tipo corpulento de poca inteligencia, también le comenta que él conoce a alguien que quizás le crea y que quizás le pueda ayudar. El delincuente le entrega una tarjeta en la cual dice “Detective privado Alfredo Salaberri” y su dirección, el delincuente le insiste en que ese detective investigara su caso... pero según la policía ya esta todo investigado y va a ser juzgado... Oscar decide escaparse e ir a hablar con el detective.

Oscar intenta hacer un agujero con sus manos pero al poco tiempo desiste, luego intenta convencer a su compañero de que doble los barrotes de la ventana, pero este no



llega a ellos, y cuando ya se estaba empezando a desesperar Oscar se percata de que una de las paredes del calabozo comunica con el exterior y en esa misma pared hay una humedad bastante grande que quizás haya reblandecido ese trozo de pared. Se le ocurre hacer un agujero en la zona donde esta esa humedad, ya que será más fácil hacerlo ahí, pero no tiene herramientas para hacerlo, así que Oscar utiliza a su compañero.

El corpulento compañero se enfada con facilidad y siempre que le dices una combinación de frases se lanza a por ti embistiendo hacia la posición en la que estas. Oscar se pone justo delante de la humedad, enfada al compañero de celda y este con toda su potencia atraviesa la pared y queda KO. Romero aprovecha este momento para escaparse y refugiarse en un descampado.

Aun nervioso y con la ropa del hospital, Oscar decide dormir un poco en aquel descampado, detrás de unos matorrales tras los cuales no le descubrirán, para así relajarse un poco y seguir con su tarea al día siguiente.

Romo (que así se hace llamar Oscar cuando es dominado por el alma mala)se levanta y decide salir de “marcha”. No puede ir con esa ropa así que roba la ropa a 3 jóvenes que se estaban riendo de él por llevar esa pinta y además se lleva su coche. Ya con esto se vio lo suficientemente decente como para ir a algún bar.

Es entonces cuando pasó por delante de “La luna llena” en dicho bar trabajaba una bonita chica, que aunque no era exuberante, tenía atractivo. Romo se decidió a entrar y descubre que esa camarera se llama Susana o “Susi” para los amigos. Romo pasa la noche con Susi para luego dejarla y volver al descampado, en la vuelta estrella el coche y en el accidente la ropa queda muy malograda, él apenas un rasguño.

### **3.3.3.Video 1**

Aparecen el inspector Ringo y Dute recibiendo la orden de encontrar a Oscar Romero como sea

### **3.3.4.Fase 2: Vaya pintas!!**

Oscar se despierta algo aturdido y cansado. De repente le entra miedo, quizás alguien le vio y le inyectó algo para aturdirlo.

No lo duda un segundo y decide marcharse de allí e ir directamente a hablar con el detective, pero no puede ir con esas pintas, llama mucho la atención. Tiene que pasar por su casa a coger algo de ropa (al salir del descampado aparece el mini-mapa, en el cual estarán activas las localizaciones de su casa, el despacho del detective y el descampado, solo podrá ir a su casa, ya que al despacho del detective no quiere ir con esas pintas y en el descampado tiene miedo a quedarse).

Tras conseguir llegar hasta el portal de su casa, piensa que quizás haya alguien en su casa y tras mirar un poco mejor se da cuenta de que los inspectores Ringo y Dute





están intentando observar indicios de que haya pasado por allí, por suerte se encuentran fuera, en el rellano y la casa por dentro esta vacía, pero ¿cómo puede conseguir entrar?. Tiene que despistarlos.

En el portal coge una escoba que seguramente pertenece al portero.

Se le ocurre usar la caldera general del edificio, la cual esta situada en el sótano, para entrar en ella se necesitan las llaves del portero el cual por suerte esta dormido, lo cual es aprovechado por Oscar para cogerle una gorra que tenia sobre la mesa, pero aún así tiene las llaves enganchadas por un candado de contraseña a su cinturón, Oscar tiene que acertar ese numero de 4 cifras, pero es prácticamente imposible... hasta que se le ocurre mirar la gorra y por la parte de abajo traía un numero escrito el cual efectivamente era la contraseña del candado.

Ya con la llave puede entrar en la caldera, en ella cogerá unas tenazas de gran tamaño y, usándolas, pondrá la temperatura de la caldera al máximo, esto hace que Ringo y Dute empiecen a sentir muchísimo calor y a Dute se le ocurre que un refresco o algo de agua no le vendría mal y manda a Ringo a por ello. Pero Ringo no sabe donde hay algún supermercado o tienda para comprarla, así que Dute le sugiere que hable con el portero (se queda en el portal esperando al portero). Oscar oye esto y se pone la gorra del portero, lo cual hace que Ringo lo confunda con el portero y le pregunta por algún sitio donde comprar agua, Romero aprovecha la ocasión para mandarle a una zona inventada, pero eso si lejísimos; antes de irse, Ringo se engancha con la puerta del portal y se le cae un trozo grande de hilo que Oscar recoge.

Ahora el problema pasa a ser Dute, que sin duda no caería en la trampa del portero. A Oscar se le ocurre ponerle una trampa más sutil, un bocado seguro que no lo rechaza Dute, este bocado esta en una mesa al otro lado del portero, pero para cogerlo tendría que molestar al portero y por tanto despertarlo. Lo que hace es unir la escoba al hilo con lo cual ya llega hasta el bocado, pero aun así, cuando llega a él no puede cogerlo, por que sin un anzuelo es imposible “pescar”, así que usando las tenazas con el mecanismo anterior y utilizándolo adecuadamente consigue hacerse con el bocado.

Va de nuevo hasta donde esta Dute, el cual se esta moviendo y hay veces que se queda sin mirar en la posición de Oscar, Oscar no puede darle el bocado a él, ya que lo vería, así que decide dejarlo en un lugar cercano a la vista de Dute en un momento en el que Dute no este mirando. Cuando vuelve a mirar y ve el bocado, lo coge, se percata de que puede ser una posible prueba, pero decide que la comida es lo primero y sale del edificio a comerse el bocado. Es este el momento en el que Oscar ya puede entrar en su casa.



### 3.3.5.Video 2

En este video se muestra a Oscar cogiendo ropa nueva para ponerse y luego ya sale vestido con ella. Justo cuando iba a salir de casa recibe una llamada de una tal Susi que le empieza a regañar e insultar por haberse ido y haberla dejado sola... Oscar piensa que es una loca y directamente la cuelga.

### 3.3.6.Fase 3: Maldito gato

Vuelve a salir del mismo modo al que entro y va a ver al detective. Al llegar al edificio donde esta el detective, al lado del portal hay una tienda de alimentación china. No lo duda y sube, en la puerta dice “Detective privado Alfredo Salaberri”. Entra y cuando se dirigía a hablar con el detective al gato se le eriza la piel y se prepara para atacar a Oscar, entonces este retrocede y oye las palabras del detective “si no le caes bien a mi gato, a mi menos”. Parece que tiene que caerle bien al gato.

Baja a la tienda de alimentación, el dependiente es de nacionalidad china, y carácter risueño y charlatán. A Oscar se le ha ocurrido comprar un pescado para sobornar al gato, pero no tiene dinero, así que se hace pasar por un funcionario, el dependiente dice algo que no debe y como compensación le regala el pescado (el pescado que le da tiene bastante mala pinta).

Oscar sube al despacho del detective y le da el pescado al gato pero el gato ni si quiera se acerca al plato donde depositaste el pescado ¿por qué?; se le ocurre que quizás el gato solo se lo come si se lo das bien preparado en pequeños trozos.

En la basura coge un palo que le podría ser útil más adelante.

Oscar baja de nuevo a la calle y vuelve al descampado, allí usa el palo para tirarlo contra una farola que había por esa zona, al hacer esto, rompe el cristal donde esta la bombilla y caen pedazos de cristal, Romero coge uno que parece estar bastante afilado y recoge de nuevo el palo, vuelve a ir al despacho del detective y con las mismas sube y corta el pescado en cachos, pero nada el gato no se acerca. Quizás sea el pescado.

Baja y vuelve a hablar con el dependiente el cual entre risas te dice que al gato del detective solo le gusta el sushi, pescado que si no lo cortas bien puede producir la muerte, pero que es muy caro. Nuestro protagonista no tiene dinero así que sigue hablando con el dependiente y este le cuenta que va a batir un record guiness, el de permanencia en cámara frigorífica para bebidas. Romero le pide que le haga una demostración y cuando se mete en la cámara (muy poquito tiempo) rápidamente utiliza el palo para dejar bloqueada la puerta y que no pueda salir. Entonces coge el sushi sube al despacho y lo pone en el plato de comida del gato, además lo corta (pero de cualquier modo mal cortao) el gato no duda en comérselo e instantes después se muere. El detective Alfredo piensa que se ha dormido lo cual es una buena señal de que Oscar le cae bien, entonces acepta hablar con él.



Después de contarle todo, el detective decide investigar el caso y empezar en ese mismo momento, además le dice a Oscar que puede pasar allí la noche.

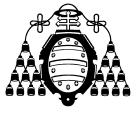
Romo se levanta y empieza a rebuscar en el despacho, allí encuentra una pistola y sale sin pensárselo a robar a alguien. No hay nadie por la calle así que se cuela en un portal y elige al azar una puerta en la que llamar, el propietario abre con la puerta con el seguro de cadena echado, pero Romo utiliza ese mínimo hueco para poner el pie en la puerta (para que no se pueda cerrar), apunta con la pistola al propietario y le manda abrir. Cuando estaba llevándose todo lo de valor, los propietarios de la casa aprovechan para llamar a la policía, al poco tiempo aparecen Ringo y Dute. Ringo de un solo golpea deja KO a Romo y Dute lo esposó y se lo llevan a comisaría.

### **3.3.7.Video Final**

Oscar Romero se acuesta y se duerme rápido, la pantalla se pone en negro. Una voz le despierta, esta de nuevo en el calabozo!!!! Es el carcelero que le trae la comida, no hay forma de salir... le van a juzgar.

Una semana después Oscar es condenado a 5 años de cárcel. Romero mirando al suelo y los grilletes puestos, está preparado para el traslado a la prisión de la localidad.

CONTINUARA ...





## **4. Análisis**

### **4.1. Especificación de requerimientos**

#### **4.1.1.Requisitos generales**

- La aplicación estará enfocada a 2 grupos de usuarios, jugadores y desarrolladores.
- El genero elegido es el de las aventuras gráficas, con una historia de 3 fases y multitud de puzzles que superar, videos que ver, personajes con los que hablar y objetos con los que interactuar.
- El juego estará preparado para ser ejecutado en teléfonos móviles con tecnología Java MIDP 2.0.
- No se limitará a una marca de teléfonos móviles en concreto. Y se intentará hacer lo más portable, entre teléfonos móviles, posible.
- Deberá ser totalmente personalizable, configurable e internacionalizable por XML.
- La aplicación estará optimizada para pantallas de 128x128 y de 176x200 pixeles.

#### **4.1.2.Requisitos del menú**

- La aplicación contará con dos tipos de menús, uno clásico, adaptado a las características nativas del propio móvil, y otro avanzado, con una presentación y un diseño especialmente preparados para este juego.
- El menú principal contará con las opciones “Seguir Jugando”, “Nueva Partida”, “Memoria”, “Opciones”, “Ayuda”, “Acerca de”.
- En el menú de memoria podremos cargar y guardar partidas.
- En el menú de opciones podremos cambiar entre los 2 tipos de menús.
- En el menú de ayuda podremos informarnos sobre los distintos detalles a la hora de jugar.
- En el menú de acerca de... obtendremos información sobre los creadores del juego.
- Toda la información que aparece en el menú, ya sean las imágenes, contenido, nombre de las opciones, etc, serán modificables por XML.



### **4.1.3.Requisitos de la interfaz**

- Ya que los jugadores no suelen leer la ayuda, se procurará que se habitúen al interfaz por medio de pequeños tutoriales que saltan en determinados momentos del juego, por ejemplo la primera vez que entras al inventario.
- El jugador dispondrá en todo momento, de un acceso directo al menú principal.
- En las conversaciones, videos y pantalla de mensaje, se le indicará que función tiene en ese momento tocar el botón de acción.
- Si el jugador sitúa el puntero sobre un ítem o personaje, al pulsar el botón de acción se mostrarán las acciones disponibles para ese ítem.
- Se dispondrá de un inventario, donde estarán aquellos objetos que el jugador haya cogido.
- Habrá una zona de texto que nos indique en cada momento lo que estamos seleccionando.
- Tanto el inventario como las opciones de un objeto, desaparecerán de la pantalla cuando nosotros lo decidamos, para así no ocupar espacio de pantalla.

### **4.1.4.Requisitos de la historia**

- Se dispondrá de una historia bien elaborada, en tono de humor, y que consiga entretener.
- Se establecerán multitud de puzzles y situaciones a superar, teniendo en cuenta que cada una tiene que ser un poco distinta a la anterior, para así no hacer el juego repetitivo.
- Ante determinadas acciones complicada, o puzzles difíciles de resolver, se establecerán premios, de tal manera que se vea recompensado el esfuerzo del jugador. Cuando hablo de premio me refiero a un video, o un mensaje gracioso, o algo que rompa la monotonía del juego.
- Se buscará que los puzzles sean variados, así encontraremos puzzles de intercambio, de combinación, de distraer y arrebatar, de huida, de disfraces, de secuencias lógicas o de conversación.

### **4.1.5.Requisitos de jugador**

- El jugador podrá elegir entre dos tipos de menú.
- Una vez que inicie una partida tiene la posibilidad de reiniciarla, o grabar la partida.



- Siempre será informado de los periodos de carga y de guardado.
- El jugador deberá resolver por medio de la lógica, una serie de interrogantes que le permitan avanzar en la historia.
- En todo momento podrá volver al menú principal pulsando el acceso directo.

#### 4.1.6.Requisitos de desarrollador

- Cualquier persona con conocimientos en XML, y basándose en los esquemas XSD que acompañan al código fuente, podrá crear su propia aventura gráfica sin necesidad de tocar ni una sola línea de código, o de recompilar.
- Así mismo un usuario con conocimiento en XML, podrá traducir el juego a cualquier idioma sin necesidad de recompilar ni de tocar código.
- El desarrollador que quiera crear su propia aventura gráfica, deberá preparar todas las imágenes del juego, crearse una historia, abrir el ejecutable (.jar) y modificar los archivos básicos.xml y los FaseX.xml, usando para ello la información contenida en esta documentación y siguiendo los esquemas XSD. Una vez modificados deberá introducirlos de nuevo en el .jar, sin necesidad de recompilar, ya estará preparado para ejecutar su propio juego en el móvil.

### 4.2. Identificación de actores

**Jugador.** Será el usuario final del juego. Se entiende por jugador no solo la persona física que juega utilizando el móvil, si no también su propia representación en el juego.

### 4.3. Identificación de los casos de uso

En el siguiente diagrama se muestran los distintos casos de uso para el jugador. Uno de los casos de uso es “Usar ítem”, y aunque realmente las opciones de interactuar con un ítem son ilimitadas, ya que se definen por XML, pongo este como ejemplo, junto con “mirar ítem” y “coger ítem”, ya que son las acciones típicas que podremos realizar sobre la mayoría de los ítems.

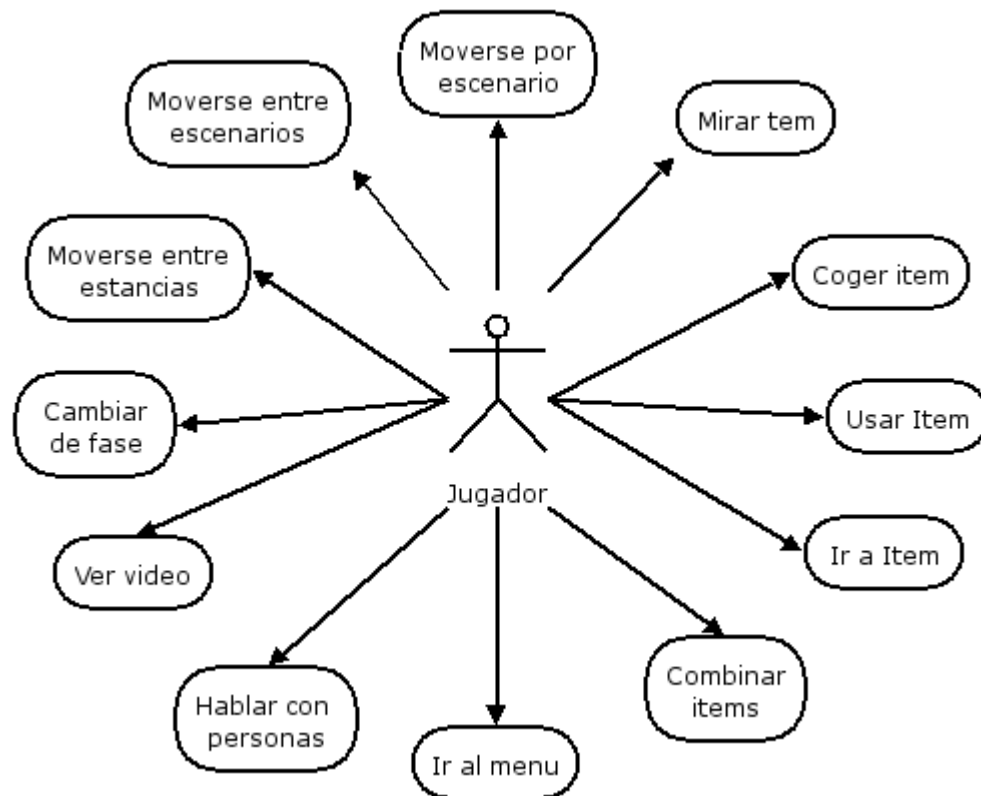


Imagen 19: Diagrama de casos de uso

### 4.3.1.Mirar ítem

<b>Nombre:</b>	Mirar ítem
<b>Actores:</b> Jugador	
<b>Descripción:</b> Permite al jugador obtener una descripción sobre un ítem.	
<b>Precondiciones:</b> El ítem (ya sea un objeto o un personaje) deberá poder ser seleccionado con el puntero.	
<b>Flujo:</b> 1.- El usuario sitúa el puntero sobre el ítem. 2.- En la zona de texto nos aparecerá el nombre de dicho ítem. 3.- El jugador pulsa el botón de acción. 4.- Se le muestran las distintas acciones disponibles para ese ítem. 5.- El jugador elige la opción mirar. 6.- Se muestra por pantalla un mensaje con una cierta información derivada de la opción de mirar.	
<b>Poscondiciones:</b> El jugador deberá haber obtenido una información sobre el ítem a mirar.	





### 4.3.2.Coger ítem

<b>Nombre:</b>	Coger ítem
<b>Actores:</b> Jugador	
<b>Descripción:</b> Permite al jugador coger un ítem.	
<b>Precondiciones:</b> El ítem (ya sea un objeto o un personaje) deberá poder ser seleccionado con el puntero, y deberá de tener entre sus opciones la de coger.	
<b>Flujo:</b> 1.- El usuario sitúa el puntero sobre el ítem. 2.- En la zona de texto nos aparecerá el nombre de dicho ítem. 3.- El jugador pulsa el botón de acción. 4.- Se le muestran las distintas acciones disponibles para ese ítem. 5.- El jugador elige la opción coger.	
<b>Poscondiciones:</b> En caso de que tenga éxito, el ítem pasará al inventario.	

### 4.3.3.Usar ítem

<b>Nombre:</b>	Usar ítem
<b>Actores:</b> Jugador	
<b>Descripción:</b> Permite al jugador coger un ítem.	
<b>Precondiciones:</b> El ítem (ya sea un objeto o un personaje) deberá poder ser seleccionado con el puntero, y deberá de tener entre sus opciones la de usar.	
<b>Flujo:</b> 1.- El usuario sitúa el puntero sobre el ítem. 2.- En la zona de texto nos aparecerá el nombre de dicho ítem. 3.- El jugador pulsa el botón de acción. 4.- Se le muestran las distintas acciones disponibles para ese ítem. 5.- El jugador elige la opción usar.	
<b>Poscondiciones:</b> En caso de que tenga éxito, se ejecutarán una serie de acciones.	



#### 4.3.4.Ir a ítem

<b>Nombre:</b>	Ir a ítem
<b>Actores:</b> Jugador	
<b>Descripción:</b> Permite al jugador acercarse a un ítem	
<b>Precondiciones:</b> El ítem (ya sea un objeto o un personaje) deberá poder ser seleccionado con el puntero.	
<b>Flujo:</b> 1.- El usuario sitúa el puntero sobre el ítem. 2.- En la zona de texto nos aparecerá el nombre de dicho ítem. 3.- El jugador pulsa el botón de acción. 4.- Se le muestran las distintas acciones disponibles para ese ítem. 5.- El jugador elige la opción Ir a... 6.- Se haya la ruta hasta el la posición caminable más cercana al ítem. 7.- El personaje principal se desplaza a dicha posición, mostrando una secuencia de frames que dan la sensación de andar.	
<b>Poscondiciones:</b> Ninguna.	



### 4.3.5. Combinar items

<b>Nombre:</b>	Combinar items
<b>Actores:</b> Jugador	
<b>Descripción:</b> Permite al jugador hacer combinaciones entre 2 items que den como resultado una secuencia de acciones.	
<b>Precondiciones:</b> El ítem deberá pertenecer al inventario, tener entre sus opciones la de usar y ser un objeto combinable.	
<b>Flujo:</b> 1.- El usuario entra al inventario. 2.- Viaja entre los distintos objetos hasta encontrar el que quiere combinar. 3.- El jugador pulsa el botón de acción. 4.- Se le muestran las distintas acciones disponibles para ese ítem. 5.- El jugador elige la opción usar 6.- En la zona de texto aparece USAR loquesea CON X, siendo X el objeto que estemos seleccionando en ese momento en el inventario, o con el puntero en la pantalla de juego. 7.- Cuando encuentra el objeto con el que completar la combinación, el jugador pulsa el botón de acción. 8.- Se produce la combinación.	
<b>Poscondiciones:</b> Ninguna.	

### 4.3.6. Ir al menú

<b>Nombre:</b>	Ir al menú
<b>Actores:</b> Jugador	
<b>Descripción:</b> Permite al jugador pasar al menú principal en cualquier momento de la partida.	
<b>Precondiciones:</b> No deberá estar cargando una fase.	
<b>Flujo:</b> 1.- El usuario pulsa sobre el acceso directo, visible en la pantalla. 2.- Se nos muestra el menú principal.	
<b>Poscondiciones:</b> Ninguna.	



### 4.3.7. Hablar con personas

<b>Nombre:</b>	Hablar con personas
<b>Actores:</b> Jugador	
<b>Descripción:</b> Permite al jugador hablar con personajes, los cuales realmente son items del escenario.	
<b>Precondiciones:</b> El personaje deberá tener activada la opción hablar.	
<b>Flujo:</b> 1.- El usuario sitúa el puntero sobre el personaje. 2.- En la zona de texto nos aparecerá el nombre de dicho personaje. 3.- El jugador pulsa el botón de acción. 4.- Se le muestran las distintas acciones disponibles para ese personaje. 5.- El jugador elige la opción hablar. 6.- Comienza la conversación.	
<b>Poscondiciones:</b> Ninguna.	

### 4.3.8. Ver video

<b>Nombre:</b>	Ver video
<b>Actores:</b> Jugador	
<b>Descripción:</b> Un jugador podrá ver distintos videos a lo largo de la partida.	
<b>Precondiciones:</b> El jugador deberá haber llegado a un punto de la historia, o haber realizado cierta acción, para que salten estos videos.	
<b>Flujo:</b> 1.- El usuario resuelve un puzzle, se pasa una fase, o hace una acción importante. 2.- Entre las consecuencias de resolver ese puzzle, de pasar de fase, o de realizar esa acción, esta el de mostrar un video. 3.- Se muestra una serie de viñetas con texto explicatorio. 4.- El jugador podrá ver el contenido completo del texto pulsando arriba y abajo, y pasará de viñeta dando al botón de acción. 5.- Al dar al botón de acción en la última viñeta, volveremos a la partida.	
<b>Poscondiciones:</b> Ninguna.	



### 4.3.9. Moverse por escenario

<b>Nombre:</b>	Moverse por escenario
<b>Actores:</b> Jugador	
<b>Descripción:</b> Permite al jugador moverse por un escenario	
<b>Precondiciones:</b> Para poder llevar a cabo este caso de uso, deberemos estar en la pantalla de juego.	
<b>Flujo:</b> 1.- El usuario sitúa el puntero en el lugar del escenario al que quiere ir. 2.- Pulsa el botón de acción, lo que ejecuta la orden ir a... 3.- Se haya la ruta para ir hasta el punto indicado, o en caso de que este no sea caminable, hasta el lugar más cercano. 4.- Se procede a mover al personaje, mostrando una sucesión de frames que den la sensación de que realmente anda. 5.- Se llega al destino	
<b>Poscondiciones:</b> Ninguna.	

### 4.3.10. Moverse entre escenarios

<b>Nombre:</b>	Moverse entre escenarios.
<b>Actores:</b> Jugador	
<b>Descripción:</b> Permite al jugador moverse por distintos escenarios, dentro de una misma estancia.	
<b>Precondiciones:</b> Para poder llevar a cabo este caso de uso, la estancia en la que este el personaje, deberá tener más de un escenario.	
<b>Flujo:</b> 1.- El usuario sitúa el puntero en una puerta o salida. 2.- El jugador pulsa el botón de acción. 3.- Se le muestran las distintas acciones disponibles para ese ítem. 4.- El jugador elige la opción Ir a.. o abrir, o algo similar. 5.- Se carga un nuevo escenario, con unos nuevos ítems y nuevos personajes. 6.- El personaje formará parte ahora de este nuevo escenario.	
<b>Poscondiciones:</b> Ninguna.	



#### 4.3.11. Moverse entre estancias

<b>Nombre:</b>	Moverse entre estancias.
<b>Actores:</b> Jugador	
<b>Descripción:</b> Permite al jugador moverse por distintas estancias.	
<b>Precondiciones:</b> El jugador deberá estar en el mapa.	
<b>Flujo:</b> 1.- El usuario sitúa el puntero en una determinada posición del mapa, y el puntero cambia de color indicando que es una estancia a la que podemos intentar ir. 2.- En la zona de texto aparece el nombre de dicha estancia. 2.- El jugador pulsa el botón de acción. 3.- Se le muestran las distintas acciones disponibles para esa estancia. 4.- El jugador elige la opción Ir a... 5.- Se carga el escenario activo para esa estancia. 6.- El personaje formará parte ahora de este nuevo escenario.	
<b>Poscondiciones:</b> En caso de que no se pueda ir a la estancia, seguiremos en el mapa.	

#### 4.3.12. Cambiar de fase

<b>Nombre:</b>	Cambiar de fase
<b>Actores:</b> Jugador	
<b>Descripción:</b> La aventura discurrirá por varias fases.	
<b>Precondiciones:</b> El jugador ha cumplido los objetivos de la fase actual.	
<b>Flujo:</b> 1.- El usuario cumple o supera los puzzles de la fase actual. 2.- Realiza la acción final, la cual tiene implícita el cambio de fase. 2.- Se carga una nueva fase.	
<b>Poscondiciones:</b> Esta nueva fase puede comenzar con un video, o ponernos directamente en el mapa, o incluso en un escenario.	



## 4.4. Especificaciones técnicas

A continuación se detalla el tipo de dispositivo al que va destinado el juego:

**Dispositivo:** Teléfono móvil o emulador.

**Java:** El móvil deberá tener Java, con MIDP 2.0 y CLDC 1.0.

**Pantalla:** Ya que sin utilizar un sdk propietario de una marca (como el de nokia), no se puede adaptar el juego a todos los tamaños de pantalla, y con la idea de que el juego pueda ser ejecutado por la mayoría de los móviles, y a la vez se aproveche el tamaño de las pantallas más grandes, se ha decidido, realizando previamente un análisis de los tamaños de pantalla más habituales en estos dispositivos, adaptar el juego (las imágenes) a dos tamaños de pantalla, medido en píxeles:

- o 128x128 → Aquellos móviles con tamaños de pantalla, ya sea de ancho o de alto, menores a estos no podrán ejecutar el juego.
- o 176x200 → Aquellos móviles con tamaños de pantalla, ya sea de ancho o de alto, mayores o igual a estos, utilizarán las imágenes de este tamaño.

**Memoria de almacenamiento:** En esta memoria se almacenará el archivo jar del juego, debido a la gran cantidad de imágenes que utiliza el juego, el tamaño del jar es grande, así que se recomienda que el móvil no tenga limitado el tamaño de este o sino el juego no podrá ser ejecutado.

**Memoria de trabajo:** Es la zona de memoria del móvil donde se almacenan los datos durante la ejecución del mismo. Con la misma idea de que el juego pueda ser utilizado en la mayoría de los móviles que dispongan de MIDP 2.0, como mucho se utilizarán 500 KB de esta memoria.

## 4.5. Diagrama de flujo del menú

Cuando estemos en el menú podremos pasar por los siguiente

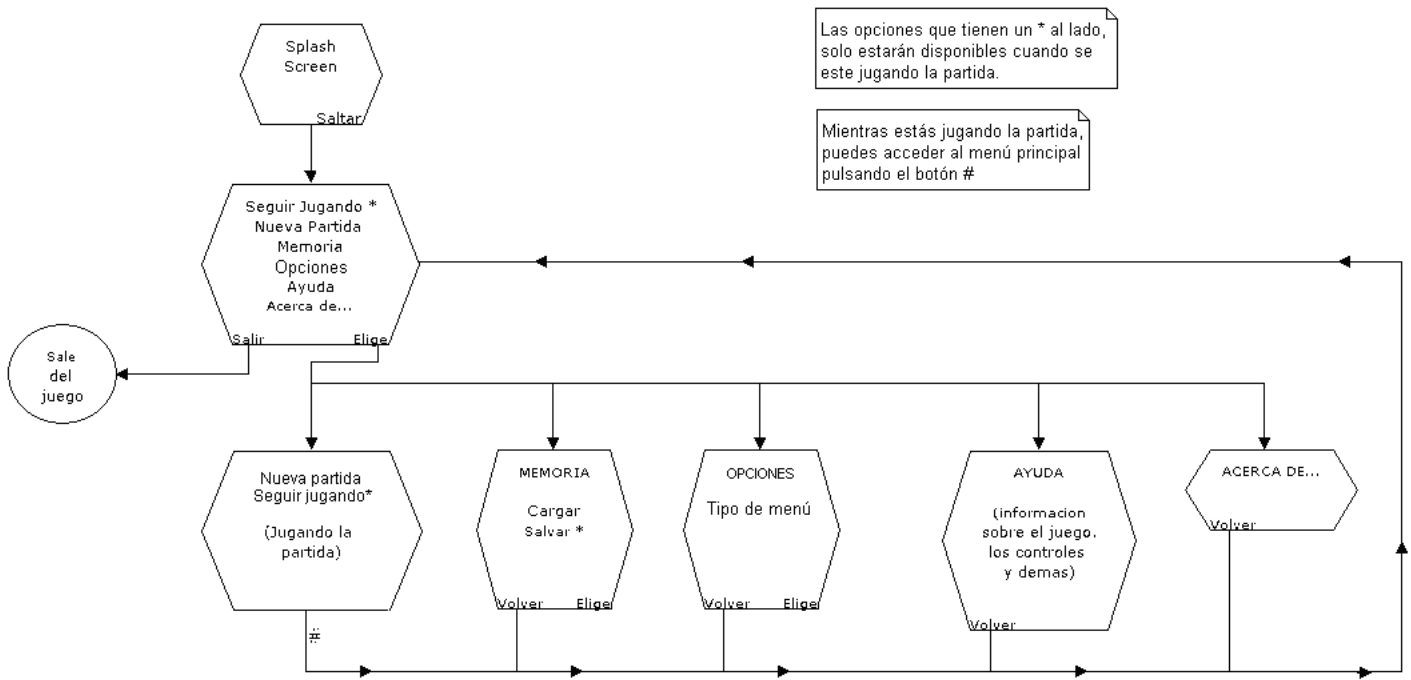


Imagen 20: Diagrama de flujo del menú

## 4.6. Diagrama de estados

A continuación se detallan los principales estados por los que pasa la aplicación.

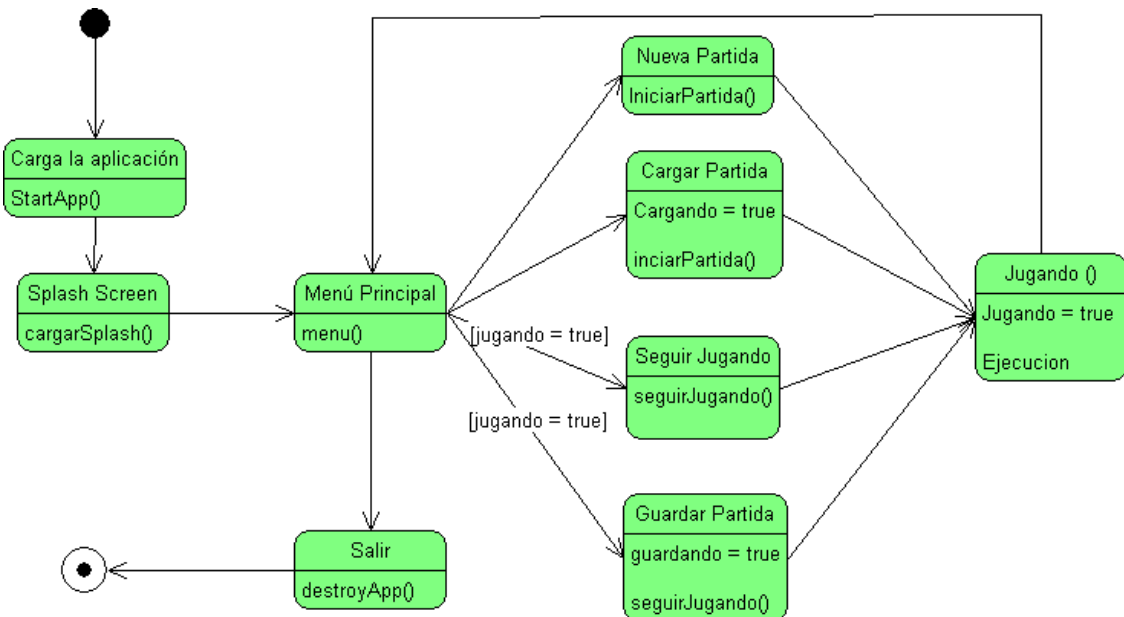


Imagen 21: Diagrama de estados





Como se puede ver, los estados de “Seguir Jugando” y “Guardar partida”, sólo estarán disponibles si jugando es igual a true, es decir, si en ese momento hay una partida en curso.

Otro detalle es que, aunque lo normal es salir de la aplicación a través del menú, es posible salir en cualquier momento en cualquiera de los estados. Esto es por que los móviles suelen traer por defecto que si, estando ejecutando una aplicación, das al botón de colgar la llamada, el móvil cerrará automáticamente el juego.

## 4.7. Patrones de diseño

Teniendo en cuenta que en el mundo móvil la memoria está muy limitada, patrones como el MVC, el cual conlleva un aumento del numero de clases considerable, hay que restringirlos o usarlos con mucho cuidado sólo en aquellos casos en que sea imprescindible. Esto no significa que haya que hacerlo todo en una clase, pero hay que tener en mente que una buena práctica para disminuir la memoria utilizada es limitar el numero de clases que usaremos.

El único que consideró que me puede venir muy bien, y tengo pensado utilizar, es el patrón Factory, ya que me permitiría adaptar los menús a las características de los distintos tipos de dispositivos móviles. El modo en que se implementará este patrón se puede ver en el diagrama de clases del diseño del menú (apartado 4.1.1)

A parte de este, y después de estudiar otros posibles patrones que me podrían venir bien como el Command o el Visitor, he considerado que ninguno me aporta lo suficiente como para sacrificar más memoria.

## 4.8. Estructuras de datos

En este apartado, el objetivo es decidir que estructuras de datos utilizaré teniendo en cuenta que, debido a que el juego es para móviles y que las operaciones que más se van a realizar son las de inserción, borrado y búsqueda, se busca máxima eficiencia (en las operaciones de inserción, borrado y búsqueda) y mínimo consumo de memoria posible.

Las familias de estructuras de datos que tuve en cuenta a la hora de hacer este análisis fueron las siguiente:

**Estructuras lineales (Listas)** → No me sirven, ya que generalmente son poco eficientes, puesto que por ejemplo, para buscar un nodo debería examinar antes todos los precedentes.

**Estructuras jerárquicas (Árboles)** → Son más eficientes, pudiendo conseguir complejidades logarítmicas, para las operaciones que a mí me interesan, si utilizase un



árbol de tipo AVL. El problema es que si el árbol tiene muchos elementos puede llegar a consumir demasiada memoria.

**Estructuras diccionario** → Está bien el hecho de que son muy eficientes, pero consumen demasiada memoria. De estas estructuras me vendría muy bien usar una tabla hash ya que tiene complejidad 1 para inserción, borrado y búsqueda, pero tengo que analizar el consumo de memoria que estas conllevan.

**Estructuras de red o malla** → No me sirven, y esto es debido a que los algoritmos de recuperación de información son complejos y costosos.

A partir de lo anterior, decido que en cuanto a **EFICIENCIA** las que mejor vienen son las siguientes (ordenadas de mejor a peor):

- o Tabla Hash.
- o Árbol AVL.
- o Vector normal y corriente.

Y En cuanto a **CONSUMO DE MEMORIA** el orden de mejor a peor estructura de datos sería:

- o Vector
- o Árbol AVL.
- o Tabla Hash.

También hay que tener en cuenta la memoria que estoy dispuesto a utilizar, mi objetivo es el siguiente:

**Memoria de almacenamiento** → En esta memoria se almacenará el archivo jar del juego, y mi idea es que para que pueda ser utilizado en la mayoría de los móviles que dispongan de MIDP 2.0, el jar no ocupe mas de 500 KB.

**Memoria de trabajo** → Es la zona de memoria del móvil donde se almacenan los datos durante la ejecución del mismo. Con la misma idea de que el juego pueda ser utilizado en la mayoría de los móviles que dispongan de MIDP 2.0, como mucho se utilizarán 500 KB de esta memoria.

Teniendo en cuenta todo esto, he decidido desechar las tablas hash, y decidirme por implementar un AVL. Pero si una vez implementado se ven problemas de memoria, entonces de lo primero que se tirará o se cortará será de los AVL transformándolos en vectores, ya que en las aventuras graficas la velocidad a la que se mueve el juego no es determinante, sin embargo la memoria utilizada si lo es.



## 4.9. Base de datos

En la programación Java para móviles, tenemos prohibido, por la KVM, la creación, eliminación o modificación de ficheros, pero esto no quiere decir que el juego no vaya a usar una base de datos.

Para solucionar esto, J2ME nos proporciona la API RMS (Record Management System). RMS es un pequeño sistema de bases de datos muy sencillo, pero que nos permite añadir información en una memoria no volátil del móvil. RMS no tiene nada que ver con JDBC debido a las limitaciones de los dispositivos J2ME, por lo tanto, el acceso y almacenamiento de la información se hace a mucho más bajo nivel. RMS no puede ser consultado con sentencias SQL ni nada parecido. En una base de datos RMS, el elemento básico es el registro (record), el cual es la unidad de información más pequeña que puede ser almacenada.

Gracias a esta API podremos almacenar el tipo de menú preferido por el jugador, y las partidas guardadas.

## 4.10. Prototipos

Normalmente los prototipos se hacen para que el usuario, que quiere tener una nueva aplicación, y el desarrollador que ha sido contratado, se pongan de acuerdo en lo que buscan. Pero en este caso no había un usuario externo, sino que el mismo que ha tenido la idea la ha desarrollado, por lo tanto estaba muy claro lo que se quería.

Aun así se realizó un prototipo, mas bien una maqueta, ya que no es un ejecutable, sino un video. En él se ve como será la interfaz y como será la interacción en el juego en todos los puntos de este.

Una vez terminado fue muy útil para que el director del proyecto viese la idea y diese su visto bueno.

A continuación se muestra un conjunto de imágenes sacadas de la maqueta, en la cual, aunque realizado a base de bocetos, se podía ver cual era la idea. Desde esta idea inicial se ha realizado muchas variaciones, que también pueden ser apreciadas en las imágenes que siguen:



Imagen 22: Imágenes de la maqueta

Como se puede ver, respecto a estas imágenes, el menú y la conversación han sufrido poca variación, sin embargo la pantalla de juego ha cambiado bastante, ya que ahora el inventario no aparece constantemente en pantalla, lo cual restaba mucho espacio disponible, y las opciones no aparecen en el medio de la pantalla, ni están limitadas a sólo esas seis.



## 5. Diseño

### 5.1. Diseño del menú

#### 5.1.1. Diagrama de clases

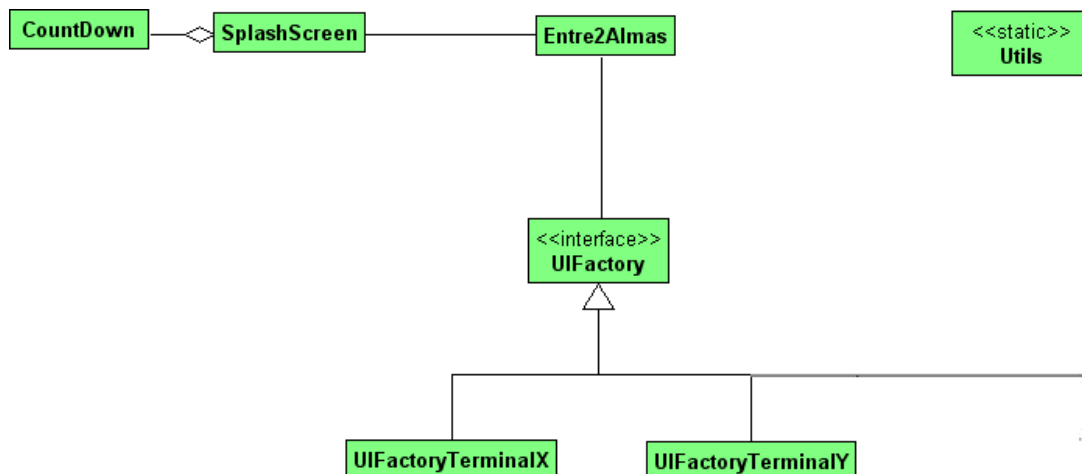


Imagen 23: Diagrama de clases del menú

#### 5.1.2. Descripción detallada de las clases

##### 5.1.2.1. Countdown

Clase llamada desde el SplashScreen, cuando el splash es mostrado en pantalla y ha pasado el "dismissTime". Será el contador que controle cuanto tiempo debe permanecer, cada una de las imágenes de la presentación, en la pantalla.

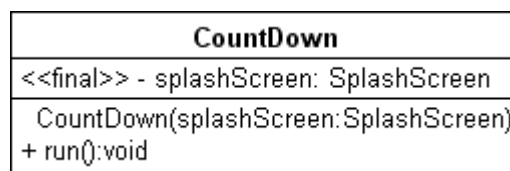


Imagen 24: Clase CountDown

#### Atributos

private final SplashScreen **splashScreen**

#### Constructor

**CountDown**(SplashScreen splashScreen)



## Métodos

public void **run()**

Aquí se establece la acción a ser realizada por un TimerTask

### 5.1.2.2. Entre2Almas

Clase principal del juego Entre2Almas, extiende de MIDlet y por ello esta clase será la que primero se ejecute al iniciar el midlet.

Entre2Almas
- display: Display - isSplash: bool=true + tipoMenu: int - avanzado: UITerminalSTBG - clasico: UITerminalST - rs: RecordStore + menuActivo: int + gameCanvas: Ejecucion - parser: KXmlParser + preguardado: Vector
+ startApp():void + getDisplay():Display + pauseApp():void + destroyApp(unconditional:bool):void - cargarSplash():void + menu():void + getTipoMenu():string + IniciarPartida():void + seguirJugando():void + salir():void - getMenuRs():int + setMenuRs():void + cargarDatos():void + getDatosImagen():StringBuffer + cargarApariencia():void + preguardar(res:Resultado):void + guardarDatos():void - insertarRegistro(baos:ByteArrayOutputStream):void + cargaDatosEsenciales():byte[] + cargarResultados():void + puedeGuardar():bool + hayGuardada():bool + mostrarAlerta(texto:string,tipo:AlertType):void

Imagen 25: Clase Entre2Almas

## Atributos

private javax.microedition.lcdui.Display **display**

Representa el manejador de la pantalla, y solo habrá una instancia de Display por midlet



---

private boolean **isSplash**

Indica si se debe mostrar el Splash Screen

---

public int **tipoMenu**

Indica si tenemos un menú clásico (0) o avanzado (1);

---

private UITerminalSTBG **avanzado**

La interfaz de usuario del menú, de tipo estándar con fondo de pantalla. (Avanzado)

---

private UITerminalST **clasico**

La interfaz de usuario del menú, de tipo estándar. (Clásico)

---

private javax.microedition.rms.RecordStore **rs**

Será la base de datos del juego.

---

private javax.microedition.rms.RecordStore **rsTemp**

Será la base de datos temporal del juego. Se utilizará para a la hora de guardar la partida por si acaso ocurriese que el móvil se queda sin batería, o que se apaga de repente, o algo raro, el guardar partida se hace primero en un rms temporal, y luego, cuando se ha completado, se pasa al bueno.

---

public int **menuActivo**

Si es 0 indica que el menú activo es clásico y viceversa

---

public Ejecucion **gameCanvas**

Donde se desarrollará el juego

---

private org.kxml2.io.KXmlParser **parser**

Parser utilizado para cargar los datos básicos del juego. Como son el splash screen y las imágenes del menú.

---

public java.util.Vector **preguardado**

Guarda aquellos datos necesarios para restaurar el juego por donde estaba en un futura partida.

---

<b>Constructor</b>
--------------------

public **Entre2Almas()**



## Métodos

public void **startApp()**

Método llamado cuando se inicia la aplicación.(heredado de MIDlet)

---

public javax.microedition.lcdui.Display **getDisplay()**

devuelve la referencia al display

**Devuelve:**

Referencia al display de la aplicación.

---

public void **pauseApp()**

Método llamado cuando se pausa el juego (heredado de MIDlet)

---

public void **destroyApp**(boolean unconditional)

Método llamado cuando se destruye el juego (heredado de MIDlet)

---

private void **cargarSplash()**

Carga la pantalla de presentación

---

public void **menu()**

A partir de lo que el usuario elija en las opciones, cargará el menú clásico o el avanzado

---

public java.lang.String **getTipoMenu()**

Devuelve el tipo de menú que en ese momento esta seleccionado en las opciones

**Devuelve:**

El tipo de menú que en ese momento esta seleccionado en las opciones

---

public void **IniciarPartida()**

Iniciamos la partida.

---

public void **seguirJugando()**

Reanuda la partida, que previamente se había parado para salir al menú principal

---

public void **salir()**

salimos de la aplicación.





---

private int **getMenuRs()**

A partir de un chorro de bytes que había en un registro, nos devuelve el ultimo menú que utilizamos en la anterior partida.

**Devuelve:**

el tipo de menú

---

public void **setMenuRs()**

Actualiza la base de datos de menú

---

private void **cargarDatos()**

Carga los datos necesarios, del archivo XML, para mostrar el menú y el splash screen

**Throws:**

org.xmlpull.v1.XmlPullParserException - Excepción del parser

java.io.IOException - Excepción de entrada/salida

---

public java.lang.StringBuffer **getDatosImagen()**

Recupera, a partir de un archivo XML, los datos descriptivos de una imagen

**Throws:**

java.io.IOException - Excepción de entrada/salida

org.xmlpull.v1.XmlPullParser - Exception Excepción en el XML

org.xmlpull.v1.XmlPullParserException

---

private void **cargaOpMenuPrin()**

Carga del archivo basics.xml, las nombre de las opciones del Menú Principal.

**Throws:**

org.xmlpull.v1.XmlPullParserException

java.io.IOException

---

private void **cargaOpMemo()**

Carga del archivo basics.xml, las nombre de las opciones del Menú Memoria.

**Throws:**

org.xmlpull.v1.XmlPullParserException

java.io.IOException

---

private void **cargaOpAyuda()**

Carga del archivo basics.xml, las nombre de las opciones del Menú Ayuda. Además también cargará el contenido de cada una de esta sub-ayudas.

**Throws:**

org.xmlpull.v1.XmlPullParserException

java.io.IOException

---



---

public void **cargarApariencia()**

Carga una apariencia de nuestro personaje principal

**Throws:**

org.xmlpull.v1.XmlPullParserException  
java.io.IOException

---

public void **pregarduar**(Resultado res)

Guardará en pregardado los resultados necesarios para restablecer el juego al momento de la última partida guardada.

**Parámetros:**

res - Resultado a almacenar.

---

public void **guardarDatos()**

Guarda, los datos necesarios de la partida actual, en la base de datos.

---

private void **insertarRegistro**(java.io.ByteArrayOutputStream baos)

Inserta un registro en el RMS que está actualmente abierto.

**Parámetros:**

baos - Stream donde está la información a guardar

**Throws:**

javax.microedition.rms.RecordStoreNotOpenException  
javax.microedition.rms.InvalidRecordIDException  
javax.microedition.rms.RecordStoreFullException  
javax.microedition.rms.RecordStoreException

---

public byte[] **cargaDatosEsenciales()**

A la hora de cargar una partida, la cual fue guardada anteriormente, este método nos devuelve los datos esenciales de la carga. (nFase,mapaActual, escenario y estancia actual y parteJuego).

**Devuelve:**

Un array de bytes que contiene todos los datos esenciales

---

public void **cargarResultados()**

A la hora de cargar una partida, la cual fue guardada anteriormente, este método carga los resultados que hay que ejecutar para que la partida se quede en el mismo estado que cuando se guardo.

---

public boolean **puedeGuardar()**

Indica, atendiendo al estado actual de la partida, si se puede guardar o no la partida.

**Devuelve:**

Si se puede guardar o no la partida.

---



---

```
public boolean hayGuardada()
```

Indica si hay alguna partida guardada.

**Devuelve:**

Indica si hay alguna partida guardada.

---

```
public void mostrarAlerta(java.lang.String texto, microedition.lcdui.AlertType tipo)
```

Muestra una alerta por pantalla

**Parámetros:**

texto - El texto que contendrá la alerta.

tipo - El tipo de alerta a mostrar

### 5.1.2.3. SplashScreen

La pantalla de presentación. Hereda de Canvas.

SplashScreen
- display: Display - next: Displayable - timer: Timer - sprite: Sprite - dismissTime: int <<static>> - frames: int
+ SplashScreen(midlet:Entre2Almas,next:Displayable,dismissTime:int) <<static>> + access(splashScreen:SplashScreen):void - cont():void - dismiss():void # keyPressed():void # paint(g:Graphics):void # pointerPressed(x:int,y:int):void # showNotify():void

Imagen 26: Clase SplashScreen

#### Atributos

```
private javax.microedition.lcdui.Display display
```

Manejador de la pantalla

---

```
private javax.microedition.lcdui.Displayable next
```

Objeto con capacidad para ser situado en la pantalla (display), y representa a lo siguiente que aparecerá en la pantalla cuando desaparezca se quite el splash

---

```
private java.util.Timer timer
```

Controlará el tiempo de ejecución



---

private javax.microedition.lcdui.game.Sprite **sprite**  
Sprite que contiene todos los frames del Splash screen

---

private int **dismissTime**  
Tiempo que se verá cada frame del sprite

---

private static int **frames**  
Numero de frames que tiene el sprite

---

## Constructor

public **SplashScreen**(Entre2Almas midlet,  
                        javax.microedition.lcdui.Displayable next,  
                        int dismissTime)

Constructor del SplashScreen

**Parámetros:**

midlet - Midlet del juego.

next - siguiente pantalla a mostrar una vez que pase el splash

dismissTime - tiempo que se mostrará cada frame

## Métodos

public static void **access**(SplashScreen splashScreen)  
Comprueba si después de mostrar una imagen del splash, un determinado tiempo, todavía quedan imágenes por mostrar o no

**Parámetros:**

splashScreen - el splash screen que estamos mostrando

---

private void **cont**()  
Continúa mostrando la siguiente imagen que le corresponde al splash screen

---

private void **dismiss**()  
Anula el timer que controlaba el tiempo que se mostraba cada splash y muestra la siguiente pantalla (el menú)

---

protected void **keyPressed**(int keyCode)  
Método llamado cuando una tecla es presionada.

---

protected void **paint**(javax.microedition.lcdui.Graphics g)  
Pinta la pantalla

---



---

```
protected void pointerPressed(int x,  
                                int y)  
    Método llamado si un puntero es pulsado
```

---

```
protected void showNotify()
```

Este método salta inmediatamente después de que el Canvas sea visible en la pantalla.

#### 5.1.2.4. UIFactory

Interfaz que define que métodos deberá usar un menú. Es un patrón factory. El patrón factory es aplicado aquí, ya que, las diferencias, en cuanto a interfaz de usuario, de los móviles que disponen J2ME, son muy grandes, y sería bueno que el juego pudiese aprovechar al máximo la interfaz de cada terminal.

Así que para solucionar esto, tengo 2 opciones: Utilizar componentes de interfaz de usuario de alto nivel o utilizar el patrón Factory. Me gusta más esta última opción ya que así podré aprovechar más las capacidades de los terminales y además quedará más "bonito", puesto que a alto nivel no se puede personalizar demasiado (hablando de imágenes y colores) el interfaz.

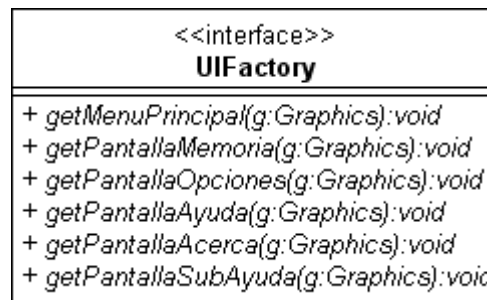


Imagen 27: Clase UIFactory

#### Métodos

```
void getMenuPrincipal(javax.microedition.lcdui.Graphics g)  
    Dibuja la pantalla del menú principal  
Parámetros:  
    g - Objeto Graphics
```

---

```
void getPantallaMemoria(javax.microedition.lcdui.Graphics g)  
    Muestra las opciones que hay dentro de "Memoria"  
Parámetros:  
    g - Objeto Graphics
```

---



void **getPantallaOpciones**(javax.microedition.lcdui.Graphics g)

Entrar en el menú de opciones

**Parámetros:**

g - Objeto Graphics

---

void **getPantallaAyuda**(javax.microedition.lcdui.Graphics g)

Dibuja el contenido de la "Ayuda"

**Parámetros:**

g - Objeto Graphics

---

void **getPantallaSubAyuda**(javax.microedition.lcdui.Graphics g)

Dibuja el contenido de las distintas opciones de la ayuda

**Parámetros:**

g - Objeto Graphics

---

void **getPantallaAcerca**(javax.microedition.lcdui.Graphics g)

Muestra el contenido de la "Acerca de..."

**Parámetros:**

g - Objeto Graphics

### 5.1.2.5. UITerminalST

Es el menú de alto nivel, el cual se adapta a las propiedades nativas de los móviles.

UITerminalST
<ul style="list-style-type: none"><li>- midlet: Entre2Almas</li><li>- display: Display</li><li>- volverCommand: Command</li><li>- salirCommand: Command</li><li>- acceptCommand: Command</li><li>opciones: List</li><li>formulario: Form</li><li>opcionOpciones: ChoiceGroup</li><li>mensaje: StringItem</li></ul>
<ul style="list-style-type: none"><li>+ UITerminalST(midlet: Entre2Almas)</li><li>+ getMenuPrincipal(g: Graphics): void</li><li>+ getPantallaMemoria(g: Graphics): void</li><li>+ getPantallaOpciones(g: Graphics): void</li><li>+ getPantallaAyuda(g: Graphics): void</li><li>+ getPantallaSubAyuda(g: Graphics): void</li><li>+ getPantallaAcerca(g: Graphics): void</li><li>+ commandAction(c: Command, d: Displayable): void</li><li># getPantallaAcerca(g: Graphics): void</li></ul>

Imagen 28: Clase UITerminalST



## Atributos

private Entre2Almas **midlet**

Midlet del juego

private javax.microedition.lcdui.Display **display**

Manejador de la pantalla

private javax.microedition.lcdui.Command **volverCommand**

Comando volver, el cual tendrá prioridad 0. Se utiliza en la pantalla de ayuda, de opciones y de memoria

private javax.microedition.lcdui.Command **salirCommand**

Comando Salir, el cual tendrá prioridad 0. Se utiliza en el menú principal

private javax.microedition.lcdui.Command **acceptCommand**

Comando aceptar, el cual tendrá prioridad 1. Se utiliza en la pantalla de opciones

javax.microedition.lcdui.List **opciones**

Lista que será utilizada para mostrar las opciones de los distintos menús.

javax.microedition.lcdui.Form **formulario**

Formulario en que se mostrarán StringItems y demás.

javax.microedition.lcdui.ChoiceGroup **opcionOpciones**

ChoiceGroup utilizado en la pantalla de opciones.

javax.microedition.lcdui.StringItem **mensaje**

Mensaje que se mostrará en la ayuda y en el acerca de... Un StringItem va dentro de un Form

## Constructor

public **UITerminalST**(Entre2Almas midlet)

Constructor de la clase que representa el interfaz clásico.



## Métodos

```
public void getMenuPrincipal(javax.microedition.lcdui.Graphics g)
```

Dibuja la pantalla del menú principal

**Parámetros:**

g - objeto Graphics

```
public void getPantallaMemoria(javax.microedition.lcdui.Graphics g)
```

Muestra las opciones que hay dentro de "Memoria"

**Parámetros:**

g - Objeto Graphics

```
public void getPantallaOpciones(javax.microedition.lcdui.Graphics g)
```

Dibuja el contenido de la pantalla "Opciones"

**Parámetros:**

g - Objeto Graphics

```
public void getPantallaAyuda(javax.microedition.lcdui.Graphics g)
```

Dibuja el contenido de la "Ayuda"

**Parámetros:**

g - Objeto Graphics

```
public void getPantallaSubAyuda(javax.microedition.lcdui.Graphics g)
```

Dibuja el contenido de la "Ayuda"

**Parámetros:**

g - Objeto Graphics

```
public void getPantallaAcerca(javax.microedition.lcdui.Graphics g)
```

Muestra el contenido de la "Acerca de..."

**Parámetros:**

g - Objeto Graphics

```
public void commandAction(javax.microedition.lcdui.Command c,  
                           javax.microedition.lcdui.Displayable d)
```

Cuando se pulsa un comando, viene a este método.

```
protected void paint(javax.microedition.lcdui.Graphics g)
```

Método llamado por el display cuando pone en pantalla este Canvas





### 5.1.2.6. UITerminalSTBG

Menú de bajo nivel, es decir, que será igual en todos los móviles, pero es más colorido, más bonito, más vistoso.

UITerminalSTBG
<ul style="list-style-type: none"><li>- midlet: Entre2Almas</li><li>- sprite: Sprite</li><li>- volverCommand: Command</li><li>- selectCommand: Command</li><li>- salirCommand: Command</li><li>- acceptCommand: Command</li><li>- menuOpldx: int</li><li>- menuMemldx: int</li><li>- opciones: Vector</li></ul>
<ul style="list-style-type: none"><li>+ UITerminalSTBG(midlet:Entre2Almas)</li><li>+ getMenuPrincipal(g:Graphics):void</li><li>+ getPantallaMemoria(g:Graphics):void</li><li>+ getPantallaGuardando(g:Graphics):void</li><li>+ getPantallaAyuda(g:Graphics):void</li><li>+ getPantallaAcerca(g:Graphics):void</li><li>+ getPantallaOpciones(g:Graphics):void</li><li>+ getPantallaSubAyuda(g:Graphics):void</li><li>+ cargar(g:Graphics,pantalla:int):void</li><li># paint(g:Graphics):void</li><li># keyPressed(code:int):void</li><li>+ commandAction(c:Command,d:Displayable):void</li><li>+ ajustarVariables():void</li></ul>

Imagen 29: Clase UITerminalSTBG

#### Atributos

private Entre2Almas **midlet**  
Midlet del juego

private javax.microedition.lcdui.game.Sprite **sprite**  
sprite que albergará las imágenes de fondo.

private javax.microedition.lcdui.Command **volverCommand**  
Comando volver, el cual tendrá prioridad 0.Se utiliza en la pantalla de ayuda, de opciones y de memoria

private javax.microedition.lcdui.Command **selectCommand**  
Comando Elegir, el cual tendrá prioridad 1.Se utiliza en el menú principal



---

private javax.microedition.lcdui.Command **salirCommand**

Comando Salir, el cual tendrá prioridad 0. Se utiliza en el menú principal

---

private javax.microedition.lcdui.Command **acceptCommand**

Comando aceptar, el cual tendrá prioridad 1. Se utiliza en la pantalla de opciones

---

private int **menuOpIdx**

Indica cual de las opciones del menú opciones está seleccionada.

---

private int **menuMemIdx**

Indica cual de las opciones del menú memoria está seleccionada.

---

private java.util.Vector **opciones**

Opciones de los menús

---

## Consttrecutor

public **UITerminalSTBG**(Entre2Almas midlet)

## Métodos

public void **getMenuPrincipal**(javax.microedition.lcdui.Graphics g)

Dibuja la pantalla del menú principal

**Parámetros:**

g - objeto Graphics

---

public void **getPantallaMemoria**(javax.microedition.lcdui.Graphics g)

Muestra las opciones que hay dentro de "Memoria"

**Parámetros:**

g - Objeto Graphics

---

public void **getPantallaGuardando**(javax.microedition.lcdui.Graphics g)

Muestra la pantalla de guardando

---

public void **getPantallaAyuda**(javax.microedition.lcdui.Graphics g)

Dibuja el contenido de la "Ayuda"

**Parámetros:**

g - Objeto Graphics

---

public void **getPantallaAcerca**(javax.microedition.lcdui.Graphics g)

Muestra el contenido de la "Acerca de..."

---



**Parámetros:**

g - Objeto Graphics

---

public void **getPantallaOpciones**(javax.microedition.lcdui.Graphics g)

Dibuja el contenido de la pantalla "Opciones"

**Parámetros:**

g - Objeto Graphics

---

public void **getPantallaSubAyuda**(javax.microedition.lcdui.Graphics g)

Dibuja el contenido de las distintas opciones de la ayuda

**Parámetros:**

g - Objeto Graphics

---

private void **cargar**(javax.microedition.lcdui.Graphics g,  
int pantalla)

Decide que y cuando cargar en cada momento

**Parámetros:**

g - Objeto Graphics

pantalla - pantalla que se desea cargar (ejemplo ayuda, opciones...)

---

protected void **paint**(javax.microedition.lcdui.Graphics g)

Método llamado por el display cuando pone en pantalla este Canvas

---

protected void **keyPressed**(int code)

Captura la pulsación de teclas en el menú

---

public void **commandAction**(javax.microedition.lcdui.Command c,  
javax.microedition.lcdui.Displayable d)

Cuando se pulsa un comando, viene a este metodo.

---

public void **ajustarVariables**()

Reajusta algunas variables como son el ancho y alto de la pantalla. Carga el numero de letras que caben en el ancho de la pantalla, y el numero de líneas que caben en el alto de la pantalla

### 5.1.2.7. Utils

En esta clase estática se realizarán tareas repetitivas como createImage, se guardarán los identificadores de las imágenes del menú, algunas variables importantes como la que indica la posición inicial a la que empezar a escribir, también tendrá variables que indican si se está jugando, guardando o cargando una partida en ese momento, etc.



La representación de la clase Utils, que se muestra a continuación, estará partida en 2 partes, ya que es demasiado grande.

Además no se incluyen los atributos estáticos que identifican los resultados, ya que son demasiados, y no cabrían. Se pueden ver en el punto 7.8.2.13.

<<static>> <b>Utils</b>	
<pre> grupo: int {static} imagen: Image=NULL {static} ancholm: int {static} altolm: int {static} nomImagen: StringBuffer {static} imagenes: String[] {static} splash: string {static} imMenuPrin: string {static} imOpciones: string {static} imMemoria: string {static} imAyuda: string {static} imAcerca: string {static} spCargando: string {static} imGuardando: string {static} spPuntero: string {static} imPunCam: string {static} spOscarLado: string {static} spOscarEspa: string {static} spOscarFren: string {static} spOscarCara: string {static} imTmplnv: string {static} apariencias: string[] {static} &lt;&lt;final&gt;&gt; lowFont: Font {static} &lt;&lt;final&gt;&gt; highFont: Font {static} &lt;&lt;final&gt;&gt; widthLF: int {static} &lt;&lt;final&gt;&gt; lowColor: int=0xFFFFFFFF {static} &lt;&lt;final&gt;&gt; highColor: int=0x00000000 {static} startHeight: int {static} startWidth: int {static} &lt;&lt;final&gt;&gt; spaceTextItemPeq: int=14 {static} &lt;&lt;final&gt;&gt; spaceTextItemGra: int=22 {static} spacing: int {static} spaceMen: int {static} numLineas: int {static} linea: int {static} </pre>	<pre> mensaje: StringBuffer {static} numLineasMensaje: int {static} palabrasMensaje: Vector {static} width: int {static} height: int {static} inicioPantallaX: int {static} inicioPantallaY: int {static} jugando: bool=false {static} cargandoPartida: bool=false {static} guardandoPartida: bool=false {static} menuPrinIdx: int {static} menuAyudaIdx: int {static} + parteMenu: int {static} + cargada: int {static} &lt;&lt;final&gt;&gt; + menuPrin: int=0 {static} &lt;&lt;final&gt;&gt; + memoria: int=1 {static} &lt;&lt;final&gt;&gt; + ayuda: int=2 {static} &lt;&lt;final&gt;&gt; + acerca: int=3 {static} &lt;&lt;final&gt;&gt; + opcion: int=4 {static} &lt;&lt;final&gt;&gt; + ayudaTematica: int=5 {static} &lt;&lt;final&gt;&gt; + ayudaTeclas: int=6 {static} &lt;&lt;final&gt;&gt; + ayudaVideo: int=7 {static} &lt;&lt;final&gt;&gt; + ayudaInventario: int=8 {static} &lt;&lt;final&gt;&gt; + ayudaMapa: int=9 {static} &lt;&lt;final&gt;&gt; + ayudaConversacion: int=10 {static} + titulo: string {static} + opMenuPrin: string[] {static} + opMenuMemo: string[] {static} + opMenuAyuda: string[] {static} + txtSubAyuda: string[] {static} + conteAcerca: string {static} + tiposMenu: string[] {static} + tiposMenuTxt: string {static} + numOps: int {static} &lt;&lt;final&gt;&gt; + aumentaX: int {static} &lt;&lt;final&gt;&gt; + aumentaY: int {static} </pre>
	<pre> - Utils() &lt;&lt;static&gt;&gt; + getGrupo():void &lt;&lt;static&gt;&gt; + getTamGrupoX():int &lt;&lt;static&gt;&gt; + getTamGrupoY():int &lt;&lt;static&gt;&gt; + getFrames(peticion:string):int[] &lt;&lt;static&gt;&gt; + getFrameInicial():int &lt;&lt;static&gt;&gt; + getNumFrames(peticion:string):int &lt;&lt;static&gt;&gt; - getDatosImagen(peticion:string):string &lt;&lt;static&gt;&gt; - getIdx(datosImagen:string):int &lt;&lt;static&gt;&gt; - getRuta(nombreImagen:string):string &lt;&lt;static&gt;&gt; - getAncho(nombreImagen:string):int &lt;&lt;static&gt;&gt; - getAlto(nombreImagen:string):int &lt;&lt;static&gt;&gt; + cargarImagen(nomSprite:string):void &lt;&lt;static&gt;&gt; - creada(nombre:string):bool &lt;&lt;static&gt;&gt; - creaImagen(nombre:string):Image &lt;&lt;static&gt;&gt; + getSpaceTextItem():int &lt;&lt;static&gt;&gt; + actualizarApariencia(activa:int):void &lt;&lt;static&gt;&gt; + getIdxResultado(resultado:string):int </pre>

Imagen 30: Clase Utils



## Atributos

### static int **grupo**

Habr  dos tama os de im genes, dependiendo del tama o de pantalla del m vil. As  tendremos 2 grupos: El de 128x128 --> Aquellos m viles con pantallas iguales o superiores a 128x128 --> Su valor int es 0 El de 176x200 --> Aquellos m viles con pantallas iguales o superiores a 176x200 --> Su valor int es 1

### static javax.microedition.lcdui.Image **imagen**

Solo habr  una imagen cargada en cada momento, cada imagen puede contener los frames de muchos sprites, lo cual hace que sean muy grandes y por eso solo se mantiene una cargada en memoria.

### static int **anchoIm**

Adem s tambi n se guardan sus datos de alto y ancho de cada uno de los frames que la forman.

### static int **altoIm**

Adem s tambi n se guardan sus datos de alto y ancho de cada uno de los frames que la forman.

### static java.lang.StringBuffer **nomImagen**

Y tambi n el nombre que tiene la imagen, para as  saber cual tenemos creada y as  puede que nos encontremos con casos en los que no es necesario cargar la imagen porque ya est  cargada

### static java.lang.String[] **imagenes**

Array de im genes que contiene descripciones de im genes cuyos par metros son: Antes del primer punto y coma se indicar  la ruta de la imagen despu s se indicar  el ancho de cada uno de los frames de la imagen y por ultimo se indicar  el alto de cada uno de los frames de la imagen

### static java.lang.String **splash**

### static java.lang.String **imMenuPrin**

### static java.lang.String **imOpciones**

### static java.lang.String **imMemoria**



---

```
static java.lang.String imAyuda
```

---

```
static java.lang.String imAcerca
```

---

```
static java.lang.String spCargando
```

---

```
static java.lang.String imGuardando
```

---

```
static java.lang.String spPuntero
```

---

```
static java.lang.String imPunCami
```

---

```
static java.lang.String spOscarLado
```

---

```
static java.lang.String spOscarEspa
```

---

```
static java.lang.String spOscarFren
```

---

```
static java.lang.String spOscarCara
```

---

```
static java.lang.String imTmpInv
```

---

```
static java.lang.String[] apariencias
```

---

```
static final javax.microedition.lcdui.Font lowFont  
    Fuentes que vamos a utilizar en el menú
```

---

```
static final javax.microedition.lcdui.Font highFont
```

---

```
static final int widthLF  
    Ancho de lowFont
```

---

```
static final int lowColor  
    Establecemos el color de las letras del menú
```

---

```
static final int highColor
```

---



static int **startHeight**

Altura y anchura a la que se comienzan a imprimir las letras

---

static int **startWidth**

---

static final int **spaceTextItemPeq**

Marca el espacio que se deja para el nombre de los items en la pantalla de juego, para los móviles de grupo pequeño

---

static final int **spaceTextItemGra**

Marca el espacio que se deja para el nombre de los items en la pantalla de juego, para los móviles de grupo grande

---

static int **spacing**

Delimita el espacio entre líneas de caracteres

---

static int **spaceMen**

Espacio que se deja para imprimir el mensaje

---

static int **numLineas**

Numero de líneas que caben en una pantalla para el grupo al que pertenece el móvil

---

static int **linea**

De entre las líneas que forman un mensaje, este atributo define cual se mostrará primero por pantalla

---

static java.lang.StringBuffer **mensaje**

Mensaje a imprimir

---

static int **numLineasMensaje**

Numero de líneas que ocupa el mensaje a imprimir

---

static java.util.Vector **palabrasMensaje**

las palabras que forman el mensaje a imprimir

---

static int **width**

Ancho de la pantalla



---

static int **height**

Alto de la pantalla

---

static int **inicioPantallaX**

Marca el píxel, a partir del cual empieza la coordenada X del cuadro de 128x128 o de 176x200, y este cuadro centrado estará en el móvil. Para establecer su valor se utiliza el primer sprite de tamaño 128x128 o 176x200 (según grupo) y se aplica la siguiente formula  $(Utils.width / 2) - (sprite.getWidth() / 2)$

---

static int **inicioPantallaY**

Marca el píxel, a partir del cual empieza la coordenada Y del cuadro de 128x128 o de 176x200, y este cuadro centrado estará en el móvil. Para establecer su valor se utiliza el primer sprite de tamaño 128x128 o 176x200 (según grupo) y se aplica la siguiente formula  $(Utils.height / 2) - (sprite.getHeight() / 2)$

---

static boolean **jugando**

Indica si el juego está en ejecución, lo cual significa que debemos mostrar el menú con la opción Seguir Jugando.

---

static boolean **cargandoPartida**

Indica si se esta cargando una partida guardada con anterioridad.

---

static boolean **guardandoPartida**

Indica si se esta guardando una partida.

---

static int **menuPrinIdx**

Indica cual de las opciones del menú principal está seleccionada.

---

static int **menuAyudaIdx**

Indica cual de las opciones del menú ayuda está seleccionada.

---

public static int **parteMenu**

Parte del menú en el que nos encontramos

---

public static int **cargada**

Almacena el identificador de la pantalla que está actualmente guardada

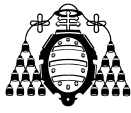
---

public static final int **menuPrin**

Cada pantalla estará identificada con un entero

---





---

public static final int **memoria**

---

public static final int **ayuda**

---

public static final int **acerca**

---

public static final int **opcion**

---

public static final int **ayudaTematica**

---

public static final int **ayudaTeclas**

---

public static final int **ayudaVideo**

---

public static final int **ayudaInventario**

---

public static final int **ayudaMapa**

---

public static final int **ayudaConversacion**

---

public static java.lang.String **titulo**

---

public static java.lang.String[] **opMenuPrin**  
Array que almacenará las opciones del menú principal.

---

public static java.lang.String[] **opMenuMemo**  
Array que almacenará las opciones del menú memoria.

---

public static java.lang.String[] **opMenuAyuda**  
Array que almacenará las opciones del menú ayuda.

---

public static java.lang.String[] **txtSubAyuda**  
Array que almacenará el texto de cada una de las sub-ayudas.

---

public static java.lang.String **conteAcerca**  
Almacenará el contenido de acerca de...

---



---

```
public static java.lang.String[] tiposMenu
```

Almacena el nombre de los tipos de menú que puede haber - En la posición 0 estará el nombre para el menú de alto nivel, el cual está adaptado a la interfaz nativa de cada móvil. - En la posición 1 estará el nombre para el menú de bajo nivel, es decir, que será igual en todos los móviles, pero es más colorido, más bonito, más vistoso, pero no adaptado a la interfaz nativa de cada móvil

---

```
public static java.lang.String tiposMenuTxt
```

Nombre de la opción elegir tipo de menú

---

```
public static final int reSalir
```

---

```
public static final int reMensaje
```

---

```
public static final int reAddInvent
```

---

```
public static final int reDelInvent
```

---

```
public static final int reVideo
```

---

```
public static final int reCombinando
```

---

```
public static final int reMapa
```

---

```
public static final int reIrEstancia
```

---

```
public static final int reConversacion
```

---

```
public static final int reRespuesta
```

---

```
public static final int reRespuOscar
```

---

```
public static final int reJuego
```

---

```
public static final int reCargaOpSup
```

---

```
public static final int reCargaOpInf
```

---



public static final int **reQuitarOp**

---

public static final int **reQuitarOpYSup**

---

public static final int **reCamActivaOpIt**

---

public static final int **reCamActivaOpEs**

---

public static final int **reCamActivaOpCon**

---

public static final int **reCamFase**

---

public static final int **reEliminarItem**

---

public static final int **reEliminarEstan**

---

public static final int **reCamNombreIt**

---

public static final int **reCamNombreEs**

---

public static final int **reCaminarBlo**

---

public static final int **reSalirJuego**

---

public static final int **reCamConsecuCom**

---

public static final int **reCamConsecuCon**

---

public static final int **reCamActivaCom**

---

public static final int **reCambiaEscenario**

---

public static final int **reCamConsecuIt**

---

public static final int **reCamConsecuEs**

---

public static final int **reCambiaVisibleIt**

---



public static final int **reCambiaVisibleEs**

---

public static final int **reCaminar**

---

public static final int **reCamMapa**

---

public static final int **reCamApariencia**

---

public static final int **reAddInventSin**

---

public static int **numOps**

---

public static final int **aumentaX**

---

public static final int **aumentaY**

## Constructor

private **Utils()**

No se podrá hacer instancias de la clase.

## Métodos

public static void **getGrupo**(int ancho,  
int alto)

Establece el grupo al que pertenece un móvil, si a los que usaran imágenes con tamaño de 128x128 o a los que usarán las de 176x200

### Parámetros:

ancho - Ancho de la pantalla del móvil

alto - Alto de la pantalla del móvil

---

public static int **getTamGrupoX()**

Nos muestra el ancho de pantalla que le corresponde al grupo

### Devuelve:

el ancho de pantalla para el grupo

---

public static int **getTamGrupoY()**

Nos muestra el alto de pantalla que le corresponde al grupo

### Devuelve:

el alto de pantalla para el grupo



---

```
public static int[] getFrames(java.lang.String peticion)
```

Este método devolverá la secuencia de frames que forman el objeto pedido.

**Parámetros:**

`peticion` - el sprite que se pide.

**Devuelve:**

secuencia de frames que forman el objeto pedido

---

```
public static int getFrameInicial(java.lang.String peticion)
```

Devuelve el frame inicial de un Sprite

**Parámetros:**

`peticion` - el Sprite del que se quiere sacar el frame inicial

**Devuelve:**

el frame inicial de un Sprite

---

```
public static int getNumFrames(java.lang.String peticion)
```

Devuelve el numero de frames para una petición determinada

**Parámetros:**

`peticion` - Sprite del que se quieren saber los frames

**Devuelve:**

el numero de frames que tiene ese Sprite

---

```
private static java.lang.String getDatosImagen(java.lang.String peticion)
```

Devuelve los datos de la imagen en la que está almacenada la petición o Sprite

**Parámetros:**

`peticion` - Sprite del que se quiere saber la imagen

**Devuelve:**

los datos de la imagen en el que esta el Sprite

---

```
private static int getId(java.lang.String datosImagen)
```

Devuelve el identificador de una imagen

**Parámetros:**

`datosImagen` - datos de la imagen de los que se quiere extraer el identificador

**Devuelve:**

identificador la imagen

---

```
private static java.lang.String getRuta(java.lang.String nombreImagen)
```

Devuelve la ruta de una imagen

**Parámetros:**

`nombreImagen` - datos de la imagen de los que se quiere extraer la ruta

**Devuelve:**

ruta de directorios donde está la imagen

---



---

```
private static int getAncho(java.lang.String nombreImagen)
```

Devuelve el ancho de una cada uno de los frame de una imagen

**Parámetros:**

`nombreImagen` - Datos de la imagen.

**Devuelve:**

EL ancho de cada uno de los frames de una imagen

---

```
private static int getAlto(java.lang.String nombreImagen)
```

Devuelve el alto de una cada uno de los frame de una imagen

**Parámetros:**

`nombreImagen` - Datos de la imagen.

**Devuelve:**

EL alto de cada uno de los frames de una imagen

---

```
public static void cargarImagen(java.lang.String nomSprite)
```

Carga los datos mas importante de una imagen en la que está el Sprite solicitado

**Parámetros:**

`nomSprite` - nombre del sprite cuya imagen el la que está contenido se quiere cargar

---

```
private static boolean creada(java.lang.String nombre)
```

Indica si una imagen (que se necesita cargar para un sprite) ya esta creada. Esto puede pasar porque puede ser que el sprite que se cargo anteriormente a este, estaba contenido en la misma imagen.

**Parámetros:**

`nombre` - Nombre de la imagen que se quiere cargar

**Devuelve:**

si esta creada o no

---

```
private static javax.microedition.lcdui.Image createImage(java.lang.String nombre)
```

Método que devuelve un objeto Image a partir de una imagen PNG, la cual tiene que estar en la carpeta res.

**Parámetros:**

`nombre` - Nombre de la imagen en la carpeta res

**Devuelve:**

Image con la imagen ya creada.

---

```
public static int getSpaceItem()
```

Devuelve el espacio reservado para el nombre de los Items en la pantalla de juego, teniendo en cuenta si el móvil es de un grupo u otro

**Devuelve:**

El espacio reservado para el nombre de los Items en la pantalla de juego.



```
public static void actualizarApariencia(int activa)
```

Actualiza a la apariencia activa del personaje principal.

### Parámetros:

activa - La apariencia activa del personaje principal

```
public static int getIdResultado(java.lang.String resultado)
```

Devuelve el identificador que corresponde a un determina resultado por ejemplo  
a Salir le corresponde reSalir

### Parámetros:

resultado - String con el nombre del resultado.

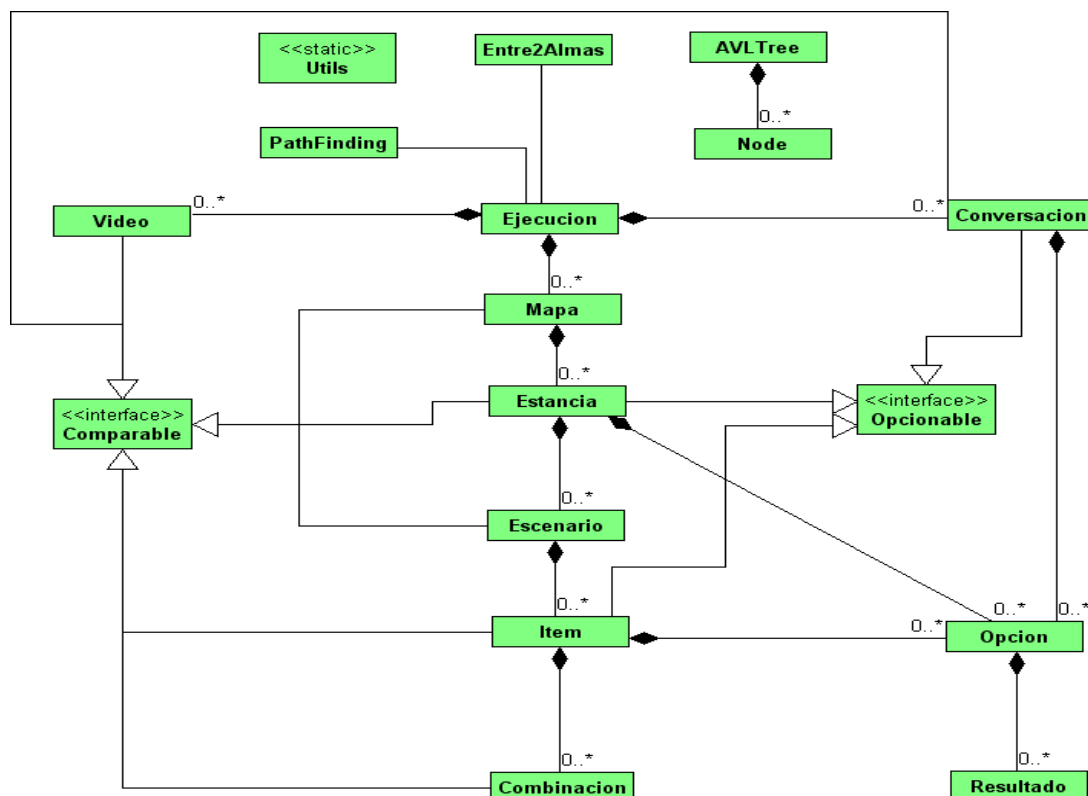
**Devuelve:**

Identificador que corresponde a una determinado resultado

## 5.2. Diseño de la estructura

### 5.2.1.Diagrama de clases

La mayoría de las conexiones entre clases, se establecen como composiciones, y entiendo por composición cuando, un objeto X esta compuesto por una objeto Y, el cual solo puede pertenecer a ese objeto X. Al destruir el objeto X se destruye el objeto Y.



**Imagen 31: Diagrama de clases de la estructura**



## 5.2.2.Descripción detallada de las clases

### 5.2.2.1. Entre2Almas

Clase principal y que será la encargada de poner en marcha la clase Ejecucion. Explicada paso a paso en el punto 4.1.2.2

### 5.2.2.2. Utils

En esta clase estática se realizarán tareas repetitivas como createImage, se guardarán los identificadores de las imágenes del menú, algunas variables importantes como la que indica la posición inicial a la que empezar a escribir, también tendrá variables que indican si se está jugando, guardando o cargando una partida en ese momento, etc. Explicada paso a paso en el punto 4.1.2.7

### 5.2.2.3. AVLTree

Implementa un árbol AVL.

El código que usé, y que se puede ver en el apéndice A, lo hallé en el libro “Introduction to Algorithm (2nd edition), McGraw-Hill (2001), ISBN: 0070131511”.

AVLTree
+ root: Node + almacenados: Vector
+ AVLTree() + search(k:int):Node + search(node:Node,k:int):Node + treeMinimum(n:Node):Node + successor(n:Node):Node - iterateUpAfterInsert(iter:Node):void - iterateUpAfterDelete(iter:Node):void + insert(data:Comparable):Node # treeInsert(z:Node):void + deleteKey(k:int):void + deleteAllElements():void + delete(node:Node):void + inorden(n:Node):void - addVector(n:Node):void + next():Node

Imagen 32: Clase AVLTree

#### Atributos

public Node **root**  
Root of the AVL tree.





---

private java.util.Vector **almacenados**

## Constructor

public **AVLTree()**

The constructor creating a binary search tree with just a `null`, which is the root.

## Métodos

public Node **search**(int k)

Searches the tree for a node with a given key. If such a node exists, prints the value of that node.

**Parámetros:**

`k` - The key being searched for.

**Devuelve:**

A reference to a `Node` object with key `k` if such a node exists. An exception `NoSuchElementException` is thrown if no node exists.

---

public Node **search**(Node node,  
int k)

Searches the subtree rooted at a given node for a node with a given key.

**Parámetros:**

`node` - Root of the subtree.

`k` - The key being searched for.

**Devuelve:**

A reference to a `Node` object with key `k` if such a node exists. An exception `NoSuchElementException` is thrown if no node exists.

---

protected Node **treeMinimum**(Node x)

Returns the node with the minimum key in the subtree rooted at a node.

**Parámetros:**

`x` - Root of the subtree.

**Devuelve:**

A `Node` object with the minimum key in the tree, or the sentinel `nil` if the tree is empty.

---

public Node **successor**(Node node)

Returns the successor of a given node in an inorder walk of the tree.

**Parámetros:**

`node` - The node whose successor is returned.

**Devuelve:**

If `node` has a successor, it is returned. Otherwise, return the sentinel `null`.



---

private void **iterateUpAfterInsert**(Node iter)

Iterates up a tree updating balance factors and performing rotations after an insert. The method returns when all the necessary updates have been performed.

---

private void **iterateUpAfterDelete**(Node iter)

Iterates up a tree updating balance factors and performing rotations after a delete. The method returns when all the necessary updates have been performed.

---

public Node **insert**(Comparable data)

Inserts a data item into the tree, creating a new node for this data.

**Parámetros:**

data - Data to be inserted into the tree.

**Devuelve:**

A reference to the `Node` object created.

---

protected void **treeInsert**(Node z)

Inserts a node into the tree.

**Parámetros:**

z - The node to insert.

---

public void **deleteKey**(int k)

Removes a node with the given key from the tree.

**Parámetros:**

k - The key to be removed.

---

public void **deleteAllElements**()

Removes all nodes from the tree.

---

public void **delete**(Node node)

Removes a node from the tree.

**Parámetros:**

node - The node to be removed. do nothing when deleting a null node `null`.

---

public void **inorden**(Node n)

---

private void **addVector**(Node n)

---

public Node **next**()

---



#### 5.2.2.4. Combinacion

Un objeto de la clase Item, podrá combinarse con otros objetos, estas posibles combinaciones son objetos de la clase Combinación.

Combinacion
idItem: int activa: bool consecuencias: Vector conseActiva: int
+ Combinacion(idItem:int,activa:bool,consecuencias:Vector) + getKey():int

Imagen 33: Clase Combinacion

#### Atributos

int **idItem**

Identificador del Item con el que ocurre algo al intentar combinar.

boolean **activa**

Indica si esta combinación está activa para un determinado Objeto

java.util.Vector **consecuencias**

Consecuencias de una determinada combinación. En este vector se almacenarán vectores de objetos de la clase Resultado. Un Objeto puede tener consecuencias distintas en momentos distintos, entonces aquí almaceno todas las consecuencias que puede tener y para cada consecuencia almacenamos el Vector de Resultados, en el orden en que ocurrirán. Una vez que una consecuencia ya no se necesita, pues se borra del array

int **conseActiva**

Indica la consecuencia que está activa, de entre las disponibles.

#### Constructor

```
public Combinacion(int idItem,  
                    boolean activa,  
                    java.util.Vector consecuencias)
```

Constructor de la clase Combinación

#### Parámetros:

*idItem* - Identificador del Item con el que ocurre algo al intentar combinar  
*activa* - indica si esta combinación está activa para un determinado Objeto  
*consecuencias* - Consecuencias de una determinada opción

**Métodos**

public int **getKey()**

Devuelve la clave del comparable

**Devuelve:**

La clave del comparable identificador del comparable.

**5.2.2.5. Comparable**

Interfaz. Todo aquel que lo implemente significará que es comparable, es decir, que se puede comparar con objetos de otras clases que también implementen la interfaz Comparable.

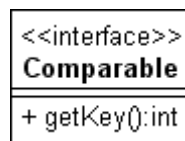


Imagen 34: Clase Comparable

**Métodos**

int **getKey()**

Devuelve la clave del comparable

**Devuelve:**

La clave del comparable identificador del comparable.

**5.2.2.6. Conversacion**

Una conversación se dará entre nuestro personaje y un objeto Item. Implementa las interfaces Opcionable y Comparable.

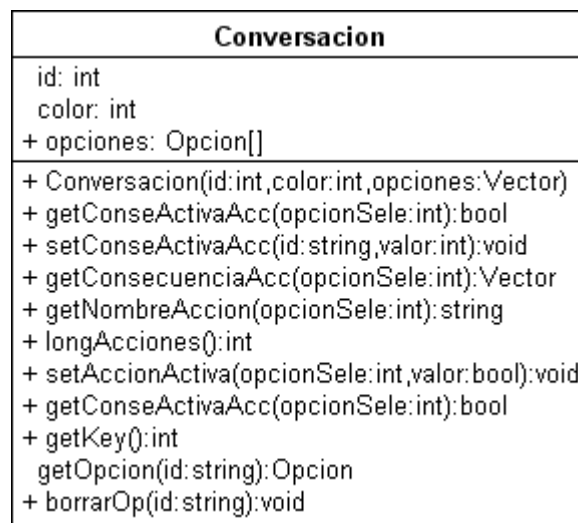


Imagen 35: Clas Conversacion



## Atributos

int **id**

identificador de la conversación

int **color**

Color con el que se identificará el personaje con el que habla Oscar

public Opcion[] **opciones**

Array que indica que las opciones disponibles.

## Constructor

```
public Conversacion(int id,  
                    int color,  
                    java.util.Vector opciones)  
Constructor de la clase Conversación
```

### Parámetros:

**id** - identificador de la conversación

**color** - Color con el que se identificará el personaje con el que habla Oscar

**opciones** - indica que las opciones disponibles.

## Métodos

```
public boolean getConseActivaAcc(int opcionSele)  
Indica si una determinada acción tiene o no alguna consecuencia
```

### Parámetros:

**opcionSele** - La opción seleccionada en el juego

### Devuelve:

boolean indicando si esa opción o acción tiene alguna consecuencia si es true es que no hay consecuencias, y viceversa.

```
public void setConseActivaAcc(java.lang.String id,  
                             int valor)
```

Establece la consecuencia activa para una determinada acción.

### Parámetros:

**id** - Identificador de la opción a modificar.

**valor** - el nuevo valor de conseActiva

```
public java.util.Vector getConsecuenciaAcc(int opcionSele)
```

Devuelve la consecuencia para una determinada acción

### Parámetros:

**opcionSele** - La opción seleccionada en el juego

### Devuelve:

Vector con la consecuencia activa para una determinada acción



---

```
public java.lang.String getNombreAccion(int opcionSele)
```

El nombre de una determinada acción.

**Parámetros:**

`opcionSele` - La opción seleccionada en el juego

**Devuelve:**

nombre de la acción.

---

```
public int longAcciones()
```

Devuelve el numero de acciones disponibles para un Seleccionable

**Devuelve:**

el numero de acciones disponibles para un Seleccionable

---

```
public void setAccionActiva(int opcionSele,  
                             boolean valor)
```

Establece si una acción esta activa o no

**Parámetros:**

`opcionSele` - La opción seleccionada en el juego

`valor` - boolean indicando si está activa o no.

---

```
public boolean getAccionActiva(int opcionSele)
```

Indica si una determinada acción esta activa o no.

**Parámetros:**

`opcionSele` - La opción seleccionada en el juego

**Devuelve:**

boolean indicando si la opción está o no activa.

---

```
public int getKey()
```

Devuelve la clave del comparable

**Devuelve:**

La clave del comparable identificador del comparable.

---

```
public Opcion getOpcion(java.lang.String id)
```

Devuelve la opción cuyo id coincide con el que se pasa como parámetro

**Parámetros:**

`id` - String con el identificador de la opción

**Devuelve:**

La opción cuyo id coincide con el de la opción.

---

```
public void borrarOp(java.lang.String id)
```

Borra una opción de entra las disponibles

**Parámetros:**

`id` - Identificador de la opción.

---



### 5.2.2.7. Ejecucion

Lleva toda la dirección de lo que es el núcleo del juego, aquí se decide cuando mirar colisiones, cuando ejecutar acciones, como buscar una Ruta, cuales son las funciones de cada tecla en cada momento, etc.

Ejecucion	
<ul style="list-style-type: none"><li>- midlet: Entre2Almas</li><li>t: Thread</li><li>- LM: LayerManager</li><li>parser: KXmlParser</li><li>buscaRuta: PathFinding</li><li>- caminando: bool</li><li>- direccion: int</li><li>- pasoActual: int</li><li>mapas: AVLTree</li><li>+ mapaActual: int</li><li>+ anchoCelda: int</li><li>+ altoCelda: int</li><li>+ escenActual: Escenario</li><li>- ultimoEscenario: StringBuffer</li><li>- sprite: Sprite</li><li>- oscar: Sprite</li><li>- puntero: Sprite</li><li>- escenDetras: Sprite</li><li>- escenCaminable: Sprite</li><li>- itemsInventario: Vector</li><li>- itemsInventario: bool</li><li>- invSelec: int</li><li>- seleccionado: Opcionable</li><li>- mostrarOpciones: bool</li><li>- colisionado: bool</li><li>&lt;&lt;static&gt;&gt; - anchoInventario: int</li><li>&lt;&lt;static&gt;&gt; - altoInventario: int</li><li>&lt;&lt;static&gt;&gt; - anchoOpciones: int</li><li>&lt;&lt;static&gt;&gt; - altoOpciones: int</li><li>&lt;&lt;static&gt;&gt; - anchoMensaje: int</li><li>&lt;&lt;static&gt;&gt; - altoMensaje: int</li><li>- opcionSele: int</li><li>- mostrarMensaje: bool</li><li>- videosFase: AVLTree</li><li>- videoSiguiente: int</li><li>- converFase: AVLTree</li><li>- converSelec: int</li><li>- participante: Item</li><li>- longIdOp: int</li><li>- inildOp: StringBuffer</li><li>- habParti: bool</li><li>- teclaPulsada: bool</li><li>+ nFase: int</li><li>- nomFase: StringBuffer</li><li>+ parteJuego: int</li><li>- video: int {final}</li><li>- juego: int {final}</li><li>- inventario: int {final}</li><li>- conversacion: int {final}</li><li>- mapa: int {final}</li><li>+ actual: int</li><li>- tmpAcumulado: int</li><li>- cargando: bool</li><li>resultados: Vector</li><li>- ocupado: bool</li><li>- redibujar: bool</li><li>- irMenu: bool</li><li>- posXMenu: int</li><li>- posYMenu: int</li><li>- anchoMenu: int</li><li>- altoMenu: int</li><li>- combinando: bool</li><li>- intentaCombinar: bool</li><li>- combinado: Item</li><li>salir: bool</li><li>letra: int</li><li>disponible: int</li><li>numLetras: int</li></ul>	<ul style="list-style-type: none"><li>+ Ejecucion(midlet:Entre2Almas)</li><li>+ start():void</li><li>+ run():void</li><li>+ keyPressed(keyCode:int):void</li><li>- input():void</li><li>- drawScreen(g:Graphics):void</li><li>- cargarFase(g:Graphics,fase:int):void</li><li>- cargarVideosFase():void</li><li>- cargarTextoVideo():Vector</li><li>- cargarConversacionFase():void</li><li>- cargarMapaFase():void</li><li>- cargarEstanciaMapa():Estancia</li><li>- cargarEscenarioEstancia():Escenario</li><li>- cargarItemEscenario():Item</li><li>- cargarCombinacionItem():Combinacion</li><li>- cargarOpcionObjeto():Opcion</li><li>- cargarConsecuenciaAccion():Vector</li><li>+ getDatosImagen():StringBuffer</li><li>- cargar(g:Graphics,pantalla:int):void</li><li>+ getVideo(g:Graphics):void</li><li>+ getJuego(g:Graphics):void</li><li>+ getInventario(g:Graphics):void</li><li>+ getConversacion(g:Graphics):void</li><li>+ getMapa(g:Graphics):void</li><li>+ getAccMenu(g:Graphics):void</li><li>- cargarEscenario():void</li><li>- cargarAreaCaminable():void</li><li>- descargarEscenario():void</li><li>- cargarMapa():void</li><li>- descargarMapa():void</li><li>- descargarConversacion():void</li><li>- colisionPunItem():bool</li><li>- colisionPunEstan():bool</li><li>- animarItems():void</li><li>- opcionesIO():void</li><li>- conseCombinacion():void</li><li>- ejecutarResultado():void</li><li>+ getOpciones(g:Graphics):void</li><li>+ getMensaje(g:Graphics):void</li><li>+ ajustarVariables():void</li><li>- getParte(parte:string):int</li><li>- getColorHex(color:string):int</li><li>+ mostrarMensaje(mensaje:string,finMensaje:string,g:Graphics):void</li><li>- getLineatexto(g:Graphics,texto:string):void</li><li>- moverBoca():void</li><li>- cerrarBoca():void</li><li>- buscarRuta():bool</li><li>+ getCeldaX(pixel:int):int</li><li>+ getCeldaY(pixel:int):int</li><li>+ getPixelX(celda:int):int</li><li>+ getPixelY(celda:int):int</li><li>- buscaEstancia(id:int):Estancia</li><li>- buscaltem(id:int):Item</li><li>- buscaltemEsc(auxEscenario:Escenario,id:int,comprueba:bool):Item</li><li>- buscaltemEst(auxEstancia:Estancia,id:int):Item</li><li>- buscaltemM(auxMapa:Mapa,id:int):Item</li><li>- buscaltemTM(id:int):Item</li><li>- avanzarPaso():void</li><li>+ puedeGuardar():bool</li><li>+ guardarPartida(g:Graphics):void</li><li>- insertarRegistro(rs:ByteArrayOutputStream,rs:RecordStore):void</li><li>- salir():void</li></ul>

Imagen 36: Clase Ejecucion



## Atributos

private Entre2Almas **midlet**

Referencia al midlet.

java.lang.Thread **t**

Hilo de ejecución del juego

private javax.microedition.lcdui.game.LayerManager **LM**

Será el controlador de las distintas capas del juego.

org.kxml2.io.KXmlParser **parser**

Parser del XML.

PathFinding **buscaRuta**

El pathFinding A\*

private boolean **caminando**

Estará a true cuando Oscar está caminando.

private int **direccion**

Si es 0, es que va de frente, 1 va de espaldas, 2 derecha, 3 izquierda.

private int **pasoActual**

Indice del array Path que contiene la posición del siguiente paso a ejecutar por oscar.

AVLTree **mapas**

Son los distintos mapas del juego

public int **mapaActual**

Indica el identificador del mapa actual;

private int **anchoCelda**

El ancho y el alto en píxeles que tiene cada celda en las que esta dividido el mapa

private int **altoCelda**

El ancho y el alto en píxeles que tiene cada celda en las que esta dividido el mapa





---

public Escenario **escenActual**

Escenario que actualmente está cargado en memoria.

---

private java.lang.StringBuffer **ultimoEscenario**

Guarda el nombre del último escenario cargado

---

private javax.microedition.lcdui.game.Sprite **sprite**

Sprite general

---

private javax.microedition.lcdui.game.Sprite **oscar**

Sprite de Oscar

---

private javax.microedition.lcdui.game.Sprite **puntero**

Sprite del puntero.

---

private javax.microedition.lcdui.game.Sprite **escenDetras**

Sprite formado por la imagen correspondiente a la parte de atrás del escenario actual.

---

private javax.microedition.lcdui.game.Sprite **escenCaminable**

Sprite formado por la imagen correspondiente a la parte caminable del escenario actual.

---

private java.util.Vector **itemsInventario**

Vector que almacena los Items que tenemos en el inventario

---

private boolean **mostrarInventario**

Indica si hay que mostrar o no, el inventario

---

private int **invSelec**

Indica el objeto que está mostrándose en ese momento en el inventario.

---

private Opcionable **seleccionado**

Indica el Seleccionable que esta siendo seleccionado en la pantalla de juego

---

private boolean **mostrarOpciones**

Indica si hay que mostrar o no, las opciones de un objeto

---



---

private boolean **colisionado**

Indica si el puntero ha colisionado con algo.

---

private static int **anchoInventario**

Marca el espacio de ancho que se deja para el inventario en la pantalla de juego

---

private static int **altoInventario**

Marca el espacio de alto que se deja para el inventario en la pantalla de juego

---

private static int **anchoOpciones**

Ancho del panel donde se van a mostrar las opciones

---

private static int **altoOpciones**

Alto del panel donde se van a mostrar las opciones

---

private static int **anchoMensaje**

Ancho del panel donde se van a mostrar los mensajes

---

private static int **altoMensaje**

Alto del panel donde se van a mostrar los mensajes

---

private int **opcionSele**

indica la opción que está seleccionada ahora mismo para un Item.

---

private boolean **mostrarMensaje**

Indica si hay que mostrar o un mensaje.

---

private AVLTree **videosFase**

Vector con los videos correspondientes a una fase

---

private int **videoSiguiente**

Identificador del siguiente video a mostrar.

---

private AVLTree **converFase**

AVL con las conversaciones correspondientes a una fase

---

private int **converSelec**

Identificador de la conversación a mostrar

---



---

private Item **participante**

Participante en la conversación.

---

private int **longIdOp**

Indica la longitud que tendrá el id de la opción a mostrar en la conversación

---

private java.lang.StringBuffer **iniIdOp**

Indica las letras que tendrá que tener en su inicio el id de la opción, para que entonces esta sea mostrada. En caso de que iniIdOp no contenga ninguna letra, se cargarán todas las opciones que tengan un id que coincida con longIdOp

---

private boolean **habParti**

Indica si el que esta hablando es el participante (no Oscar), en cuyo caso estará a true.

---

private boolean **teclaPulsada**

Indica si una tecla está pulsada en este momento.

---

public int **nFase**

Numero de fase en la que nos encontramos

---

private java.lang.StringBuffer **nomFase**

Nombre de la fase en la que nos encontramos.

---

public int **parteJuego**

Parte o pantalla de juego en la que nos encontramos. Las posibilidades son: Video, Juego, inventario, conversación, minijuego1, minijuego2 o minijuego 3

---

private final int **video**

Variables estáticas que representan los distintos tipos de pantalla que existen

---

private final int **juego**

---

private final int **inventario**

---

private final int **conversacion**

---

private final int **mapa**

---



---

public int **actual**

Indica que pantalla estamos viendo actualmente.

---

private int **tmpAcumulado**

Tiempo de delays acumulado. En cada bucle del hilo se hace un delay por ejemplo de 20 milsg, pues en tmpAcumulado vamos sumando esto.

---

private boolean **cargando**

Indica que se está cargando una fase

---

java.util.Vector **resultados**

Vector que guardará los resultados a ejecutar a una determinada acción.

---

private boolean **ocupado**

Indica cuando el programa está ocupado ejecutando una acción.

---

private boolean **redibujar**

Indica cuando es necesario redibujar la pantalla.

---

private boolean **irMenu**

Indica que queremos ir al menú.

---

private int **posXMenu**

Posiciones X e Y en las que aparece el enlace directo al menú

---

private int **posYMenu**

Posiciones X e Y en las que aparece el enlace directo al menú

---

private int **anchoMenu**

Ancho y alto del acceso directo al menú.

---

private int **altoMenu**

Ancho y alto del acceso directo al menú.

---

private boolean **combinando**

Indica que un ítem está preparado en este momento para ser combinado con otros ítems.



---

private boolean **intentaCombinar**

Indica que el ítem que está preparado en este momento para ser combinado, ya está intentando ser combinado con otro ítem.

---

private Item **combinado**

Ítem que está siendo combinando

---

boolean **salir**

Nos indicara cuando salir del bucle de ejecución, y por tanto finalizar el hilo

---

int **letra**

Es la letra inicial desde la que hay que empezar mostrar el mensaje en la línea de texto. Por defecto su valor será -6. El porque es para que tarden más en empezar a moverse las letras.

---

int **disponible**

Espacio disponible en píxeles en la línea de texto

---

int **numLetras**

Numero de letras que caben en la línea de texto

---

## Constructor

public **Ejecucion**(Entre2Almas midlet)

Constructor de la clase Ejecución

**Parámetros:**

midlet - El midlet del juego. Entre2Almas

## Métodos

public void **start**()

---

public void **run**()

---



---

```
public void keyPressed(int keyCode)
```

KeyPressed es llamado cuando una tecla es pulsada, pero solo lo usaremos cuando sea una non-game Key. Las teclas de números (1,2,3,...), el \* y la #, son game Keys y tienen positivos KeyCodes. Así las non-game keys tienen keyCodes negativos. Cuando se pulse una de estas teclas especiales, iremos al menú.

**Parámetros:**

keyCode - keyCode de la tecla pulsada.

---

```
private void input()
```

Método para controlar las entradas de teclado

**Throws:**

java.lang.InterruptedException

---

```
private void drawScreen(javax.microedition.lcdui.Graphics g)
```

**Throws:**

org.xmlpull.v1.XmlPullParserException  
java.io.IOException

---

```
private void cargarFase(javax.microedition.lcdui.Graphics g,  
int fase)
```

Método encargado de cargar el estado del juego al inicio de una fase. Además llama a drawScreen para que pinte la imagen de cargando.

**Parámetros:**

g - objeto Graphics necesario para llamar a drawScreen y que de esta manera se cargue la imagen de cargando

fase - a cargar

**Throws:**

java.io.IOException  
org.xmlpull.v1.XmlPullParserException

---

```
private void cargarVideosFase()
```

Carga los videos de una determinada Fase.

**Throws:**

org.xmlpull.v1.XmlPullParserException  
java.io.IOException

---

```
private java.util.Vector cargarTextoVideo()
```

Carga el texto de un video

**Devuelve:**

vector con los textos

**Throws:**

org.xmlpull.v1.XmlPullParserException  
java.io.IOException

---



---

private void **cargarConversacionFase()**

Carga las conversaciones de una determinada Fase.

**Throws:**

org.xmlpull.v1.XmlPullParserException  
java.io.IOException

---

private void **cargarMapaFase()**

Carga las estancias de una fase

**Throws:**

org.xmlpull.v1.XmlPullParserException  
java.io.IOException

---

private Estancia **cargarEstanciaMapa()**

Carga las estancias de una fase

**Devuelve:**

Estancia que forma parte de un Mapa

**Throws:**

org.xmlpull.v1.XmlPullParserException  
java.io.IOException

---

private Escenario **cargarEscenarioEstancia()**

Carga un escenario de una determinada Estancia

**Devuelve:**

Escenario que forma parte de una Estancia

**Throws:**

org.xmlpull.v1.XmlPullParserException  
java.io.IOException

---

private Item **cargarItemEscenario()**

Carga un escenario de una determinada Estancia

**Devuelve:**

Escenario que forma parte de una Estancia

**Throws:**

org.xmlpull.v1.XmlPullParserException  
java.io.IOException

---

private Combinacion **cargarCombinacionItem()**

Carga una combinación de un determinado Item con otro Item

**Devuelve:**

Combinacion de un determinado Item con otro Item

**Throws:**

org.xmlpull.v1.XmlPullParserException  
java.io.IOException



---

```
private Opcion cargarOpcionObjeto()
```

Carga la opción de un determinado Objeto

**Devuelve:**

opción que forma parte de un Objeto

**Throws:**

```
org.xmlpull.v1.XmlPullParserException  
java.io.IOException
```

---

```
private java.util.Vector cargarConsecuenciaAccion()
```

Cargar las consecuencias de una acción

**Devuelve:**

Vector con un conjunto de resultados que formarán una consecuencia

**Throws:**

```
org.xmlpull.v1.XmlPullParserException  
java.io.IOException
```

---

```
public java.lang.StringBuffer getDatosImagen()
```

Recupera, a partir de un archivo XML, los datos descriptivos de una imagen

**Throws:**

```
java.io.IOException - Excepción de entrada/salida  
org.xmlpull.v1.XmlPullParser - Exception Excepción en el XML  
org.xmlpull.v1.XmlPullParserException
```

---

```
private void cargar(javax.microedition.lcdui.Graphics g,  
int pantalla)
```

Decide que y cuando cargar en cada momento

**Parámetros:**

```
g - Objeto Graphics  
pantalla - pantalla que se desea cargar
```

---

```
public void getVideo(javax.microedition.lcdui.Graphics g)
```

Dibuja la pantalla del video

**Parámetros:**

```
g - objeto Graphics
```

---

```
public void getJuego(javax.microedition.lcdui.Graphics g)
```

Dibuja la pantalla de juego

**Parámetros:**

```
g - objeto Graphics
```

---

```
public void getInventario(javax.microedition.lcdui.Graphics g)
```

Dibuja el inventario

**Parámetros:**

```
g - objeto Graphics
```

---





---

```
public void getConversacion(javax.microedition.lcdui.Graphics g)
```

Dibuja la pantalla de la conversación

**Parámetros:**

g - objeto Graphics

---

```
public void getMapa(javax.microedition.lcdui.Graphics g)
```

Dibuja la pantalla del mapa

**Parámetros:**

g - objeto Graphics

---

```
public void getAccMenu(javax.microedition.lcdui.Graphics g)
```

Dibuja el acceso directo al menú.

**Parámetros:**

g - objeto Graphics

---

```
private void cargarEscenario()
```

Realiza la carga del escenario de juego.

---

```
private void cargarAreaCaminable()
```

Carga el área caminable del escenario actual.

---

```
private void descargarEscenario()
```

Realiza la descarga del escenario de juego.

---

```
private void cargarMapa()
```

Realiza la carga del mapa de juego.

---

```
private void descargarMapa()
```

Realiza la descarga del mapa de juego.

---

```
private void descargarConversacion()
```

Realiza la descarga de la conversación

---

```
private boolean colisionPunItem()
```

Comprueba si hay colisión entre el puntero y algún Item del escenario actual, también se incluye a Oscar Solo hace falta que colisione con uno, para que devuelva verdadero

**Devuelve:**

true si hay colisión entre el puntero y un Item del escenario actual



---

private boolean **colisionPunEstan()**

Comprueba si hay colisión entre el puntero y alguna Estancia del mapa. Solo hace falta que colisione con una, para que devuelva verdadero

**Devuelve:**

true si hay colisión entre el puntero y una Estancia del mapa

---

private void **animarItems()**

Para aquellos Items que están establecidos como animados, este método comprueba si ha llegado su hora de cambiar de frame.

---

private void **opcionesIO()**

método que salta como respuesta a la pulsación de una acción, de entre las disponibles para un objeto.

---

private void **conseCombinacion()**

método que salta como respuesta al intento de combinar dos Items

---

private void **ejecutarResultado()**

Método encargado de dirigir la ejecución de cada uno de los resultados de una accion

---

public void **getOpciones**(javax.microedition.lcdui.Graphics g)

Muestra las opciones del objeto seleccionado en la pantalla de juego

**Parámetros:**

g - objeto Graphics

---

public void **getMensaje**(javax.microedition.lcdui.Graphics g)

Muestra Un mensaje en la pantalla de juego

**Parámetros:**

g - objeto Graphics

---

public void **ajustarVariables()**

Reajusta algunas variables como son el ancho y alto de la pantalla. Carga el numero de letras que caben en el ancho de la pantalla, y el numero de líneas que caben en el alto de la pantalla

---

private int **getParte**(java.lang.String parte)

Devuelve el entero que identifica a una parte del juego

**Parámetros:**

parte - Parte del juego de la que quiero saber el identificador

**Devuelve:**

el identificador de la parte del juego

---



---

```
private int getColorHex(java.lang.String color)
```

Haya el valor hexadecimal de un determinado color.

**Parámetros:**

color - Color del que queremos saber su valor hexadecimal

**Devuelve:**

el valor en hexadecimal de un determinado color

---

```
public static void mostrarMensaje(java.lang.String mensaje,  
    java.lang.String finMensaje,  
    javax.microedition.lcdui.Graphics g)
```

Imprime un mensaje por pantalla

**Parámetros:**

mensaje - Mensaje a imprimir

finMensaje - Cadena de texto a poner al final del mensaje para indicar que el mensaje se ha terminado

g - Objeto Graphics.

---

```
private void getLineatexto(javax.microedition.lcdui.Graphics g,  
    java.lang.String texto)
```

Imprime un mensaje en la línea de textos

**Parámetros:**

g - objeto Graphics

texto - el mensaje a mostrar.

---

```
private void moverBoca()
```

Mueve la boca del personaje que esta hablando

---

```
private void cerrarBoca()
```

Cierra la boca del personaje que esta hablando

---

```
private boolean buscarRuta()
```

Se encarga de buscar la ruta adecuada, para llevar a nuestro persona desde su posición actual, hasta el sitio de la pantalla en el que hayamos pulsado.

**Devuelve:**

Boolean indicando si existe ruta hasta una determinada celda o no.

---

```
private int getCeldaX(int pixel)
```

Nos devuelve la posición X de la celda en la que esta situada un determinado píxel

**Parámetros:**

pixel - La posición en píxeles de la que se quiere sabe a que celda corresponde

**Devuelve:**

la posición X de la celda a la que está apuntando el puntero.

---



---

private int **getCeldaY**(int pixel)

Nos devuelve la posición Y de la celda en la que esta situada un determinado píxel

**Parámetros:**

pixel - La posición en píxeles de la que se quiere sabe a que celda corresponde

**Devuelve:**

la posición Y de la celda a la que está apuntando el puntero.

---

private int **getPixelX**(int celda)

Nos devuelve la posición X del píxel en el que esta situado una determinada celda

**Parámetros:**

celda - La celda de la que se quiere sabe a que píxel corresponde

**Devuelve:**

el píxel X de la celda a la que está apuntando el puntero.

---

private int **getPixelY**(int celda)

Nos devuelve la posición Y del píxel en el que esta situado una determinada celda

**Parámetros:**

celda - La celda de la que se quiere sabe a que píxel corresponde

**Devuelve:**

el píxel Y de la celda a la que está apuntando el puntero.

---

private Estancia **buscaEstancia**(int id)

Haya la Estancia cuyo identificador coincide con el parámetro que le pasamos.

**Parámetros:**

id - Identificador de la estancia a buscar

**Devuelve:**

La estancia buscado.

---

private Item **buscaItem**(int id)

Haya el Item cuyo identificador coincide con el parámetro que le pasamos.

**Parámetros:**

id - Identificador del Item a buscar

**Devuelve:**

El Item buscado.



---

```
private Item buscaItemEsc(Escenario auxEscenario,  
    int id,  
    boolean comprueba)
```

Busca un Item en un escenario que se pasa por parámetro, cuyo identificador coincida con el que pasamos por parámetro

**Parámetros:**

auxEscenario - Escenario donde buscar el Item

id - Identificador del Item a buscar.

comprueba - Indica se comprueba el Item seleccionado o no

**Devuelve:**

El item encontrado o null en caso de no encontrarlo.

---

```
private Item buscaItemEst(Estancia auxEstancia,  
    int id)
```

Busca un Item en un estancia que se pasa por parámetro, cuyo identificador coincida con el que pasamos por parámetro

**Parámetros:**

auxEstancia - Estancia donde buscar el Item

id - Identificador del Item a buscar.

**Devuelve:**

El item encontrado o null en caso de no encontrarlo.

---

```
private Item buscaItemM(Mapa auxMapa,  
    int id)
```

Busca un Item en un mapa que se pasa por parámetro, cuyo identificador coincida con el que pasamos por parámetro

**Parámetros:**

auxMapa - Mapa donde buscar el Item

id - Identificador del Item a buscar.

**Devuelve:**

El item encontrado o null en caso de no encontrarlo.

---

```
private Item buscaItemTM(int id)
```

Busca un Item en todos los mapa, cuyo identificador coincida con el que pasamos por parámetro

**Parámetros:**

id - Identificador del Item a buscar.

**Devuelve:**

El item encontrado o null en caso de no encontrarlo.

---

```
private void avanzarPaso()
```

Hace avanzar un paso a Oscar, en su camino hacia el destino.



---

```
public boolean puedeGuardar()
```

Indica, atendiendo al estado actual de la partida, si se puede guardar o no la partida.

**Devuelve:**

Si se puede guardar o no la partida.

---

```
public void guardarPartida(javax.microedition.lcdui.Graphics g)  
    throws org.xmlpull.v1.XmlPullParserException,  
        java.io.IOException
```

Guarda en la memoria del móvil, la partida en curso.

**Parámetros:**

g - objeto Graphics necesario para llamar a drawScreen y que de esta manera se cargue la imagen de cargando

**Throws:**

java.io.IOException  
org.xmlpull.v1.XmlPullParserException

---

```
private void insertarRegistro(java.io.ByteArrayOutputStream baos,  
                               javax.microedition.rms.RecordStore rs)
```

Inserta un registro en el RMS que está actualmente abierto.

**Parámetros:**

baos - Stream donde está la información a guardar

**Throws:**

javax.microedition.rms.RecordStoreNotOpenException  
javax.microedition.rms.InvalidRecordIDException  
javax.microedition.rms.RecordStoreFullException  
javax.microedition.rms.RecordStoreException

---

```
private void salir()
```

Establece a null todos los objetos del GameCanvas



### 5.2.2.8. Escenario

Un objeto Estancia, puede estar formado de objetos Escenario. Implementa la interfaz Comparable.

Escenario
id: int nombre: string posInicialX: int posInicialY: int + representa: Sprite imDelante: int imDetras: int imCaminable: int delante: bool items: AVLTree
+ Escenario(id:int,nombre:string,imagen:string,posInicialX:int,posInicialY:int,items:AVLTree,delante:bool) + getPosInicialX():int + getPosInicialY():int + getKey():int

Imagen 37: Clase Escenario

#### Atributos

int **id**

Identificador del escenario

java.lang.String **nombre**

Nombre del escenario.

int **posInicialX**

Posición inicial X, en la que empieza el personaje principal en el escenario

int **posInicialY**

Posición inicial Y, en la que empieza el personaje principal en el escenario

public javax.microedition.lcdui.game.Sprite **representa**

Imagen que representará al Escenario. La imagen de cada escenario estará formado por 3 frames, el 1º será la representación del escenario que se verá por pantalla (delante del personaje), el 2º es el escenario que se ve por detrás del personaje y el 3º es el área caminable del escenario

int **imDelante**

Indices de, entre las imágenes que forman el sprite representa, las imágenes que corresponden a la parte de atrás, a la de delante y a la caminable.



---

int **imDetras**

---

int **imCaminable**

---

boolean **delante**

Indica si un escenario, tiene alguna parte del escenario, la cual está por delante del personaje En caso de que esté a true, el escenario tendrá 3 frames: - 1º para lo que se ve delante - 2º para lo que se ve detrás - 3º el área caminable En caso contrario solo tendrá 2: - 1º la parte del escenario que se ve detrás. - 2º el área caminable.

---

AVLTree **items**

Items que va a haber en el escenario

## Constructor

```
public Escenario(int id,  
    java.lang.String nombre,  
    java.lang.String imagen,  
    int posInicialX,  
    int posInicialY,  
    AVLTree items,  
    boolean delante)
```

Constructor de Escenario

### Parámetros:

`id` - Identificador del Escenario

`nombre` - Nombre del escenario

`imagen` - identificador de Utils que define la imagen que representa este escenario

`posInicialX` - Posición inicial X, en la que empieza el personaje principal en el escenario

`posInicialY` - Posición inicial Y, en la que empieza el personaje principal en el escenario

`items` - Items que va a haber en el escenario

`delante` - Indica si un escenario, tiene alguna parte del escenario, la cual está por delante del personaje

## Métodos

```
public int getPosInicialX()
```

### Devuelve:

Devuelve el posInicialX.





---

```
public int getPosInicialY()
```

**Devuelve:**

Devuelve el posInicialY.

---

```
public int getKey()
```

Devuelve la clave del comparable

**Devuelve**

La clave del comparable identificador del comparable.

### 5.2.2.9. Estancia

Un objeto Mapa, tendrá objetos Estancia. Implementa la interfaz Comparable y Opcionable.

Estancia
id: int + escenarios: AVLTree nombre: string + actual: int + representa: Sprite + posX: int + acciones: Opcion[] + Estancia(id:int,imagen:string,nombre:string,actual:int,posX:int,posY:int,acciones:Vector,escenarios:AVLTree,visible:bool) + getEscenActual():Escenario + getConseActivaAcc(opcionSele:int):bool + getConseActivaAcc(id:string,valor:int):void + getConsecuenciaAcc(opcionSele:int):Vector + getNombreAccion(opcionSele:int):string + longAcciones(opcionSele:int):int + setAccionActiva(opcionSele:int,valor:bool):void + getKey():int + getOpcion(id:string):Opcion

**Imagen 38: Clase Estancia**

Atributos
-----------

int id

Identificador de la **Estancia**

---

```
public AVLTree escenarios
```

Escenarios disponibles para una determinada estancia

---

```
java.lang.String nombre
```

Nombre de la estancia

---

```
public int actual
```

Indica el id del escenario activo actual.



---

public javax.microedition.lcdui.game.Sprite **representa**  
Imagen que representará a la estancia en el mapa

---

public int **posX**  
Posición X en la que la imagen que representa esta estancia, se colocará en el mapa

---

public int **posY**  
Posición Y en la que la imagen que representa esta estancia, se colocará en el mapa

---

public Opcion[] **acciones**  
Array que indica que opciones está o no disponibles

---

## Constructor

public **Estancia**(int id,  
java.lang.String imagen,  
java.lang.String nombre,  
int actual,  
int posX,  
int posY,  
java.util.Vector acciones,  
AVLTree escenarios,  
boolean visible)

### Parámetros:

id - Identificador de la estancia.  
imagen - identificador de Utils que define la imagen que representa esta estancia  
nombre - Nombre de la estancia  
actual - Escenario activo actualmente  
posX - posición X del sprite de esta estancia en el mapa  
posY - posición Y del sprite de esta estancia en el mapa  
acciones - Array que indica que opciones tendrá o no disponibles el Item  
escenarios - Escenarios que forman parte de la Estancia.  
visible - Indica si la estancia será visible en el mapa

## Métodos

public Escenario **getEscenActual()**  
Devuelve el escenario que actualmente esta activo en esta estancia.

### Devuelve:

el escenario activo para una estancia



---

```
public boolean getConseActivaAcc(int opcionSele)
```

Indica si una determinada acción tiene o no alguna consecuencia

**Parámetros:**

`opcionSele` - La opción seleccionada en el juego

**Devuelve:**

boolean indicando si esa opción o acción tiene alguna consecuencia si es true es que no hay consecuencias, y viceversa.

---

```
public void setConseActivaAcc(java.lang.String id,  
                             int valor)
```

Establece la consecuencia activa para una determinada acción.

**Parámetros:**

`id` - Identificador de la opción a modificar.

`valor` - el nuevo valor de conseActiva

---

```
public java.util.Vector getConsecuenciaAcc(int opcionSele)
```

Devuelve la consecuencia para una determinada acción

**Parámetros:**

`opcionSele` - La opción seleccionada en el juego

**Devuelve:**

Vector con la consecuencia activa para una determinada acción

---

```
public java.lang.String getNombreAccion(int opcionSele)
```

El nombre de una determinada acción.

**Parámetros:**

`opcionSele` - La opción seleccionada en el juego

**Devuelve:**

nombre de la acción.

---

```
public int longAcciones()
```

Devuelve el numero de acciones disponibles para un Seleccionable

**Devuelve:**

El numero de acciones disponibles para un Seleccionable

---

```
public void setAccionActiva(int opcionSele,  
                             boolean valor)
```

Establece si una acción esta activa o no

**Parámetros:**

`opcionSele` - La opción seleccionada en el juego

`valor` - boolean indicando si está activa o no.

---



---

```
public boolean getAccionActiva(int opcionSele)
```

Indica si una determinada acción esta activa o no.

**Parámetros:**

`opcionSele` - La opción seleccionada en el juego

**Devuelve:**

boolean indicando si la opción está o no activa.

---

```
public int getKey()
```

Devuelve la clave del comparable

**Devuelve:**

La clave del comparable identificador del comparable.

---

```
public Opcion getOpcion(java.lang.String id)
```

Devuelve la opción cuyo id coincide con el que se pasa como parámetro

**Parámetros:**

`id` - String con el identificador de la opción

**Devuelve:**

La opción cuyo id coincide con el de la opción.

---

### 5.2.2.10. Item

Un objeto de tipo Item, formará parte de otro objeto de tipo Escenario. Implementa la interfaz Comparable y Opcionable.

Item
<code>nombre: string</code> <code>animado: bool</code> <code>tmpAnima: int</code> <code>imagenEspecial: string</code> <code>+ acciones: Opcion[]</code> <code>+ combinaciones: AVLTree</code> <code>id: int</code> <code>posición: int</code>
<code>+ Item()</code> <code>+ getConseActivaAcc(opcionSele:int):bool</code> <code>+ setConseActivaAcc(id:string,valor:int):void</code> <code>+ getConsecuenciaAcc(opcionSele:int):Vector</code> <code>+ getNombreAccion(opcionSele:int):string</code> <code>+ longAcciones():int</code> <code>+ setAccionActiva(opcionSele:int,valor:bool):void</code> <code>+ getAccionActiva(opcionSele:int):bool</code> <code>+ getConseActivaCom(indice:int):bool</code> <code>+ setConseActivaCom(id:string,valor:int):void</code> <code>+ getConsecuenciaCom(id:int):Vector</code> <code>+ setComActiva(id:string,valor:int):void</code> <code>+ getComActiva(id:int):bool</code> <code>+ getKey():int</code> <code>+ getOpcion(id:string):Opcion</code>

Imagen 39: Clase Item



## Atributos

java.lang.String **nombre**

Nombre del Item, el cual será mostrado cuando el puntero este encima del Item.

---

boolean **animado**

indica si el Item es animado.

---

int **tmpAnima**

Tiempo que debe transcurrir para que un item animado, cambie a su siguiente frame.

---

java.lang.String **imagenEspecial**

Sprite que almacena unos frames especiales del Item, como puede ser la imagen que un objeto tendrá en el inventario, o los frames de los personajes en las conversaciones.

---

public Opcion[] **acciones**

Array que indica que opciones está o no disponibles

---

public AVLTree **combinaciones**

Las posibles combinaciones que este Item tiene con otros items.

---

int **id**

Identificador del Item

---

int **posicion**

Indica si el Item se vera delante o detrás del personaje principal, es decir, que si el personaje y el item están en la misma posición, este atributo indica quien se mostrará delante. Y si el Item se verá delante o detrás del escenario Si posición vale... - 0 --> El item estará por detrás del escenario (Objetos no cogibles) - 1 --> El item estará entre el escenario y el personaje principal - 2 --> El item estará por delante del personaje principal



## Constructor

```
public Item(java.lang.String nombre,  
            java.lang.String image,  
            int posX,  
            int posY,  
            boolean visible,  
            boolean animado,  
            int tmpAnima,  
            java.lang.String imagenEspecial,  
            java.util.Vector acciones,  
            int posicion,  
            int id,  
            boolean combinable,  
            AVLTree combinaciones)
```

Constructor de la clase. Se usa cuando el Item es animado.

### Parámetros:

`nombre` - Nombre del Item.  
`image` - identificador de Utils que define la imagen que representa este Item  
`posX` - posición X del sprite de este Item  
`posY` - posición Y del sprite de este Item  
`visible` - indica si el Sprite es visible  
`animado` - indica si el Item es animado.  
`tmpAnima` - Tiempo que debe transcurrir para que un item animado, cambie a su siguiente frame.  
`imagenEspecial` - identificador de Utils que define el Sprite especial del Item  
`acciones` - Array que indica que opciones tendrá o no disponibles el Item  
`posicion` - Indica como de visible será el Item  
`id` - identificador que tendrá el Item  
`combinable` - Indica si un Item es combinable con otros Items.  
`combinaciones` - Las posibles combinaciones que este Item tiene con otros items.

## Métodos

```
public boolean getConseActivaAcc(int opcionSele)
```

Indica si una determinada acción tiene o no alguna consecuencia

### Parámetros:

`opcionSele` - La opción seleccionada en el juego

### Devuelve:

boolean indicando si esa opción o acción tiene alguna consecuencia si es true es que no hay consecuencias, y viceversa.



---

```
public void setConseActivaAcc(java.lang.String id,  
                             int valor)
```

Establece la consecuencia activa para una determinada acción.

**Parámetros:**

`id` - Identificador de la opción a modificar.  
`valor` - el nuevo valor de `conseActiva`

---

```
public java.util.Vector getConsecuenciaAcc(int opcionSele)
```

Devuelve la consecuencia para una determinada acción

**Parámetros:**

`opcionSele` - La opción seleccionada en el juego

**Devuelve:**

Vector con la consecuencia activa para una determinada acción

---

```
public java.lang.String getNombreAccion(int opcionSele)
```

El nombre de una determinada acción.

**Parámetros:**

`opcionSele` - La opción seleccionada en el juego

**Returns:**

nombre de la acción.

---

```
public int longAcciones()
```

Devuelve el numero de acciones disponibles para un Seleccionable

**Devuelve:**

el numero de acciones disponibles para un Seleccionable

---

```
public void setAccionActiva(int opcionSele,  
                             boolean valor)
```

Establece si una acción esta activa o no

**Parámetros:**

`opcionSele` - La opción seleccionada en el juego  
`valor` - boolean indicando si está activa o no.

---

```
public boolean getAccionActiva(int opcionSele)
```

Indica si una determinada acción esta activa o no.

**Parámetros:**

`opcionSele` - La opción seleccionada en el juego

**Devuelve:**

boolean indicando si la opción está o no activa.



---

```
public boolean getConseActivaCom(int indice)
```

Indica si una determinada combinación tiene o no alguna consecuencia

**Parámetros:**

`indice` - Índice de la combinación seleccionada en el juego

**Devuelve:**

boolean indicando si esa combinación tiene alguna consecuencia si es true es que no hay consecuencias, y viceversa.

---

```
public void setConseActivaCom(int id,  
                             int valor)
```

Establece la consecuencia activa para una determinada combinación.

**Parámetros:**

`id` - Id de la combinación seleccionada en el juego

`valor` - el nuevo valor de conseActiva

---

```
public java.util.Vector getConsecuenciaCom(int id)
```

Devuelve la consecuencia para una determinada combinación

**Parámetros:**

`id` - id de la combinación seleccionada en el juego

**Devuelve:**

Vector con la consecuencia activa para una determinada combinación

---

```
public void setComActiva(int id,  
                        boolean valor)
```

Establece si una combinación esta activa o no

**Parámetros:**

`id` - Id de la combinación seleccionada en el juego

`valor` - boolean indicando si está activa o no.

---

```
public boolean getComActiva(int id)
```

Indica si una determinada combinación esta activa o no.

**Parámetros:**

`id` - Id de la combinación seleccionada en el juego

**Devuelve:**

boolean indicando si la combinación está o no activa.

---

```
public int getKey()
```

Devuelve la clave del comparable

**Devuelve:**

La clave del comparable identificador del comparable.

---





```
public Opcion getOpcion(java.lang.String id)
```

Devuelve la opción cuyo id coincide con el que se pasa como parámetro

**Parámetros:**

id - String con el identificador de la opción

**Devuelve:**

La opción cuyo id coincide con el de la opción.

### 5.2.2.11. Mapa

El juego tendrá al menos un objeto de tipo Mapa. Implementa la interfaz Comparable.

Mapa
id: int + estancias: AVLTree + activa: int representa: Sprite
+ Mapa(id:int,representa:string,estancias:AVLTree,activa:int) + getEstanActual():Estancia + getEscenActual():Escenario + getKey():int

Imagen 40: Clase Epigrafe

#### Atributos

int **id**

Identificador del Mapa

```
public AVLTree estancias
```

Arbol AVL con las estancias activas del mapa

```
public int activa
```

Variable que indica el id de la estancia está activa

```
javax.microedition.lcdui.game.Sprite representa
```

Imagen de fondo del mapa



## Constructor

```
public Mapa(int id,  
             java.lang.String representa,  
             AVLTree estancias,  
             int activa)
```

Constructor del Mapa

### Parámetros:

`id` - identificador del mapa.  
`representa` - Imagen de fondo del mapa  
`estancias` - Estancias activas del mapa  
`activa` - Variable que indica cual de las estancias está activa

## Métodos

```
public Estancia getEstanActual()
```

Devuelve la estancia actual

### Devuelve:

la estancia que actualmente tiene que ser mostrado

```
public Escenario getEscenActual()
```

Devuelve el escenario a mostrar

### Devuelve:

el escenario que actualmente tiene que ser mostrado

```
public int getKey()
```

Devuelve la clave del comparable

### Devuelve:

La clave del comparable identificador del comparable.

### 5.2.2.12. Node

Representa a cada uno de los nodos que formarán parte del árbol AVL.

El código que usé, y que se puede ver en el apéndice A, lo hallé en el libro “Introduction to Algorithm (2nd edition), McGraw-Hill (2001), ISBN: 0070131511”.

## Atributos

```
public Comparable data
```

The data stored in the node.

```
public Node parent
```

The node's parent.



---

public int **bf**

The node's balanceFactor.

---

public Node **left**

The node's left child.

---

public Node **right**

The node's right child.

---

### Constructor

public **Node**(Comparable data)

Initializes a node with the key and the data and makes other pointers null. The Balance Factor is initialized to be 0.

**Parámetros:**

data - Data to save in the node.

### Métodos

public Node **rotateLeft**()

Performs a left rotation of the subtree rooted at Node a, and returns the new root. The pointers and balance factors are updated as specified in the AVL handout.

---

public Node **rotateRight**()

Performs a right rotation of the subtree rooted at Node a, and returns the new root. The pointers and balance factors are updated as specified in the AVL handout.

---

public Node **rotateRightLeft**()

Performs a double right left rotation of the subtree rooted at Node a, and returns the new root. The pointers and balance factors are updated as specified in the AVL handout.

---

public Node **rotateLeftRight**()

Performs a double left right rotation of the subtree rooted at Node a, and returns the new root. The pointers and balance factors are updated as specified in the AVL handout.



### 5.2.2.13. Opcion

El programa podrá mostrar opciones de muchos objetos. Esas opciones serán objetos del tipo Opción

Opcion
activa: bool texto: string id: string consecuencias: Vector conseActiva: int
+ Opcion(activa:bool,texto:string,id:string,consecuencias:Vector)

Imagen 41: Clase Opcion

#### Atributos

boolean **activa**

Indica si está opción está activa para un determinado Objeto

java.lang.String **texto**

Texto de la opción

java.lang.String **id**

Identificador de la opción.

java.util.Vector **consecuencias**

Consecuencias de una determinada opción. En este vector se almacenarán vectores de objetos de la clase Resultado. Un Objeto puede tener consecuencias distintas en momentos distintos, entonces aquí almaceno todas las consecuencias que puede tener y para cada consecuencia almacenamos el Vector de Resultados, en el orden en que ocurrirán. Una vez que una consecuencia ya no se necesita, pues se borra del array

int **conseActiva**

Indica la consecuencia que está activa, de entre las disponibles.



## Constructor

```
public Opcion(boolean activa,  
              java.lang.String texto,  
              java.lang.String id,  
              java.util.Vector consecuencias)
```

Constructor de la clase Opción

### Parámetros:

*activa* - indica si está opción está activa para un determinado Objeto  
*texto* - texto de la opción  
*id* - Identificador de la opción  
*consecuencias* - Consecuencias de una determinada opción

### 5.2.2.14. Opcionable

Interfaz. Aquellas clases cuyos objetos pueden disponer de opciones para mostrar, deberán implementar la interfaz Opcionable.

<b>&lt;&lt;interface&gt;&gt; Opcionable</b>
+ getOpcion(): Opcion + getConseActivaAcc(opcionSele:int):bool + getConsecuenciaAcc(opcionSele:int):Vector + setConseActivaAcc(id:string,valor:int):void + getNombreAccion(opcionSele:int):string + longAcciones():int + setConseActivaAcc(opcionSele:string,valor:int):void + getConseActivaAcc(opcionSele:int):bool

Imagen 42: Clase Opcionable

## Métodos

Opcion **getOpcion**(java.lang.String id)

Devuelve la opción cuyo id coincide con el que se pasa como parámetro

### Parámetros:

*id* - String con el identificador de la opción

### Devuelve:

La opción cuyo id coincide con el de la opción.

boolean **getConseActivaAcc**(int opcionSele)

Indica si una determinada acción tiene o no alguna consecuencia

### Parámetros:

*opcionSele* - La opción seleccionada en el juego

### Devuelve:

boolean indicando si esa opción o acción tiene alguna consecuencia si es true es que no hay consecuencias, y viceversa.



---

java.util.Vector **getConsecuenciaAcc**(int opcionSele)

Devuelve la consecuencia para una determinada acción

**Parámetros:**

opcionSele - La opción seleccionada en el juego

**Devuelve:**

Vector con la consecuencia activa para una determinada acción

---

void **setConseActivaAcc**(java.lang.String id,  
int valor)

Establece la consecuencia activa para una determinada acción.

**Parámetros:**

id - Identificador de la opción a modificar.

valor - el nuevo valor de conseActiva

---

java.lang.String **getNombreAccion**(int opcionSele)

El nombre de una determinada acción.

**Parámetros:**

opcionSele - La opción seleccionada en el juego

**Devuelve:**

nombre de la acción.

---

int **longAcciones**()

Devuelve el numero de acciones disponibles para un Seleccionable

**Devuelve:**

el numero de acciones disponibles para un Seleccionable

---

void **setAccionActiva**(int opcionSele,  
boolean valor)

Establece si una acción esta activa o no

**Parámetros:**

opcionSele - La opción seleccionada en el juego

valor - boolean indicando si está activa o no.

---

boolean **getAccionActiva**(int opcionSele)

Indica si una acción esta activa o no

**Parámetros:**

opcionSele - La opción seleccionada en el juego

**Devuelve:**

boolean indicando si está activa o no



### 5.2.2.15. PathFinding

Yo como algoritmo de búsqueda decidí usar el A\*, el cual es un algoritmo de búsqueda inteligente que busca el camino más corto desde un estado inicial al estado meta a través de un espacio de problema, usando una heurística óptima. Como ignora los pasos más cortos en algunos casos rinde una solución subóptima.

#### Atributos

int **mapWidthMax**

int **mapHeightMax**

int **mapWidth**

int **mapHeight**

int **imax**

boolean[] **walkability**

int[] **paths**

int **paths\_count**

int **shortest\_path**

int[] **paths\_coord**

byte[] **parent**

byte **direction\_up**

byte **direction\_down**

byte **direction\_left**

byte **direction\_right**



int[] **paths\_guessed\_way**

---

boolean **path\_found**

---

int[] **path**

---

int **path\_count**

---

int **path\_total\_length**

---

int **manhattan**

---

int **real**

---

int **real\_diagonalvalue**

---

int **heuristic**

## Constructor

```
public PathFinding(int width,  
                    int height)
```

## Métodos

```
public boolean findPath(int startx,  
                        int starty,  
                        int targetx,  
                        int targety)
```

---

```
int getIndex(int x,  
             int y)
```

---

```
int getX(int index)
```

---

```
int getY(int index)
```

---

```
void addPoint(int n,  
              int x, int y)
```





### 5.2.2.16. Resultado

Un objeto de tipo Opción, tendrá unas consecuencias, y esas consecuencias tendrán objetos de tipo Resultado.

Resultado
id: int info: string
+ Resultado(id:int,info:string)

Imagen 43: Clase Resultado

#### Atributos

int **id**

Identificador del resultado a realizar (ejemplo Utils.reMensaje es un numero que aquí servirá como id)

java.lang.String **info**

Aquí se almacenarán, en los casos en que sea necesario, el texto del resultado a realizar, por ejemplo un mensaje

#### Constructor

```
public Resultado(int id,  
                 java.lang.String info)
```

Constructor de la clase Resultado

**Parámetros:**

id - Identificador del resultado a realizar

info - información añadida un resultado

### 5.2.2.17. Video

Un video será una sucesión de imágenes con una explicación. Podrá haber infinitos o ningún video, y se actualizarán, completamente, cada vez que se cambie de fase. Implementa el interfaz Comparable.

Video
id: string vinyeta: string texto: Object[] sigParte: int
+ Video(id:int,vinyeta:string,texto:Vector,sigParte:int)

Imagen 44: Clase Video



## Atributos

int **id**

Identificador del Video

java.lang.String **vinyeta**

Nombre del conjunto de viñetas al que hay que llamar.

java.lang.Object[] **texto**

Texto a mostrar en cada una de las viñetas

int **sigParte**

Indica a la parte de juego a la que se pasa después del video

## Constructor

```
public Video(int id,  
             java.lang.String vinyeta,  
             java.util.Vector texto,  
             int sigParte)
```

Constructor de la clase Video

### Parámetros:

**id** - Identificador del video.

**vinyeta** - conjunto de viñetas al que hay que llamar.

**texto** - Texto a mostrar en cada una de las viñetas

**sigParte** - parte de juego a la que se pasa después del video

## Métodos

```
public int getKey()
```

Devuelve la clave del comparable

### Devuelve:

La clave del comparable identificador del comparable.



## 5.3. Diagramas de secuencia

### 5.3.1.Mirar ítem

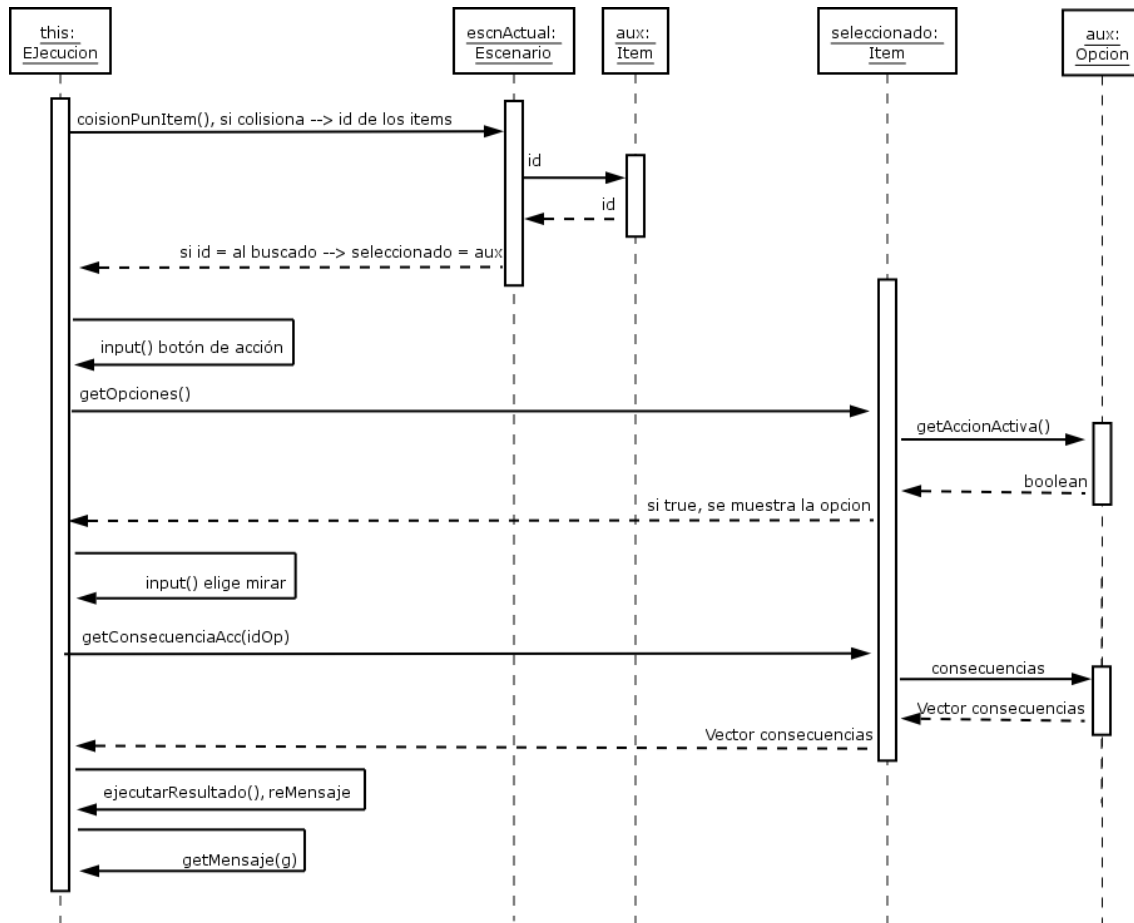


Imagen 45: Diagrama de secuencia de "Mirar ítem"

En el vector consecuencias habrá múltiples resultados. En este caso suponemos, ya que es el caso más común, que uno de los resultados será el de mostrar un mensaje por pantalla (reMensaje).



### 5.3.2.Coger ítem

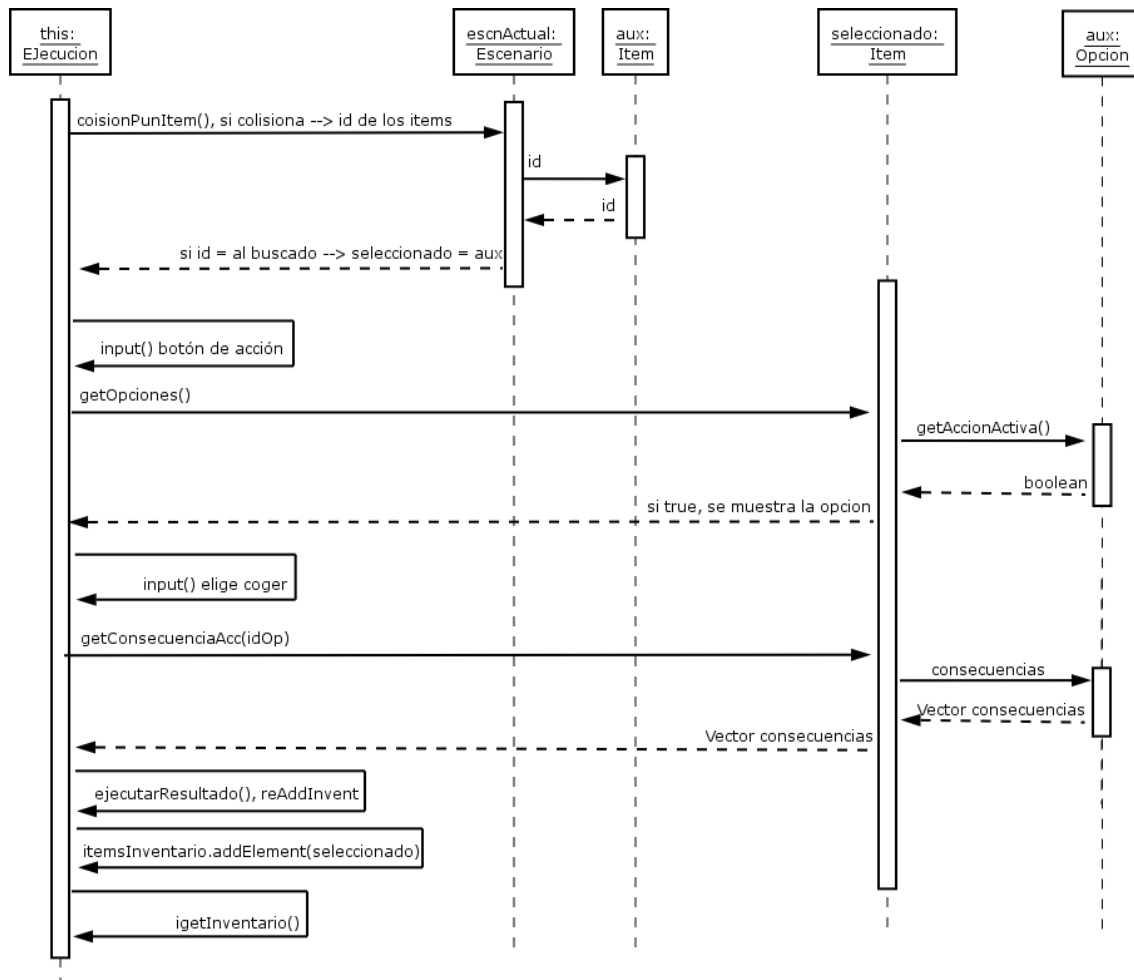


Imagen 46: Digrama de secuencia de "Coger ítem"

En el vector consecuencias habrá múltiples resultados. En este caso suponemos, ya que es el caso más común, que uno de los resultado será el de añadir al inventario (reAddInvent).



### 5.3.3. Usar ítem

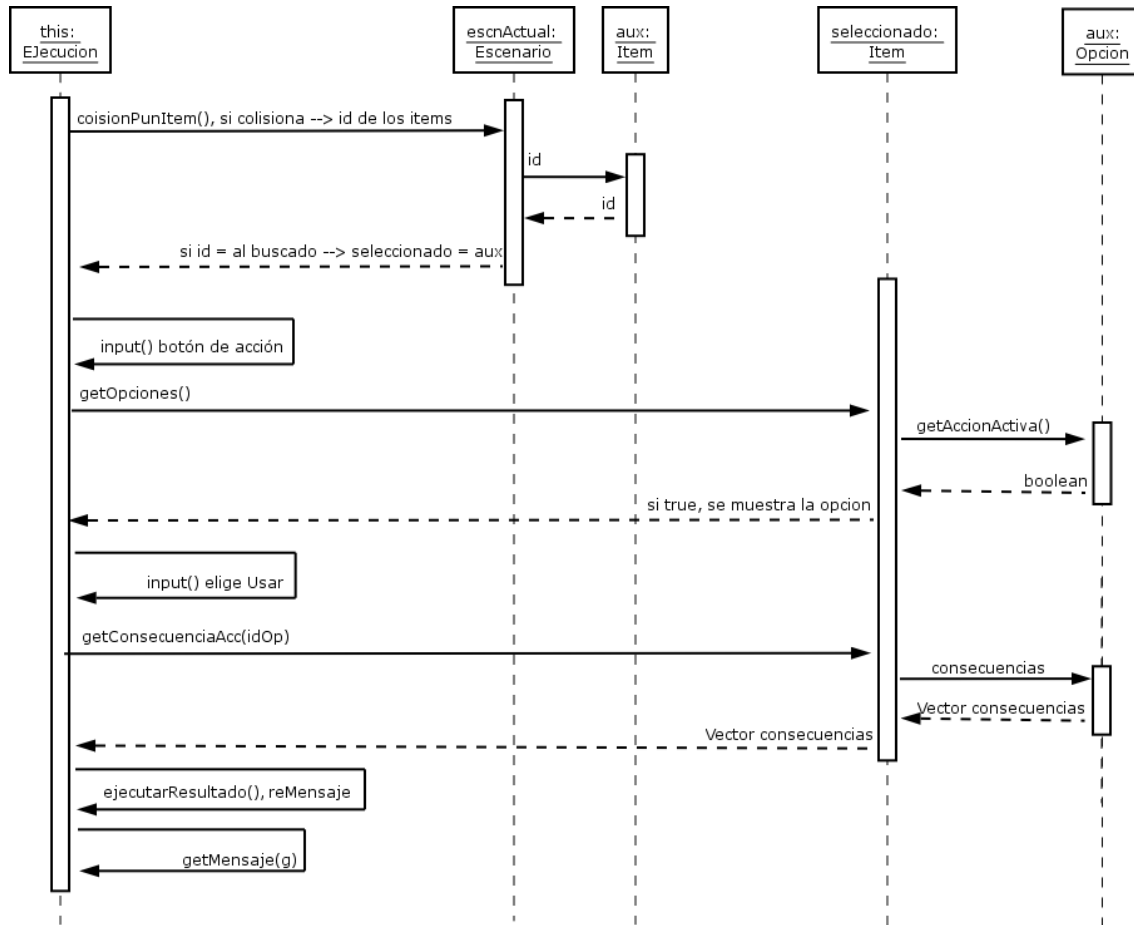


Imagen 47: Diagrama de secuencia de "Usar ítem"

Al igual que los dos diagramas anteriores, en el vector consecuencias habrá múltiples resultados. En este caso suponemos que uno de los resultado será el de mostrar un mensaje(reMensaje).



### 5.3.4.Ir a Ítem

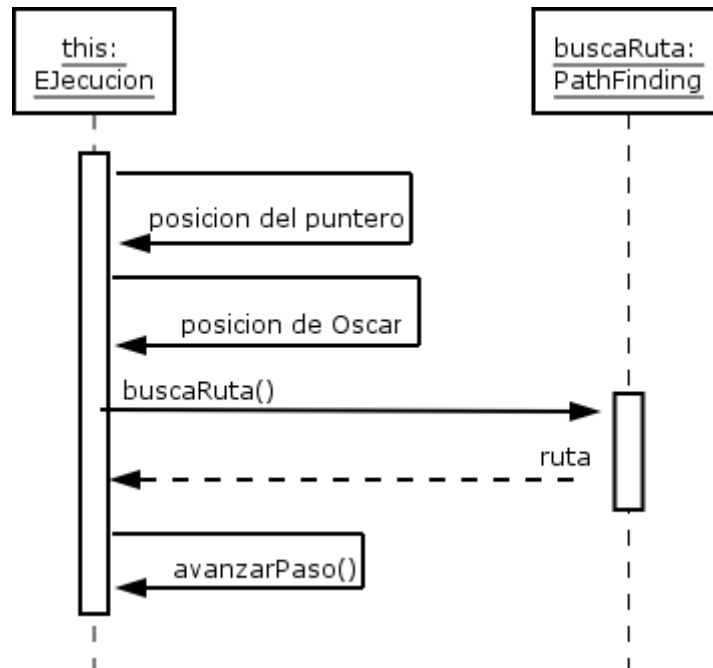
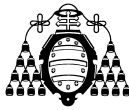


Imagen 48: Diagrama de secuencia de "Ir a ítem"

Teniendo en cuenta que el escenario está dividido en celdas, lo que se hace es calcular la celda en que está el puntero, la celda en la que está Oscar, y buscar la ruta entre esas dos celdas. Una vez encontrada no queda más que llamar al método `avanzarPaso()`, el cual hace que Oscar se vaya desplazando hasta la posición del puntero.

Destacar que Oscar no se moverá a la celda del ítem, sino a la celda donde estaba el puntero cuando pulso la opción Ir a... del ítem.



### 5.3.5. Combinar items

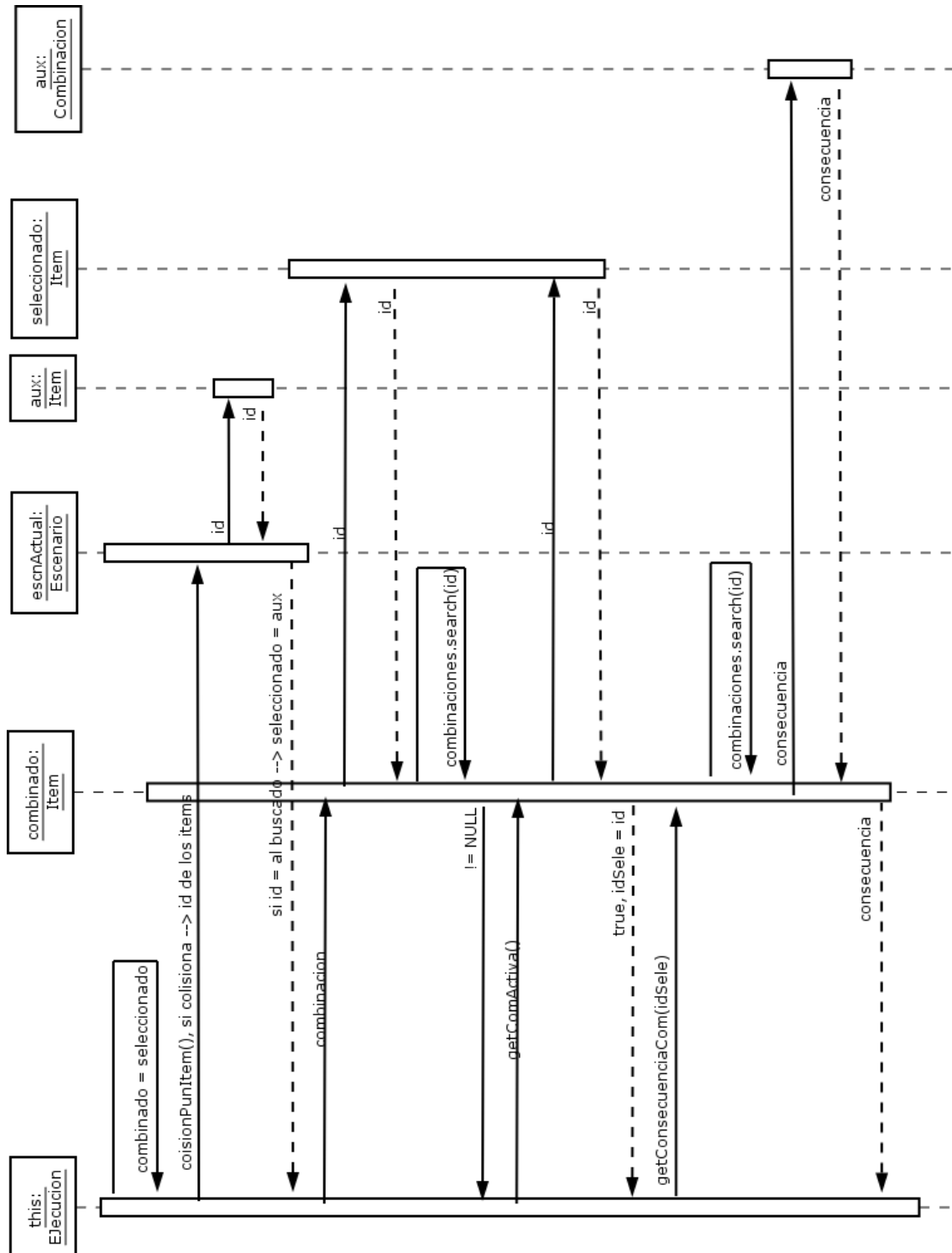


Imagen 49: Diagrama de secuencia de "Combinar items"



### 5.3.6.Ir al menú

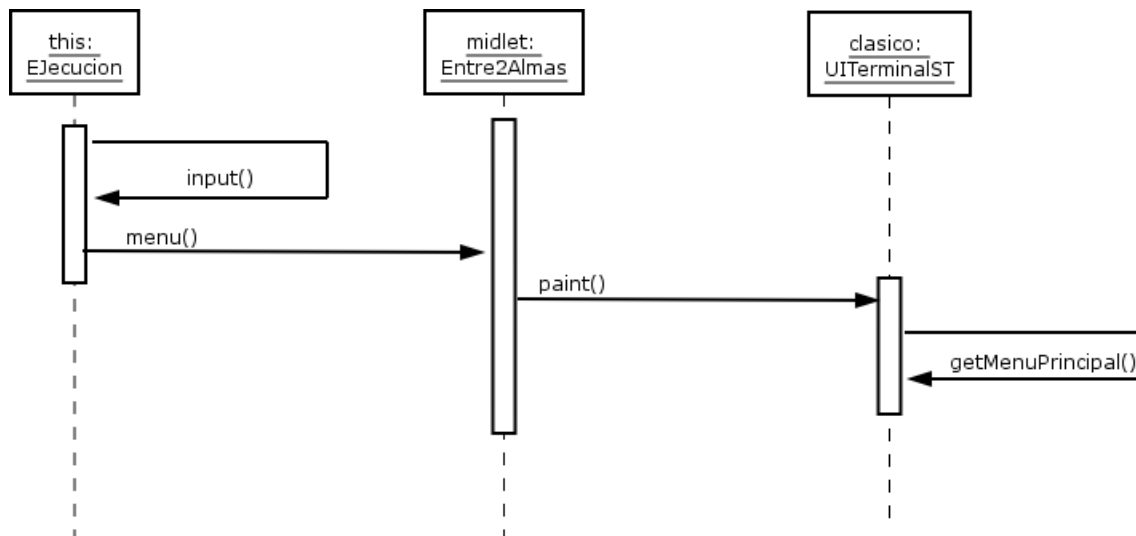


Imagen 50: Diagrama de secuencia de "Ir al menú"

Para que ocurra esto, durante el `input()` se deberá detectar que se ha pulsado la tecla correspondiente al acceso directo al menú.

También tener en cuenta que, aunque aquí solo se representa uno, hay dos tipos de menús (`UITerminalST` y `UITerminalSTBG`).





### 5.3.7.Hablar con personas

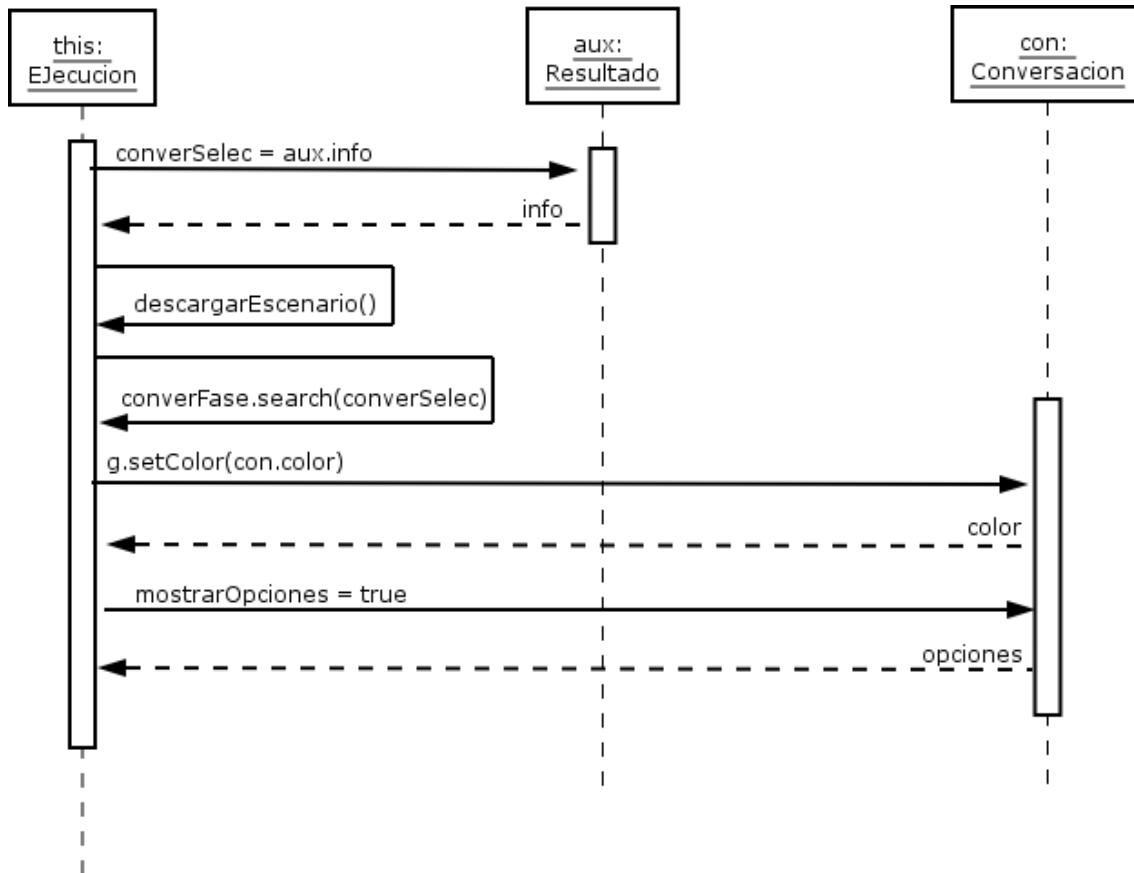


Imagen 51: Diagrama de secuencia de "Hablar con personas"



### 5.3.8.Ver video

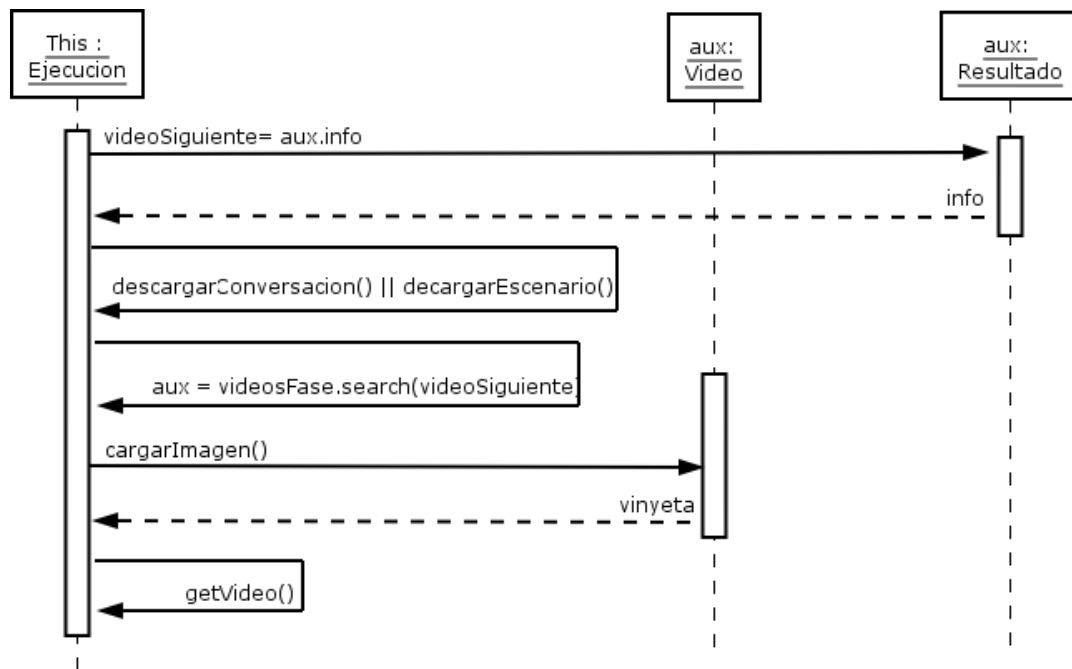


Imagen 52: Diagrama de secuencia de "Ver video"

### 5.3.9.Moverse por escenario

Su diagrama será igual que el de "Ir a ítem", ya que este último no va hasta la posición del ítem, sino hasta la posición del puntero.

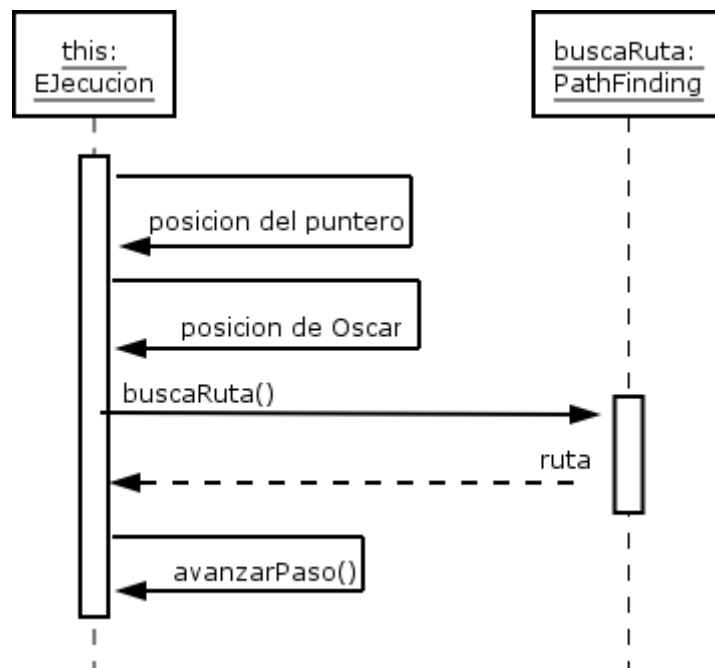


Imagen 53: Diagrama de secuencia de "Moverse por escenario"



### 5.3.10. Moverse entre escenarios

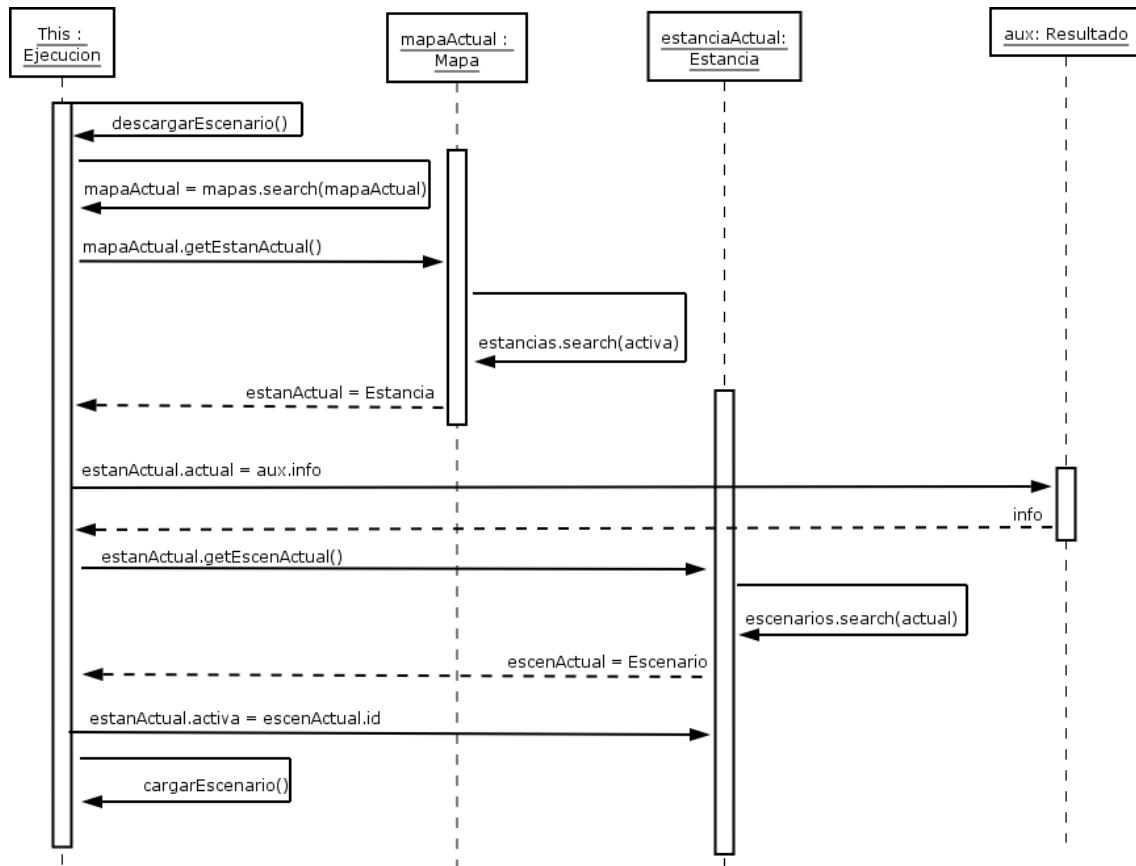


Imagen 54: Diagrama de secuencia de "Moverse entre escenario"



### 5.3.11. Moverse entre estancias

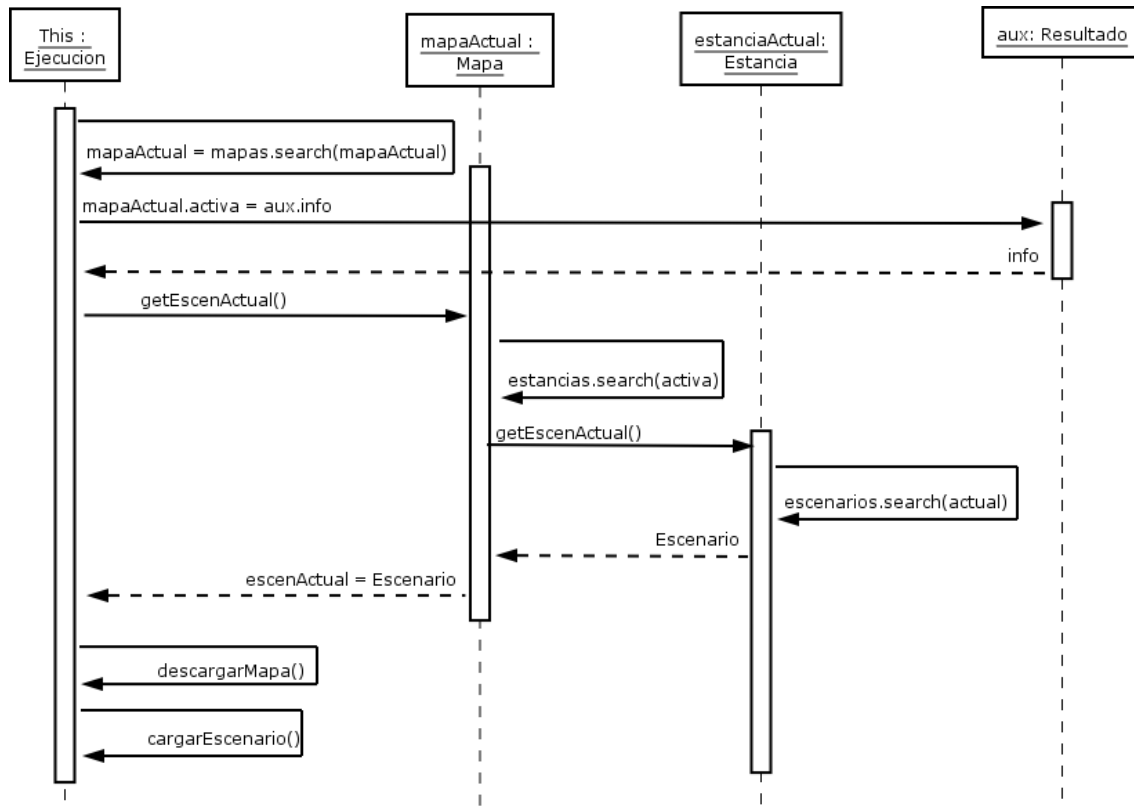


Imagen 55: Diagrama de secuencia de "Moverse entre estancias"

### 5.3.12. Cambiar de fase

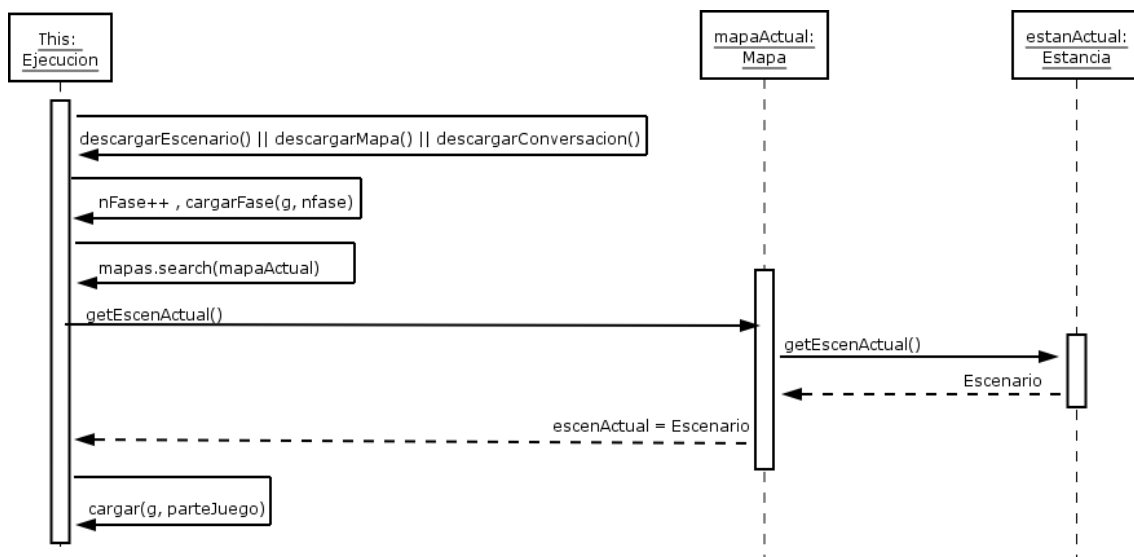


Imagen 56: Diagrama de secuencia de "Cambiar de fase"



## 5.4. Descripción de los ficheros XML

Uno de los objetivos a desarrollar, es que la aplicación sea modificable por XML, de tal manera que cualquier persona pueda crear su propia aventura gráfica, sin tocar el código fuente ni recompilar, y solo editando unos ficheros XML. Los motivos por los que se decidió usar XML fueron:

- Reducción y simplificación considerable del código fuente.
- Simplicidad de aprendizaje, incluso para los que no están familiarizados con la programación
- Modificaciones mucho más fáciles y rápidas, ya que los documentos XML son más claros y estructurados que el código fuente.
- La idea de que cualquiera persona pueda hacer su propia aventura gráfica para móvil, sin necesidad de tocar el código fuente.

Un detalle que quiero aclarar sobre el emulador de Java para móvil es que no permite la creación ni la modificación de ficheros, de tal manera que los archivos XML que forman el juego, nunca podrán ser modificados en tiempo ejecución. Esto supone un inconveniente, sobre todo a la hora de implementar el guardar partida. Para suplir esta limitación, el emulador Java proporciona una base de datos basada en registro, formados de chorros de bytes, bajo la API rms.

En total sería necesario modificar dos ficheros para poder actualizar el juego a tu gusto, estos ficheros son los siguiente:

***basicos.xml*** → Aquí se detallarán todos los datos base.

***FaseX.xml*** → Aquí se especifica la lógica de la aventura, siendo X el numero de la fase, así si tenemos 3 fases, tendremos 3 archivos llamados Fase1.xml, Fase2.xml y Fase3.xml.

### 5.4.1. Estructura y función de basicos.xml

Como dije antes, aquí se detallarán los datos base, como son las imágenes que formarán el juego, cual de estas le corresponde a cada parte del menú, cual es la imagen de cargando y guardando, cuales son las de la pantalla de presentación, etc.

Este XML será el primero en ser cargado nada más arrancar la aplicación, incluso antes de que se muestre la pantalla de presentación. A continuación se muestra, a través del diagrama de su XSD, la estructura del XML. Como este diagrama es muy grande, se ha dividido en partes:



### 5.4.1.1. Visión general

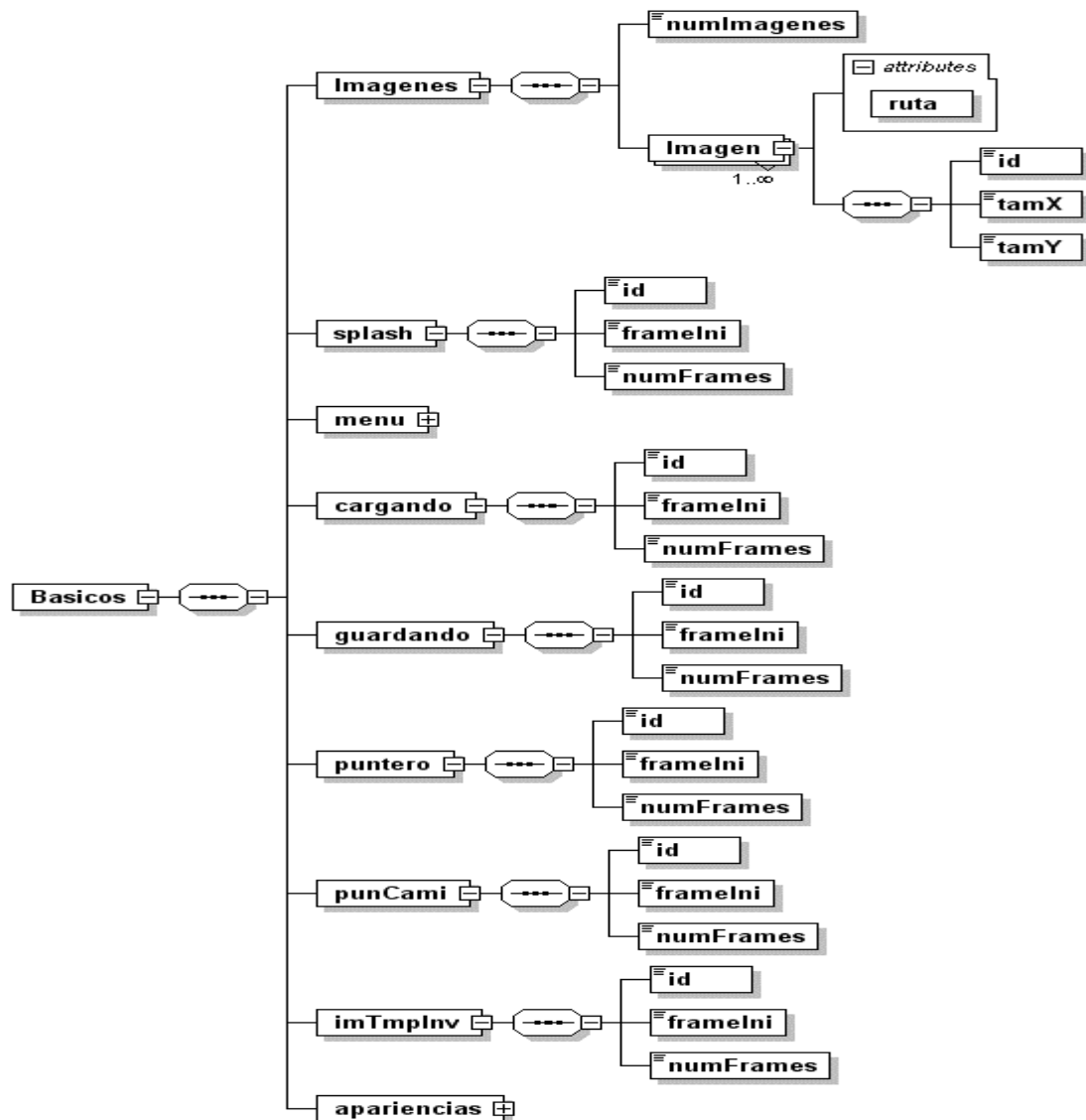


Imagen 57: basics.xml (Visión general)

**Imagenes** → Imágenes que serán utilizadas en el juego. Tener en cuenta que las imágenes son para 2 grupos de pantallas, las de 128x128 y las de 176x200. Y para que el juego sea valido para los 2 grupos de móviles, por cada imagen que se ponga para el móvil "de pantalla pequeña" SU SIGUIENTE IMAGEN será para el móvil de "pantalla grande".

- numImagenes → Número de imágenes a definir.
- Imagen → Cada una de las imágenes que formarán el juego.
  - o ruta → Es la ruta de directorios en la que puede ser encontrada la imagen, partiendo de que la raíz es el directorio res.
  - o id → El id de cada imagen será único.



- o tamX → Ancho en píxeles, de cada uno de los frames de esta imagen.
- o tamY → Alto en píxeles, de cada uno de los frames de esta imagen.

**splash** → Pantalla de presentación del juego.

- id → Identificador de la imagen, que habrá sido anteriormente definida en el apartado imágenes, que contiene los frames de la pantalla de presentación.
- frameIni → Frame inicial, en un determinado sprite, a partir del cual empiezan los frames del splash.
- numFrames → Numero de frames de los que está compuesta la pantalla de presentación..

**menu** → Configuración del menú.

**cargando** → Imágenes que se mostrarán cuando se está cargando una partida.

- id → Identificador de la imagen, que habrá sido anteriormente definida en el apartado imágenes, que contiene los frames de cargando.
- frameIni → Frame inicial, en un determinado sprite, a partir del cual empiezan los frames de cargando.
- numFrames → Número de frames de los que está compuesta la pantalla de cargando.

**guardando** → Imágenes que se mostrarán cuando se está cargando una partida.

- id → Identificador de la imagen, que habrá sido anteriormente definida en el apartado imágenes, que contiene los frames de guardando.
- frameIni → Frame inicial, en un determinado sprite, a partir del cual empiezan los frames de guardando.
- numFrames → Número de frames de los que está compuesta la pantalla de guardando.

**puntero** → La representación del puntero.

- id → Identificador de la imagen, que habrá sido anteriormente definida en el apartado imágenes, que contiene los frames del puntero.
- frameIni → Frame inicial, en un determinado sprite, a partir del cual empiezan los frames del puntero.
- numFrames → Número de frames de los que está compuesta el puntero. Su valor será dos frames, uno para cuando no colisiona con ningún ítem, y otro para cuando si.

**punCami** → Esta imagen, o especie de puntero, se usará para ir comprobando el area caminable de un escenario. Lo normal es que mida 2x2 píxeles.



- *id* → Identificador de la imagen, que habrá sido anteriormente definida en el apartado imágenes, que contiene los frames del puntero.
- *frameIni* → Frame inicial, en un determinado sprite, a partir del cual empiezan los frames del puntero.
- *numFrames* → Número de frames de los que está compuesta el puntero. Su valor será 2 frames, uno para cuando no colisiona con ningún ítem, y otro para cuando sí.

*imTmpInv* → Esta imagen, se usará como imagen temporal del inventario, es decir, para poder cargar el inventario cuando no hay ninguna imagen en él. Así que recomiendo usar la imagen, definida anteriormente en imágenes, cuyo *id* coincide con el usado para este juego.

- *id* → Identificador de la imagen.
- *frameIni* → Frame inicial, en un determinado sprite.
- *numFrames* → Número de frames de los que está compuesta la imagen.

*apariencias* → El personaje principal podrá cambiar de apariencia a lo largo del juego.

#### **5.4.1.2. Basicos / menu**

Todas las sub partes de menú tiene los elementos *id*, *frameIni* y *numFrames*. Como ya se ha explicado unas cuantas veces atrás, se decide no volver a explicarlos y se da por echo que se sabe que están ahí dichos elementos.



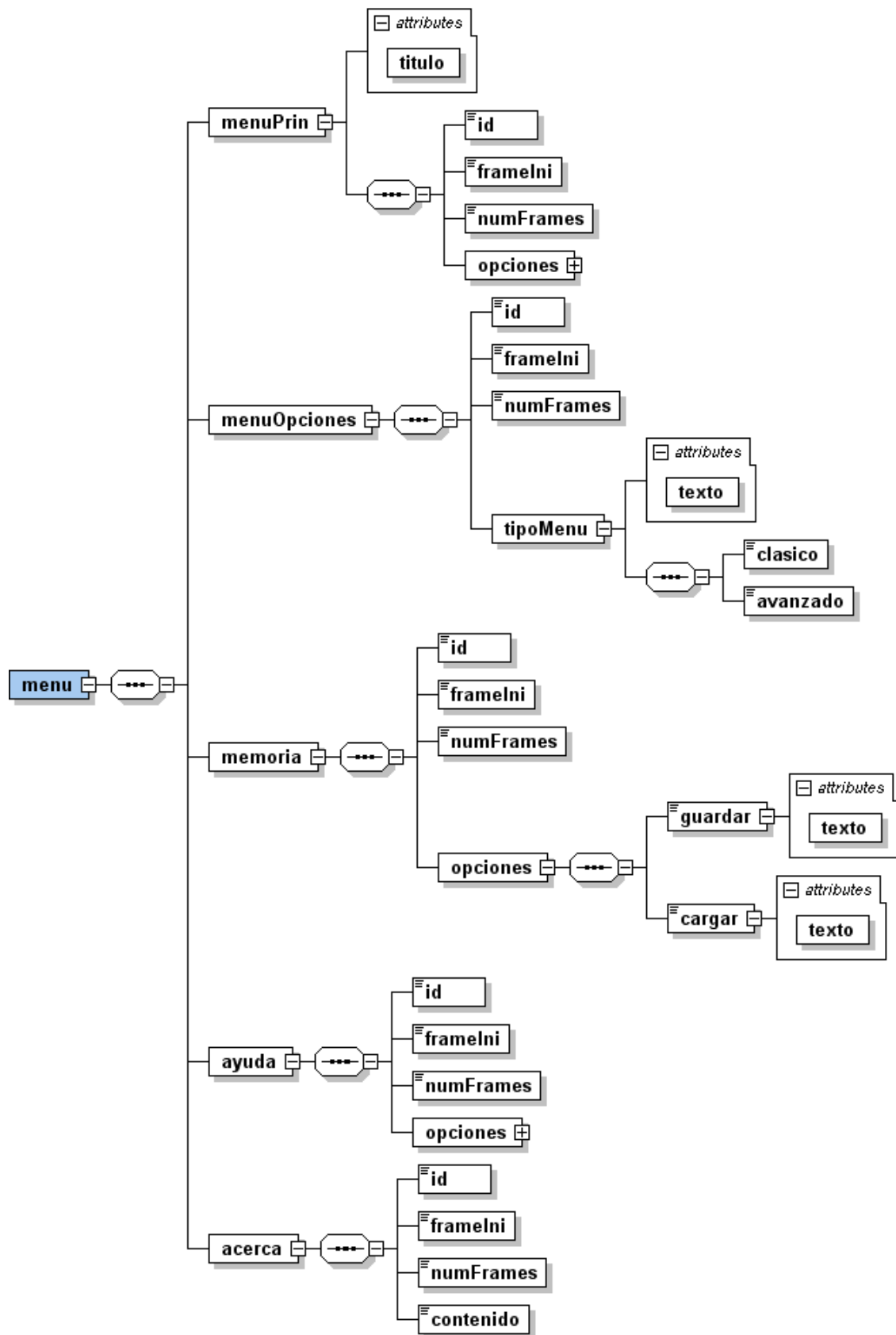


Imagen 58: basics.xml (Basicos / menu)



**menuPrin** → Configuración del menú principal

- titulo → Es el título que se mostrará en el menú principal.
- opciones → Opciones que se mostrarán en el menú principal

**menuOpciones** → Configuración del menú opciones.

- tipoMenu → Es la opción que encontraremos en este menú. Y nos permite elegir el tipo de menú que queremos, uno clásico o uno avanzado.
  - o texto → El texto que se mostrará para identificar esta opción.
  - o clásico → El texto que representará al menú adaptado al interfaz nativa del móvil.
  - o avanzado → El texto que representará un menú colorido y preparado exclusivamente para el juego.

**memoria** → Configuración del menú de memoria, que es donde se harán las tareas de cargar y guardar partida.

**opciones** → Opciones que encontraremos en este menú.

- guardar → Opción guardar partida, que en su atributo texto se especifica la opción de guardar partida.
- cargar → Opción cargar partida.

**ayuda** → Configuración del menú de ayuda.

- opciones → Opciones que encontraremos en este menú.

**acerca** → Configuración del menú Acerca de...

- contenido → Aquí se especificará el contenido que tendrá el menú Acerca de...



### 5.4.1.3. menu / menuPrin / opciones

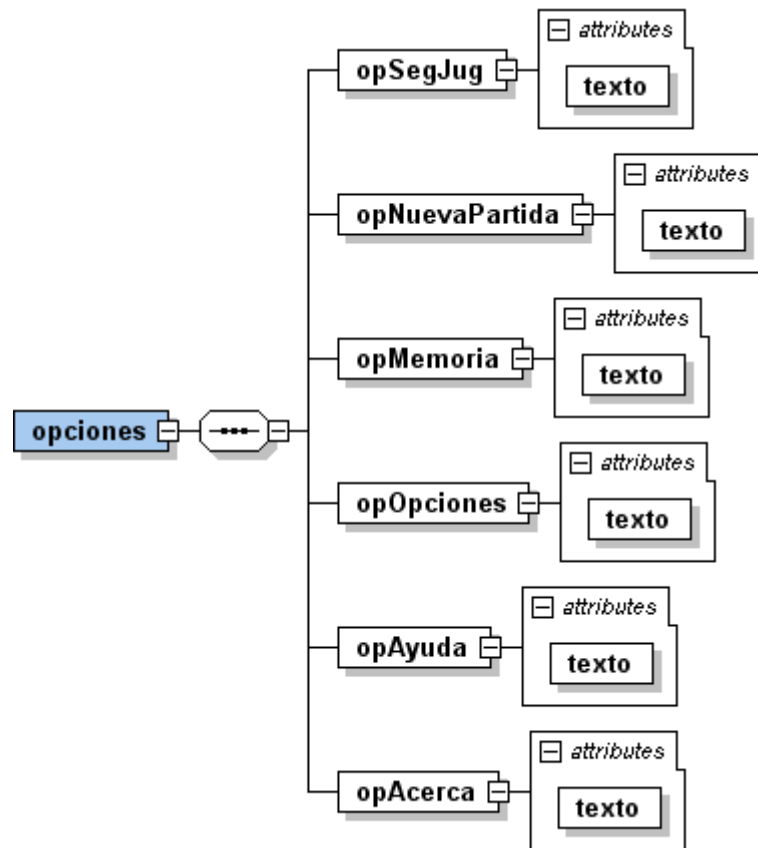


Imagen 59: basics.xml (menu / menuPrin / opciones )

**opSegJug** → Como atributo tendrá al texto que se pondrá a la opción cuya función es que la partida, ya iniciada, continúe.

**opNuevaPartida** → Como atributo tendrá al texto que se pondrá a la opción cuya función es iniciar una nueva partida.

**opMemoria** → Como atributo tendrá al texto que se pondrá a la opción cuya función es ir al menú de cargar y guardar partidas.

**opOpciones** → Como atributo tendrá al texto que se pondrá a la opción cuya función es ir al menú de opciones.

**opAyuda** → Como atributo tendrá al texto que se pondrá a la opción cuya función es ir a la ayuda.

**opAcerca** → Como atributo tendrá al texto que se pondrá a la opción cuya función es ir al Acerca de...



#### 5.4.1.4. menu / ayuda / opciones

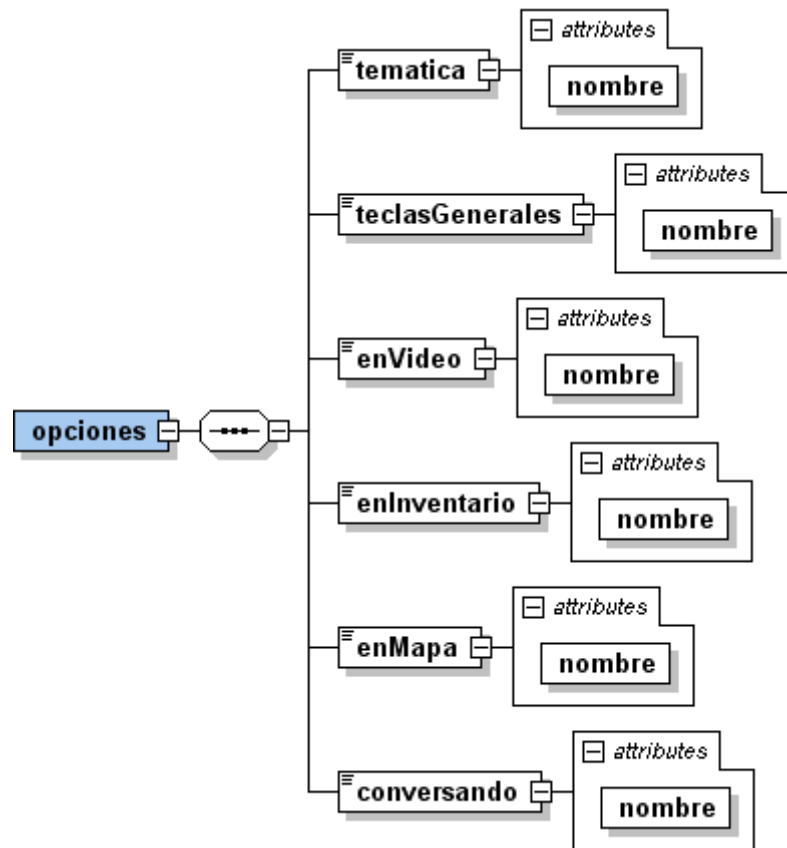


Imagen 60: basics.xml (menu / ayuda / opciones )

**tematica** → Como atributo tendrá el nombre que se pondrá a la opción cuya función explicar la temática del juego. Luego como contenido se pondrá esa temática.

**teclasGenerales** → Como atributo tendrá el nombre que se pondrá a la opción cuya función explicar la temática del juego. Luego como contenido se pondrá esa temática.

**enVideo** → Como atributo tendrá el nombre que se pondrá a la opción cuya función explicar la temática del juego. Luego como contenido se pondrá esa temática.

**enInventario** → Como atributo tendrá el nombre que se pondrá a la opción cuya función explicar la temática del juego. Luego como contenido se pondrá esa temática.

**enMapa** → Como atributo tendrá el nombre que se pondrá a la opción cuya función explicar la temática del juego. Luego como contenido se pondrá esa temática.

**conversando** → Como atributo tendrá el nombre que se pondrá a la opción cuya función explicar la temática del juego. Luego como contenido se pondrá esa temática.



### 5.4.1.5. Basicos / apariencias

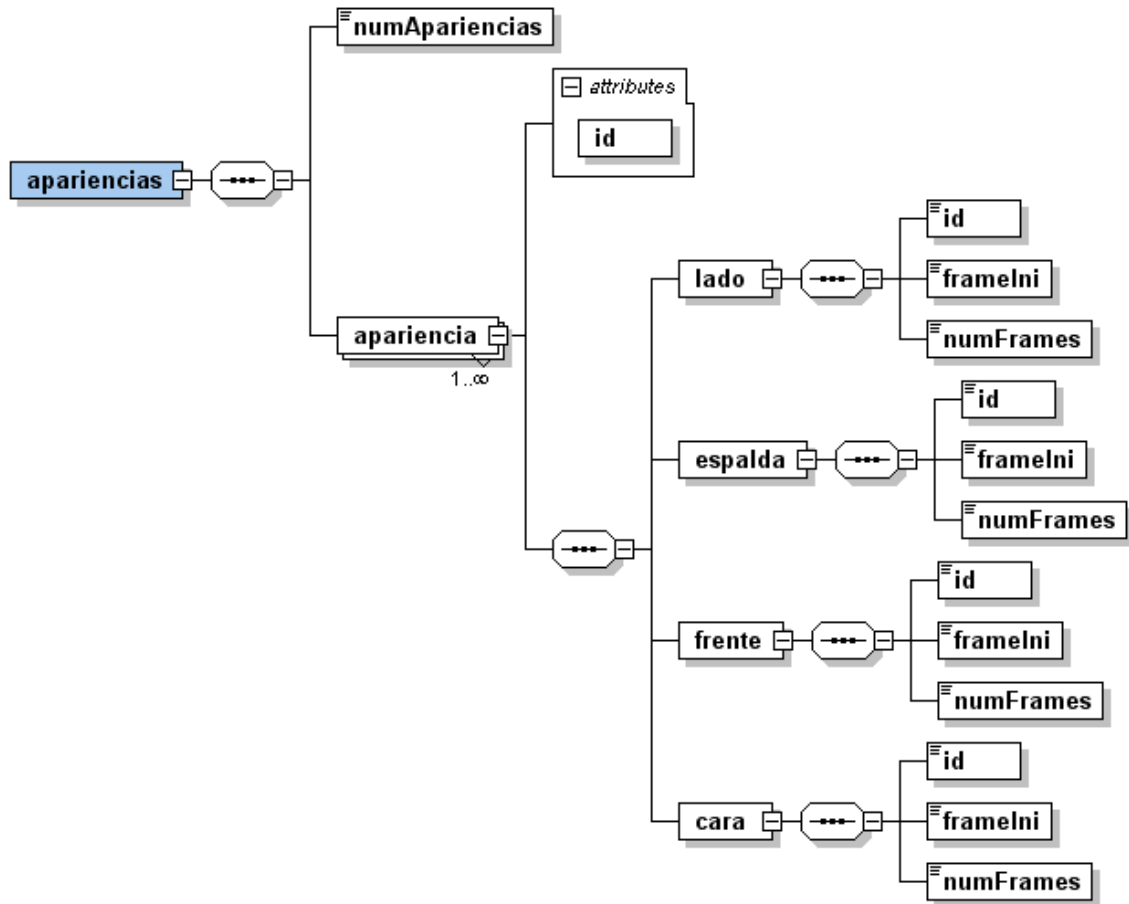


Imagen 61: basicos.xml (Basicos / apariencias)

**numApariencias** → Número de apariencias a definir.

**apariencia** → Cada una de las apariencias que puede tener el personaje principal.

- **id** → Identificador de la apariencia, el cual será único para cada apariencia.
- **lado** → Sprite del protagonista andando de lado. Se define id, frameIni y numFrames.
- **espalda** → Sprite del protagonista andando de espaldas. Se define id, frameIni y numFrames.
- **frente** → Sprite del protagonista andando de frente. Se define id, frameIni y numFrames.
- **cara** → Zoom de la cara del protagonista. Este sprite sería bueno que estuviese formado por varios frames, con dos, uno con la boca abierta y otro con la boca cerrada, valdría, así se consigue hacer el efecto de que está hablando moviendo la boca. Se define id, frameIni y numFrames.



## 5.4.2. Estructura y función de FaseX.xml

FaseX.xml es una forma de llamar al conjunto de ficheros que especifican cada una de las fases del juego. X será el número de la fase, lo cual quiere decir que si el juego tiene 3 fases, deberíamos tener 3 archivos XML, cuyos nombres serían Fase1.xml, Fase2.xml y Fase3.xml, empezando por el número 1.

En este fichero se especificará la lógica de la aventura, por ejemplo que pasa si intentas coger un objeto, o que opciones tendrá en la conversación con otro personaje, que items habrá en un escenario, cuando saltará cada video, etc.

Este XML se cargará en cuanto inicies o cargues una nueva partida, o cuando cambies de fase, lo que quiere decir que, una vez cargada la fase, ya no habrá cargas, ni si quiera al cambiar de una estancia a otra, lo cual es algo de agradecer en la aventuras gráfica.

A continuación se muestra, , a través del diagrama de su XSD, la estructura del XML. Como este diagrama es muy grande, se ha dividido en partes:

### 5.4.2.1. Visión general

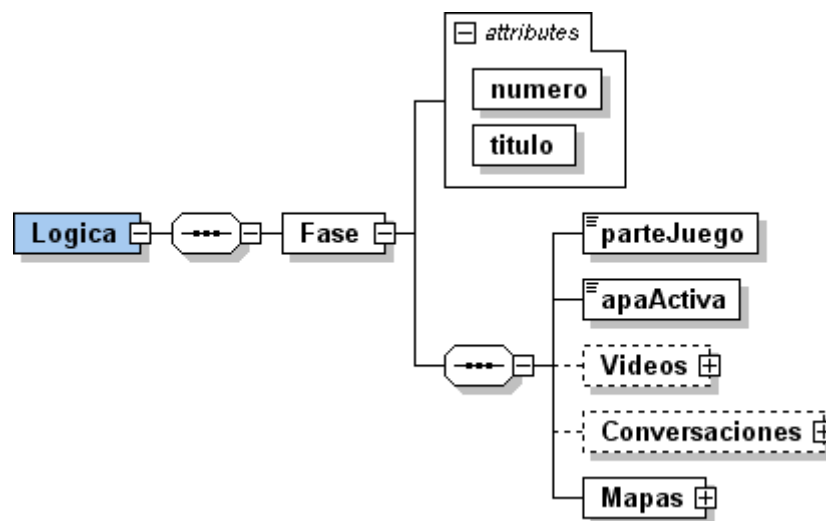


Imagen 62: FaseX.xml (Visión general)

**Fase** → Lógica de la fase.

- **numero** → Atributo de “Fase”. Cada fase deberá tener un número propio, no se podrá repetir. Empezará por 1.
- **titulo** → Es el título de la fase, que será utilizado, junto con el número, en el momento en que se cargue una fase, ya que sobre la imagen de cargando, aparecerá número y nombre de la fase a cargar.
- **parteJuego** → Parte juego indica la parte de juego en la comenzarás al empezar la fase. Puede ser video, conversación o mapa.



- *apaActiva* → Apariencia activa, para el personaje principal, al inicio de esta fase.
- *videos* → Una fase podrá tener videos, pero es opcional.
- *conversaciones* → Una fase podrá tener conversaciones, pero es opcional.
- *mapas* → Una fase deberá tener al menos un mapa.

#### 5.4.2.2. Fase / Videos

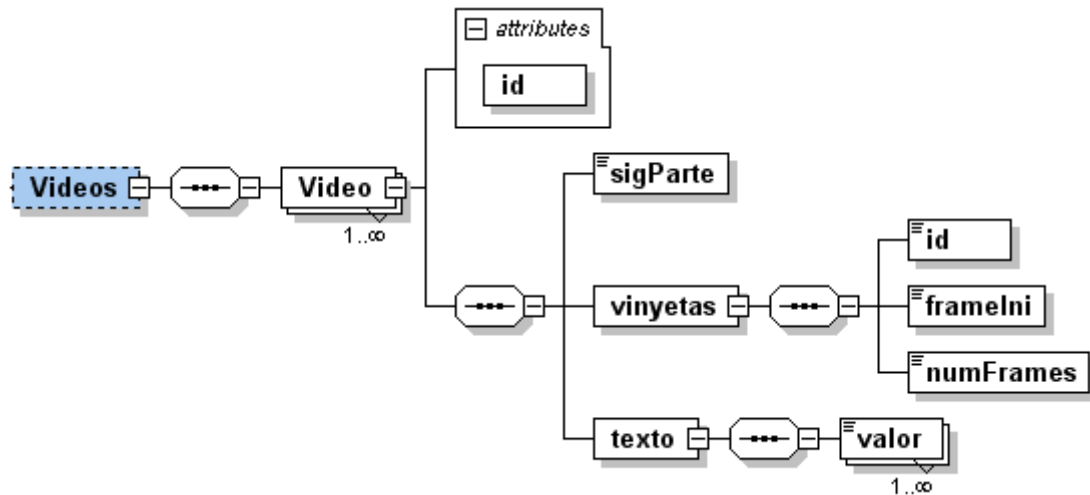


Imagen 63: FaseX.xml (Fase / Videos)

**Video** → En caso de que haya “Videos”, entonces deberá haber por lo menos 1, y como mucho infinitos.

- *id* → Atributo de “Video”. Cada video tendrá un identificador, que será único, pero sólo para cada fase, es decir en la Fase 1 puede haber los videos con id 0,1 y 2, y en la Fase 2 podríamos tener los videos con id 0, 1 ...
- *sigParte* → Elemento de Video. “sigParte” indica la parte de juego siguiente a un video. Puede tener exclusivamente los valores video, juego o mapa.
- *vinyetas* → Para cada viñeta que forma el video, habrá que especificar:
  - o *id* → El id hará referencia al identificador de la imagen que contiene las viñetas a mostrar en el video. Dicha imagen, y su identificador, está definida en *basicos.xml*
  - o *frameIni* → Frame inicial, en un determinado sprite, a partir del cual empiezan las viñetas de un video.
  - o *numFrames* → Numero de frames de los que está compuesto el video. Una viñeta tendrá al menos 1 frame
- *texto* → Los textos que va a llevar cada viñeta.



- o valor → En "valor" se almacenará el texto que le corresponde a cada una de las viñetas, lo cual quiere decir que, lo normal, es que el número de elementos "valor" sea igual que el número de frames del video.

#### 5.4.2.3. Fase / Conversaciones

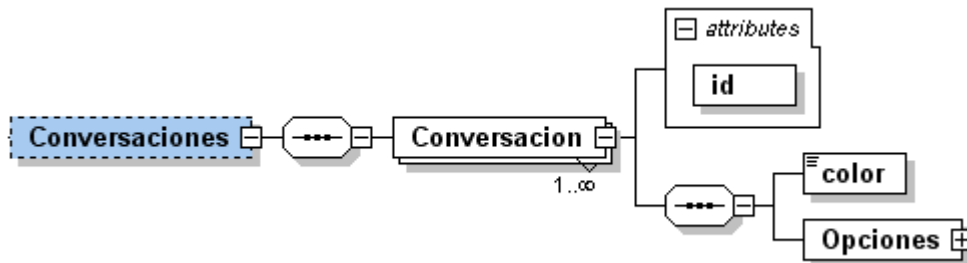


Imagen 64: FaseX.xml (Fase / Conversaciones)

**Conersacion** → Si hay “Conversaciones”, entonces deberá haber por lo menos un elemento de tipo “Conversación”, y el limite será infinito.

- id → Cada conversación tendrá un identificador, que será único, pero sólo para cada fase, es decir en la Fase 1 puede haber las conversaciones con id 0,1 y 2, y en la Fase 2 podríamos hacer las conversaciones con id 0, 1 ...
- color → Será el color de las letras, y el fondo del zoom, del personaje con el que hablamos. La elección del color estará limitada a los siguientes: Rojo, Verde, Crema, Amarillo, Marrón, Azul, Negro, Naranja y Rosa.
- Opciones → Las opciones (preguntas o frases que podemos decir) disponibles en una conversación.

#### 5.4.2.4. Conversaciones / Conversación / Opciones

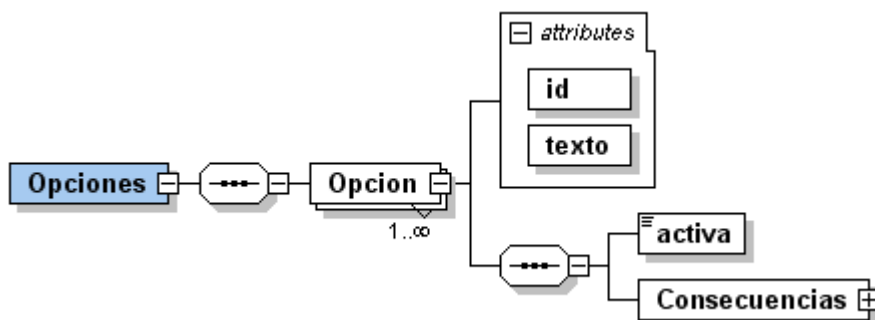


Imagen 65: FaseX.xml (Conversaciones / Conversación / Opciones)

**Opcion** → Deberá haber al menos 1 opción, y no hay limite.

- id → Identificador de la opción, el cual no será un número, si no que será, por ejemplo, a (pregunta principal), aa (pregunta a la que se llega tras decir la a), b (pregunta a la misma altura que a)...





- texto → Texto de la opción. La pregunta o frase que se nos ofrecerá decir a otro personaje.
- activa → Su valor podrá ser SI o NO. Si está activa, la opción se mostrará en el juego.
- Consecuencias → Una opción tendrá unas consecuencias.

#### 5.4.2.5. Conversación / Opciones / Opcion / Consecuencias

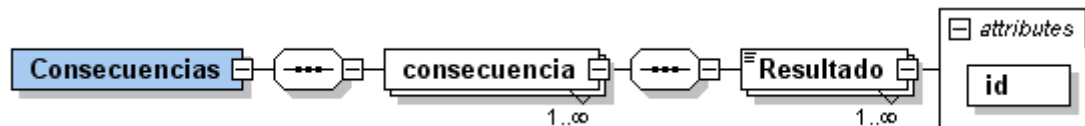


Imagen 66: FaseX.xml (Conversación / Opciones / Opcion / Consecuencias)

*consecuencia* → Deberá haber al menos una consecuencia. No tiene limite.

- Resultado → Una consecuencia esta formada por un conjunto de resultados.
  - o id → Atributo de “Resultado”, el cual indica a la aplicación que acciones debe llevar a cabo. Hay muchos posibles identificadores para un Resultado, podrá ver estos en el punto 7.8.2.13

#### 5.4.2.6. Fase / Mapas

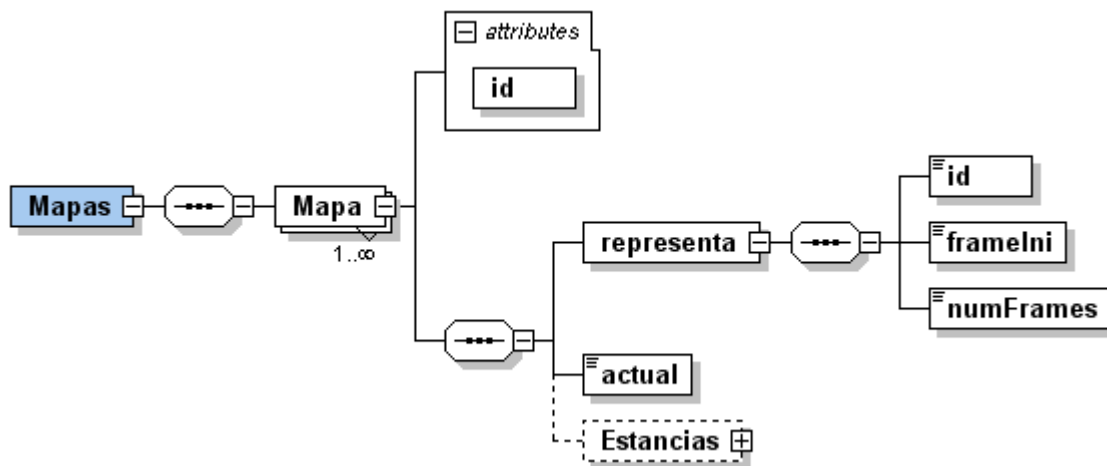


Imagen 67: FaseX.xml (Fase / Mapas)

*Mapa* → Una fase deberá tener al menos 1 mapa, y no habrá limite.

- id → Atributo de Mapa.Cada mapa tendrá un identificador, que será único, pero sólo para cada fase, es decir en la Fase 1 puede haber mapas con id 0,1 y 2, y en la Fase 2 podríamos hacer los mapas con id 0, 1 ...
- representa → El dibujo o la imagen del mapa.



- o id → El id hará referencia a el identificador de la imagen que contiene el dibujo del mapa. Dichas imagenes están definidas en basics.xml.
- o frameIni → Frame inicial, en un determinado sprite, a partir del cual está la imagen del mapa.
- o numFrames → Número de frames de los que está compuesto el mapa.
- actual → Indica el id de la estancia actual de un mapa.
- Estancias → Estancias del mapa. Será opcional.

#### 5.4.2.7. Mapas / Mapa /Estancias

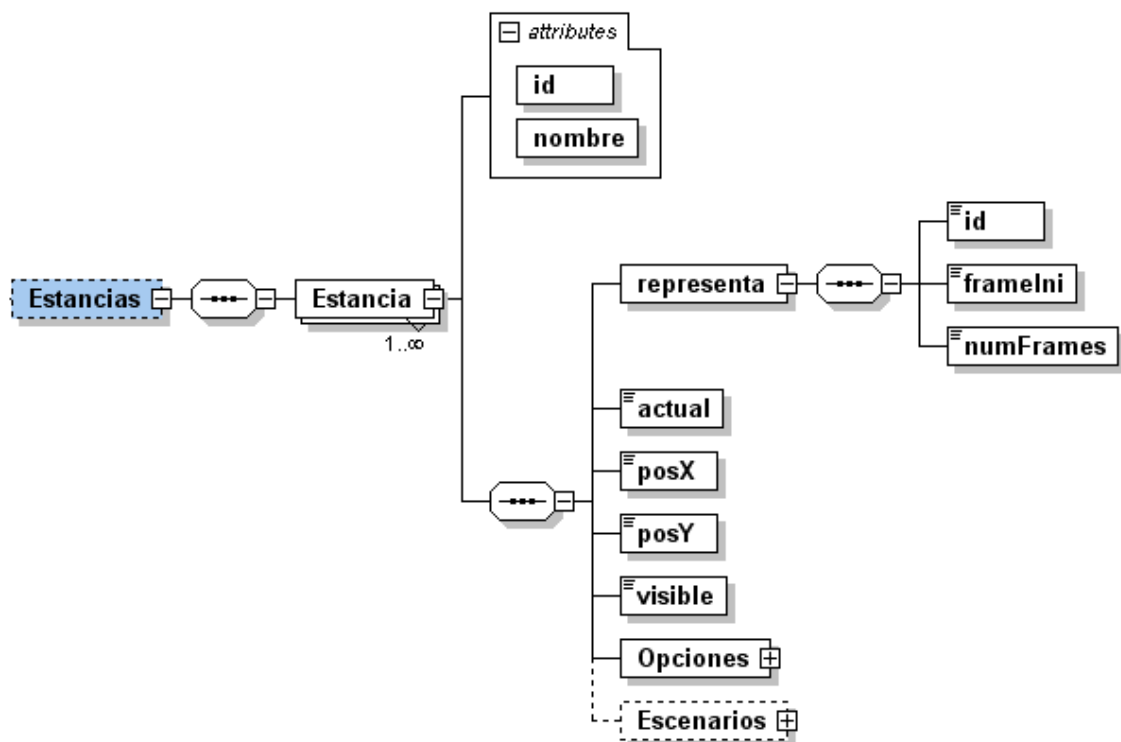


Imagen 68: FaseX.xml (Mapas / Mapa /Estancias)

**Estancia** → Habrá entre 1 e infinitas estancias.

- id → Cada estancia tendrá un identificador, que será único, pero sólo para cada fase, es decir en la Fase 1 puede haber estancias con id 0,1 y 2, y en la Fase 2 podríamos hacer las estancias con id 0, 1 ...
- nombre → Atributo de “Estancia”. Es el nombre que se nos mostrará cuando pasemos el puntero, en la pantalla del mapa, por encima de la estancia.
- representa → Representación de la estancia en el mapa.
  - o id → El id hará referencia a el identificador de la imagen que contiene la representación de la estancia en el mapa. Dichas imagenes están definidas en basics.xml.



- o frameIni → Frame inicial, en un determinado sprite, a partir del cual está la imagen de la estancia.
- o numFrames → Número de frames de los que está compuesta la estancia.
- actual → Indica el identificador del escenario actual de la estancia.
- posX → Valor de X para la posición donde aparecerá, esta estancia, dentro del mapa. A este punto hay que sumarle el ancho que ocupe la imagen que representa la estancia.
- posY → Valor de Y para la posición donde aparecerá, esta estancia, dentro del mapa. A este punto hay que sumarle el alto que ocupe la imagen que representa la estancia.
- visible → Indica si la estancia será visible en el mapa. Sus valores serán SI o NO.
- Opciones → Las opciones que se nos mostrarán, en la pantalla del mapa, al pulsar con el puntero sobre la estancia.
- Escenarios → No es obligatorio que una estancia tenga escenarios.

#### 5.4.2.8. Estancias / Estancia / Opciones

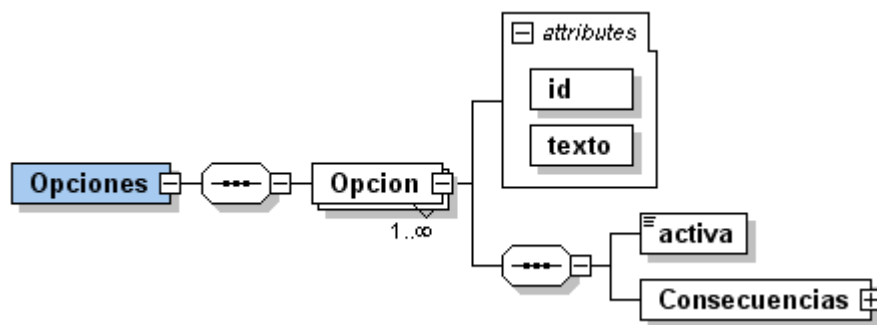


Imagen 69: FaseX.xml (Estancias / Estancia / Opciones)

**Opcion** → Deberá haber al menos 1 opción, y no hay limite.

- id → Identificador de la opción, el cual será un número, único para cada estancia.
- texto → Texto de la opción. El texto de la opción que se nos ofrecerá.
- activa → Su valor podrá ser SI o NO. Si está activa, la opción se mostrará en el juego.
- Consecuencias → Una opción tendrá unas consecuencias.



#### 5.4.2.9. Estancia / Opciones / Opcion / Consecuencias

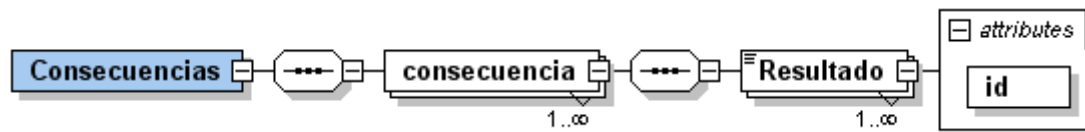


Imagen 70: FaseX.xml (Estancia / Opciones / Opcion / Consecuencias)

*consecuencia* → Deberá haber al menos una consecuencia. No tiene limite.

- Resultado → Una consecuencia esta formada por un conjunto de resultados.
  - o id → Atributo de “Resultado”, el cual indica a la aplicación que acciones debe llevar a cabo. Hay muchos posibles identificadores para un Resultado, podrá ver estos en el punto 7.8.2.13.

#### 5.4.2.10. Estancias / Estancia / Escenarios

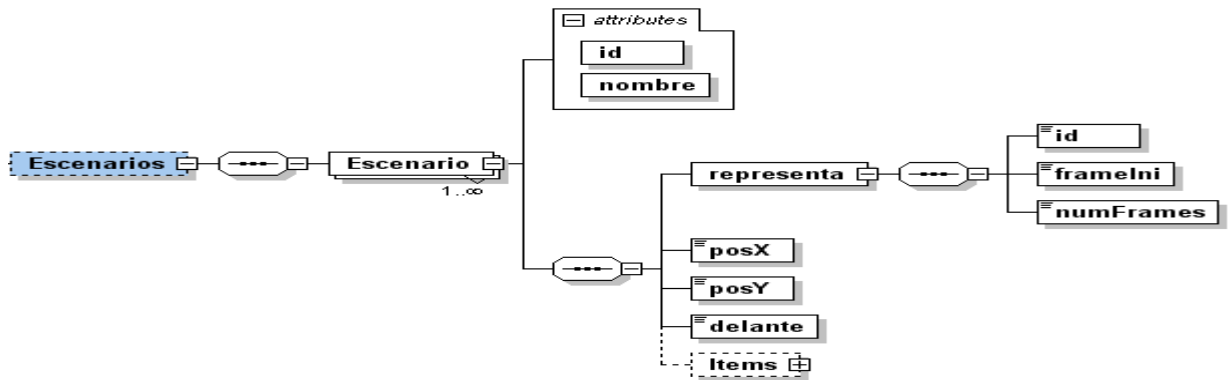


Imagen 71: FaseX.xml (Estancias / Estancia / Escenarios)

*Escenario* → Habrá entre 1 e infinitas estancias.

- id → Cada escenario tendrá un identificador, que será único, pero sólo para cada fase, es decir en la Fase 1 puede haber los escenarios con id 0,1 y 2, y en la Fase 2 podríamos encontrarnos los escenarios con id 0, 1 ...
- nombre → Atributo de “Escenario”. Este nombre no se mostrará en ningún momento, pero se mantiene ahí, ya que facilita la búsqueda de un escenario, cuando estamos programando.
- representa → Representación del escenario en el mapa.
  - o id → El id hará referencia a el identificador de la imagen que contiene el sprite del escenario. Dichas imagenes están definidas en basics.xml
  - o frameIni → Frame inicial, en un determinado sprite, a partir del cual está la imagen del escenario.



- o numFrames → Número de frames de las que está compuesto el sprite del escenario. La imagen de cada escenario podrá estar formado por 3 frames:
  - el 1º será la representación del escenario que se verá por pantalla delante del personaje (este es opcional).
  - el 2º es el escenario que se verá por detrás del personaje.
  - el 3º es el area caminable del escenario.
- posX → Valor de X para la posición donde aparecerá nuestro protagonista, en el escenario, nada más cargar el escenario. A este punto hay que sumarle el ancho que ocupe la imagen que representa al protagonista.
- posY → Valor de Y para la posición donde aparecerá nuestro protagonista, en el escenario, nada más cargar el escenario. A este punto hay que sumarle el alto que ocupe la imagen que representa al protagonista
- delante→ Indica si alguna parte del escenario estará por delante de nuestro personaje.
- Items → No es obligatorio que un escenario tenga ítems.



### 5.4.2.11. Escenarios / Escenario / Items

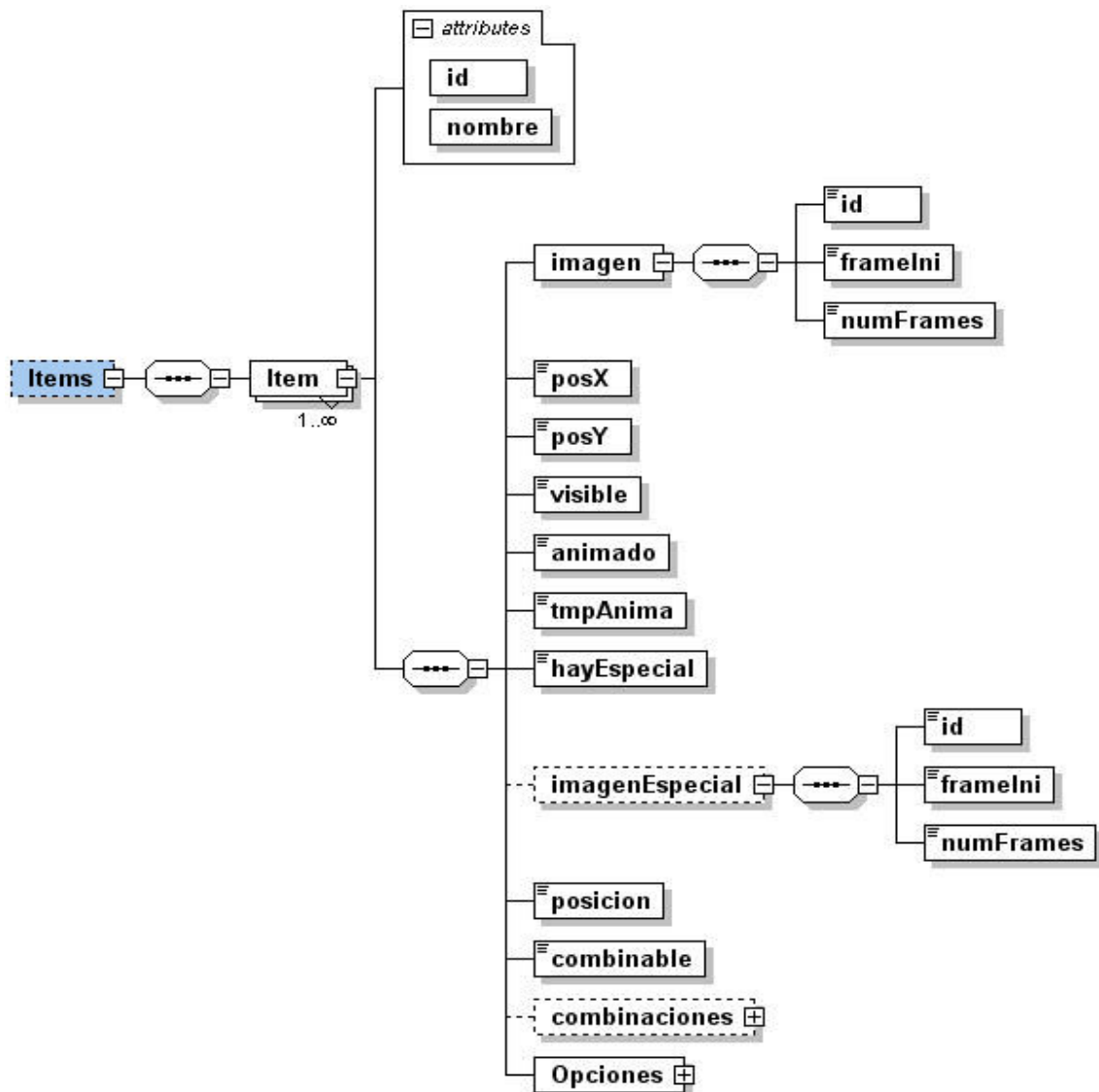


Imagen 72: FaseX.xml (Escenarios / Escenario / Items)

**Item** → Un escenario podrá tener infinitos ítems.

- **id** → Cada ítem tendrá un identificador, que será único, pero sólo para cada fase, es decir en la Fase 1 puede haber times con id 0,1 y 2, y en la Fase 2 podríamos encontrarnos con times con id 0, 1 ...
- **nombre** → Atributo de “Item”. Es el nombre que se nos mostrará cuando nos situemos, en la pantalla de juego o en el inventario, sobre el ítem.
- **imagen** → Representación de ítem en el escenario.
  - o **id** → El id hará referencia a el identificador de la imagen que contiene el sprite del ítem. Dichas imágenes están definidas en basics.xml



- o frameIni → Frame inicial, en un determinado sprite, a partir del cual está la imagen del ítem.
- o numFrames → Número de frames de las que está compuesto el sprite del ítem. Lo normal es que el número de frames sea mas de uno cuando el ítem es animado.
- posX → Valor de X para la posición donde se mostrará el ítem en el escenario
- posY → Valor de Y para la posición donde se mostrará el ítem en el escenario
- visible → Indica si el ítem será visible en el escenario.
- Opciones → Las opciones que se nos mostrarán, en la pantalla del mapa, al pulsar con el puntero sobre la estancia.
- animado → Indica si el ítem tiene algún tipo de animación. Se podrá poner SI o NO.
- tmpAnima → En el caso de que animado sea igual a SI, tmpAnima almacenará el tiempo que debe pasar entre un frame y otro del sprite.
- hayEspecial → Indica si el ítem tiene alguna imagen especial. Como puede ser el zoom de un personaje, o su representación en el inventario.
- imagenEspecial → En el caso de que hayEspecial sea SI, imagenEspecial guardará la información sobre esa imagen especial.
  - o id → El id hará referencia a el identificador de la imagen que contiene la imagen especial del ítem. Dichas imágenes están definidas en basics.xml
  - o frameIni → Frame inicial, en un determinado sprite, a partir del cual está la imagen especial del ítem.
  - o numFrames → Número de frames de las que está compuesto la imagen especial del ítem.
- posición → Indica si el ítem se verá delante o detrás del personaje principal, o si el ítem se verá delante o detrás del escenario. Sus posibles valores son:
  - o 0 → El ítem estará por detrás del escenario
  - o 1 → El ítem estará entre el escenario y el personaje principal
  - o 2 → El ítem estará por delante del personaje principal
- combinable → Indica si un ítem se puede combinar con otros ítems.
- combinaciones → En caso de que combinable sea igual a SI, aquí aparecerán esas posibles combinaciones.
- Opciones → Las opciones que se nos mostrarán al pulsar sobre el ítem. Como se puede ver en la última parte del diagrama (Imagen 37), opciones se puede



extender, pero como es igual que las Opciones de Estancia, os remitimos al punto 4.4.2.13, donde se puede ver de nuevo la explicación de la parte Opciones.

#### 5.4.2.12. Items / Item / combinaciones

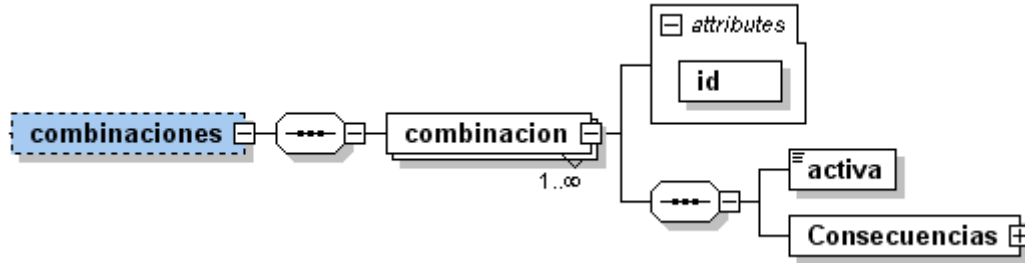


Imagen 73: FaseX.xml (Items / Item / combinaciones)

**combinación** → En caso de que el ítem sea combinable, deberá haber por lo menos una combinación.

- id → Atributo de “combinación”. En este atributo se especificará el identificador del ítem X, con el que el ítem, que estamos definiendo, puede combinarse dando como resultado unas consecuencias. Además, usaremos el id = -1, para definir las consecuencias de combinar, el ítem que estamos definiendo, con otro ítem que no esta entre las combinaciones.
- activa → Si está activa, esta combinación podrá ocurrir en el juego.
- consecuencias → Consecuencia de una combinación. Consecuencias ya se explico anteriormente, en el punto 4.4.2.12, la estructura de consecuencias, así que les remito a ese punto.

#### 5.4.2.13. Identificadores de resultados.

Como hemos indicado anteriormente, los resultados de una consecuencia, tienen un identificador, que es utilizado para decirle a la aplicación que debe hacer en cada momento. Pero todavía no hemos explicado la función y la forma de uso de cada uno de los identificadores permitidos para un “Resultado”.

**Salir** → Sale del juego.

**Mensaje** → Se muestra en una ventana de mensajes (imagen 7) el texto que ponemos en “Resultado”. Ejemplo,

`<Resultado id="Mensaje">Es la comisaría de la ciudad de Alborá. </Resultado>`

**Video** → Salta el video cuyo identificador es igual al que ponemos en “Resultado”. Ejemplo,

`<Resultado id="Video">2</Resultado>`





**Mapa** → Nos lleva al mapa cuyo identificador es igual al que le pasamos como parámetro.

**Conversación** → Nos lleva a la conversación cuyo identificador es igual al que le pasamos como parámetro. Este resultado tendrá que ser, en caso de que aparezca, el último, del total de los resultados, de la consecuencia a la que pertenece. Este resultado debe ir, en el caso de que se use, como último resultado en una consecuencia.

**Juego** → Nos lleva a la pantalla de juego. Es útil si por ejemplo estamos en una conversación, y la respuesta a una opción X es salir de esa conversación, entonces como consecuencia de dicha opción X, añadimos un resultado cuyo id sea igual a Juego, lo cual hará que salgamos de la conversación.

**Respuesta** → Se utiliza en las conversaciones, e indica la respuesta que, el personaje con el que hablamos, nos dirá cuando le hagamos una determinada pregunta o frase (Opcion).

**RespuOscar** → Se utiliza en las conversaciones, e indica la respuesta que, nosotros mismos, haremos cuando le hagamos una determinada pregunta o frase (Opcion) al personaje con el que hablamos.

**cargaOpSup** → Se utiliza en las conversaciones, y su función es cargar las opciones superiores a una determinada opción. El número de niveles que sube se pasará por parámetro.

Ejemplo, Si desde la opción con id “aaa” (a la que hemos llegado tras elegir la opción “a” y luego la “aa”) queremos subir al grupo de “a, b, c”, es decir, subir dos niveles, haríamos:

```
<Opcion id="aaa" texto="lo que sea">
  <activa>SI</activa>
  <Consecuencias>
    <consecuencia>
      <Resultado id="cargaOpSup">2</Resultado>
    </consecuencia>
  </Consecuencias>
</Opcion>
```

**cargaOpInf** → Se utiliza en las conversaciones, y su función es cargar las opciones inferiores de una determinada opción. El número de niveles que baja se pasará por parámetro.

**quitarOp** → Se utiliza en las conversaciones. Como parámetro se le pasaría, separado por punto y coma, el id de la conversación donde está la opción a quitar, y el id de dicha opción.

Ejemplo, para eliminar la opción con id igual a ‘ab’, de la conversación con id igual a ‘0’:

```
<Resultado id="quitarOp">0;ab</Resultado>
```



**quitarOpYSup** → Se utiliza en las conversaciones, y es una combinación de “cargaOpSup” y “quitarOp”. Por parámetro se le pasará, el id de la conversación a la que afecta, el id de la opción a quitar, y el numero de niveles a subir, todo ellos separado por punto y coma.

Entonces lo que hará será borrar, de la conversación con id igual al pasado por parámetro, la opcion cuyo id es igual al pasado por parámetro. Y luego, en caso de que ya no quede ninguna opción inferior, para la opción superior de la pasada por parámetro, entonces se borrará está ultima, y seguirá subiendo hasta el numero de niveles indicado.

Ejemplo, *Eliminar la opción ‘cc’ de la conversación ‘0’, subir 1 nivel (opciones a, b, c...) y en caso de que ya no haya más opciones cuyo numero de letras del id sea igual a dos y comiencen por c, se procederá a eliminar la opción cuyo id es igual a ‘c’:*

```
<Opcion id="cc" texto="Ah vale.">
  <activa>SI</activa>
  <Consecuencias>
    <consecuencia>
      <Resultado id="quitarOpYSup">0;cc;1</Resultado>
    </consecuencia>
  </Consecuencias>
</Opcion>
```

**salirJuego** → Sale del juego.

**Caminar** → Se utiliza en la pantalla de juego, y sirve, para indicar a nuestro personaje que se mueva desde su posición, hasta la señalada por el puntero. Todo ello sin bloquear el hilo de ejecución, es decir, el resto de tareas se seguirán ejecutando.

**CaminarBlo** → Igual que el anterior, solo que este si bloquea.

**CambiaEscenario** → Utilizado para cambiar de un escenario a otro, pasándole como parámetro, al resultado, el identificador del escenario al que queremos pasar.

**camMapa** → Igual que el anterior, pero aplicado al cambio de un mapa a otro.

**irEstancia** → Utilizado en el mapa, sirve para indicar que queremos ir a la estancia, cuyo identificador hemos pasado por parámetro.

**combinando** → Indica a la aplicación que se está procediendo a intentar combinar un objeto con otro.

**camFase** → Pasa a la fase siguiente. Además si quieres que una vez en la siguiente fase, el personaje, por ejemplo, tenga tales objetos (ya que al cambiar de fase se borra todo) y muestre un mensaje, tendrás que añadir los resultados addInventario y Mensaje después de poner el camFase, y todo dentro de la misma consecuencia.

**addInventario** → Añade al inventario el ítem cuyo id se ha pasado por parámetro.

**addInventarioSin** → Igual que el anterior, solo que en este caso no muestra el inventario.



**delInventario** → Borra del inventario el ítem cuyo id se ha pasado por parámetro.

**camActivaOpIt** → Para el ítem cuyo id coincide con el pasado por parámetro, se le activa (en caso de que este desactivada) o viceversa, la opción cuyo id también es pasado por parámetro.

Ejemplo, *imaginar un ítem con id = 21, y cuya opción con id = 3 está activada. Nosotros queremos desactivar esa opción:*

```
<Resultado id="camActivaOpIt">21;3</Resultado>
```

**camActivaOpEs** → Igual que la anterior, pero aplicado a estancias.

**camActivaOpCon** → Igual que las dos anteriores, pero aplicado a conversaciones.

**eliminarItem** → Eliminar el ítem cuyo id se ha pasado por parámetro.

**eliminarEstan** → Eliminar la estancia cuyo id se ha pasado por parámetro.

**camNombreIt** → Al ítem cuyo id coincida con el pasado por parámetro, se le cambia el nombre.

Ejemplo, *Al ítem cuyo id es igual a 2, le vamos a cambiar el nombre a “Farola rota”.*

```
<Resultado id="camNombreIt">2;Farola rota</Resultado>
```

**camNombreEs** → Igual que el anterior, pero aplicado a estancias.

**camVisibleIt** → Si el ítem, cuyo id coincide con el pasado por parámetro, es visible, entonces pasa a ser NO visible. Y viceversa.

**camVisibleEs** → Igual que el anterior, pero aplicado a estancias.

**camConsecuIt** → Cambia la consecuencia de utilizar una opción de un ítem. Simplemente hay que indicarle el id del ítem, el id de la opción, y el numero de la consecuencia que queremos poner.

Ejemplo, *A un ítem, cuyo id es 26, le queremos cambiar la opción con id igual a 0, para ponerle la consecuencia 2.*

```
<Resultado id="camConsecuIt">26;0;2</Resultado>
```

**camConsecuEs** → Igual que el anterior, pero aplicado a estancias.

**camConsecuCom** → Igual que el anterior, pero aplicado a una combinación. Lo cual implica que en vez de indicar el id de la opción, habrá que indicar el id de la combinación.

**camConsecuCon** → Igual que el camConsecuIt o camConsecuEs, pero aplicado a una conversación.

**camActivaCom** → Activa o desactiva, para un determinado ítem, una combinación.

Ejemplo, *Activar para el ítem 32, su combinación con el ítem 43 (la cual está desactivada).*



```
<Resultado id="camActivaCom">32;43</Resultado>
```

***camApariencia*** → En básicos.xml definiremos las distintas apariencias que podrá tener el protagonista. Para cambiar entre una y otra usaremos *camApariencia*, pasándole como parámetro el id de la apariencia a usar.



## 6. Implementación

### 6.1. Tecnología

#### 6.1.1. Lenguaje de programación

##### 6.1.1.1. Elección

Esta decisión fue de las primera decisiones que se tomó, y en ese momento barajé dos opciones:

**Symbian** → Lenguaje de programación basado en C.

- A favor → Lenguaje utilizado por la mayoría de la industria de los videojuegos, debido a que es más eficiente que Java en cuanto a velocidad, y esta, en algunos géneros, es imprescindible. Permite crear y modificar ficheros externos.
- En contra → Estoy menos familiarizado en C que en Java, pero sobretodo Symbian, pese a su prometedor futuro, todavía no está demasiado extendido.

**J2ME** → Tecnología Java.

- A favor → El esfuerzo necesario para adaptarme a J2ME es menor que para Symbian, puesto que conozco mejor Java que C. Además J2ME es una tecnología extendida en prácticamente todos los móviles que actualmente hay en el mercado.
- En contra → Es menos eficiente que Symbian, y el emulador Java no permite modificar ni crear ficheros.

Así que teniendo en cuenta que la velocidad no es algo imprescindible en una aventura gráfica, un pequeño retraso no afecta nada, y que, en principio, quiero que el móvil pueda ser ejecutado en el mayor numero de móviles posible, he decidido utilizar J2ME.

##### 6.1.1.2. Java 2 Platform, Micro Edition (J2ME)

Esta versión de Java está enfocada a la aplicación de la tecnología Java en dispositivos electrónicos con capacidades computacionales y gráficas muy reducidas, tales como teléfonos móviles, PDAs o electrodomésticos inteligentes. Esta edición tiene unos componentes básicos que la diferencian de las otras versiones, como el uso de una máquina virtual denominada KVM (Kilo Virtual Machine, debido a que requiere sólo unos poco Kilobytes de memoria para funcionar) en vez del uso de la JVM clásica.



Imagen 74: Arquitectura de la plataforma Java 2 de SUN

Un entorno de ejecución determinado de J2ME se compone entonces de una selección de:

**Máquina virtual (KVM)** → Existen 2 configuraciones CLDC y CDC, cada una con unas características propias que veremos en profundidad más adelante. Como consecuencia, cada una requiere su propia máquina virtual. La VM (Virtual Machine) de la configuración CLDC se denomina KVM y la de la configuración CDC se denomina CVM.

En mi caso, teniendo en cuenta que los dispositivos a los que va orientado el juego son los móviles, me interesa la Máquina Virtual más pequeña desarrollada por Sun. Su nombre KVM proviene de Kilobyte (haciendo referencia a la baja ocupación de memoria, entre 40Kb y 80Kb). Se trata de una implementación de Máquina Virtual reducida y especialmente orientada a dispositivos con bajas capacidades computacionales y de memoria. La KVM está escrita en lenguaje C.

**Configuración.** → J2ME contiene una mínima parte de las APIs de Java. Esto es debido a que la edición estándar de APIs de Java ocupa 20 Mb, y los dispositivos pequeños disponen de una cantidad de memoria mucho más reducida. En concreto, J2ME usa 37 clases de la plataforma J2SE provenientes de los paquetes java.lang, java.io, java.util. Esta parte de la API que se mantiene fija forma parte de lo que se denomina “configuración”.

Las configuraciones, son un conjunto de clases básicas orientadas a conformar el corazón de las implementaciones para dispositivos de características específicas. Existen 2 configuraciones definidas en J2ME:



- Connected Limited Device Configuration (CLDC) → Enfocada a dispositivo con restricciones de procesamiento y memoria.
- Connected Device Configuration (CDC) → Enfocada a dispositivos con más recursos.

Yo utilizaré la CLDC.

**Perfil** → El perfil establece unas APIs que definen las características de un dispositivo, mientras que la configuración hace lo propio con una familia de ellos. Esto hace que a la hora de construir una aplicación se cuente tanto con las APIs del perfil como de la configuración. Tenemos que tener en cuenta que un perfil siempre se construye sobre una configuración determinada. De este modo, podemos pensar en un perfil como un conjunto de APIs que dotan a una configuración de funcionalidad específica.

Anteriormente vimos que para una configuración determinada se usaba una Máquina Virtual Java específica. Teníamos que con la configuración CDC usábamos la CVM y que con la configuración CLDC usábamos la KVM. Con los perfiles ocurre lo mismo. Existen unos perfiles que construiremos sobre la configuración CDC y otros que construiremos sobre la CLDC.

Para la configuración CDC tenemos los siguientes perfiles:

- Foundation Profile.
- Personal Profile.
- RMI Profile

Y para la configuración CLDC tenemos:

- PDA Profile.
- Mobile Information Device Profile (MIDP).



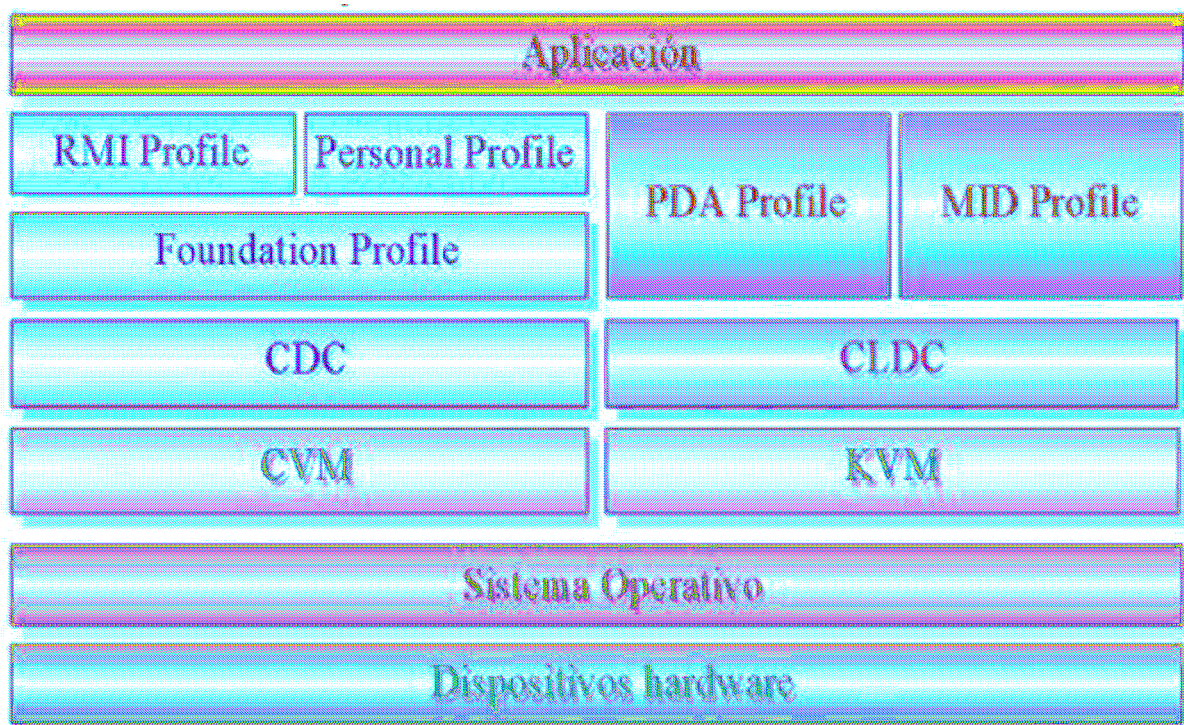


Imagen 75: Arquitectura del entorno de ejecución J2ME

Como perfil utilizaré MIDP, ya que sus características se adaptan a los teléfonos móviles.

**Paquetes Opcionales** → Paquetes opcionales que añadas tu a la aplicación.

### 6.1.1.3. Resumen

Por lo tanto como resumen del lenguaje de programación podríamos establecer que se va a usar:

- Java en su versión limitada J2ME.
- Como VM (Virtual Machine) utilizaremos la KVM.
- La configuración será la CLDC, en su versión 1.0.
- El perfil el MIDP, en su versión 2.0.

### 6.1.2. Entorno de desarrollo

A la hora de elegir la herramienta de desarrollo me planteé las siguientes opciones:

**J2ME Wireless Toolkit 2.0** → Es simplemente un emulador al que le proporcionamos las clases Java ya creadas y podemos ver el *MIDlet* (las aplicaciones, creadas con el perfil MIDP, se denominan MIDlet) en ejecución.





***Sun One Studio Mobile Edition*** → Es un entorno de desarrollo de Java con un emulador integrado. Este entorno es exactamente igual al *Sun One Studio*, pero incluye un emulador con el que podemos ver la ejecución de nuestros *MIDlets*, además de incluir las APIs propias de la configuración CLDC y el perfil MIDP. Este entorno fue utilizado en las practicas de una asignatura optativa.

***EclipseME*** → EclipseME es un plugin para Eclipse que facilita la integración de los distintos kits de desarrollo J2ME con este IDE, aprovechando su gestión de proyectos, herramientas de desarrollo y depuración dentro de emuladores J2ME.

***JBuilder Mobile Edition*** → Al igual que el resto, ofrece un entrono de desarrollo Java con un emulador integrado. El entorno de desarrollo es el de JBuilder. Lo bueno que tiene este IDE es que permite crear interfaces de alto nivel para el móvil de forma sencilla, simplemente arrastrando componentes en una pantalla de diseño.

Y finalmente de entre estas opciones, me decidí por EclipseME, ya que me encanta el entorno de desarrollo Eclipse, es gratuito, fácil de usar y hay gran cantidad de información sobre este.

### 6.1.3.Librerías y código externo

A continuación se exponen aquellas librerías y trozos de código, no generados por mí, y que son parte esencial del juego

#### 6.1.3.1. Librería KXML

A la hora de decidir que parser que iba a utilizar para el XML, me marque como objetivo que este ocupase poco y fuese fácil de usar. Algunas posibilidades eran: NanoXML, TinyXML o MinML, pero finalmente el que más me convenció fue el KXML.

KXML es un pequeño parser de XML, especialmente diseñado para entornos limitados, como MIDP. Me decidí por usar KXML porque:

- Su facilidad de uso → Aun siendo la primera vez que he usado un parser, no he tenido demasiados problemas en conseguir que funcione correctamente.
- Su escaso tamaño → Y es que esta librería solo ocupa 9 KB.
- Gran cantidad de información → Es muy fácil encontrar documentación, ejemplos e información, sobre KXML, en Internet.

#### 6.1.3.2. Árbol AVL

Como dije en el apartado 3.8, he elegido, como estructura de datos principal, los árboles AVL debido a que son más eficientes, pudiendo conseguir complejidades logarítmicas, para las operaciones que a mí me interesan.



Supuse que una estructura tan común como esta ya estaría implementada en algún sitio, y como alguna vez se me dijo durante la carrera, si algo ya está implementado y esta a disposición de todo el mundo, ¿porque volver a hacerlo?.

El código que usé, y que se puede ver en el apéndice A, lo hallé en el libro “Introduction to Algorithm (2nd edition), McGraw-Hill (2001), ISBN: 0070131511”.

### 6.1.3.3. Algoritmo A\* (A Estrella)

Nuestro protagonista, que se ve en pantalla en 3ª persona, podrá moverse por el escenario simplemente pulsando con el puntero en un determinado punto. El algoritmo de búsqueda de caminos, encargado de encontrar la ruta desde la posición de nuestro personaje hasta el punto en el que hemos pulsado con el puntero esquivando los obstáculos que encuentra en su camino, es una decisión que hay que hacer comúnmente en los juegos (ya sea una persona, un coche, un animal...).

Yo como algoritmo de búsqueda decidí usar el A\*, el cual es un algoritmo de búsqueda inteligente que busca el camino más corto desde un estado inicial al estado meta a través de un espacio de problema, usando una heurística óptima. Como ignora los pasos más cortos en algunos casos rinde una solución subóptima.

Pero un algoritmo de búsqueda (Path Finder) necesita saber cual es el área caminable de una pantalla, o en el caso de este juego, escenario. Para especificar el área caminable hice que cada escenario estuviese formado por al menos 2 frames, uno que será el de la imagen (dibujo del escenario) y otro que representará el área caminable de dicho escenario.

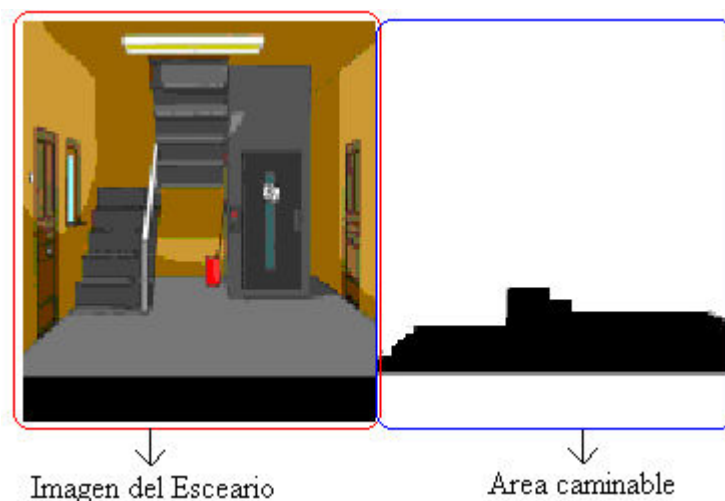


Imagen 76: Escenario con su area caminable



## 6.2. Guía de estilo de programación

A continuación expongo mi estilo de programación, este se basa, principalmente, en el estilo que se uso durante la carrera, aunque hay algún cambio:

- Se sangrará adecuadamente cada nuevo bloque de sentencias. Por ejemplo,

```
private int getCeldaX(int pixel){  
    return pixel/anchoCelda;  
}
```

- En esta aplicación en concreto, las distintas clases no se dividirán en paquetes, ya que aunque no se puede definir como una buena técnica, es una buena forma de ahorrar memoria, algo básico en aplicaciones para móviles.
- Pero si alguna clase forma parte de un paquete, la sentencia package deber ser la primera del fichero.
- Si se importan paquetes, la sentencia import debe aparecer después de la sentencia package.
- Después de lo imports se pondrá la declaración de la clase.
- La declaración irá seguida de los atributos de la clase, y estos a su vez de los metodos de la clase.
- Los nombres de las clases tendrán su primera letra en mayúsculas y el resto en minúscula, a no ser que sean una palabra compuesta, en ese caso estará en mayúscula la primera letra de cada palabra que conforma el nombre de la clase.
- Los nombres de los métodos estarán en minúsculas, a no ser que sean una palabra compuesta, en ese caso la primera letra del nombre del método estará en minúscula, y el resto de las palabras que forman ese nombre compuesto, tendrán su primera letra en mayúscula.
- Los atributos, ya sean locales o globales, estarán en minúsculas, a no ser que sean una palabra compuesta, en ese caso la primera letra del nombre del método estará en minúscula, y el resto de las palabras que forman ese nombre compuesto, tendrán su primera letra en mayúscula.
- Se intentará evitar el uso de get/set para recuperar el valor de los atributos globales de una clase, ya que es más rapido.
- Los nombre de los iteradores serán i, j, k, etc.
- No utilice la sentencia break excepto en la sentencia switch, donde es forzoso hacerlo. Nunca use la sentencia continue.



- Coloque la apertura de bloque ( { ) al final de la línea inicial de la sentencia. El fin de bloque ( } ) colóquelo en línea aparte y alineado con el principio de la sentencia. Por ejemplo,

```
static public int getSpaceItem(){  
    if(grupo==0)  
        return spaceSpaceItemPeq;  
    else  
        return spaceSpaceItemGra;  
}
```

- En la clase Utils.java hay un conjunto de atributos que representan las imágenes básicas del juego. El nombre de estos atributos ira precedido de ‘im’ si la imagen solo tiene un frame, o de ‘sp’ si la imagen es un sprite (conjunto de frames).
- En Utils.java hay un conjunto de variables estaticas, que contienen los valores de los distintos resultados a una determinada acción u opción. Estas variables empezarán por ‘re’.
- En el caso de querer usar un String, el cual va a tener varios cambios de valor, se usará StringBuffer, ya que sino cada vez que cambiemos de valor al String, se nos creará una nueva variable String, y desperdiciaremos memoria. Esto no pasará si usamos StringBuffer.
- En los arrays se limitarán los accesos a memoria.
- En los bucles, sacar todas aquellas operaciones que se repiten y son constantes.
- En J2ME no hay recolector de basura, así que asigna valor nulo a los objetos y threads que no uses, y cada cierto tiempo llama a System.gc
- Todo el código estará documentado en estilo javadoc.

### 6.3. Estructura de ficheros

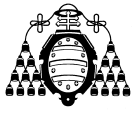
La estructura de ficheros del proyecto, que podremos encontrar dentro del cd que acompaña la documentación, es la siguiente:

- Entre\_2\_Almas → Directorio principal del proyecto.
  - o Ejecutable → Entre2Almas.jar y Entre2Almas.jad forman parte de este directorio.
  - o Imágenes → Todas las imágenes del proyecto irán incluidas aquí.
  - o Código → Todos los .java que forman el código fuente.

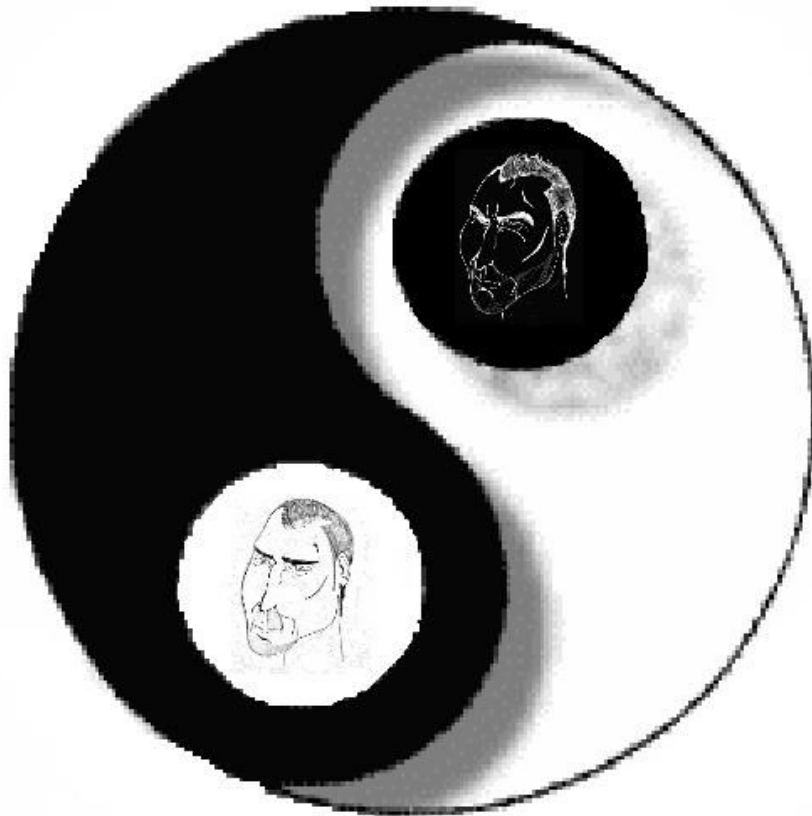


- o Configurables → XMLs y sus correspondientes XSDs.
- o Documentación → Documentación del proyecto.
- o Emulación → En este directorio estan todos aquellos programas, que a fecha de hoy, son necesarios para emular el juego en el ordenador.





## 7. Manual



# ENTRE 2 ALMAS

Manual de instrucciones



## 7.1. Requisitos mínimos

- **Dispositivo:** Teléfono móvil o emulador.
- **Java:** Java con MIDP 2.0 y CLDC 1.0.
- **Pantalla:** Mínimo de 128x128 píxeles.
- **Memoria de almacenamiento:** 500 KB
- **Memoria de trabajo:** 500 KB

## 7.2. Objetivo del juego

Entre 2 Almas es una aventura gráfica, escrita en tono de humor, que te llevará a la ciudad de Alhora a vivir la aventura de Oscar Romero.

Todo comienza con el atropello de Oscar, en un paso de cebra, y su misteriosa y rápida recuperación cuando ya le daban incluso por muerto. Una vez en el hospital se duerme tranquilo, pero a la mañana siguiente algo no va bien, ¡¡ Esta en un calabozo !!

A partir de aquí deberás ayudar a Oscar en multitud de puzzles y conversaciones que le ayuden a salir de la cárcel y descubrir que es lo que está pasando, cual es el misterio.

Todo esto aderezado por unos impresionantes gráficos 2D en color, dos tipos de menús a elegir, más de 80 objetos con los que interactuar, 3 personajes con los que relacionarte... PREPÁRATE PARA LA AVENTURA!!

## 7.3. Instalación

Debido a que el juego no esta disponible para descargar vía OTA, este deberá ser pasado a su móvil desde el ordenador, por ejemplo por bluetooth.

El ejecutable del juego, que será el que tengan que pasar a su móvil, lo pueden encontrar en la carpeta “Ejecutable” dentro del CD que acompaña esta documentación. Una vez en esta carpeta, deberá mover el archivo “Entre2Almas.jar” a su móvil, donde ya podrá ser ejecutado.

Otra forma de jugar sería a través de un emulador móvil para PC. En ese caso deberá de:

- Disponer de Microsoft Windows XP o (soportado) o Linux





- Ir a la carpeta “Emulación”, incuida en el CD que acompaña esta documentación, y proceder a ejecutar el archivo “jdk-1\_5\_0\_04-windows-i586-p-iftw.exe”.
- En la misma carpeta, y una vez instalado lo anterior, deberá ejecutar el archivo j2me\_wireless\_toolkit-2\_2-windows.exe.
- Una vez hecho esto, vaya a la carpeta Ejecutable, y ejecute el archivo “Entre2Almas.jad”. Y el juego ya estará corriendo en su ordenador.

## 7.4. El menú



Imagen 77: Menu principal (Avanzado)

Nada más ejecutar el juego, y después de pasar la pantalla de presentación, llegaremos al menú principal avanzado (ya hablaremos de los dos tipos de menús más adelante), el cual nos ofrecerá 5 opciones, que son:

**Nueva partida:** Nos permite iniciar una nueva partida de Entre 2 Almas.

**Memoria:** Nos lleva al menú de cargar y guardar partida. Donde podremos cargar nuestras partidas guardadas, o guardar la partida en curso.

- *Grabar* → Grabas tu estado actual y tu situación en el mapa, para así poder reanudar mas tarde en el mismo sitio en el que lo dejaste. Una vez guardada,



sigues jugando la partida en el mismo punto en el que lo dejaste cuando entraste al menú.

- **Cargar** → Cargas un estado y tu situación en el mapa de un tiempo pasado, para así poder reanudar en el mismo sitio en el que lo dejaste. Una vez cargada la partida, aparecerás en la pantalla de juego (no en el menú).

**Opciones:** Nos lleva al menú de opciones, donde se podrá elegir entre los dos tipos de menús disponibles.

**Ayuda:** Nos lleva a la ayuda del juego.

**Acerca de...:** Nos da información sobre la versión y los desarrolladores del juego.

Además si estamos jugando una partida, al salir al menú también se nos ofrecerá la opción:

**Seguir Jugando:** Nos permite seguir jugando la partida en curso.

Ya se ha comentado un par de veces que existen dos tipos de menús, esto se hace con la idea de que el jugador pueda elegir el tipo de menú que más se adapte a su gusto. El cambio entre uno u otro menú se realiza fácilmente a través de la pantalla de opciones, y una vez que el usuario selecciona uno, su elección se queda en la memoria del teléfono para que así, la próxima vez que vuelva a entrar al juego, no tenga que volver a tomar esta decisión. Si quisiese cambiar el tipo de menú, simplemente tendría que pasar de nuevo por las opciones. Los tipos de menús son::

**Avanzado.** (Imagen 76) Menú adaptado para el juego, con imágenes creadas para este. Este tipo de menú será igual para todos los móviles.

**Clásico.** Este tipo de menú mantendrá el formato y colores de los menús del propio móvil, es decir, será un poco distinto para cada tipo de móvil. El realizar interfaces parecidos a los nativos del propio móvil es una práctica recomendable.

## 7.5. Controles

**Botones de dirección.** Se usará el propio joystick del móvil o las teclas: ‘2’ (arriba), ‘8’ (abajo), ‘4’ (izquierda), ‘6’ (derecha). Estos botones no moverían al personaje, sino que moverían un puntero en torno a la pantalla (como el ratón hace en las aventuras gráficas del PC). Cuando estemos dentro del inventario, usaremos izquierda y derecha para movernos por los distintos objetos.

**Botón de acción ‘5’.** Serviría para numerosas acciones, como elegir que acción quieres realizar sobre un objeto o seleccionar un objeto del inventario o elegir que frase quieres decirle a un NPC...

**Botón de inventario ‘\*’.** Sirve para entrar o salir del inventario. En dicho inventario estarán todos los objetos de los que disponen nuestro personaje. Para movernos a través



de los distintos objetos utilizaríamos los botones de dirección izquierda y derecha, y para seleccionar uno usaríamos la tecla de acción.

**Botón para entrar en el menú ‘#’ o ‘Acceso directo derecho2’.** Entraríamos en el menú. Nos moveríamos a través de sus opciones con los botones de dirección ‘2’ (arriba) o ‘8’ (abajo) y para seleccionar utilizaríamos el botón de acción.

## 7.6. Pantallas

En el juego podremos encontrarnos 5 tipos de pantallas básicas, en las cuales, la forma de interactuar con la maquina cambian, al igual que también cambia la información que esta nos envía.

### 7.6.1.En un escenario

Como pantalla del juego entendemos aquella en la que se nos está mostrando un escenario, en el cual hay una serie de personajes y objetos con los cuales podemos interactuar. En ese momento en la pantalla nos encontraremos la siguiente información:

**Zona de juego:** (Ver imagen 5) Se muestra el escenario, en el que esta nuestro personaje, y todo lo que en él hay. También se mostrará el puntero (el cual manejamos con los botones de dirección), en el caso de que nuestro puntero este señalando algún objeto o personaje del juego sobre los que se puede realizar alguna acción, este cambiará de color y en la zona de texto de la pantalla, aparecerá el nombre del objeto o del personaje. Si no estamos señalando a nada importante, el puntero retomará su color original y en la zona de texto aparecerá “ Ir a...” queriendo representar que si pulsas el botón acción iras a este lugar.

**Zona de texto:** (Ver imagen 5) Una línea de texto donde aparecerá información sobre objetos o personajes que estén siendo señalados por el puntero. Si el puntero no señala nada en la zona de texto aparecerá “Ir a...”.

**Zona de inventario:** (Ver imagen 5) En caso de que pulses el botón ‘\*’, entras en el inventario. El inventario se muestra justo encima de la zona de texto, en la parte izquierda de la pantalla, mostrará solo un objeto de los disponibles y en caso de que no haya ninguno aparecerá el texto “NADA”. Por el inventario nos moveremos con las teclas de dirección izquierda y derecha y seleccionaremos el objeto deseado con el botón de acción (lo cual implica que estas teclas ya no manejaran el puntero de la zona de juego hasta que volvamos a pulsar ‘\*’). Una vez seleccionado, realizada la acción que queríamos hacer sobre los objetos que teníamos en el inventario o si simplemente queremos volver a manejar el puntero de la zona de juego, pulsaremos de nuevo la tecla ‘\*’.

---

<sup>2</sup> Acceso directo → Son los botones situados en las esquinas superiores del móvil, en este caso sería el botón situado en la esquina superior derecha del móvil .



**Opciones de un objeto:** (Ver imagen 6) Cuando tenemos el puntero situado sobre un Ítem, o cuando estamos mirando un objeto del inventario, el nombre del ítem aparecerá en la zona de texto, pero si además, pulsamos el botón de acción, nos aparecerá una lista de opciones para dicho ítem. Estas opciones ocupan la esquina inferior derecha de la pantalla, justo encima de la zona de texto. Una vez ahí usaremos las teclas arriba y abajo para movernos entre las opciones de ese objeto y elegiremos una con el botón de acción. En caso de que no queramos realizar ninguna acción, usaremos la opción ‘Cancel’.

**Ventana de mensajes:** (Ver imagen 6) Ventana que servirá para que el protagonista de Entre 2 Almas, Oscar Romero, se comuniquen con nosotros. Así cuando este nos quiera decir algo, esta ventana se abrirá y en ella aparecerá el texto que se nos quiera decir. Para deslizarnos a través del texto, usaremos las teclas de dirección arriba y abajo, y para cerrar la ventana de mensajes usaremos el botón de acción.

## 7.6.2.En una conversación

Durante el juego, tendremos la oportunidad de hablar con algunos personajes, en ese momento la pantalla se dividiría en 3 partes, la zona de imagen (que a su vez se dividiría en 2 partes y en ella irían las fotos de los que hablan,) y la de texto (Ver imagen 7).

En la parte izquierda-arriba de la pantalla aparecería el busto de nuestro personaje y un color de fondo, en la parte derecha-arriba de la pantalla aparecería el busto del NPC con el que hablamos y otro color de fondo. Cuando un personaje habla se podrá ver las distintas expresiones de su rostro, y los comentarios de cada personaje irían acompañados de su correspondiente color de fondo.

En la parte de abajo se iría mostrando lo que el NPC nos dice y cuando nos toque hablar a nosotros, desaparecerá lo que él nos dijo y en su lugar se nos mostrara las posibles respuestas, con los botones de dirección izquierda o derecha nos movemos entre ellas y con el botón de acción seleccionamos la respuesta que queremos decir, entonces aparecerá esa respuesta sola indicando que se la está diciendo.

Para salir de la conversación habrá que seleccionar una respuesta que tenga connotación de despedida o que indique que la conversación ha terminado, Ej. : “Hasta luego”.

## 7.6.3.En el mapa

Cuando vayamos a desplazarnos de una zona a otra, entre las cuales se supone hay mucha distancia, aparecerá en pantalla y a tamaño completo, un mapa con todos los puntos a los que podemos ir y el nombre de esos puntos. Para movernos entre los distintos puntos moveremos el puntero, y al situarnos sobre una localización a la que podemos desplazarnos, el puntero cambia de color, si pulsamos entonces el botón de



acción, nos aparecerán una serie de opciones, entre las que estará Ir a... que sirve para ir a ese escenario.

(Ver imagen 8)

#### 7.6.4.En un video

En determinados momentos de la partida, saltarán videos para representar una situación. Pero estos no serán videos realmente, sino que serán una sucesión de viñetas con un texto explicatorio. Para deslizarlos, a través del texto explicatorio, utilizaremos las teclas arriba y abajo, y para continuar con la siguiente viñeta, pulsaremos el botón de acción.

(Ver imagen 9)

### 7.7. Créditos

Proyecto final de carrera para la EUITIO, desarrollado en el GameLab.

#### 7.7.1.Equipo de desarrollo

- *Idea original, guión y programación* → Pedro Javier Sáez Martínez.
- *Grafista* → Juan Manuel Prada [[www.jmprada.com](http://www.jmprada.com)]

#### 7.7.2.Agradecimientos

- Víctor Lobo
- Rubén Sáez
- Pedro Sáez
- Yayo





## 8. Pruebas

Una de las últimas fases del ciclo de vida antes de entregar un programa para su explotación, es la fase de pruebas.

Una de las sorpresas que me ha deparado esta fase es que lleva un tiempo considerable. Y es que mi objetivo era encontrar todos los errores posibles, lo cual te hace repetir acciones montones de veces, pasar por estados que ya conoces a la perfección, hasta que de repente hay un fallo. Eso significará una buena noticia, ya que vale más que te ocurra a ti, que no a un usuario final, pero arreglar ese fallo puede ser desde muy fácil hasta complicadísimo, entre los que está los fallos fantasma, que aparecen y desaparecen sin darnos cuenta.

El número de pruebas realizadas, desde que comenzó el proyecto, son muy numerosas, y sería demasiado extenso poner todas y cada una de ellas, así que mejor me centraré en las última fases de pruebas, la fase alfa y la fase beta.

### 8.1. Alfa

#### 8.1.1.Objetivos

Las pruebas alfa consisten en invitar al cliente a que venga al entorno de desarrollo a probar el sistema. Se trabaja en un entorno controlado y el cliente siempre tiene un experto a mano para ayudarle a usar el sistema y para analizar los resultados.

En este caso los clientes fueron varios amigos míos, y probaron el juego en mi móvil, cuando el juego sólo tenía 1 fase, pero suficiente para probar todas las opciones de la aplicación. Yo en todo momento estuve con ellos mientras lo jugaban, y les ayudaba a entender como jugar.

#### 8.1.2.Casos de prueba

##### 8.1.2.1. Pruebas de interfaz.

PRUEBA:

Se prueba a cambiar de forma continua de menú clásico a menú avanzado, y esto llega a producir una excepción de falta de memoria.

SOLUCIÓN:



Se comprueba con un profiler, el % de uso de memoria que hace cada método, y se descubre que se producen picos de consumo importantes durante toda la ejecución.

La optimización es una parte muy importante en una aplicación J2ME, ya que una mala programación puede acabar, rápidamente, con la memoria de trabajo del móvil. Lo mejor es empezar por mejorar el diseño de tu código y de los algoritmos, lo cual significa, optimización de alto nivel. Esto consigue incrementar el rendimiento de ejecución en muchas plataformas, e incluso mejora la calidad del código.

Gracias a esta optimización, se soluciona el problema de la excepción por falta de memoria.



Imagen 78: Vista del menú avanzado y menú clásico

#### PRUEBA:

Uno de los usuarios que está probando el juego, se queda bloqueado cuando, en un momento del juego, se le introduce en el inventario y no sabe salir de él, ni si quiera cerrar el juego.

-----





**SOLUCIÓN:**

Se decide poner un acceso directo al menú, bien visible, para que en cualquier momento sepan como ir al menú principal.

Esto cumpliría una doble misión, por una parte los jugadores sabrían como salir del juego, y por otra tendrían más a mano la opción de mirar la ayuda, donde se les explica como usar el inventario.

**PRUEBA:**

Uno de los probadores me comenta que la ayuda es demasiado extensa, y que cuesta encontrar la información que se busca.

**SOLUCIÓN:**

Se decide dividir la ayuda en apartados, de esta forma se podrá ir directamente a lo que se busca.

**PRUEBA:**

A los probadores les tengo que ayudar demasiado en algunos momentos puntuales, como a la hora de pasar de una viñeta a otra (no saben que hay que pulsar el botón de acción), pero sobre todo tienen problemas en las conversaciones, no saben cuando dar al botón de acción y cuando no.

**SOLUCIÓN:**

A partir de ese momento se introducen una serie de palabras aclaratorias de lo que hay que hacer, tanto en los videos, como en las conversaciones y en las pantallas de mensajes.



### 8.1.2.2. Pruebas del núcleo de ejecución.

#### PRUEBA:

A algunos les molesta que el personaje principal, sea señalable, porque suele estar entre el puntero y un objeto y hay que moverlo constante, como si fuese un estorbo.

-----

#### SOLUCIÓN:

Me pongo de su parte, ya que realmente el poder señalar al personaje principal no aporta nada, y al ser escenarios pequeños suele molestar. Así que, como hacen en la mayoría de las aventuras gráficas, el protagonista (en nuestro caso Oscar Romero) no es seleccionable.

#### PRUEBA:

Uno de las partes más proclive a fallos son las conversaciones, ya que en cada una de ellas participan varios objetos de distinto tipo. Así que se testea a fondo, y se encuentra un error fantasma que hace que de vez en cuando, cuando estás hablando con un personaje, te salte de una opción a otra sin sentido.

-----

#### SOLUCIÓN:

Después de mucho investigar y de mucho depurar, se da con el problema. De forma resumida, el error se producía cuando se ejecutaba un resultado de cargaOpInf, y encima tenemos un mensaje pendiente. Se soluciona retocando la lógica del resultado cargaOpInf.

#### PRUEBA:

Se prueba a caminar por el escenario, probando a ir a sitios raros e imposibles. Gracias a esto nos percatamos de que si le señalamos que vaya a zonas del mapa que no son caminables, en vez de llevarnos al sitio más cercano a este sitio, hace un recorrido hacia atrás muy extraño, se podría decir, que repite la ruta hecha anteriormente.



---

#### SOLUCIÓN:

Se examina el código y se encuentra el error fácilmente, el problema era que tenía implementado que si no encontraba una ruta al destino, me diese la anterior que tenía guardada. Se modifica y vuelve a funcionar.

### 8.1.2.3. Pruebas de lógica

No hay problemas ya que como estoy yo ayudándolos, saben que hacer en cada momento. Se pasan la fase 1 bastante rápido.

## 8.2. Beta

### 8.2.1.Objetivos

Las pruebas beta vienen después de las pruebas alfa, y se desarrollan en el entorno del cliente, un entorno que está fuera de control. Aquí el cliente se queda a solas con el producto y trata de encontrarle fallos (reales o imaginarios) de los que informa al desarrollador.

En el caso concreto del juego, este se instaló en los móviles de dos familiares míos, y se les dejó solos, con una hoja donde apuntar todo lo que no entendían, lo que les parecía difícil de usar, o lo que veían mal. Una vez que me entregaban las hojas, entre todos pensábamos como se podía arreglar ese problema, por ejemplo para hacer la interfaz más fácil de usar, una vez decidido, yo arreglaba los errores y les enseñaba los resultados. Si ellos lo veían bien, continuábamos con la fase beta hasta que ya no viesen ningún error, o al menos ninguno al que yo le diese importancia.

### 8.2.2.Casos de prueba

#### 8.2.2.1. Pruebas de interfaz.

---

#### PRUEBA:

El jugador comienza una partida, sin mi ayuda, todo transcurre bien en el video, pero en la conversación ya no se da cuenta de que tiene más opciones a izquierda y derecha, aun indicándoselo con flechas, y del inventario vuelve a no saber salir. Parece que el menú no se digna a leerlo, e igual que lo hace el, lo hago yo y lo hacen muchos...

---



**SOLUCIÓN:**

Como veo que aunque haya ayuda, esta no se consulta, decido obligarles a leerla. Esto lo consigo haciendo una especie de tutorial mientras juegan. Así la primera vez que entre en el inventario le saldrá una pantalla de mensaje explicándole como funciona el inventario (el mismo contenido que en la ayuda), cuando entre en la conversación lo mismo, cuando intente combinar un objeto también, cuando esté en el mapa también.

Los resultados son bastante positivos, y por lo menos ya no se quedan atascados por las teclas.

**PRUEBA:**

Probando el cambio entre escenarios, uno de los beta testers me comenta que quizás sería más cómodo si para traspasar una puerta, en vez de señalar darle a Ir a... y luego a abrir, que valiese con darle a Ir a..., y que automáticamente traspasase la puerta.

-----  
**SOLUCIÓN:**

Creo que es buena idea, es lógica, cuando te acercas a una puerta lo normal es que sea para traspasarla.

**PRUEBA:**

Uno de los beta tester me dice que, es un poco pesado, tener que andar buscando la salida en aquellos escenarios que no la tiene clara, es decir, en aquellos cuya salida no es una puerta o algo bien definido. Me recomienda que si es un trozo de calle ponga una flecha indicando que por ahí está la salida.

-----  
**SOLUCIÓN:**

Se rechaza, eso estropearía la estética del escenario, además de que sólo el encontró ese problema, diciéndome el otro probador que no era tan difícil encontrar las salidas.

**PRUEBA:**



Como ahora los jugadores están solos, y sin mi ayuda, les cuesta más pasar las fases, esto provoca que tengan que moverse más y probar más opciones y demás. Pues me comentan que sería recomendable que la primera opción que se nos da a elegir de un objeto sea la más lógica o normal para ese objeto, por ejemplo, para una puerta Ir a... (y así traspasarla).

-----

**SOLUCIÓN:**

Estoy de acuerdo, reestructuro las opciones para que estén colocadas de una forma más lógica.

### **8.2.2.2. Pruebas del núcleo de ejecución.**

**PRUEBA:**

Como comenté antes, ahora los jugadores se mueven más y están más tiempo jugando. Esto ha provocado que sientan que el puntero va demasiado lento.

-----

**SOLUCIÓN:**

Se aumenta 1/3 la velocidad del puntero.

**PRUEBA:**

Se hacen numerosas pruebas cargar y guardar partida, y en una de ellas se dan cuenta de que si de la apariencia 0, cambia a la 1 y guardas, cuando cargas de nuevo el juego, vuelves a tener la apariencia 0.

-----

**SOLUCIÓN:**

Es debido a que el resultado camApariencia no lo introduzco entre los que se guardan. Simplemente se soluciona diciendo que si se guarde.

### **8.2.2.3. Pruebas de lógica.**



**PRUEBA:**

La fase 1, que en principio debería ser fácil de superar, ya que es la parte en la que nos familiarizamos con el juego, les causa demasiado problemas y ninguno es capaz a pasársela. Esto puede causar frustración y hacer que abandonen el juego.

---

**SOLUCIÓN:**

Se añaden algunos comentarios y pistas que facilitan la tarea. Ahora si que consiguen pasársela.



## 9. Ampliaciones

### 9.1. Minijuegos

Los mini-juegos, son sencillos juegos que lo único que intentan es romper con la monotonía de la aventura, para así sorprender al usuario con algo diferente. Estos minijuegos saltarían a lo largo de la partida, y estarían completamente acoplados a la historia, de tal manera que no de la sensación de que aparecen de manera forzada.

El uso de estos minijuego es muy común en las aventuras comerciales, y quizás uno de los más divertidos es el concurso de escupitajos de Monkey Island.

### 9.2. Sonido

Este, sin duda, sería el elemento que más apartaría al juego, ya que un buen sonido mejora la ambientación de la partida, y hace que el jugador se introduzca más en el juego.

En un principio estaba pensado el introducir sonidos, pero se tuvo que desechar por falta de espacio.

### 9.3. Videos

Y es que aunque en esta aventura mas que “videos”, vemos viñetas con texto. La idea en un principio también era poner videos, no viñetas, pero se tuvo que eliminar la idea incluso antes que la del sonido. Además el que el juego tuviese videos, obligaba a que los jugadores, para poder ejecutar el juego, tenían que tener un móvil con la Mobile Media API (MMAPI), lo cual limitaría aun más el espectro de jugadores potenciales.







## PostMorten

Desde siempre mi ilusión ha sido crear o participar en la creación de un juego, y más sobre todo si el juego es de un genero que a mi me guste. Para mi ser desarrollador de videojuegos era algo mágico... “Piensas una idea divertida, y la llevas a la pantalla”, un arte, tanto como el cine.

Con estos precedente y cuando me llego el momento de pensar en empezar el proyecto final de carrera, mi primer pensamiento fue hacer un videojuego, una aventura gráfica, que era el genero con el que había crecido y con el que más había disfrutado.

Pero respecto a la elección de dicho genero, pronto me di cuenta de que había cogido uno de los géneros más difíciles en el mundo de los videojuegos, por lo menos a mi entender. En principio lo ves como un reto, y no piensas en las dificultades, pero cuando va llegando el final, esas dificultades te pesan cada día más.

La suerte que tuve fue la de contactar con alguien como Juan Manuel Prada (Ferre), quien realizo los dibujos, y sin quien esta aventura no sería ni la mitad de lo que es, ya que gracias a su esfuerzo podemos ver un juego agradable a la vista.

Pero tanto él como yo, tuvimos una limitación muy importante en cuanto a la calidad de las imágenes, y no porque el no tuviese capacidad, que tiene muchísima, sino porque el juego decidí hacerlo para móvil. Esto tiene la ventaja de que el mundo móvil te permite crear un juego completo, con una calidad casi a la altura de los que se venden por radio, televisión, Internet... pero como pago tienes que sufrir muchísimas limitaciones, como por ejemplo en el tamaño y peso de las imágenes, en la optimización del código, en el uso de sonido...

Así que sin saber nada de cómo se hace una aventura gráfica, ni de cómo se programa en J2ME, inicié mi proyecto. Al principio básicamente me dediqué a recopilar información, hasta que empecé a tener un poco claro como iba todo, poco a poco, probando esto y quitando aquello, conseguí hacer un pequeño juego de naves, ya tenía mi base en programación para móviles.

Esto es algo que agradezco a este proyecto, ya que el tener conocimiento de J2ME te puede abrir algunas puertas, y más si presentas una aplicación tuya y funcionando.

Ahora que ya está acabado me doy cuenta de que quizás comencé demasiado tarde a diseñar e implementar, y perdí demasiado tiempo documentándome, quizás era el miedo al por donde coger un proyecto tan grande, hasta que me decidí y empecé por lo más fácil, el menú.



Ahí empezaba lo bueno, el comenzar a ver resultados es muy gratificante, ver que poco a poco va mejorando el proyecto, y un día ver aparecer en pantalla a ese personaje que hace poco solo tenías en un boceto, es una sensación muy gratificante.

Pero de lo gratificante y bonito, se puede pasar muy rápido a la desesperación. Lo peor es ese momento en el que tienes un error y que por más vueltas que le das no sabes porque es. Yo de estos he tenido muchos, y es que los móviles son, en cierto modo, horrorosos, no tienen unas característica comunes, implementa MIDP, pero luego le ponen restricciones... es decir, que más que facilitar la labor del programador, la dificultan. Encima en los móviles no se puede depurar, para eso están los emuladores claro, eso deben decir, pero yo me he pasado días pensando, buscando y probando, porque el juego se ejecutaba bien en el emulador pero luego no se ejecutaba en el móvil, y es que el emulador está en unas condiciones ideales, mientras que el móvil está restringido.

Así que el proyecto ha tenido sus momentos buenos y malos, porque aunque te lo pongan muy difícil, cuando consigues tener el juego corriendo en tu móvil, ver a un amigo, o a un familiar jugando, porque de ahí ya no pasa, es una gozada, están disfrutando de algo creado por ti, te conviertes en alguien de esos que yo hablaba al principio.

En definitiva, que aunque he terminado muy cansado, ya que un proyecto de este tipo en solitario, se te termina haciendo muy largo, estoy contento, porque creo que el resultado es bueno, y que puedo estar orgulloso de este.



## Glosario

**2D** → Algo es bidimensional si tiene dos dimensiones, por ejemplo, ancho y largo, pero no profundo. Los planos son bidimensionales, y sólo pueden contener cuerpos unidimensionales o bidimensionales.

**Boceto** → Dibujo hecho de forma esquemática y sin preocuparse de los detalles o terminaciones para representar una idea, un lugar, una persona, un aparato o cualquier cosa en general.

**CLDC** → Es la base para que los perfiles (como MIDP o PDAP) funcionen, proveyendo las APIs básicas y la máquina virtual (KVM). CLDC está diseñada para equipos microprocesadores RISC o CISC de 16 a 32 bits y con una memoria mínima de 160 KB para la pila de la tecnología Java.

**Freeware** → Aplicación informática que se puede copiar y distribuir libremente, y cuyo uso es gratis.

**J2ME** → Java 2 Platform Micro Edition. El J2ME permite que los desarrolladores usen Java y las herramientas inalámbricas J2ME para crear aplicaciones y programas para dispositivos móviles.

**JAR** → Formato de archivo utilizado para reunir todos los componentes que requiere un applet de java. Los archivos, también conocidos por la sigla JAR, simplifican la descarga de applets, dado que todos los componentes (archivos con extensión. class, imágenes, sonidos, etc.) pueden reunirse en un único archivo.

**Java** → Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems.

**KVM** → Su nombre KVM proviene de Kilobyte (haciendo referencia a la baja ocupación de memoria, entre 40Kb y 80Kb). Se trata de una implementación de Máquina Virtual reducida y especialmente orientada a dispositivos con bajas capacidades computacionales y de memoria. La KVM está escrita en lenguaje C.

**KXML** → KXML es un pequeño parser de XML, especialmente diseñado para entornos limitados, como MIDP.

**Midlet** → Aplicación escrita para MIDP.

**RMS** → Pequeña base de datos, basada en registros, utilizada en aplicaciones MIDP.

**sdk** → Kit de desarrollo de software, un conjunto de aplicaciones para desarrollar programas en un determinado lenguaje o para un determinado entorno (Software Development Kit).





## Bibliografía

***Aplicación de la teoría del puzzle*** → Artículo que para mi, es de obligada lectura a la hora de desarrollar una aventura gráfica. En <http://la-aventura.net/fan-articles/aplicacion-de-la-teoria-del-puzzle>

***Programación de juegos para móviles con J2ME*** → Una buena forma de iniciarse en la programación con J2ME. Se puede encontrar en [http://www.programacion.com/java/tutorial/ags\\_j2me/](http://www.programacion.com/java/tutorial/ags_j2me/)

***Java a tope: J2ME*** → Fantástico libro, en el que profundizarás en todos los detalles de J2ME. Escrito por Sergio González Rojas y Lucas Ortega Días de la Universidad de Málaga. Se puede encontrar en <http://www.lcc.uma.es/~galvez/J2ME.html>

***J2ME & Gaming Book de Jason Lam*** → En inglés. Te enseñará a crear un juego de naves, con J2ME, de principio a fin. Muy bueno para resolver dudas puntuales. Se puede encontrar en <http://www.jasonlam604.com/books.php>

***J2ME Game Optimization Secrets*** → Indispensable artículo de optimización en J2ME. Disponible en [http://www.developer.com/ws/j2me/article.php/10945\\_2234631\\_1](http://www.developer.com/ws/j2me/article.php/10945_2234631_1)

***Parsing XML in J2ME*** → Nos indica como parsear un archivo XML, utilizando J2ME. Disponible en <http://developers.sun.com/techtopics/mobility/midp/articles/parsingxml/>

***KXML project*** → Página web del proyecto KXML. La encontrarás en <http://kxml.objectweb.org/index.html>

***A\* Pathfinding para Principiantes*** → El primero de una serie de artículos que nos enseñan como funciona este importante algoritmo. En [http://www.policyalmanac.org/games/aStarTutorial\\_es.htm](http://www.policyalmanac.org/games/aStarTutorial_es.htm)





## Apéndice A: Código fuente

### AVLTree

```
import java.util.NoSuchElementException;
import java.util.Vector;

/**
 * Code has been taken from the course textbook: Cormen, Leiserson, Rivest,
 * Stein: Introduction to Algorithm (2nd edition). McGraw-Hill (2001), ISBN:
 * 0070131511
 */

public class AVLTree {

    /** Root of the AVL tree. */
    public Node root;

    private Vector almacenados;

    /**
     * The constructor creating a binary search tree with just a
     * <code>null</code>, which is the root.
     */
    public AVLTree() {
        root = null;
        almacenados = new Vector();
    }

    /**
     * Searches the tree for a node with a given key. If such a node exists,
     * prints the value of that node.
     *
     * @param k
     *     The key being searched for.
     * @return A reference to a <code>Node</code> object with key
     *     <code>k</code> if such a node exists. An exception
     *     NoSuchElementException is thrown if no node exists.
     */
    public Node search(int k) {
        Node node = search(root, k);
        return node;
    } //search

    /**
     * Searches the subtree rooted at a given node for a node with a given key.
     *
     * @param node
     *     Root of the subtree.
     * @param k
     *     The key being searched for.
     * @return A reference to a <code>Node</code> object with key
     *     <code>k</code> if such a node exists. An exception

```



```
*/ NoSuchElementException is thrown if no node exists.

public Node search(Node node, int k) {
    while ((node != null) && (k != node.data.getKey())) {
        if (k < node.data.getKey())
            node = node.left;
        else
            node = node.right;
    } //while

    return node;
} //search

/**
 * Returns the node with the minimum key in the subtree rooted at a node.
 *
 * @param x
 *     Root of the subtree.
 * @return A <code>Node</code> object with the minimum key in the tree, or
 *     the sentinel <code>nil</code> if the tree is empty.
 */
protected Node treeMinimum(Node x) {
    while (x.left != null)
        x = x.left;

    return x;
} //treeMinimum

/**
 * Returns the successor of a given node in an inorder walk of the tree.
 *
 * @param node
 *     The node whose successor is returned.
 * @return If <code>node</code> has a successor, it is returned.
 *     Otherwise, return the sentinel <code>null</code>.
 */
public Node successor(Node node) {
    Node x = node;

    if (x.right != null)
        return treeMinimum(x.right);

    Node y = x.parent;
    while (y != null && x == y.right) {
        x = y;
        y = y.parent;
    }

    return y;
} //successor

/**
 * Iterates up a tree updating balance factors and performing rotations
 * after an insert. The method returns when all the necessary updates have
 * been performed.
 */
private void iterateUpAfterInsert(Node iter) {
    //The parent of node iter at the start of each iteration
    Node parent;
```





```
//The new root of the rotated subtree
Node newRoot;

//The variable stop is set to true after a rotation or after a balance
//factor is set to 0. In these cases, no more balance factors need
//to be updated
boolean stop = false;

while ((iter != null) && !stop) {
    parent = iter.parent;

    if (parent == null)
        iter = iter.parent;
    else {
        //Update the parent's balance factor
        if (parent.left == iter) {
            parent.bf--;
            if (parent.bf == 0)
                stop = true;
        } else if (parent.right == iter) {
            parent.bf++;
            if (parent.bf == 0)
                stop = true;
        }

        //Traverse one node up the tree
        iter = iter.parent;
        parent = iter.parent;

        //Perform any necessary rotations
        if ((iter.bf == -2) || (iter.bf == +2)) {
            stop = true;

            if ((iter.bf == -2) && (iter.left.bf == -1))
                newRoot = iter.rotateRight();
            else if ((iter.bf == -2) && (iter.left.bf == +1))
                newRoot = iter.rotateLeftRight();
            else if ((iter.bf == +2) && (iter.right.bf == +1))
                newRoot = iter.rotateLeft();
            else
                newRoot = iter.rotateRightLeft();

            // Update the root pointer if necessary;
            // otherwise, update the pointers between the
            // newRoot and its new parent.
            if (parent == null) {
                root = newRoot;
                newRoot.parent = null;
            } else if (parent.left == iter) {
                parent.left = newRoot;
                newRoot.parent = parent;
            } else {
                parent.right = newRoot;
                newRoot.parent = parent;
            }
        }
    }
}

} //if

} //else
```



```
} //iterateUpAfterInsert
} //while

/**
 * Iterates up a tree updating balance factors and performing rotations
 * after a delete. The method returns when all the necessary updates have
 * been performed.
 */
private void iterateUpAfterDelete(Node iter) {
    // The variable stop is set to true after the root of a rotated
    // subtree is -1 or +1. In this case, no more balance
    // factors need to be updated
    boolean stop = false;

    //The parent of node iter at the start of each iteration
    Node parent;

    //The new root of the rotated subtree
    Node newRoot;

    // Exit the loop when
    // i) we reached the root (i.e. iter == null)
    // ii) the root of a rotated subtree has a balance factor of
    // -1 or +1 (i.e., stop == true), or
    // iii) the current node has a balance factor of -1 or +1

    // Note: The delete code is more complicated than the insert
    // code, since we may need to do several rotations at different
    // levels of the tree.
    while ((iter != null) && !stop && ((iter.bf != -1) || (iter.bf != +1))) {

        parent = iter.parent;

        //Perform any necessary rotations, and update the
        //parent's balance factors
        if ((iter.bf == -2) || (iter.bf == +2)) {
            if ((iter.bf == -2) && (iter.left.bf != +1))
                newRoot = iter.rotateRight();
            else if ((iter.bf == -2) && (iter.left.bf == +1))
                newRoot = iter.rotateLeftRight();
            else if ((iter.bf == +2) && (iter.right.bf != -1))
                newRoot = iter.rotateLeft();
            else
                newRoot = iter.rotateRightLeft();

            // If the balance factor of the new root is -1 or +1,
            // no more rotations need to be performed

            if ((newRoot.bf == -1) || (newRoot.bf == +1))
                stop = true;

            // Update the root pointer if necessary;
            // otherwise, update the pointers between the
            // newRoot and its new parent.
            if (parent == null) {
                root = newRoot;
                newRoot.parent = null;
            } else if (parent.left == iter) {
                parent.left = newRoot;
                newRoot.parent = parent;
            }
        }
    }
}
```



```
        if (!stop)parent.bf++;
    } else {
        parent.right = newRoot;
        newRoot.parent = parent;
        if (!stop)
            parent.bf--;
    }
} //if

// No rotations are necessary here. Simply update the
// parent's balance factor
else if (parent != null) {
    if (parent.left == iter)
        parent.bf++;
    else
        parent.bf--;
} //else if

// Traverse one node up the tree.
iter = parent;
if (iter != null)
    parent = iter.parent;

} //while
} //iterateUpAfterDelete

/**
 * Inserts a data item into the tree, creating a new node for this data.
 *
 * @param data
 *      Data to be inserted into the tree.
 * @return A reference to the <code>Node</code> object created.
 */
public Node insert(Comparable data) {
    Node z = new Node(data);
    treeInsert(z);
    return z;
} //insert

/**
 * Inserts a node into the tree.
 *
 * @param z
 *      The node to insert.
 */
protected void treeInsert(Node z) {
    Node y = null;
    Node x = root;

    while (x != null) {
        y = x;
        if (z.data.getKey() <= x.data.getKey())
            x = x.left;
        else
            x = x.right;
    } //while

    // Find the location where the node should be inserted
    z.parent = y;
```



```
        if (y == null)
            root = z; // the tree had been empty
        else {
            if (z.data.getKey() <= y.data.getKey())
                y.left = z;
            else
                y.right = z;
        } //else

        //Perform any necessary rotations
        iterateUpAfterInsert(z);

    } //treeInsert

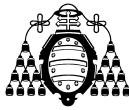
    /**
     * Removes a node with the given key from the tree.
     *
     * @param k
     *     The key to be removed.
     */
    public void deleteKey(int k) {
        Node node = (Node) search(root, k);
        if (node != null)
            delete(node);
        else
            throw new NoSuchElementException("Not Found");
    } //deleteKey

    /**
     * Removes all nodes from the tree.
     *
     */
    public void deleteAllElements() {
        Node aux = root;
        inorden(aux);
        aux = next();
        while (aux != null) {
            delete(aux);
            aux = next();
        }
    }

    /**
     * Removes a node from the tree.
     *
     * @param node
     *     The node to be removed. do nothing when deleting a null node
     *     <code>null</code>.
     */
    public void delete(Node node) {
        Node z = node;

        // Make sure that there is no attempt to delete a null node
        if (z == null)
            throw new NoSuchElementException("empty node");

        Node x; // Replaces z as the subtree's root
```



```
/* Case 1 and 2: The node to delete has 0 or 1 children. If the node has
* no children, we simply delete it. Otherwise, if the node has one
* child, we replace the node's contents with the child's contents. In
* these two cases, the balance factors of the tree may change.
*/

if ((z.left == null) || (z.right == null)) {
    if (z.left == null) // z has no left child
        x = z.right;
    else
        // z has no right child
        x = z.left;

    // Fix links between the parent of the subtree and x, and
    // update the balance factors of the parent of the deleted node
    if (x != null)
        x.parent = z.parent;
    if (root == z)
        root = x;
    else {
        if (z == z.parent.left) {
            z.parent.left = x;
            z.parent.bf++;
        } else {
            z.parent.right = x;
            z.parent.bf--;
        }
    }
} //else

// Perform any necessary rotations
iterateUpAfterDelete(z.parent);

} //if ((z.left == null) || (z.right == null))

/*
* Case #3: If the node has 2 children, then we splice out the node and
* replace it with its successor. Notice that replacing this node does
* not change any balance factors.
*/

else {
    x = (Node) successor(z); // replace z with its successor
    delete(x); // Delete the node where the successor was

    // Splice out z and put x in its place by fixing links
    // with children.
    x.left = z.left;
    x.right = z.right;
    x.bf = z.bf;
    if (x.left != null)
        x.left.parent = x;
    if (x.right != null)
        x.right.parent = x;

    // Fix links between the parent of the subtree and x.
    if (x != null)
        x.parent = z.parent;
    if (root == z)
        root = x;
}
```



```
        else {      if (z == z.parent.left)
                     z.parent.left = x;
                     else
                     z.parent.right = x;
        } //else

    } //else

} //delete

public void inorden(Node n) {
    if (n != null) {
        inorden(n.left);
        addVector(n); //Lo añadido al vector
        inorden(n.right);
    }
}

private void addVector(Node n) {
    almacenados.addElement(n);
}

public Node next() {
    Node n = null;
    if (almacenados.size() != 0) {
        n = (Node) almacenados.firstElement();
        almacenados.removeElementAt(0);
    }
    return n;
}
}
```

## Combinacion

```
import java.util.Vector;

/**
 * Entre 2 Almas [http://gamelab.uniovi.es/entre2almas]
 * -----
 * PFC para la EUITIO [http://www.euitio.uniovi.es/]
 * Desarrollado en el GameLab [http://gamelab.mieres.uniovi.es/]
 * Bajo licencia Freeware
 * -----
 *
 * 2005-2006
 *
 *
 * Un objeto de la clase Item, podrá combinarse con otros objetos, estás
 * posibles combinaciones son objetos de la clase Combinación.
 *
 * @author Pedro Javier Sáez Martínez
 * @version 1.0
 */
public class Combinacion implements Comparable {
```



```
/**
 * Identificador del Item con el que ocurre algo al intentar combinar.
 */
int idItem;

/**
 * Indica si esta combinación está activa para un determinado Objeto
 */
boolean activa;

/**
 * Consecuencias de una determinada combinación. En este vector se
 * almacenarán vectores de objetos de la clase Resultado. Un Objeto puede
 * tener consecuencias distintas en momentos distintos, entonces aquí
 * almaceno todas las consecuencias que puede tener y para cada consecuencia
 * almacenamos el Vector de Resultados, en el orden en que ocurrirán. Una
 * vez que una consecuencia ya no se necesita, pues se borra del array
 */
Vector consecuencias;

/**
 * Indica la consecuencia que está activa, de entre las disponibles.
 */
int conseActiva;

/**
 * Constructor de la clase Combinación
 */
/**
 * @param idItem
 *       Identificador del Item con el que ocurre algo al intentar
 *       combinar
 * @param activa
 *       indica si esta combinación está activa para un determinado
 *       Objeto
 * @param consecuencias
 *       Consecuencias de una determinada opción
 */
public Combinacion(int idItem, boolean activa, Vector consecuencias) {
    this.idItem = idItem;
    this.activa = activa;
    this.consecuencias = consecuencias;
    conseActiva = 0;
}

/**
 * Devuelve la clave del comparable
 */
/**
 * @return La clave del comparable identificador del comparable.
 */
public int getKey() {
    return idItem;
}
}
```



## Comparable

```
/**
 * Entre 2 Almas [http://gamelab.uniovi.es/entre2almas]
 * -----
 * PFC para la EUITIO [http://www.euitio.uniovi.es/]
 * Desarrollado en el GameLab [http://gamelab.mieres.uniovi.es/]
 * Bajo licencia Freeware
 * -----
 *
 * 2005-2006
 *
 *
 * Interfaz. Todo aquel que lo implemente significará que es comparable, es
 * decir, que se puede comparar con objetos de otras clases que también
 * implementen la interfaz Comparable.
 *
 * @author Pedro Javier Sáez Martínez
 * @version 1.0
 */
public interface Comparable {

    /**
     * Devuelve la clave del comparable
     * @return La clave del comparable identificador del comparable.
     */
    public int getKey();

}
```

## Conversacion

```
import java.util.Vector;

/**
 * Entre 2 Almas [http://gamelab.uniovi.es/entre2almas]
 * -----
 * PFC para la EUITIO [http://www.euitio.uniovi.es/]
 * Desarrollado en el GameLab [http://gamelab.mieres.uniovi.es/]
 * Bajo licencia Freeware
 * -----
 *
 * 2005-2006
 *
 *
 * Una conversación se dará entre nuestro personaje y un objeto Item. Implementa
 * las interfaces Opcionable y Comparable.
 *
 * @author Pedro Javier Sáez Martínez
 * @version 1.0
 */
public class Conversacion implements Opcionable, Comparable {

    /**
```





```
    */identificador de la conversación
    int id;

    /**
     * Color con el que se identificará el personaje con el que habla Oscar
     */
    int color;

    /**
     * Array que indica que las opciones disponibles.
     */
    public Opcion[] opciones;

    /**
     * Constructor de la clase Conversación
     *
     * @param id
     *      identificador de la conversación
     * @param color
     *      Color con el que se identificará el personaje con el que habla
     *      Oscar
     * @param opciones
     *      indica que las opciones disponibles.
     */
    public Conversacion(int id, int color, Vector opciones) {
        this.id = id;
        this.color = color;
        this.opciones = new Opcion[opciones.size()];
        opciones.copyInto(this.opciones);
    }

    /**
     * Indica si una determinada acción tiene o no alguna consecuencia
     *
     * @param opcionSele
     *      La opción seleccionada en el juego
     * @return boolean indicando si esa opción o acción tiene alguna
     *      consecuencia si es true es que no hay consecuencias, y viceversa.
     */
    public boolean getConseActivaAcc(int opcionSele) {

        return opciones[opcionSele].consecuencias.isEmpty();
    }

    /**
     * Establece la consecuencia activa para una determinada acción.
     *
     * @param id
     *      Identificador de la opción a modificar.
     * @param valor
     *      el nuevo valor de conseActiva
     */
    public void setConseActivaAcc(String id, int valor) {
        Opcion auxOp = getOpcion(id);
        auxOp.conseActiva = valor;
    }

    /**
     * Devuelve la consecuencia para una determinada acción
```



```
    * @param opcionSele
    *     La opción seleccionada en el juego
    * @return Vector con la consecuencia activa para una determinada acción
    */
    public Vector getConsecuenciaAcc(int opcionSele) {

        return (Vector) opciones[opcionSele].consecuencias
            .elementAt(opciones[opcionSele].conseActiva);
    }

    /**
     * El nombre de una determinada acción.
     *
     * @param opcionSele
     *     La opción seleccionada en el juego
     * @return nombre de la acción.
     */
    public String getNombreAccion(int opcionSele) {
        return opciones[opcionSele].texto;
    }

    /**
     * Devuelve el numero de acciones disponibles para un Seleccionable
     *
     * @return el numero de acciones disponibles para un Seleccionable
     */
    public int longAcciones() {
        return opciones.length;
    }

    /**
     * Establece si una acción esta activa o no
     *
     * @param opcionSele
     *     La opción seleccionada en el juego
     * @param valor
     *     boolean indicando si está activa o no.
     */
    public void setAccionActiva(int opcionSele, boolean valor) {
        opciones[opcionSele].activa = valor;
    }

    /**
     * Indica si una determinada acción esta activa o no.
     *
     * @param opcionSele
     *     La opción seleccionada en el juego
     * @return boolean indicando si la opción está o no activa.
     */
    public boolean getAccionActiva(int opcionSele) {
        return opciones[opcionSele].activa;
    }

    /**
     * Devuelve la clave del comparable
     *
     * @return La clave del comparable identificador del comparable.
     */
    public int getKey() {
```



```
        }        return id;

    /**
     * Devuelve la opción cuyo id coincide con el que se pasa como parámetro
     *
     * @param id
     *         String con el identificador de la opción
     * @return La opción cuyo id coincide con el de la opción.
     */
    public Opcion getOpcion(String id) {
        for (int i = 0; i < longAcciones(); i++) {
            if (opciones[i] != null) {
                if (opciones[i].id.equals(id)) {
                    return opciones[i];
                }
            }
        }
        return null;
    }

    /**
     * Borra una opción de entra las disponibles
     *
     * @param id
     *         Identificador de la opción.
     */
    public void borrarOp(String id) {
        for (int i = 0; i < longAcciones(); i++)
            if (opciones[i] != null)
                if (opciones[i].id.equals(id))
                    opciones[i] = null;
    }
}
```

## CountDown

```
import java.util.TimerTask;

/**
 * Entre 2 Almas [http://gamelab.uniovi.es/entre2almas]
 * -----
 * PFC para la EUITO [http://www.euitio.uniovi.es/]
 * Desarrollado en el GameLab [http://gamelab.mieres.uniovi.es/]
 * Bajo licencia Freeware
 * -----
 *
 * 2005-2006
 *
 *
 * Clase llamada desde el SplashScreen, cuando el splash es mostrado en pantalla
 * y ha paso el "dismissTime".
 *
 * @author Pedro Javier Sáez Martínez
 * @version 1.0
 */
```



```
class Countdown extends TimerTask {
    private final SplashScreen splashScreen;

    Countdown(SplashScreen splashScreen) {
        this.splashScreen = splashScreen;
    }

    /**
     * Aquí se establece la acción a ser realizada por un TimerTask
     */
    public void run() {
        SplashScreen.access(this.splashScreen);
    }
}
```

## Ejecucion

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.DataInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Vector;

import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Font;
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.game.GameCanvas;
import javax.microedition.lcdui.game.LayerManager;
import javax.microedition.lcdui.game.Sprite;
import javax.microedition.rms.InvalidRecordIDException;
import javax.microedition.rms.RecordStore;
import javax.microedition.rms.RecordStoreException;
import javax.microedition.rms.RecordStoreFullException;
import javax.microedition.rms.RecordStoreNotOpenException;

import org.kxml2.io.KXmlParser;
import org.xmlpull.v1.XmlPullParser;
import org.xmlpull.v1.XmlPullParserException;

/**
 * Entre 2 Almas [http://gamelab.uniovi.es/entre2almas]
 * -----
 * PFC para la EUITIO [http://www.euitio.uniovi.es/]
 * Desarrollado en el GameLab [http://gamelab.mieres.uniovi.es/]
 * Bajo licencia Freeware
 * -----
 *
 * 2005-2006
 *
 *
 * Lleva toda la dirección de lo que es el núcleo del juego, aquí se decide
 * cuando mirar colisiones, cuando ejecutar acciones, como buscar una Ruta,
 * cuales son las funciones de cada tecla en cada momento, etc.
 *
 * @author Pedro Javier Sáez Martínez
```



```
*/@version 1.0
public class Ejecucion extends GameCanvas implements Runnable {

    /**
     * Referencia al midlet.
     */
    private Entre2Almas midlet;

    /**
     * Hilo de ejecución del juego
     */
    Thread t;

    /**
     * Será el controlador de las distintas capas del juego.
     */
    private LayerManager LM;

    /**
     * Parser del XML.
     */
    KXmlParser parser;

    /**
     * El pathFinding A*
     */
    PathFinding buscaRuta;

    /**
     * Estará a true cuando Oscar está caminando.
     */
    private boolean caminando;

    /**
     * Si es 0, es que va de frente, 1 va de espaldas, 2 derecha, 3 izquierda.
     */
    private int direccion;

    /**
     * Indice del array Path que contiene la posición del siguiente paso a
     * ejecutar por oscar.
     */
    private int pasoActual;

    /**
     * Son los distintos mapas del juego
     */
    AVLTree mapas;

    /**
     * Indica el identificador del mapa actual;
     */
    public int mapaActual;

    /**
     * El ancho y el alto en píxeles que tiene cada celda en las que esta
     * dividido el mapa
     */
    private int anchoCelda, altoCelda;
```



```
/**
 * Escenario que actualmente está cargado en memoria.
 */
public Escenario escenActual;

/**
 * Guarda el nombre del último escenario cargado
 */
private StringBuffer ultimoEscenario;

/**
 * Sprite general
 */
private Sprite sprite;

/**
 * Sprite de Oscar
 */
private Sprite oscar;

/**
 * Sprite del puntero.
 */
private Sprite puntero;

/**
 * Sprite formado por la imagen correspondiente a la parte de atrás del
 * escenario actual.
 */
private Sprite escenDetras;

/**
 * Sprite formado por la imagen correspondiente a la parte caminable del
 * escenario actual.
 */
private Sprite escenCaminable;

/**
 * Vector que almacena los Items que tenemos en el inventario
 */
private Vector itemsInventario;

/**
 * Indica si hay que mostrar o no, el inventario
 */
private boolean mostrarInventario;

/**
 * Indica el objeto que está mostrándose en ese momento en el inventario.
 */
private int invSelec;

/**
 * Indica el Seleccionable que esta siendo seleccionado en la pantalla de
 * juego
 */
private Opcionable seleccionado;

/**
```



```
*/Indica si hay que mostrar o no, las opciones de un objeto
private boolean mostrarOpciones;

/**
 * Indica si el puntero ha colisionado con algo.
 */
private boolean colisionado;

/**
 * Marca el espacio de ancho que se deja para el inventario en la pantalla
 * de juego
 */
private static int anchoInventario;

/**
 * Marca el espacio de alto que se deja para el inventario en la pantalla de
 * juego
 */
private static int altoInventario;

/**
 * Ancho del panel donde se van a mostrar las opciones
 */
private static int anchoOpciones;

/**
 * Alto del panel donde se van a mostrar las opciones
 */
private static int altoOpciones;

/**
 * Ancho del panel donde se van a mostrar los mensajes
 */
private static int anchoMensaje;

/**
 * Alto del panel donde se van a mostrar los mensajes
 */
private static int altoMensaje;

/**
 * indica la opción que está seleccionada ahora mismo para un Item.
 */
private int opcionSele;

/**
 * Indica si hay que mostrar o un mensaje.
 */
private boolean mostrarMensaje;

/**
 * Vector con los videos correspondientes a una fase
 */
private AVLTree videosFase;

/**
 * Identificador del siguiente video a mostrar.
 */
private int videoSiguiente;
```



```
/**
 * AVL con las conversaciones correspondientes a una fase
 */
private AVLTree converFase;

/**
 * Identificador de la conversación a mostrar
 */
private int converSelec;

/**
 * Participante en la conversación.
 */
private Item participante;

/**
 * Indica la longitud que tendrá el id de la opción a mostrar en la
 * conversación
 */
private int longIdOp;

/**
 * Indica las letras que tendrá que tener en su inicio el id de la opción,
 * para que entonces esta sea mostrada. En caso de que iniIdOp no contenga
 * ninguna letra, se cargarán todas las opciones que tengan un id que
 * coincida con longIdOp
 */
private StringBuffer iniIdOp;

/**
 * Indica si el que esta hablando es el participante (no Oscar), en cuyo caso
 * estará a true.
 */
private boolean habParti;

/**
 * Indica si una tecla está pulsada en este momento.
 */
private boolean teclaPulsada;

/**
 * Numero de fase en la que nos encontramos
 */
public int nFase;

/**
 * Nombre de la fase en la que nos encontramos.
 */
private StringBuffer nomFase;

/**
 * Parte o pantalla de juego en la que nos encontramos. Las posibilidades
 * son: Video, Juego, inventario, conversación, minijuego1, minijuego2 o
 * minijuego 3
 */
public int parteJuego;

/**
 * Variables estáticas que representan los distintos tipos de pantalla que
```





```
*/existen
private final int video = 0;

private final int juego = 1;

private final int inventario = 2;

private final int conversacion = 3;

private final int mapa = 4;

/**
 * Indica que pantalla estamos viendo actualmente.
 */
public int actual;

/**
 * Tiempo de delays acumulado. En cada bucle del hilo se hace un delay por
 * ejemplo de 20 milsg, pues en tmpAcumulado vamos sumando esto.
 */
private int tmpAcumulado;

/**
 * Indica que se está cargando una fase
 */
private boolean cargando;

/**
 * Vector que guardará los resultados a ejecutar a una determinada acción.
 */
Vector resultados;

/**
 * Indica cuando el programa está ocupado ejecutando una acción.
 */
private boolean ocupado;

/**
 * Indica cuando es necesario redibujar la pantalla.
 */
private boolean redibujar;

/**
 * Indica que queremos ir al menú.
 */
private boolean irMenu;

/**
 * Posiciones X e Y en las que aparece el enlace directo al menú
 */
private int posXMenu, posYMenu;

/**
 * Ancho y alto del acceso directo al menú.
 */
private int anchoMenu, altoMenu;

/**
 * Indica que un item está preparado en este momento para ser combinado con
```



```
    #/otros items.
private boolean combinando;

/**
 * Indica que el item que está preparado en este momento para ser combinado,
 * ya está intentando ser combinado con otro item.
 */
private boolean intentaCombinar;

/**
 * Item que está siendo combinando
 */
private Item combinado;

/**
 * Nos indicara cuando salir del bucle de ejecución, y por tanto finalizar
 * el hilo
 */
boolean salir;

/**
 * Es la letra inicial desde la que hay que empezar mostrar el mensaje en la
 * línea de texto. Por defecto su valor será -6. El porque es para que
 * tarden más en empezar a moverse las letras.
 */
int letra;

/**
 * Espacio disponible en píxeles en la línea de texto
 */
int disponible;

/**
 * Numero de letras que caben en la línea de texto
 */
int numLetras;

/**
 * COnstructor de la clase Ejecución
 */
/**
 * @param midlet
 *      El midlet del juego. Entre2Almas
 */
public Ejecucion(Entre2Almas midlet) {
    super(true);
    this.midlet = midlet;
    //      El orden en que se ven las capas es desde la ultima a la
    //primera, es decir si lo primero que metes es una imagen
    //de un lobo, es lobo estará siempre en primera plana
    LM = new LayerManager();
    itemsInventario = new Vector();
    mapas = new AVLTree();
    mapaActual = 0;
    //Ponemos la pantalla a modo completo
    setFullScreenMode(true);
    //inicializamos el Vector
    videosFase = new AVLTree();
    converFase = new AVLTree();
}
```



```
//Cargamos el tamaño de la pantalla y el punto de inicio donde se
//van a poner las imágenes.
Utils.width = getWidth();
Utils.height = getHeight();
Utils.inicioPantallaX = (Utils.width / 2) - (Utils.getTamGrupoX() / 2);
Utils.inicioPantallaY = (Utils.height / 2) - (Utils.getTamGrupoY() / 2);
//
//Cargamos el sprite del puntero.
Utils.cargarImagen(Utils.spPuntero);
puntero = new Sprite(Utils.imagen, Utils.anchoIm, Utils.altoIm);
puntero.setFrameSequence(Utils.getFrames(Utils.spPuntero));
//Situamos el puntero en el medio.
puntero.setPosition(Utils.inicioPantallaX + (Utils.getTamGrupoX() / 2),
Utils.inicioPantallaY + (Utils.getTamGrupoY() / 2));
//Cargamos el Sprite de Oscar.
Utils.cargarImagen(Utils.spOscarFren);
oscar = new Sprite(Utils.imagen, Utils.anchoIm, Utils.altoIm);
oscar.setFrameSequence(Utils.getFrames(Utils.spOscarFren));
if (Utils.grupo == 0) { //grupo pequeño
    altoInventario = 42;
    anchoOpciones = Utils.lowFont.charWidth('a') * 8;
} else { //grupo grande
    altoInventario = 66;
    anchoOpciones = Utils.highFont.charWidth('a') * 8;
}
//
//Establecemos el ancho y alto de cada una de las celdas
anchoCelda = 2;
altoCelda = 2;
anchoInventario = Utils.width - anchoOpciones - Utils.inicioPantallaX
- (Utils.width - Utils.inicioPantallaX - Utils.getTamGrupoX());
anchoMensaje = Utils.getTamGrupoX() * 80 / 100;
altoMensaje = (Utils.getTamGrupoY() - Utils.getSpaceTextItem() - altoInventario) * 90
/ 100;

teclaPulsada = false;
colisionado = false;
mostrarOpciones = false;
opcionSele = 0;
mostrarInventario = false;
invSelec = 0;
converSelec = 0;
longIdOp = 1;
iniIdOp = new StringBuffer();
habParti = false;
mostrarMensaje = false;
cargando = false;
resultados = new Vector();
ocupado = false;
tmpAcumulado = 0;
pasoActual = 0;
direccion = 0;
combinando = false;
intentaCombinar = false;
//le pongo -6 para que tarde más en empezar a moverse las letras.
letra = -6;
ultimoEscenario = new StringBuffer();
ultimoEscenario.append("");
//Se empieza por la fase 1
actual = -1;
nFase = 1;
nomFase = new StringBuffer();
//Se empieza mostrando el video inicial
```



```
videoSiguiente = 0;
parteNegro = video;
redibujar = true;
irMenu = false;
//Medidas del acceso directo al menú.
if (Utils.grupo == 0) {
    posXMenu = Utils.inicioPantallaX + Utils.getTamGrupoX()
        - Utils.highFont.stringWidth("Menu") - 2;
    posYMenu = Utils.inicioPantallaY + Utils.getTamGrupoY()
        - Utils.getSpaceTextItem();
    anchoMenu = Utils.highFont.stringWidth("Menu") + 2;
    altoMenu = Utils.getSpaceTextItem();
} else {
    posXMenu = Utils.inicioPantallaX + Utils.getTamGrupoX()
        - Utils.highFont.stringWidth("Menu") - 2;
    posYMenu = Utils.inicioPantallaY + Utils.getTamGrupoY()
        - Utils.getSpaceTextItem();
    anchoMenu = Utils.highFont.stringWidth("Menu") + 2;
    altoMenu = Utils.getSpaceTextItem();
}
disponible = Utils.getTamGrupoX() - anchoMenu;
numLetras = disponible / Utils.widthLF;
}

public void start() {
    //crea el hilo y lo inicia (al iniciarlo se ejecuta el método run())
    t = new Thread(this);
    t.start();
}

/*
 * (non-Javadoc)
 *
 * @see java.lang.Runnable#run()
 */
public void run() {
    try {
        Graphics g = getGraphics();
        // lo pongo aquí porque si lo creo en ejecución() en el
nokia 6230
        // me
        //da outOfMemory, debido a que se junta lo que consume de memoria
        //la clase PathFinding (debido a sus numerosos arrays) y lo que
        // consumen
        //de memoria el cargar una fase.
        int numCeldasX = Utils.getTamGrupoX() / anchoCelda;
        int numCeldasY = Utils.getTamGrupoY() / altoCelda;
        buscaRuta = new PathFinding(numCeldasX, numCeldasY);
        //En caso de que estemos cargando una partida.
        ByteArrayInputStream bais = null;
        DataInputStream dis = null;
        byte[] datosEsen = null;
        if (Utils.cargandoPartida) {
            datosEsen = midlet.cargaDatosEsenciales();
            bais = new ByteArrayInputStream(datosEsen);
            dis = new DataInputStream(bais);
            nFase = dis.readInt();
        }
        cargarFase(g, nFase);
    }
}
```



dis

```
if (Utils.cargandoPartida) {
    mapaActual = dis.readInt();
    ((Mapa) mapas.search(mapaActual).data).activa = dis.readInt();
    ((Mapa) mapas.search(mapaActual).data).getEstanActual().actual =
        .readInt();
    parteJuego = dis.readInt();

    bais.close();
    dis.close();

    escenActual = ((Mapa) mapas.search(mapaActual).data)
        .getEscenActual();

    //Una vez cargada la fase e introducidos los datos esenciales,
    //introducimos los Resultados a ejecutar
    midlet.cargarResultados();
    parser = null;
    System.gc();
}

salir = false;
while (!salir) {
    synchronized (this) {
        if (Utils.guardandoPartida) {
            guardarPartida(g);
        }

        //Recogemos las pulsaciones de teclado.
        input();
        switch (parteJuego) {
            case video:

                break;
            case juego:
                animarItems();
                if (caminando)
                    avanzarPaso();
                if (resultados.size() > 0 && !ocupado)
                    ejecutarResultado();
                break;
            case conversacion:
                if (resultados.size() > 0 && !ocupado)
                    ejecutarResultado();
                if (resultados.size() == 0 && !ocupado) {
                    mostrarOpciones = true;
                    Utils.startWidth = Utils.inicioPantallaX
                        + ((oscar.getWidth() * 32)
/ 100);

                    Utils.spaceMen = Utils.getTamGrupoX()
                        - ((oscar.getWidth() * 60)
/ 100);

                    redibujar = true;
                }
                moverBoca();
                break;
            case mapa:
                if (resultados.size() > 0 && !ocupado)
                    ejecutarResultado();
```



```
        } break;
        //Redibujamos la pantalla
        if (redibujar)
            drawScreen(g);
        // El delay especifica el tiempo que se dormirá el hilo
        // entre
        // ejecución y ejecución.
        //si pusiésemos un delay menor el juego iría mas rápido
        tmpAcumulado = tmpAcumulado + 20;
        if (tmpAcumulado > 20000)
            tmpAcumulado = 0;
        Thread.sleep(20);
    }
}
salir();
} catch (InterruptedException ie) {
    System.err.println("Problemas con el Thread\n" + ie);
    midlet.mostrarAlerta("Ejecucion::run::Problemas con el Thread\n"
        + ie, AlertType.ERROR);
    ie.printStackTrace();
} catch (NullPointerException ie) {
    System.err
        .println("Ejecucion::run::Problemas con el XML\nProblemas
con un null\n"
        + ie);
    midlet
        .mostrarAlerta(
            "Ejecucion::run::Problemas con el
XML\nProblemas con un null\n",
            AlertType.ERROR);
    ie.printStackTrace();
} catch (XmlPullParserException e) {
    System.err.println("Problemas con el XML\n" + e);
    midlet.mostrarAlerta("Ejecucion::run::Problemas con el XML\n" + e,
        AlertType.ERROR);
    e.printStackTrace();
} catch (IOException e) {
    System.err.println("Problemas de entrada/salida\n" + e);
    midlet.mostrarAlerta(
        "Ejecucion::run::Problemas de entrada/salida\n" + e,
        AlertType.ERROR);
    e.printStackTrace();
} catch (Exception e) {
    System.err.println("Problemas desconocido\n" + e);
    midlet.mostrarAlerta("Ejecucion::run::Problemas desconocido\n" + e,
        AlertType.ERROR);
    e.printStackTrace();
}
}

/**
 * KeyPressed es llamado cuando una tecla es pulsada, pero solo lo usaremos
 * cuando sea una non-game Key. Las teclas de numeros (1,2,3,...), el * y la # ,
 * son game Keys y tienen positivos KeyCodes. Así las non-game keys tienen
 * keyCodes negativos. Cuando se pulse una de estas teclas especiales,
 * iremos al menú.
 *
 * @param keyCode
 *         keyCode de la tecla pulsada.
 */
```



```
public void keyPressed(int keyCode) {
    if (keyCode < 0) {
        irMenu = true;
    }
}

/**
 * método para controlar las entradas de teclado
 *
 * @throws InterruptedException
 */
private void input() throws InterruptedException {
    //Recogemos el estado de las teclas
    int keyStates = getKeyStates();
    if (irMenu)
        keyStates = GameCanvas.GAME_D_PRESSED;
    switch (parteJuego) {
        case video:
            if (((keyStates & UP_PRESSED) != 0) && Utils.linea > 0
                && !teclaPulsada) {
                Utils.linea--;
            } else if (((keyStates & DOWN_PRESSED) != 0)
                && Utils.numLineasMensaje - Utils.linea > Utils.numLineas
                && !teclaPulsada) {
                Utils.linea++;
            } else if (((keyStates & FIRE_PRESSED) != 0) && !teclaPulsada) {
                Video video = (Video) videosFase.search(videoSiguiente).data;
                // Comprobación para que
                no de vueltas y vueltas en los mismo
                //frames
                if (sprite.getFrame() == Utils.getNumFrames(video.vinyeta) - 1) {
                    parteJuego = video.sigParte;
                    ocupado = false;
                } else {
                    //Preparamos la siguiente viñeta del video
                    sprite.nextFrame();
                    Utils.linea = 0;
                }
            }
            break;
        case juego:
            if (!mostrarOpciones && !mostrarInventario && !mostrarMensaje) { //Si
                // no
                // estamos
                // mostrando
                // las
                // opciones
                // de
                // un
                // objeto...
                //lo que está comentado son ANDs que se podrían poner en
                //caso de que se quisiese limitar, por razones de ejecución,
                //las veces que se entra en cada uno de los ifs
                if (((keyStates & UP_PRESSED) != 0) /*
                    * &&
                    * puntero.getY()>Utils.inicioPantallaY
                */
            }
        }
    }
}
```

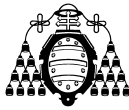


```

        */) {
            puntero.setPosition(puntero.getX(), Math.max(
                Utils.inicioPantallaY, puntero.getY() - 2
                - Utils.grupo));
            redibujar = true;
        } else if (((keyStates & DOWN_PRESSED) != 0))/*
            * &&puntero.getY()+puntero.getHeight()
            * <Utils.inicioPantallaY+Utils.tamGrupoY()
            */) {
                if (puntero.getX() + puntero.getWidth() > posXMenu)
                    puntero.setPosition(puntero.getX(), Math.min(
                        Utils.inicioPantallaY +
Utils.getTamGrupoY()
puntero.getHeight() - altoMenu,
                                puntero.getY() + 2 + Utils.grupo));
                else
                    puntero.setPosition(puntero.getX(), Math.min(
                        Utils.inicioPantallaY +
Utils.getTamGrupoY()
puntero.getHeight(), puntero.getY()
                                + 2 +
Utils.grupo));
                redibujar = true;
            } else if ((keyStates & LEFT_PRESSED) != 0 /*
                * &&
                * puntero.getX()>Utils.inicioPantallaX
                */) {
                    puntero.setPosition(Math.max(Utils.inicioPantallaX, puntero
                        .getX()
                        - 2 - Utils.grupo), puntero.getY());
                    redibujar = true;
                } else if ((keyStates & RIGHT_PRESSED) != 0 /*
                    * &&
                    * puntero.getX()+puntero.getWidth()
                    * <Utils.inicioPantallaX+Utils.tamGrupoX()
                    */) {
                        if (puntero.getY() + puntero.getHeight() > posYMenu)
                            puntero.setPosition(Math.min(Utils.inicioPantallaX
                                + Utils.getTamGrupoX() -
puntero.getWidth()
                                - anchoMenu, puntero.getX() + 2 +
Utils.grupo),
                                    puntero.getY());
                        else
                            puntero.setPosition(Math.min(Utils.inicioPantallaX
                                + Utils.getTamGrupoX() -
puntero.getWidth(),

```





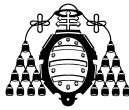
puntero

pantalla

!mostrarOpciones

```
puntero.getX() + 2 + Utils.grupo),
.getY());

redibujar = true;
} else if (((keyStates & FIRE_PRESSED) != 0) && !teclaPulsada) {
    if (colisionado) {
        if (combinando) {
            conseCombinacion();
            intentaCombinar = true;
        }
        //Si está ocupado no se podrán mostrar las opciones
        //de un objeto, debido al tema de que al cambiar de
        //escenario podría cascar.
        else {
            if (seleccionado != null && !ocupado)
                mostrarOpciones = true;
        }
        /*
        * else podría ser nuestro personaje principal y aquí
        * por ejemplo hacer que muestre un mensaje por
        */
    } else {
        if (buscarRuta())
            caminando = true;
    }
} else if (((keyStates & GameCanvas.GAME_C_PRESSED) != 0)
&& !teclaPulsada) {
    mostrarInventario = true;
    if (colisionado) {
        colisionado = false;
        letra = -6;
        puntero.nextFrame();
        seleccionado = null;
    }
}
//Comprobamos si hay colisión, solo cuando la última pulsación
// de tecla no sirvió ni para entrar en inventario ni para
// entrar en las opciones.
//Es decir que sirvió para mover el puntero.
//Y solo lo comprobamos cuando no está ocupado ejecutando un
//resultado, ya que hay algunos resultados que necesitan que
//el objeto con el colisiona el puntero, no cambie.
if (resultados.size() == 0) {
    if (!colisionado && !mostrarInventario &&
        && teclaPulsada) {
        if (colisionPunItem()) {
            colisionado = true;
            puntero.nextFrame();
        }
    } else if (colisionado && !mostrarInventario
        && !mostrarOpciones && teclaPulsada) {
        if (!colisionPunItem()) {
            colisionado = false;
            letra = -6;
            seleccionado = null;
            System.gc();
            puntero.nextFrame();
        }
    }
}
```



Utils.numLineas

```
    }
} else if (mostrarOpciones) { //si estamos mostrando las opciones de
    // un objeto...
    if ((keyStates & UP_PRESSED) != 0 && !teclaPulsada) {
        if (opcionSele > 0)
            opcionSele--;
        else
            opcionSele = Utils.numOps - 1;
    } else if ((keyStates & DOWN_PRESSED) != 0 && !teclaPulsada) {
        if (opcionSele < Utils.numOps - 1)
            opcionSele++;
        else
            opcionSele = 0;
    } else if (((keyStates & FIRE_PRESSED) != 0) && !teclaPulsada) {
        mostrarOpciones = false;
        opcionesIO();
    }
} else if (mostrarMensaje) {
    if (((keyStates & UP_PRESSED) != 0) && Utils.linea > 0
        && !teclaPulsada) {
        Utils.linea--;
    } else if (((keyStates & DOWN_PRESSED) != 0)
        && Utils.numLineasMensaje - Utils.linea >
        && !teclaPulsada) {
        Utils.linea++;
    } else if (((keyStates & FIRE_PRESSED) != 0) && !teclaPulsada) {
        mostrarMensaje = false;
        ocupado = false;
        Utils.linea = 0;
    }
} else if (mostrarInventario) {
    if (((keyStates & LEFT_PRESSED) != 0) && !teclaPulsada) {
        if (itemsInventario.size() > 1) {
            if (invSelec > 0)
                invSelec--;
            else
                invSelec = itemsInventario.size() - 1;
            letra = -6;
        }
    } else if (((keyStates & RIGHT_PRESSED) != 0) && !teclaPulsada)
    {
        if (itemsInventario.size() > 1) {
            if (invSelec < itemsInventario.size() - 1)
                invSelec++;
            else
                invSelec = 0;
            letra = -6;
        }
    } else if (((keyStates & FIRE_PRESSED) != 0) && !teclaPulsada) {
        if (combinando) {
            conseCombinacion();
            intentaCombinar = true;
        }
        //Si está ocupado no se podrán mostrar las opciones
        //de un objeto, debido al tema de que al cambiar de
        //escenario podría cascar.
        else {
            if (itemsInventario.size() != 0)
```



```
        }
        } else if (((keyStates & GameCanvas.GAME_C_PRESSED) != 0)
        && !teclaPulsada) {
            mostrarInventario = false;
            invSelec = 0;
            letra = -6;
        }
    }

    break;
case conversacion:
    Conversacion con = (Conversacion) seleccionado;
    if (((keyStates & UP_PRESSED) != 0) && Utils.linea > 0
        && !teclaPulsada) {
        Utils.linea--;
    } else if (((keyStates & DOWN_PRESSED) != 0))
        && Utils.numLineasMensaje - Utils.linea > Utils.numLineas
        && !teclaPulsada) {
        Utils.linea++;
    } else if (((keyStates & FIRE_PRESSED) != 0) && !teclaPulsada) {
        Utils.linea = 0;
        if (mostrarOpciones) {
            mostrarOpciones = false;
            Utils.startWidth = Utils.inicioPantallaX;
            Utils.spaceMen = Utils.getTamGrupoX();
            ocupado = true;
            opcionesIO();
        } else {
            //
            boca del que esta hablando, en caso de que
            // alguien hable
            cerrarBoca();
            if (habParti)
                habParti = false;
            if (ocupado)
                ocupado = false;
        }
    } else if (((keyStates & LEFT_PRESSED) != 0) && !teclaPulsada
        && mostrarOpciones == true) {
        if (con.opciones.length != 0) {
            if (opcionSele > 0)
                opcionSele--;
            else
                opcionSele = con.opciones.length - 1;
            //
            Ahora
            hayamos las opciones que pueden ser mostradas
            //salir se pondrá a true cuando se cumplan las condiciones
            // de una opción
            //NOTA: Esto no se puede quitar, ya que aunque en
            // getConversacion() hacemos
            //algo parecido, no es exactamente igual, ya que aquel vale
            //para el caso de pulsar acción o derecha (ya que siempre
            // hace
            //opcionSele++, mientras que este vale para izquierda,
            // siempre opcionSele--;
            boolean salir = false;
            while (!salir) {
                if (con.opciones[opcionSele] != null) {
```



```

longIdOp) {
    if (con.opciones[opcionSele].id.length() ==
        if (longIdOp == 1)
            salir = true;
        else {
            if
                0,
            salir = true;
        else {
            if (opcionSele >
                else
            }
        } else {
            if (opcionSele > 0)
                opcionSele--;
            else
                opcionSele =
        }
    } else {
        if (opcionSele > 0)
            opcionSele--;
        else
            opcionSele = con.opciones.length -
    }
    if (con.opciones[opcionSele] != null) {
        if
            && salir == true) {
                if (opcionSele > 0)
                    opcionSele--;
                else
                    opcionSele =
                salir = false;
            }
        }
        Utils.linea = 0;
    }
} else if (((keyStates & RIGHT_PRESSED) != 0) && !teclaPulsada
    && mostrarOpciones == true) {
    if (con.opciones.length != 0) {
        if (opcionSele < con.opciones.length - 1)
            opcionSele++;
        else
            opcionSele = 0;
        Utils.linea = 0;
    }
}
    con.opciones.length - 1;
    (!seleccionado.getAccionActiva(opcionSele)
    1;
    con.opciones.length - 1;
    con.opciones.length - 1;
    opcionSele = con.opciones.length - 1;
    opcionSele--;
    0)
    .toString()))
    longIdOp - 1).equals(iniIdOp
    ((con.opciones[opcionSele].id.substring(
    longIdOp) {

```



```
        break;
    case mapa:
        if (!mostrarOpciones && !mostrarMensaje) { // Si no estamos mostrando
            // las opciones de un
            // objeto...
            // lo que está comentado son ANDs que se podrían poner en
            // caso de que se quisiese limitar, por razones de ejecución,
            // las veces que se entra en cada uno de los ifs
            if (((keyStates & UP_PRESSED) != 0) /*

                * &&

                * puntero.getY() > Utils.inicioPantallaY

                */) {

                    puntero.setPosition(puntero.getX(), Math.max(
                        Utils.inicioPantallaY, puntero.getY() - 2
                        - Utils.grupo));

                    redibujar = true;
                } else if (((keyStates & DOWN_PRESSED) != 0) /*

                    * && puntero.getY() + puntero.getHeight()

                    * < Utils.inicioPantallaY + Utils.tamGrupoY()

                    */) {

                        puntero.setPosition(puntero.getX(), Math.min(
                            Utils.inicioPantallaY +

Utils.getTamGrupoY()

                                - puntero.getHeight(),

puntero.getY() + 2

                                    + Utils.grupo));

                        redibujar = true;
                    } else if ((keyStates & LEFT_PRESSED) != 0 /*

                        * &&

                        * puntero.getX() > Utils.inicioPantallaX

                        */) {

                            puntero.setPosition(Math.max(Utils.inicioPantallaX, puntero
                                .getX()
                                - 2 - Utils.grupo), puntero.getY());

                            redibujar = true;
                        } else if ((keyStates & RIGHT_PRESSED) != 0 /*

                            * &&

                            * puntero.getX() + puntero.getWidth()

                            * < Utils.inicioPantallaX + Utils.tamGrupoX()

                            */) {

                                puntero.setPosition(Math.min(Utils.inicioPantallaX
                                    + Utils.getTamGrupoX() -

puntero.getWidth(),

                                        puntero.getX() + 2 + Utils.grupo),

puntero.getY());

                                    redibujar = true;
```



```
        } else if (((keyStates & FIRE_PRESSED) != 0) && !teclaPulsada) {  
            if (colisionado) {  
                if (seleccionado != null)  
                    mostrarOpciones = true;  
            }  
        }  
        // Comprobamos si hay  
colisión, solo cuando la última pulsación  
        // de tecla no sirvió ni para entrar en inventario ni para  
        // entrar en las opciones.  
        //Es decir que sirvió para mover el puntero.  
        if (!colisionado && !mostrarInventario && !mostrarOpciones  
            && teclaPulsada) {  
            if (colisionPunEstan()) {  
                colisionado = true;  
                puntero.nextFrame();  
            }  
        } else if (colisionado && !mostrarInventario  
            && !mostrarOpciones && teclaPulsada) {  
            if (!colisionPunEstan()) {  
                colisionado = false;  
                letra = -6;  
                seleccionado = null;  
                System.gc();  
                puntero.nextFrame();  
            }  
        }  
    } else if (mostrarOpciones) { //si estamos mostrando las opciones de  
        // un objeto...  
        if ((keyStates & UP_PRESSED) != 0 && !teclaPulsada) {  
            if (opcionSele > 0)  
                opcionSele--;  
            else  
                opcionSele = Utils.numOps - 1;  
        } else if ((keyStates & DOWN_PRESSED) != 0 && !teclaPulsada) {  
            if (opcionSele < Utils.numOps - 1)  
                opcionSele++;  
            else  
                opcionSele = 0;  
        } else if (((keyStates & FIRE_PRESSED) != 0) && !teclaPulsada) {  
            mostrarOpciones = false;  
            opcionesIO();  
        }  
    } else if (mostrarMensaje) {  
        if (((keyStates & UP_PRESSED) != 0) && Utils.linea > 0  
            && !teclaPulsada) {  
            Utils.linea--;  
        } else if (((keyStates & DOWN_PRESSED) != 0)  
            && Utils.numLineasMensaje - Utils.linea >  
Utils.numLineas  
            && !teclaPulsada) {  
            Utils.linea++;  
        } else if (((keyStates & FIRE_PRESSED) != 0) && !teclaPulsada) {  
            mostrarMensaje = false;  
            ocupado = false;  
            Utils.linea = 0;  
        }  
    }  
    }  
    break;  
}
```



```
if ((keyStates & GameCanvas.GAME_D_PRESSED) != 0) {
    if (actual != menu) {
        this.wait();
    }
    if (keyStates != 0) {
        if (!teclaPulsada)
            redibujar = true;
        teclaPulsada = true;
    } else
        //KeyStates será 0 cuando no haya ninguna tecla pulsada.
        teclaPulsada = false;
}

private void drawScreen(Graphics g) throws XmlPullParserException,
    IOException {
    if (cargando) { //si estamos cargando fase
        //pintamos la imagen de cargando
        sprite.paint(g);
        sprite.nextFrame();
        Font f = g.getFont();
        g.setFont(Utils.lowFont);
        if (Utils.grupo == 0) //grupo pequeño no pone suprimimos la primera
            // palabra Fase
            g.drawString(nFase + ": " + nomFase,
                (Utils.width - Utils.lowFont.stringWidth(nFase + ": " +
                    nomFase.toString())) / 2,
                Utils.inicioPantallaY, Graphics.TOP |
Graphics.LEFT);
        else
            g.drawString("Fase " + nFase + ": " + nomFase,
                (Utils.width - Utils.lowFont.stringWidth("Fase " +
                    nFase + ": " +
                    nomFase.toString())) / 2,
                Utils.inicioPantallaY, Graphics.TOP |
Graphics.LEFT);
        g.setFont(f);
    } else if (!Utils.cargandoPartida && !Utils.guardandoPartida) {
        redibujar = false;
        switch (parteJuego) {
            case video: //Menú principal
                if (actual != video)
                    cargar(g, video);
                else
                    getVideo(g);
                break;
            case juego: //Memoria:
                if (actual != juego)
                    cargar(g, juego);
                else
                    getJuego(g);
                break;
            case conversacion: //Acerca de...
                if (actual != conversacion)
                    cargar(g, conversacion);
                else
                    getConversacion(g);
                break;
            case mapa:
                if (actual != mapa)
```



```
        else    cargar(g, mapa);
                getMapa(g);
        break;
    case -1:
        cargarFase(g, nFase);
        break;
    }
    //Ahora dibujo el enlace a menú que estará siempre activo.
    getAccMenu(g);
}
flushGraphics();
}

/**
 * Método encargado de cargar el estado del juego al inicio de una fase.
 * Además llama a drawScreen para que pinte la imagen de cargando.
 *
 * @param g
 *      objeto Graphics necesario para llamar a drawScreen y que de
 *      esta manera se cargue la imagen de cargando
 * @param fase
 *      a cargar
 * @throws IOException
 * @throws XmlPullParserException
 */
private void cargarFase(Graphics g, int fase)
    throws XmlPullParserException, IOException {
    //Cargamos el sprite de Cargar Fase, que va a ser lo primero que
    // hagamos
    Utils.cargarImagen(Utils.spCargando);
    sprite = new Sprite(Utils.imagen, Utils.anchoIm, Utils.altoIm);
    sprite.setFrameSequence(Utils.getFrames(Utils.spCargando));
    sprite.setPosition(Utils.inicioPantallaX, Utils.inicioPantallaY);
    // el pintar la pantalla de negro solo se hará una vez.
    g.setColor(0x00000000);
    g.fillRect(0, 0, Utils.width, Utils.height);
    cargando = true;
    //Servirá de contador para saber cuantas veces pedimos que se pinte la
    // imagen de
    //cargando, solo se podrá
    int aux = 0;
    //Pintamos la imagen de cargando
    drawScreen(g);
    aux++;
    //      Inicializamos el parser
    parser = new KXmlParser();
    parser.setInput(new InputStreamReader(this.getClass()
        .getResourceAsStream("/Fase" + nFase + ".xml")));
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "Logica");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "Fase");
    nomFase.delete(0, nomFase.length());
    nomFase.append(parser.getAttributeValue(1));
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "parteJuego");
    String parte = parser.nextText();
}
```





```
parteJuego = getParte(parte);
parser.require(XmlPullParser.END_TAG, null, "parteJuego");
parser.nextTag();
parser.require(XmlPullParser.START_TAG, null, "apaActiva");
// Cargamos el array de apariencias
int apaActiva = Integer.parseInt(parser.nextText());
parser.require(XmlPullParser.END_TAG, null, "apaActiva");
Utils.actualizarApariencia(apaActiva);
Utils.cargarImagen(Utils.spOscarFren);
oscar.setImage(Utils.imagen, Utils.anchoIm, Utils.altoIm);
oscar.setFrameSequence(Utils.getFrames(Utils.spOscarFren));
while (parser.nextTag() != XmlPullParser.END_TAG) {
    if (parser.getName().equals("Videos")) {
        parser.require(XmlPullParser.START_TAG, null, "Videos");
        while (parser.nextTag() != XmlPullParser.END_TAG) {
            parser.require(XmlPullParser.START_TAG, null, "Video");
            cargarVideosFase();
            if (aux < 3) {
                drawScreen(g);
                aux++;
            }
        }
        parser.require(XmlPullParser.END_TAG, null, "Videos");
    } else if (parser.getName().equals("Conversaciones")) {
        parser.require(XmlPullParser.START_TAG, null, "Conversaciones");
        while (parser.nextTag() != XmlPullParser.END_TAG) {
            parser.require(XmlPullParser.START_TAG, null,
                "Conversacion");
            cargarConversacionFase();
            if (aux < 4) {
                drawScreen(g);
                aux++;
            }
        }
        parser.require(XmlPullParser.END_TAG, null, "Conversaciones");
    } else if (parser.getName().equals("Mapas")) {
        parser.require(XmlPullParser.START_TAG, null, "Mapas");
        while (parser.nextTag() != XmlPullParser.END_TAG) {
            parser.require(XmlPullParser.START_TAG, null, "Mapa");
            cargarMapaFase();
        }
        parser.require(XmlPullParser.END_TAG, null, "Mapas");
        drawScreen(g);
    }
}
parser.require(XmlPullParser.END_TAG, null, "Fase");

cargando = false;
// Para que compruebe colisiones.
teclaPulsada = true;
if (!Utils.cargandoPartida) {
    parser = null;
    System.gc();
    escenActual = ((Mapa) mapas.search(mapaActual).data)
        .getEscenActual();
}
redibujar = true;
}

/**
```



```
    * Carga los videos de una determinada Fase.
    * @throws XmlPullParserException
    * @throws IOException
    */
    private void cargarVideosFase() throws XmlPullParserException, IOException {

        //Vector auxiliar
        Vector textos = new Vector();

        int id = Integer.parseInt(parser.getAttributeValue(0));
        parser.nextTag();
        parser.require(XmlPullParser.START_TAG, null, "sigParte");
        int sigParte = getParte(parser.nextText());
        parser.require(XmlPullParser.END_TAG, null, "sigParte");
        parser.nextTag();
        parser.require(XmlPullParser.START_TAG, null, "vinyetas");
        StringBuffer vinyeta = new StringBuffer();
        vinyeta = getDatosImagen();
        parser.nextTag();
        parser.require(XmlPullParser.END_TAG, null, "vinyetas");
        if (parser.nextTag() == XmlPullParser.START_TAG) {
            parser.require(XmlPullParser.START_TAG, null, "texto");
            textos = cargarTextoVideo();
            parser.nextTag();
            parser.require(XmlPullParser.END_TAG, null, "Video");
        } else
            parser.require(XmlPullParser.END_TAG, null, "Video");

        videosFase.insert(new Video(id, vinyeta.toString(), textos, sigParte));
    }

    /**
     * Carga el texto de un video
     *
     * @return vector con los textos
     * @throws XmlPullParserException
     * @throws IOException
     */
    private Vector cargarTextoVideo() throws XmlPullParserException,
        IOException {

        Vector aux = new Vector();

        while (parser.nextTag() == XmlPullParser.START_TAG) {
            parser.require(XmlPullParser.START_TAG, null, "valor");
            String valor = parser.nextText();
            aux.addElement(valor);
            parser.require(XmlPullParser.END_TAG, null, "valor");
        }
        parser.require(XmlPullParser.END_TAG, null, "texto");

        return aux;
    }

    /**
     * Carga las conversaciones de una determinada Fase.
     *
     * @throws XmlPullParserException
     * @throws IOException
     */
}
```



```
private void cargarConversacionFase() throws XmlPullParserException,
    IOException {
    // Vector auxiliar
    Vector opciones = new Vector();

    int id = Integer.parseInt(parser.getAttributeValue(0));
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "color");
    String color = parser.nextText();
    parser.require(XmlPullParser.END_TAG, null, "color");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "Opciones");
    while (parser.nextTag() == XmlPullParser.START_TAG) {
        parser.require(XmlPullParser.START_TAG, null, "Opcion");
        opciones.addElement(cargarOpcionObjeto());
    }
    parser.require(XmlPullParser.END_TAG, null, "Opciones");
    parser.nextTag();
    parser.require(XmlPullParser.END_TAG, null, "Conversacion");

    converFase.insert(new Conversacion(id, getColorHex(color), opciones));
}

/**
 * Carga las estancias de una fase
 *
 * @throws XmlPullParserException
 * @throws IOException
 */
private void cargarMapaFase() throws XmlPullParserException, IOException {
    AVLTree estancias = new AVLTree();
    int id = Integer.parseInt(parser.getAttributeValue(0));
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "representa");
    StringBuffer representa = new StringBuffer();
    representa = getDatosImagen();
    parser.nextTag();
    parser.require(XmlPullParser.END_TAG, null, "representa");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "actual");
    int actual = Integer.parseInt(parser.nextText());
    parser.require(XmlPullParser.END_TAG, null, "actual");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "Estancias");
    while (parser.nextTag() == XmlPullParser.START_TAG) {
        parser.require(XmlPullParser.START_TAG, null, "Estancia");
        estancias.insert(cargarEstanciaMapa());
    }
    parser.require(XmlPullParser.END_TAG, null, "Estancias");
    parser.nextTag();
    parser.require(XmlPullParser.END_TAG, null, "Mapa");
    mapas.insert(new Mapa(id, representa.toString(), estancias, actual));
}

/**
 * Carga las estancias de una fase
 *
 * @return Estancia que forma parte de un Mapa
 * @throws XmlPullParserException
 */
```



```
    */@throws IOException
private Estancia cargarEstanciaMapa() throws XmlPullParserException,
    IOException {
    AVLTree escenarios = new AVLTree();

    Vector acciones = new Vector();
    int id = Integer.parseInt(parser.getAttributeValue(0));
    String nombre = parser.getAttributeValue(1);
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "representa");
    StringBuffer representa = new StringBuffer();
    representa = getDatosImagen();
    parser.nextTag();
    parser.require(XmlPullParser.END_TAG, null, "representa");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "actual");
    int actual = Integer.parseInt(parser.nextText());
    parser.require(XmlPullParser.END_TAG, null, "actual");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "posX");
    int posX = Integer.parseInt(parser.nextText());
    parser.require(XmlPullParser.END_TAG, null, "posX");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "posY");
    int posY = Integer.parseInt(parser.nextText());
    parser.require(XmlPullParser.END_TAG, null, "posY");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "visible");
    boolean visible = false;
    if (parser.nextText().equals("SI"))
        visible = true;
    parser.require(XmlPullParser.END_TAG, null, "visible");

    while (parser.nextTag() == XmlPullParser.START_TAG) {
        if (parser.getName().equals("Opciones")) {
            parser.require(XmlPullParser.START_TAG, null, "Opciones");
            while (parser.nextTag() != XmlPullParser.END_TAG) {
                parser.require(XmlPullParser.START_TAG, null, "Opcion");
                acciones.addElement(cargarOpcionObjeto());
            }
        } else if (parser.getName().equals("Escenarios")) {
            parser.require(XmlPullParser.START_TAG, null, "Escenarios");
            while (parser.nextTag() != XmlPullParser.END_TAG) {
                parser.require(XmlPullParser.START_TAG, null,
"Escenario");
                escenarios.insert(cargarEscenarioEstancia());
            }
        }
    }

    parser.require(XmlPullParser.END_TAG, null, "Estancia");
    return new Estancia(id, representa.toString(), nombre, actual, posX,
        posY, acciones, escenarios, visible);
}

/**
 * Carga un escenario de una determinada Estancia
 */
```



```

    * @return Escenario que forma parte de una Estancia
    * @throws XmlPullParserException
    */
private Escenario cargarEscenarioEstancia() throws XmlPullParserException,
    IOException {
    AVLTree items = new AVLTree();

    int id = Integer.parseInt(parser.getAttributeValue(0));
    String nombre = parser.getAttributeValue(1);
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "representa");
    StringBuffer representa = new StringBuffer();
    representa = getDatosImagen();
    parser.nextTag();
    parser.require(XmlPullParser.END_TAG, null, "representa");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "posX");
    int posX = Integer.parseInt(parser.nextText());
    parser.require(XmlPullParser.END_TAG, null, "posX");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "posY");
    int posY = Integer.parseInt(parser.nextText());
    parser.require(XmlPullParser.END_TAG, null, "posY");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "delante");
    boolean delante = false;
    if (parser.nextText().equals("SI"))
        delante = true;
    parser.require(XmlPullParser.END_TAG, null, "delante");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "Items");
    while (parser.nextTag() == XmlPullParser.START_TAG) {
        parser.require(XmlPullParser.START_TAG, null, "Item");
        items.insert(cargarItemEscenario());
    }
    parser.require(XmlPullParser.END_TAG, null, "Items");
    parser.nextTag();
    parser.require(XmlPullParser.END_TAG, null, "Escenario");
    System.gc();
    return new Escenario(id, nombre, representa.toString(), posX, posY,
        items, delante);
}

/**
 * Carga un escenario de una determinada Estancia
 *
 * @return Escenario que forma parte de una Estancia
 * @throws XmlPullParserException
 * @throws IOException
 */
private Item cargarItemEscenario() throws XmlPullParserException,
    IOException {
    Vector acciones = new Vector();
    AVLTree combinaciones = new AVLTree();

    int id = Integer.parseInt(parser.getAttributeValue(0));
    String nombre = parser.getAttributeValue(1);
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "imagen");

```



```
StringBuffer imagen = new StringBuffer();
imagen.append(getDatosImagen());
parser.nextTag();
parser.require(XmlPullParser.END_TAG, null, "imagen");
parser.nextTag();
parser.require(XmlPullParser.START_TAG, null, "posX");
int posX = Integer.parseInt(parser.nextText());
parser.require(XmlPullParser.END_TAG, null, "posX");
parser.nextTag();
parser.require(XmlPullParser.START_TAG, null, "posY");
int posY = Integer.parseInt(parser.nextText());
parser.require(XmlPullParser.END_TAG, null, "posY");
parser.nextTag();
parser.require(XmlPullParser.START_TAG, null, "visible");
boolean visible = false;
if (parser.nextText().equals("SI"))
    visible = true;
parser.require(XmlPullParser.END_TAG, null, "visible");
parser.nextTag();
parser.require(XmlPullParser.START_TAG, null, "animado");
boolean animado = false;
if (parser.nextText().equals("SI"))
    animado = true;
parser.require(XmlPullParser.END_TAG, null, "animado");
parser.nextTag();
parser.require(XmlPullParser.START_TAG, null, "tmpAnima");
int tmpAnima = Integer.parseInt(parser.nextText());
parser.require(XmlPullParser.END_TAG, null, "tmpAnima");
parser.nextTag();
parser.require(XmlPullParser.START_TAG, null, "hayEspecial");
StringBuffer imagenEspecial = new StringBuffer();
boolean hayImEspecial = false;
imagenEspecial.append(parser.nextText());
parser.require(XmlPullParser.END_TAG, null, "hayEspecial");
if (imagenEspecial.toString().equals("SI")) {
    imagenEspecial.delete(0, imagenEspecial.length());
    hayImEspecial = true;
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "imagenEspecial");
    imagenEspecial = getDatosImagen();
    parser.nextTag();
    parser.require(XmlPullParser.END_TAG, null, "imagenEspecial");
}
parser.nextTag();
parser.require(XmlPullParser.START_TAG, null, "posicion");
int posicion = Integer.parseInt(parser.nextText());
parser.require(XmlPullParser.END_TAG, null, "posicion");
parser.nextTag();
parser.require(XmlPullParser.START_TAG, null, "combinable");
boolean combinable = false;
if (parser.nextText().equals("SI"))
    combinable = true;
parser.require(XmlPullParser.END_TAG, null, "combinable");
if (combinable) {
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "combinaciones");
    while (parser.nextTag() == XmlPullParser.START_TAG) {
        parser.require(XmlPullParser.START_TAG, null, "combinacion");
        combinaciones.insert(cargarCombinacionItem());
    }
}
```



```
        }
        parser.require(XmlPullParser.END_TAG, null, "combinaciones");
        parser.nextTag();
        parser.require(XmlPullParser.START_TAG, null, "Opciones");
        while (parser.nextTag() == XmlPullParser.START_TAG) {
            parser.require(XmlPullParser.START_TAG, null, "Opcion");
            acciones.addElement(cargarOpcionObjeto());
        }
        parser.require(XmlPullParser.END_TAG, null, "Opciones");
        parser.nextTag();
        parser.require(XmlPullParser.END_TAG, null, "Item");
        Utils.cargarImagen(imagen.toString());
        if (hayImEspecial) {
            return new Item(nombre, imagen.toString(), posX, posY, visible,
                            animado, tmpAnima, imagenEspecial.toString(), acciones,
                            posicion, id, combinable, combinaciones);
        } else {
            return new Item(nombre, imagen.toString(), posX, posY, visible,
                            animado, tmpAnima, null, acciones, posicion, id,
                            combinable, combinaciones);
        }
    }

/**
 * Carga una combinación de un determinado Item con otro Item
 *
 * @return Combinacion de un determinado Item con otro Item
 * @throws XmlPullParserException
 * @throws IOException
 */
private Combinacion cargarCombinacionItem() throws XmlPullParserException,
        IOException {
    Vector aux = new Vector();

    int id = Integer.parseInt(parser.getAttributeValue(0));
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "activa");
    boolean activa = false;
    if (parser.nextText().equals("SI"))
        activa = true;
    parser.require(XmlPullParser.END_TAG, null, "activa");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "Consecuencias");
    while (parser.nextTag() == XmlPullParser.START_TAG) {
        parser.require(XmlPullParser.START_TAG, null, "consecuencia");
        aux.addElement(cargarConsecuenciaAccion());
    }
    parser.require(XmlPullParser.END_TAG, null, "Consecuencias");
    parser.nextTag();
    parser.require(XmlPullParser.END_TAG, null, "combinacion");
    // Runtime rt=Runtime.getRuntime() ;
    // midlet.mostrarAlerta("Combinacion libre= "+id+" ::
    // "+(rt.freeMemory()),AlertType.ERROR);
    return new Combinacion(id, activa, aux);
}

/**
 * Carga la opción de un determinado Objeto
 *
 * @return opción que forma parte de un Objeto
 */
```



```

    * @throws XmlPullParserException
    * @throws IOException
    */
private Opcion cargarOpcionObjeto() throws XmlPullParserException,
    IOException {
    Vector aux = new Vector();

    String id = parser.getAttributeValue(0);
    String texto = parser.getAttributeValue(1);
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "activa");
    boolean activa = false;
    if (parser.nextText().equals("SI"))
        activa = true;
    parser.require(XmlPullParser.END_TAG, null, "activa");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "Consecuencias");
    while (parser.nextTag() == XmlPullParser.START_TAG) {
        parser.require(XmlPullParser.START_TAG, null, "consecuencia");
        aux.addElement(cargarConsecuenciaAccion());
    }
    parser.require(XmlPullParser.END_TAG, null, "Consecuencias");
    parser.nextTag();
    parser.require(XmlPullParser.END_TAG, null, "Opcion");
    // Runtime rt=Runtime.getRuntime();
    // midlet.mostrarAlerta("Opcion libre= "+id+" ::
    // "+(rt.freeMemory()),AlertType.ERROR);
    return new Opcion(activa, texto, id, aux);
}

/**
 * Cargar las consecuencias de una acción
 *
 * @return Vector con un conjunto de resultados que formarán una
 *         consecuencia
 * @throws XmlPullParserException
 * @throws IOException
 */
private Vector cargarConsecuenciaAccion() throws XmlPullParserException,
    IOException {

    Vector resultado = new Vector();
    StringBuffer id = new StringBuffer();
    StringBuffer info = new StringBuffer();
    while (parser.nextTag() == XmlPullParser.START_TAG) {
        id.delete(0, id.length());
        info.delete(0, info.length());
        parser.require(XmlPullParser.START_TAG, null, "Resultado");
        id.append(parser.getAttributeValue(0));
        info.append(parser.nextText());
        parser.require(XmlPullParser.END_TAG, null, "Resultado");
        resultado.addElement(new Resultado(Utils.getIdResultado(id
            .toString()), info.toString()));
    }
    parser.require(XmlPullParser.END_TAG, null, "consecuencia");
    // Runtime rt=Runtime.getRuntime();
    // midlet.mostrarAlerta("Escenario libre= "+id+" ::
    // "+(rt.freeMemory()),AlertType.ERROR);
    return resultado;
}

```





```
}
/**
 * Recupera, a partir de un archivo XML, los datos descriptivos de una
 * imagen
 *
 * @throws IOException
 *         Excepción de entrada/salida
 * @throws XmlPullParserException
 *         Exception Excepción en el XML
 */
public StringBuffer getDatosImagen() throws XmlPullParserException,
        IOException {

    StringBuffer imagen = new StringBuffer();
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "id");
    imagen.append(parser.nextText() + ";");
    parser.require(XmlPullParser.END_TAG, null, "id");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "frameIni");
    imagen.append(parser.nextText() + ";");
    parser.require(XmlPullParser.END_TAG, null, "frameIni");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "numFrames");
    imagen.append(parser.nextText());
    parser.require(XmlPullParser.END_TAG, null, "numFrames");
    return imagen;
}

/**
 * Decide que y cuando cargar en cada momento
 *
 * @param g
 *         Objeto Graphics
 * @param pantalla
 *         pantalla que se desea cargar
 */
private void cargar(Graphics g, int pantalla) {
    switch (pantalla) {
        case video:
            if (actual == conversacion) {
                descargarConversacion();
            }
            if (actual == juego) {
                descargarEscenario();
            }
            actual = video;
            midlet.mostrarAlerta("llega al punto 1", AlertType.ERROR);
            Video aux = (Video) videosFase.search(videoSiguiente).data;
            midlet.mostrarAlerta("llega al punto 2", AlertType.ERROR);
            //cargamos las imágenes del video
            Utils.cargarImagen(aux.vinyeta);
            sprite.setImage(Utils.imagen, Utils.anchoIm, Utils.altoIm);
            sprite.setFrameSequence(Utils.getFrames(aux.vinyeta));
            sprite.setPosition(Utils.inicioPantallaX, Utils.inicioPantallaY);
            //Espacio entre cada una de las líneas de la viñeta
            Utils.spacing = (Utils.lowFont.getHeight() * 85) / 100;
            //Inicializamos línea
            Utils.linea = 0;
    }
}
```



```
g.setFont(Utils.lowFont);
g.setColor(Utils.highColor);
//Calcula la altura y anchura a la que empieza el texto de la
// viñeta
Utils.startHeight = (Utils.height / 2) - (Utils.getTamGrupoY() / 2);
Utils.startWidth = (Utils.width / 2) - (Utils.getTamGrupoX() / 2)
+ Utils.lowFont.getSize();
//Calculamos el numero de líneas y de letras que caben
//Cambiamos el numero de líneas que caben en pantalla para los
// videos
//el 33% de 128 es 42 (33% de 200 es 66) mas o menos, y ese es
//el espacio que dejaremos para el texto de la viñeta.
int espacioTexto = (Utils.getTamGrupoY() * 33) / 100;
Utils.numLineas = espacioTexto / Utils.lowFont.getHeight() - 1;
Utils.spaceMen = Utils.getTamGrupoX();
break;
case juego:
    if (actual == mapa)
        descargarMapa();
    else if (actual == conversacion) {
        descargarConversacion();
    }
    actual = juego;
    //Inicializamos línea
    Utils.linea = 0;
    if (Utils.grupo == 1)//Si el movil es de grupo grande
        g.setFont(Utils.highFont);
    //Calcula la altura y el ancho a la que se mostrará el nombre de
    // los items
    Utils.startHeight = (Utils.height / 2) - (Utils.getTamGrupoY() / 2)
+ (Utils.getTamGrupoY() - Utils.getSpaceTextItem());
    Utils.startWidth = (Utils.width / 2) - (Utils.getTamGrupoX() / 2)
+ Utils.lowFont.getSize();

    sprite = null;
    //cargamos sprite con una imagen del tamaño de inventario
    Utils.cargarImagen(Utils.imTmpInv);
    sprite = new Sprite(Utils.imagen, Utils.anchoIm, Utils.altoIm);
    puntero.setPosition(Utils.inicioPantallaX
+ (Utils.getTamGrupoX() / 2), Utils.inicioPantallaY
+ (Utils.getTamGrupoY() / 2));

    seleccionado = null;
    cargarEscenario();
    break;
case conversacion:
    if (actual == juego) {
        descargarEscenario();
    }
    actual = conversacion;
    tmpAcumulado = 0;
    longIdOp = 1;
    participante = (Item) seleccionado;
    seleccionado = (Opcionable) converFase.search(converSelec).data;
    Utils.spacing = (Utils.lowFont.getHeight() * 85) / 100;
    //Cargamos y colocamos el Sprite de Oscar.
    Utils.cargarImagen(Utils.spOscarCara);
    oscar.setImage(Utils.imagen, Utils.anchoIm, Utils.altoIm);
    oscar.setFrameSequence(Utils.getFrames(Utils.spOscarCara));
    oscar.setPosition(Utils.inicioPantallaX, Utils.inicioPantallaY);
    //Cargamos y colocamos el sprite del otro que habla
```



el texto de la

```
sprite = null;
Utils.cargarImagen(participante.imagenEspecial);
sprite = new Sprite(Utils.imagen, Utils.anchoIm, Utils.altoIm);
sprite.setFrameSequence(Utils
    .getFrames(participante.imagenEspecial));
sprite.setPosition(Utils.inicioPantallaX + oscar.getWidth(),
    Utils.inicioPantallaY);
g.setFont(Utils.lowFont);
//                                     Calcula la altura y anchura a la que empieza
// conversación
Utils.startHeight = Utils.inicioPantallaY + oscar.getHeight();
Utils.startWidth = Utils.inicioPantallaX
    + ((oscar.getWidth() * 32) / 100);
Utils.numLineas = (Utils.getTamGrupoY() - oscar.getHeight())
    / Utils.lowFont.getHeight() - 1;
Utils.spaceMen = Utils.getTamGrupoX()
    - ((oscar.getWidth() * 60) / 100);
opcionSele = 0;
mostrarOpciones = true;
break;
case mapa:
    if (actual == conversacion) {
        descargarConversacion();
    }
    if (actual == juego) {
        descargarEscenario();
    }
    actual = mapa;
    if (Utils.grupo == 1)//Si el movil es de grupo grande
        g.setFont(Utils.highFont);
    //Calcula la altura y el ancho a la que se mostrará el nombre de
    // los items
    Utils.startHeight = (Utils.height / 2) - (Utils.getTamGrupoY() / 2)
        + (Utils.getTamGrupoY() - Utils.getSpaceTextItem());
    Utils.startWidth = (Utils.width / 2) - (Utils.getTamGrupoX() / 2)
        + Utils.lowFont.getSize();
    puntero.setPosition(Utils.inicioPantallaX
        + (Utils.getTamGrupoX() / 2), Utils.inicioPantallaY
        + (Utils.getTamGrupoY() / 2));
    seleccionado = null;
    cargarMapa();
    break;
}
redibujar = true;
}

/**
 * Dibuja la pantalla del video
 *
 * @param g
 *      objeto Graphics
 */
public void getVideo(Graphics g) {
    midlet.mostrarAlerta("llega al punto 3", AlertType.ERROR);
    Video video = (Video) videosFase.search(videoSiguiente).data;
    midlet.mostrarAlerta("llega al punto 4", AlertType.ERROR);
    //pintamos el sprite
    sprite.paint(g);
    midlet.mostrarAlerta("llega al punto 5", AlertType.ERROR);
}
```



```
//Imprimimos el mensaje
mostrarMensaje((String) video.texto[sprite.getFrame()], "SIGUIENTE", g);
midlet.mostrarAlerta("llega al punto 6", AlertType.ERROR);
}

/**
 * Dibuja la pantalla de juego
 *
 * @param g
 *      objeto Graphics
 */
public void getJuego(Graphics g) {

    //          Color blanco para los nombres de los Items
    g.setColor(Utils.lowColor);
    LM.paint(g, 0, 0);
    //el inventario no se muestra hasta que no estén todos los resultados
    //de un determinado Item, ejecutados, porque si no se puede dar el caso
    //de que hagan reAddInventInt y entonces cambia el seleccionado
    //A no ser que el inventario ya estuviese siendo mostrado
    if (mostrarInventario) {
        getInventario(g);
        if (itemsInventario.size() != 0) {
            g.setColor(0x00DDDDDD);
            getLineatexto(g, ((Item) seleccionado).nombre);
        }
        if (mostrarOpciones)
            getOpciones(g);
    }

    if (mostrarMensaje)
        getMensaje(g);

    //si lo seleccionado es un Item
    if (colisionado && seleccionado != null) {
        g.setColor(0x00DDDDDD);
        getLineatexto(g, ((Item) seleccionado).nombre);
        if (mostrarOpciones) {
            getOpciones(g);
        }
        g.setColor(Utils.lowColor);
    }
    //si la colisión no ha sido con un Item, pues habrá sido con Oscar
    else if (colisionado && seleccionado == null)
        getLineatexto(g, "Oscar Romero");
    else if (!mostrarInventario && !mostrarOpciones)//si no señalas nada
        getLineatexto(g, "Ir a...");
}

/**
 * Dibuja el inventario
 *
 * @param g
 *      objeto Graphics
 */
public void getInventario(Graphics g) {
    //          Dibujamos el rectángulo marrón
    g.setColor(0x996633);
    int tempX = Utils.inicioPantallaX;
    int tempY = Utils.inicioPantallaY + Utils.getTamGrupoY()
```



```
g.fillRect(tempX, tempY, Utils.getSpaceTextItem(1, altoInventario);
//Dibujamos el cuadrado rojo (tiene mismo ancho y alto claro)
g.setColor(0x00DD0000);
int tempX2 = Utils.inicioPantallaX + (anchoInventario / 2)
            - (altoInventario / 2);
g.fillRect(tempX2, tempY, altoInventario, altoInventario);
//Ahora las flechas
g.setColor(Utils.highColor);
g.fillTriangle(tempX2 - (anchoInventario / 20), tempY
              + (altoInventario / 14), Utils.inicioPantallaX, tempY
              + (altoInventario / 2), tempX2 - (anchoInventario / 20), tempY
              + (altoInventario) - (altoInventario / 14));

g.fillTriangle(tempX2 + altoInventario + (anchoInventario / 20), tempY
              + (altoInventario / 14), Utils.inicioPantallaX
              + anchoInventario, tempY + (altoInventario / 2), tempX2
              + altoInventario + (anchoInventario / 20), tempY
              + (altoInventario) - (altoInventario / 14));
//Ahora mostramos el inventario
if (itemsInventario.size() != 0) {
    //Si por casualidad acabamos de sacar del inventario, el objeto que
    //estaba seleccionado por invSelec, lo corregimos aquí.
    if (invSelec >= itemsInventario.size())
        invSelec = 0;
    seleccionado = (Item) itemsInventario.elementAt(invSelec);
    // cargamos la imagen del inventario para ese objeto
    Utils.cargarImagen(((Item) seleccionado).imagenEspecial);
    sprite.setImage(Utils.imagen, Utils.anchIm, Utils.altoIm);
    tempY = tempY + (altoInventario / 2) - (sprite.getHeight() / 2);
    tempX = tempX2 + (altoInventario / 2) - (sprite.getWidth() / 2);
    sprite.setPosition(tempX, tempY);
    sprite.paint(g);
} else {
    tempY = tempY + (altoInventario / 2) - (sprite.getHeight() / 2);
    tempX = tempX2 + (altoInventario / 2) - (sprite.getWidth() / 2);
    //dibujamos cuadrado negro con la palabra Vacío en el centro.
    g.setColor(Utils.highColor);
    g.fillRect(tempX, tempY, sprite.getWidth(), sprite.getHeight());
    g.setColor(Utils.lowColor);
    g.drawString("VACIO", tempX + sprite.getWidth() / 2
                - (Utils.highFont.charWidth('a') * ("Vacio".length() / 2)),
                tempY + (sprite.getHeight() / 2)
                - Utils.highFont.getHeight(), Graphics.TOP
                | Graphics.LEFT);
}

}

/**
 * Dibuja la pantalla de la conversación
 *
 * @param g
 *      objeto Graphics
 */
public void getConversacion(Graphics g) {
    //indica que tecla se ha tocado, si el botón izquierda o el derecha.
    boolean izquierda = false;
    //indica si se ha comprobado si el botón pulsado fue el izquierda o el
    // derecha.
    boolean comprobado = false;
```



```
Conversacion con = ((Conversacion) seleccionado);
// Ponemos la pantalla en blanco
g.setColor(0xFFFFFFFF);
g.fillRect(Utils.inicioPantallaX, Utils.inicioPantallaY, Utils
    .getTamGrupoX(), Utils.getTamGrupoY());
// Dibujamos el cuadrado, en el color correspondiente,
// del otro que habla
g.setColor(con.color);
g.fillRect(Utils.inicioPantallaX + oscar.getWidth(),
    Utils.inicioPantallaY, sprite.getWidth(), sprite.getHeight());
g.setColor(Utils.highColor);
if (mostrarOpciones) {
    // Dibujamos los rectangulos marrones
    g.setColor(0x996633);
    g.fillRect(Utils.inicioPantallaX, Utils.inicioPantallaY
        + oscar.getHeight(), (oscar.getWidth() * 30) / 100, Utils
        .getTamGrupoY()
        - oscar.getHeight());
    g.fillRect(Utils.inicioPantallaX + Utils.getTamGrupoX()
        - ((oscar.getWidth() * 30) / 100), Utils.inicioPantallaY
        + oscar.getHeight(), (oscar.getWidth() * 30) / 100, Utils
        .getTamGrupoY()
        - oscar.getHeight());
    // Dibujamos las flechas
    g.setColor(Utils.highColor);
    g.fillTriangle(Utils.inicioPantallaX
        + ((oscar.getWidth() * 25) / 100), Utils.inicioPantallaY
        + oscar.getHeight() + 2, Utils.inicioPantallaX,
        Utils.inicioPantallaY + oscar.getHeight()
        + ((Utils.getTamGrupoY() -
oscar.getHeight()) / 2),
        Utils.inicioPantallaX + ((oscar.getWidth() * 25) / 100),
        Utils.inicioPantallaY + Utils.getTamGrupoY() - 2);

    int tempX = Utils.inicioPantallaX + Utils.getTamGrupoX();
    g.fillTriangle(tempX - ((oscar.getWidth() * 25) / 100),
        Utils.inicioPantallaY + oscar.getHeight() + 2, tempX,
        Utils.inicioPantallaY + oscar.getHeight()
        + ((Utils.getTamGrupoY() -
oscar.getHeight()) / 2),
        tempX - ((oscar.getWidth() * 25) / 100),
        Utils.inicioPantallaY + Utils.getTamGrupoY() - 2);
    // Ahora hayamos las opciones que pueden ser mostradas
    // salir se pondrá a true cuando se cumplan las condiciones de una
    // opción
    boolean salir = false;
    while (!salir) {
        if (con.opciones[opcionSele] != null) {
            if (con.opciones[opcionSele].id.length() == longIdOp) {
                if (longIdOp == 1)
                    salir = true;
                else {
                    if
((con.opciones[opcionSele].id.substring(0,
longIdOp -
1).equals(iniIdOp.toString()))
                    salir = true;
                else {
                    if (opcionSele <
con.opciones.length - 1)
```



```

else        opcionSele++;
            opcionSele = 0;
        }
    } else {
        if (opcionSele < con.opciones.length - 1)
            opcionSele++;
        else
            opcionSele = 0;
    }
} else {
    if (opcionSele < con.opciones.length - 1)
        opcionSele++;
    else
        opcionSele = 0;
}
if (con.opciones[opcionSele] != null) {
    if (!seleccionado.getAccionActiva(opcionSele)
        && salir == true) {
        if (opcionSele < con.opciones.length - 1)
            opcionSele++;
        else
            opcionSele = 0;
        salir = false;
    }
}
//
// Ahora imprimimos la opción seleccionada
Utils.mensaje.delete(0, Utils.mensaje.length());
Utils.mensaje.append(con.opciones[opcionSele].texto);
}
//Dibujamos una línea negra entre las 2 caras
g.drawLine(Utils.inicioPantallaX + oscar.getWidth(),
            Utils.inicioPantallaY,
            Utils.inicioPantallaX + oscar.getWidth(), Utils.inicioPantallaY
            + oscar.getHeight());
//Dibujamos una línea negra entre el espacio de las caras y el del
//texto
g.drawLine(Utils.inicioPantallaX, Utils.inicioPantallaY
            + oscar.getHeight(), Utils.inicioPantallaX
            + Utils.getTamGrupoX(), Utils.inicioPantallaY
            + oscar.getHeight());
//introducimos sus caras
oscar.paint(g);
sprite.paint(g);
if (habParti)
    g.setColor(con.color);
//mostramos el mensaje
if (mostrarOpciones)
    mostrarMensaje(Utils.mensaje.toString(), "DECIR", g);
else
    mostrarMensaje(Utils.mensaje.toString(), "CONTINUAR", g);
}

/**
 * Dibuja la pantalla del mapa
 *
 * @param g
 *      objeto Graphics

```



```
*/public void getMapa(Graphics g) {
    //          Color blanco para los nombres de las Estancia
    g.setColor(Utils.lowColor);
    LM.paint(g, 0, 0);

    if (mostrarMensaje)
        getMensaje(g);

    //si lo seleccionado es una Estancia
    if (colisionado && seleccionado != null) {
        g.setColor(0x00DDDDDD);
        getLineatexto(g, ((Estancia) seleccionado).nombre);
        if (mostrarOpciones) {
            getOpciones(g);
        }
        g.setColor(Utils.lowColor);
    }
}

/**
 * Dibuja el acceso directo al menú.
 */
@param g
objeto Graphics
*/
public void getAccMenu(Graphics g) {
    int fontStyle = g.getFont().getStyle();
    g.setFont(Utils.highFont);
    /*
     * if(Utils.grupo==0) g.setFont(Utils.lowFont);
     */
    g.setColor(getColorHex("Crema"));
    g.fillRect(posXMenu, posYMenu, anchoMenu, altoMenu);
    g.setColor(getColorHex("Rojo"));
    g.drawRect(posXMenu, posYMenu, anchoMenu, altoMenu);
    g.setColor(Utils.highColor);
    g.drawString("Menu", posXMenu + 2, posYMenu - 2, Graphics.TOP
        | Graphics.LEFT);
    //en caso de que sea necesario ponemos la antigua fuente
    if (fontStyle == Font.STYLE_PLAIN)
        g.setFont(Utils.lowFont);
}

/**
 * Realiza la carga del escenario de juego.
 */
private void cargarEscenario() {
    if (!escenActual.nombre.equals(ultimoEscenario.toString())) {
        //para que no entre en el if que hay mas abajo y asi lo ponga en la
        //posición inicial del mapa
        pasoActual = 0;
        cargarAreaCaminable();
        ultimoEscenario.delete(0, ultimoEscenario.length());
        ultimoEscenario.append(escenActual.nombre);
    }
    escenDetras = null;
    escenDetras = new Sprite(escenActual.representa);
    escenDetras.setFrame(escenActual.imDetras);
}
```





```
LM.append(puntero);
Item temp2;
Node aux = escenActual.items.root;
escenActual.items.inorden(aux);
aux = escenActual.items.next();
while (aux != null) {
    temp2 = (Item) aux.data;
    if (temp2.posicion == 2)
        //Primero metemos los Items de delante
        LM.append(temp2);
    aux = escenActual.items.next();
}
//Luego el escenario de delante
if (escenActual.delante) //si tiene alguna parte del escenario que va
    // delante...
    LM.append(escenActual.representa);
//introducimos a Oscar
//Carga la posición de oscar en el escenario
oscar.setPosition(Utils.inicioPantallaX + escenActual.getPosInicialX(),
    Utils.inicioPantallaY + escenActual.getPosInicialY());
//pero si nos encontramos en el mismo escenario que antes
//y oscar se había movido, cargamos su ultima posición
if (pasoActual > 0) {
    int x = getPixelX(buscaRuta.getX(buscaRuta.path[pasoActual]));
    int y = getPixelY(buscaRuta.getY(buscaRuta.path[pasoActual]));
    oscar.setPosition(x, y);
}
LM.append(oscar);
aux = escenActual.items.root;
escenActual.items.inorden(aux);
aux = escenActual.items.next();
while (aux != null) {
    temp2 = (Item) aux.data;
    if (temp2.posicion == 1)
        //
        //Luego añadimos los Items de
        //detrás de Oscar
        LM.append(temp2);
    aux = escenActual.items.next();
}
//
//Luego el escenario de detrás del prota
LM.append(escenDetras);
//el siguiente sprite será el área caminable
//LM.append(escenCaminable); //El área caminable ya no lo metemos,
// porque
//esta precalculado.
aux = escenActual.items.root;
escenActual.items.inorden(aux);
aux = escenActual.items.next();
while (aux != null) {
    temp2 = (Item) aux.data;
    if (temp2.posicion == 0)
        //
        //Por último los items de detrás del
        //escenario.
        LM.append(temp2);
    aux = escenActual.items.next();
}
//Para que compruebe colisiones.
teclaPulsada = true;
letra = -6;
}
```



```
/**
 * Carga el área caminable del escenario actual.
 */
private void cargarAreaCaminable() {
    //lo primero en cargar es el sprite del puntero que comprueba el área
    // caminable
    //el cual tendrá un tamaño de 2x2
    //guardamos la posición actual del puntero
    int tempX = puntero.getX();
    int tempY = puntero.getY();
    int frame = puntero.getFrame();

    Utils.cargarImagen(Utils.imPunCami);
    puntero.setImage(Utils.imagen, Utils.anchoIm, Utils.altoIm);
    LM.append(puntero);

    escenCaminable = null;
    escenCaminable = new Sprite(escenActual.representa);
    escenCaminable.setFrame(escenActual.imCaminable);

    //el siguiente sprite será el área caminable
    LM.append(escenCaminable);
    //Calculamos para ahorrárnoslo en el bucle for
    int temPosIniX = Utils.inicioPantallaX + (anchoCelda / 2);
    int temPosIniY = Utils.inicioPantallaY + (altoCelda / 2);

    int j = 0;
    boolean caminable = false;
    int maxi = buscaRuta.walkability.length;
    int maxj = buscaRuta.mapWidth;

    //walkability hay que modificarlo para indicar cuales de las celdas son
    //caminables y cuales no. Si establecemos true, es caminable , y si no
    // pues no.
    //En este for lo que hace es comprobar una celda y si esa celda es
    // caminable,
    //todas las conectadas directamente a ella, tb lo serán. Y viceversa.
    //Es decir si el ancho y alto de cada celda es de 2x2, entonces aquí
    //simularemos que son de 14x14.Un cuadrado de celdas de 7x7
    //Ejemplo, empezamos por la celda 4,4 --> Si es caminable, pues 0,0 0,1
    // 0,2
    //0,3 0,4 0,5 0,6 1,0 1,1 1,2... 6,3 6,4 6,5 6,6 tb serán caminables, y
    // la siguiente en mirar
    //será la celda 4,11. Si llegásemos al final de la fila 4, pasaríamos a
    // mirar
    //la celda 11,4
    //Puede pasar que las ultimas filas (como mucho 4)queden sin mirar, si
    // son
    //caminables o no, pero no importa, ya que esas filas corresponden al
    // área
    //donde se pone el nombre de los items, y eso nunca es caminable.

    //Lado del cuadrado imaginario que vamos a formar para de 7x7.
    //Debe ser siempre un numero impar, para que la celda del centro este
    //justo en el centro
    int ladoCuadrado = 7;
    for (int i = 3 * maxj; i < maxi; i++) {
        for (j = 3; j < maxj; j = j + ladoCuadrado) {
            puntero.setPosition(temPosIniX + j * anchoCelda, temPosIniY
```



```

caminable = puntero.colidesWith(escenCaminable, true);
int numRecorre = (ladoCuadrado / 2) + 1;
for (int k = 0; k < numRecorre; k++) {
    for (int z = 0; z < numRecorre; z++) {
        if (i + j + k * maxj + z < maxi)
            buscaRuta.walkability[i + j + k * maxj + z]
= caminable;

        if (i + j + k * maxj - z < maxi)
            buscaRuta.walkability[i + j + k * maxj - z]
= caminable;

        if (i + j - k * maxj + z < maxi && k != 0)
            buscaRuta.walkability[i + j - k * maxj + z]
= caminable;

        if (i + j - k * maxj - z < maxi && k != 0)
            buscaRuta.walkability[i + j - k * maxj - z] =
caminable;
    }
}
//Multiplicamos por 3, porque al mirar en cuadrado, iremos mirando
//la fila 1 , la 4, la 7...
i = i + (maxj * 7) - 1;
}
LM.remove(puntero);
LM.remove(escenCaminable);
//restablecemos la posición e imagen del puntero
Utils.cargarImagen(Utils.spPuntero);
puntero.setImage(Utils.imagen, Utils.anchoIm, Utils.altoIm);
puntero.setFrameSequence(Utils.getFrames(Utils.spPuntero));
puntero.setFrame(frame);
puntero.setPosition(tempX, tempY);
}

/**
 * Realiza la descarga del escenario de juego.
 */
private void descargarEscenario() {
    //lo primero en cargar es el sprite del puntero
    LM.remove(puntero);
    Item temp2;

    Node aux = escenActual.items.root;
    escenActual.items.inorden(aux);
    aux = escenActual.items.next();
    while (aux != null) {
        temp2 = (Item) aux.data;
        if (temp2.posicion == 2)
            //Primero metemos los Items de delante
            LM.remove(temp2);
        aux = escenActual.items.next();
    }
    //Luego el escenario de delante
    if (escenActual.delante) //si tiene alguna parte del escenario que va
        // delante...
        LM.remove(escenActual.representa);

    LM.remove(oscar);

    aux = escenActual.items.root;

```



```
escenActual.items.inorden(aux);
aux = escenActual.items.next();
while (aux != null) {
    temp2 = (Item) aux.data;
    if (temp2.posicion == 1)
        // Luego borramos los Items de
detrás de Oscar
        LM.remove(temp2);
    aux = escenActual.items.next();
}
// Luego el escenario de detrás del prota
LM.remove(escenDetras);

aux = escenActual.items.root;
escenActual.items.inorden(aux);
aux = escenActual.items.next();
while (aux != null) {
    temp2 = (Item) aux.data;
    if (temp2.posicion == 0)
        //Por último los items de detrás del escenario.
        LM.remove(temp2);
    aux = escenActual.items.next();
}
}

/**
 * Realiza la carga del mapa de juego.
 */
private void cargarMapa() {
    //Se pone a 0, si al salir al mapa vuelves a entrar al mismo escenario
    //que antes, pues te pondrá en la posición inicial del escenario
    //y no en la posición que tenías, en ese escenario, antes de salir al
    // mapa.
    pasoActual = 0;
    Mapa temp = (Mapa) mapas.search(mapaActual).data;
    //lo primero en cargar es el sprite del puntero
    LM.append(puntero);
    Estancia temp2;

    // Cargamos el Mapa
    LM.append(temp.representa);

    //Cargamos la representación de los distintas estancias, que
    //irán detrás del mapa de tal manera que al pasar el puntero
    //por encima, estas colisionen con el
    Node aux = ((Mapa) mapas.search(mapaActual).data).estancias.root;
    ((Mapa) mapas.search(mapaActual).data).estancias.inorden(aux);
    aux = ((Mapa) mapas.search(mapaActual).data).estancias.next();
    while (aux != null) {
        temp2 = (Estancia) aux.data;
        LM.append(temp2.representa);
        aux = ((Mapa) mapas.search(mapaActual).data).estancias.next();
    }

    // Para que compruebe colisiones.
    teclaPulsada = true;
}

/**
 * Realiza la descarga del mapa de juego.
```



```
private void descargarMapa() {
    Mapa temp = (Mapa) mapas.search(mapaActual).data;
    //lo primero en cargar es el sprite del puntero
    LM.remove(puntero);
    Estancia temp2;

    // Descargamos el Mapa
    LM.remove(temp.representa);

    //Descargamos la representación de los distintas estancias, que
    //irán detrás del mapa de tal manera que al pasar el puntero
    //por encima, estas colisionen con el
    Node aux = ((Mapa) mapas.search(mapaActual).data).estancias.root;
    ((Mapa) mapas.search(mapaActual).data).estancias.inorden(aux);
    aux = ((Mapa) mapas.search(mapaActual).data).estancias.next();
    while (aux != null) {
        temp2 = (Estancia) aux.data;
        LM.remove(temp2.representa);
        aux = ((Mapa) mapas.search(mapaActual).data).estancias.next();
    }
}

/**
 * Realiza la descarga de la conversación
 */
private void descargarConversacion() {
    seleccionado = participante;
    participante = null;
    // Cargamos el Sprite de Oscar.
    Utils.cargarImagen(Utils.spOscarFren);
    oscar.setImage(Utils.imagen, Utils.anchoIm, Utils.altoIm);
    oscar.setFrameSequence(Utils.getFrames(Utils.spOscarFren));
    mostrarOpciones = false;
}

/**
 * Comprueba si hay colisión entre el puntero y algún Item del escenario
 * actual, también se incluye a Oscar Solo hace falta que colisione con uno,
 * para que devuelva verdadero
 *
 * @return true si hay colisión entre el puntero y un Item del escenario
 * actual
 */
private boolean colisionPunItem() {
    Item aux;
    Node auxN = escenActual.items.root;
    escenActual.items.inorden(auxN);
    auxN = escenActual.items.next();
    // primero miramos a ver si colisiona primero con los que están delante
    // de oscar.
    while (auxN != null) {
        aux = (Item) auxN.data;
        if (aux.posicion == 2 && aux.isVisible()) {
            if (puntero.collidesWith(aux, true)) {
                //Si el objeto es distinto al actualmente seleccionado
                //entonces reiniciamos letra
                if (seleccionado != null
                    && ((Item) seleccionado).id != aux.id)
                    letra = -6;
            }
        }
    }
}
```



```
        seleccionado = aux;
        System.gc();
        return true;
    }
}
auxN = escenActual.items.next();
}

//Si queréis habilitar la opción de seleccionar a Oscar con el puntero
//Descomentar la siguiente sentencia if
//      En caso de que antes no colisionase,a ver si colisiona con Oscar
/*
* if(puntero.collidesWith(oscar,true)){ seleccionado=null; letra=-6;
* System.gc(); return true; }
*/

//detrás de Oscar.
auxN = escenActual.items.root;
escenActual.items.inorden(auxN);
auxN = escenActual.items.next();
//      Luego los que están entre Oscar y el escenario.
while (auxN != null) {
    aux = (Item) auxN.data;
    if (aux.posicion == 1 && aux.isVisible()) {
        if (puntero.collidesWith(aux, true)) {
            //Si el objeto es distinto al actualmente seleccionado
            //entonces reiniciamos letra
            if (seleccionado != null
                && ((Item) seleccionado).id != aux.id)
                letra = -6;
            seleccionado = aux;
            System.gc();
            return true;
        }
    }
    auxN = escenActual.items.next();
}
//Detrás del escenario.
auxN = escenActual.items.root;
escenActual.items.inorden(auxN);
auxN = escenActual.items.next();
//      Por último los que están por detrás del escenario
while (auxN != null) {
    aux = (Item) auxN.data;
    if (aux.posicion == 0 && aux.isVisible()) {
        if (puntero.collidesWith(aux, true)) {
            //Si el objeto es distinto al actualmente seleccionado
            //entonces reiniciamos letra
            if (seleccionado != null
                && ((Item) seleccionado).id != aux.id)
                letra = -6;
            seleccionado = aux;
            System.gc();
            return true;
        }
    }
    auxN = escenActual.items.next();
}
return false;
}
```



```
/**
 * Comprueba si hay colisión entre el puntero y alguna Estancia del mapa.
 * Solo hace falta que colisione con una, para que devuelva verdadero
 */
* @return true si hay colisión entre el puntero y una Estancia del mapa
*/
private boolean colisionPunEstan() {
    Sprite aux;
    Node auxN = ((Mapa) mapas.search(mapaActual).data).estancias.root;
    ((Mapa) mapas.search(mapaActual).data).estancias.inorden(auxN);
    auxN = ((Mapa) mapas.search(mapaActual).data).estancias.next();
    while (auxN != null) {
        aux = ((Estancia) auxN.data).representa;
        if (puntero.collidesWith(aux, true)) {
            seleccionado = (Estancia) auxN.data;
            return true;
        }
        auxN = ((Mapa) mapas.search(mapaActual).data).estancias.next();
    }
    return false;
}

/**
 * Para aquellos Items que están establecidos como animados, este método
 * comprueba si ha llegado su hora de cambiar de frame.
 */
private void animarItems() {
    Item aux;
    Node auxN = escenActual.items.root;
    escenActual.items.inorden(auxN);
    auxN = escenActual.items.next();
    while (auxN != null) {
        aux = (Item) auxN.data;
        if (aux.animado && tmpAcumulado % aux.tmpAnima == 0) {
            aux.nextFrame();
            redibujar = true;
        }
        auxN = escenActual.items.next();
    }
}

/**
 * método que salta como respuesta a la pulsación de una acción, de entre
 * las disponibles para un objeto.
 */
private void opcionesIO() {
    int idOpActivas[] = new int[50];
    int contador = 0;
    int idOp = opcionSele;
    if (actual == juego) {
        for (int i = 0; i < seleccionado.longAcciones(); i++) {
            if (seleccionado.getAccionActiva(i)) {
                idOpActivas[contador] = i;
                contador++;
            }
        }
        idOp = idOpActivas[opcionSele];
    }
    Vector aux;
```



```
        if (!seleccionado.getConseActivaAcc(idOp)) {
            aux = (Vector) seleccionado.getConseguenciaAcc(idOp);
            for (int i = 0; i < aux.size(); i++)
                resultados.addElement(aux.elementAt(i));
        }
    }

    /**
     * método que salta como respuesta al intento de combinar dos Items
     */
    private void conseCombinacion() {
        AVLTree arbol;
        Vector aux;
        int idSele = -1; // Combinaciones con id=-1 (la de por defecto)
        if (seleccionado != null
            && // si no es oscar y
            combinado.combinaciones.search(((Item) seleccionado).id) != null)
// es
            // un
            // objeto
            // combinable
            // con
            // este
            if (combinado.getComActiva(((Item) seleccionado).id)) // Si la
                // combinación
                // está activa.
                idSele = ((Item) seleccionado).id;

        aux = (Vector) combinado.getConseguenciaCom(idSele);
        for (int i = 0; i < aux.size(); i++)
            resultados.addElement(aux.elementAt(i));
    }

    /**
     * Método encargado de dirigir la ejecución de cada uno de los resultados de
     * una accion
     */
    private void ejecutarResultado() {
        Resultado aux = (Resultado) resultados.firstElement();
        switch (aux.id) {
            case Utils.reSalir:
                mostrarOpciones = false;
                opcionSele = 0;
                break;
            case Utils.reCamFase:
                nFase++;
                switch (actual) {
                    case juego:
                        descargarEscenario();
                        break;
                    case mapa:
                        descargarMapa();
                        break;
                    case conversacion:
                        descargarConversacion();
                        break;
                }
                videoSiguiente = 0;
                mapas.deleteAllElements();
            }
    }
}
```





```
itemsInventario.removeAllElements();
videoSiguiente.removeAllElements();
converFase.deleteAllElements();
midlet.preguardado.removeAllElements();
parteJuego = -1;
actual = -1;
redibujar = true;
break;
case Utils.reMensaje:
    ocupado = true;
    mostrarMensaje = true;
    redibujar = true;
    Utils.mensaje.delete(0, Utils.mensaje.length());
    Utils.mensaje.append(aux.info);
    break;
case Utils.reAddInvent:
    itemsInventario.addElement(buscaItem(Integer.parseInt(aux.info)));
    if (!Utils.cargandoPartida) {
        mostrarInventario = true;
        parteJuego = juego;
    }
    redibujar = true;
    invSelec = itemsInventario.size() - 1;
    midlet.pregarduar(aux);
    break;
case Utils.reAddInventSin:
    itemsInventario.addElement(buscaItem(Integer.parseInt(aux.info)));
    invSelec = itemsInventario.size() - 1;
    midlet.pregarduar(aux);
    break;
case Utils.reDelInvent:
    Item auxItem;
    int id = Integer.parseInt(aux.info);
    int size = itemsInventario.size();
    for (int i = 0; i < size; i++) {
        auxItem = (Item) itemsInventario.elementAt(i);
        if (auxItem.id == id)
            itemsInventario.removeElementAt(i);
        i = size;
    }
    midlet.pregarduar(aux);
    break;
case Utils.reVideo:
    videoSiguiente = Integer.parseInt(aux.info);
    parteJuego = video;
    ocupado = true;
    redibujar = true;
    break;
case Utils.reIrEstancia:
    id = Integer.parseInt(aux.info);
    ((Mapa) mapas.search(mapaActual).data).activa = id;
    escenActual = ((Mapa) mapas.search(mapaActual).data)
        .getEscenActual();
    parteJuego = juego;
    //no lo pongo ocupado, ya que se supone que tiene carácter de
    //ultima acción
    break;
case Utils.reCamConsecuCon:
    int fin = 0;
    int inicio = 0;
```



```
while (aux.info.charAt(fin) != ';') {
    fin++;
}
id = Integer.parseInt(aux.info.substring(inicio, fin));
inicio = fin + 1;
fin++;
//identificador de la opción seleccionada
while (aux.info.charAt(fin) != ';') {
    fin++;
}
String idOp = aux.info.substring(inicio, fin);
inicio = fin + 1;
fin++;
Conversacion auxCon = (Conversacion) converFase.search(id).data;
if (auxCon.id == id)
    auxCon.setConseActivaAcc(idOp, Integer.parseInt(aux.info
        .substring(inicio)));
else {
    AVLTree arbol = converFase;
    auxCon = (Conversacion) arbol.search(id).data;
    if (auxCon != null)
        auxCon.setConseActivaAcc(idOp, Integer.parseInt(aux.info
            .substring(inicio)));
}
midlet.pregarduar(aux);
break;
case Utils.reCamConsecuIt:
    fin = 0;
    inicio = 0;
    while (aux.info.charAt(fin) != ';') {
        fin++;
    }
    auxItem = buscaItem(Integer.parseInt(aux.info
        .substring(inicio, fin)));
    inicio = fin + 1;
    fin++;
    //
    //                                identificador de la opción seleccionada
    while (aux.info.charAt(fin) != ';') {
        fin++;
    }
    idOp = aux.info.substring(inicio, fin);
    inicio = fin + 1;
    fin++;
    auxItem.setConseActivaAcc(idOp, Integer.parseInt(aux.info
        .substring(inicio)));
    midlet.pregarduar(aux);
    break;
case Utils.reCamConsecuEs:
    fin = 0;
    inicio = 0;
    while (aux.info.charAt(fin) != ';') {
        fin++;
    }
    Estancia auxEstancia = buscaEstancia(Integer.parseInt(aux.info
        .substring(inicio, fin)));
    inicio = fin + 1;
    fin++;
    //
    //                                identificador de la opción seleccionada
    while (aux.info.charAt(fin) != ';') {
```



```
        }        fin++;
        idOp = aux.info.substring(inicio, fin);
        inicio = fin + 1;
        fin++;
        auxEstancia.setConseActivaAcc(idOp, Integer.parseInt(aux.info
            .substring(inicio)));
        midlet.pregarduar(aux);
        break;
case Utils.reCamConsecuCom:
    fin = 0;
    inicio = 0;
    while (aux.info.charAt(fin) != ';') {
        fin++;
    }
    auxItem = buscaItem(Integer.parseInt(aux.info
        .substring(inicio, fin)));
    inicio = fin + 1;
    fin++;
    //identificador de la combinación que se busca
    while (aux.info.charAt(fin) != ';') {
        fin++;
    }
    int idCom = Integer.parseInt(aux.info.substring(inicio, fin));
    inicio = fin + 1;
    fin++;
    auxItem.setConseActivaCom(idCom, Integer.parseInt(aux.info
        .substring(inicio)));
    midlet.pregarduar(aux);
    break;
case Utils.reMapa:
    parteJuego = mapa;
    break;
case Utils.reCamMapa:
    mapaActual = Integer.parseInt(aux.info);
    if (actual == mapa) {
        //Le digo que el actual es video, ya que como actual es mapa,
        // pues
        //al entrar en drawScreen, recargara el escenario
        actual = video;
        redibujar = true;
    }
    break;
case Utils.reConversacion:
    converSelec = Integer.parseInt(aux.info);
    parteJuego = conversacion;
    redibujar = true;
    break;
case Utils.reRespuesta:
    Utils.mensaje.delete(0, Utils.mensaje.length());
    Utils.mensaje.append(aux.info);
    habParti = true;
    ocupado = true;
    break;
case Utils.reRespuOscar:
    Utils.mensaje.delete(0, Utils.mensaje.length());
    Utils.mensaje.append(aux.info);
    ocupado = true;
    break;
case Utils.reJuego:
```



```
        parteJuego = juego;
        break;
case Utils.reCargaOpSup:
    auxCon = ((Conversacion) seleccionado);
    longIdOp = auxCon.opciones[opcionSele].id.length()
                - Integer.parseInt(aux.info);
    iniIdOp.delete(0, iniIdOp.length());
    iniIdOp.append(auxCon.opciones[opcionSele].id.substring(0,
        longIdOp - 1));
    break;
case Utils.reCargaOpInf:
    auxCon = ((Conversacion) seleccionado);
    longIdOp = auxCon.opciones[opcionSele].id.length() + 1;
    iniIdOp.delete(0, iniIdOp.length());
    iniIdOp.append(auxCon.opciones[opcionSele].id);
    break;
case Utils.reQuitarOp:
    fin = 0;
    inicio = 0;
    while (aux.info.charAt(fin) != ';') {
        fin++;
    }
    auxCon = (Conversacion) converFase.search(Integer.parseInt(aux.info
        .substring(inicio, fin))).data;
    inicio = fin + 1;
    auxCon.borrarOp(aux.info.substring(inicio));
    midlet.pregarduar(aux);
    break;
case Utils.reQuitarOpYSup:
    fin = 0;
    inicio = 0;
    while (aux.info.charAt(fin) != ';') {
        fin++;
    }
    auxCon = (Conversacion) converFase.search(Integer.parseInt(aux.info
        .substring(inicio, fin))).data;
    inicio = fin + 1;
    fin++;
    while (aux.info.charAt(fin) != ';') {
        fin++;
    }
    Opcion auxOp = auxCon.getOpcion(aux.info.substring(inicio, fin));
    inicio = fin + 1;
    //Contador del numero de opciones inferiores, para una determinada
    // opción
    int contador = 0;
    longIdOp = auxOp.id.length()
                - Integer.parseInt(aux.info.substring(inicio));
    iniIdOp.delete(0, iniIdOp.length());
    iniIdOp.append(auxOp.id.substring(0, longIdOp));
    for (int i = 0; i < auxCon.opciones.length; i++) {
        if (auxCon.opciones[i] != null) {
            if ((auxCon.opciones[i].id.substring(0, longIdOp)
                .equals(iniIdOp.toString()))
                && auxCon.opciones[i].id.length() >
longIdOp) {
                contador++;
            }
        }
    }
}
```



```
//Si no hay mas inferiores, aparte de la de volver a las
//principales...
if (contador <= 1) {
    //Ponemos la Principal y la actual a null
    for (int i = 0; i < auxCon.opciones.length; i++) {
        if (auxCon.opciones[i] != null) {
            if ((auxCon.opciones[i].id.substring(0, longIdOp)
                .equals(iniIdOp.toString())) {
                //
                //la opción seleccionada será lo ultimo en
                // borrar, porque sino mas abajo nos daría
                //problemas con null
                if (auxCon.opciones[i].id != auxOp.id)
                    auxCon.opciones[i] = null;
            }
        }
    }
}
//
// Contador del numero de opciones
superiores, para una determinada
// opción
//esto es necesario ya que el cargaOpSup viene implícito en este
// resultado
//y si resulta que al eliminar al sup, el nivel superior se queda
// sin ninguna
//opción, entonces deberemos ir otro nivel por encima.
int temp = contador;
contador = 0;
for (int i = 0; i < auxCon.opciones.length; i++) {
    if (auxCon.opciones[i] != null) {
        if ((auxCon.opciones[i].id.length() == longIdOp)) {
            contador++;
        }
    }
}
if (contador == 0) {
    //Si no hay ninguna inmediatamente superior a la actual,
    // tendremos
    //que ir a orta 2 niveles por encima, ej del grupo
    //de opciones aba,abb ... , al grupo de opciones a, b...
    longIdOp = auxOp.id.length() - 2;
    iniIdOp.delete(0, iniIdOp.length());
    iniIdOp.append(auxOp.id.substring(0, longIdOp));
}
if (temp <= 1)
    auxCon.borrarOp(auxOp.id);
midlet.pregarduar(aux);
break;
case Utils.reCamActivaOpIt:
    fin = 0;
    inicio = 0;
    while (aux.info.charAt(fin) != ';') {
        fin++;
    }
    auxItem = buscaItem(Integer.parseInt(aux.info
        .substring(inicio, fin)));
    inicio = fin + 1;
    auxOp = auxItem.getOpcion(aux.info.substring(inicio));
    if (auxOp.activa)
        auxOp.activa = false;
```



```
        else        auxOp.activa = true;
        midlet.pregarduar(aux);
        break;
case Utils.reCamActivaOpEs:
    fin = 0;
    inicio = 0;
    while (aux.info.charAt(fin) != ';') {
        fin++;
    }
    auxEstancia = buscaEstancia(Integer.parseInt(aux.info.substring(
        inicio, fin)));
    inicio = fin + 1;
    auxOp = auxEstancia.getOpcion(aux.info.substring(inicio));
    if (auxOp.activa)
        auxOp.activa = false;
    else
        auxOp.activa = true;
    midlet.pregarduar(aux);
    break;
case Utils.reCamActivaOpCon:
    fin = 0;
    inicio = 0;
    while (aux.info.charAt(fin) != ';') {
        fin++;
    }
    id = Integer.parseInt(aux.info.substring(inicio, fin));
    inicio = fin + 1;
    idOp = aux.info.substring(inicio);

    auxCon = (Conversacion) converFase.search(id).data;
    if (auxCon.id == id) {
        auxOp = auxCon.getOpcion(idOp);
        if (auxOp.activa)
            auxOp.activa = false;
        else
            auxOp.activa = true;
    } else {
        AVLTree arbol = converFase;
        auxCon = (Conversacion) arbol.search(id).data;
        if (auxCon != null) {
            auxOp = auxCon.getOpcion(idOp);
            if (auxOp.activa)
                auxOp.activa = false;
            else
                auxOp.activa = true;
        }
    }
    midlet.pregarduar(aux);
    break;
case Utils.reCaminar:
    if (buscarRuta())
        caminando = true;
    break;
case Utils.reCamActivaCom:
    fin = 0;
    inicio = 0;
    while (aux.info.charAt(fin) != ';') {
        fin++;
    }
```



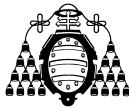
```
auxItem = buscaItem(Integer.parseInt(aux.info
    .substring(inicio, fin)));
inicio = fin + 1;
idCom = Integer.parseInt(aux.info.substring(inicio));
Combinacion auxCom = (Combinacion) auxItem.combinaciones
    .search(idCom).data;
if (auxCom.activa == false)
    auxCom.activa = true;
else
    auxCom.activa = false;
midlet.pregarduar(aux);
break;
case Utils.reEliminarItem:
    boolean eliminado = false;
    id = Integer.parseInt(aux.info);
    //Si eliminamos un Item sin antes descargarlo del escenario,
    //el Item aparecerá en todos los escenarios, formará
    //parte de LM para siempre
    if (actual == juego)
        descargarEscenario();

    //haya el escenario al que pertenece el Item

    //1º miramos si es un Item perteneciente al escenario actual
    if (buscaItemEsc(escenActual, id, true) != null) {
        escenActual.items.deleteKey(id);
        eliminado = true;
    }

    if (!eliminado) {
        //2º miramos si es un Item perteneciente a la estancia actual.
        Node auxN = ((Mapa) mapas.search(mapaActual).data)
            .getEstanActual().escenarios.root;
        Escenario auxEscenario;
        ((Mapa) mapas.search(mapaActual).data).getEstanActual().escenarios
            .inorden(auxN);
        auxN = ((Mapa)
            mapas.search(mapaActual).data).getEstanActual().escenarios
            .next();
        while (auxN != null && eliminado == false) {
            auxEscenario = (Escenario) auxN.data;
            //Si el escenario no es el actual, que ya comprobamos
            // antes...
            if (auxEscenario.id != escenActual.id) {
                if (buscaItemEsc(auxEscenario, id, true) != null) {
                    auxEscenario.items.deleteKey(id);
                    eliminado = true;
                }
            }
            auxN = ((Mapa) mapas.search(mapaActual).data)
                .getEstanActual().escenarios.next();
        }
    }

    if (!eliminado) {
        //3º Miramos si es un Item perteneciente a el mapa actual.
        Node auxN = ((Mapa) mapas.search(mapaActual).data).estancias.root;
        ((Mapa) mapas.search(mapaActual).data).estancias.inorden(auxN);
        auxN = ((Mapa) mapas.search(mapaActual).data).estancias.next();
        while (auxN != null && eliminado == false) {
```



```
auxEstancia = (Estancia) auxN.data;      Si la estancia no
es la actual, que ya comprobamos

// antes...
if (auxEstancia.id != ((Mapa)
    .getEstanActual().id) {
    Node auxN2 = auxEstancia.escenarios.root;
    auxEstancia.escenarios.inorden(auxN2);
    auxN2 = auxEstancia.escenarios.next();
    while (auxN2 != null) {
        Escenario auxEscenario = (Escenario)
        if (buscaItemEsc(auxEscenario, id, true) !=
            auxEscenario.items.deleteKey(id);
            eliminado = true;
        }
        auxN2 = auxEstancia.escenarios.next();
    }
}
auxN = ((Mapa) mapas.search(mapaActual).data).estancias
    .next();
}
}

if (!eliminado) {
    //Por último miramos si es de otro mapa.
    Node auxN = mapas.root;
    Mapa auxMapa;
    mapas.inorden(auxN);
    auxN = mapas.next();
    while (auxN != null && eliminado == false) {
        auxMapa = (Mapa) auxN.data;
        if (auxMapa.id != ((Mapa)
            mapas.search(mapaActual).data).id) {
            Node auxN2 = auxMapa.estancias.root;
            auxMapa.estancias.inorden(auxN2);
            auxN2 = auxMapa.estancias.next();
            while (auxN2 != null) {
                auxEstancia = (Estancia) auxN2.data;
                Node auxN3 = auxEstancia.escenarios.root;
                auxEstancia.escenarios.inorden(auxN3);
                auxN3 = auxEstancia.escenarios.next();
                while (auxN3 != null) {
                    Escenario auxEscenario =
                    if (buscaItemEsc(auxEscenario, id,
                        true) != null) {
                        auxEscenario.items.deleteKey(id);
                        eliminado = true;
                    }
                    auxN3 =
                    auxEstancia.escenarios.next();
                }
                auxN2 = auxMapa.estancias.next();
            }
        }
    }
    auxN = mapas.next();
}
```





```
    }  
    }  
  
    size = itemsInventario.size();  
    //Y también lo borramos del inventario  
    for (int i = 0; i < size; i++) {  
        auxItem = (Item) itemsInventario.elementAt(i);  
        if (auxItem.id == id) {  
            itemsInventario.removeElementAt(i);  
            i = size; //salimos del for  
        }  
    }  
    //Una vez borrado el Item, volvemos a cargar el escenario.  
    if (actual == juego)  
        cargarEscenario();  
    midlet.pregarduar(aux);  
    break;  
case Utils.reEliminarEstan:  
    eliminado = false;  
    id = Integer.parseInt(aux.info);  
    //Si eliminamos una estancia sin antes descargarlo el mapa,  
    //la estancia aparecerá en todos los mapa, formará  
    //parte de LM para siempre  
    if (actual == mapa)  
        descargarMapa();  
  
    //haya el mapa al que pertenece la estancia  
  
    //1º Miramos si es la estancia actual  
    auxEstancia = (Estancia) seleccionado;  
    if (auxEstancia.id == id) {  
        ((Mapa) mapas.search(mapaActual).data).estancias.deleteKey(id);  
        eliminado = true;  
    }  
  
    if (!eliminado) {  
        //2º Miramos si es una Estancia perteneciente a el mapa actual.  
        Node auxN = ((Mapa) mapas.search(mapaActual).data).estancias.root;  
        ((Mapa) mapas.search(mapaActual).data).estancias.inorden(auxN);  
        auxN = ((Mapa) mapas.search(mapaActual).data).estancias.next();  
        while (auxN != null && eliminado == false) {  
            auxEstancia = (Estancia) auxN.data;  
            // Si la estancia no  
            es la actual, que ya comprobamos  
            // antes...  
            if (auxEstancia.id != ((Mapa)  
mapas.search(mapaActual).data)  
                .getEstanActual().id) {  
                if (auxEstancia.id == id) {  
                    ((Mapa)  
mapas.search(mapaActual).data).estancias  
                        .delete(auxN);  
                    eliminado = true;  
                }  
            }  
            auxN = ((Mapa) mapas.search(mapaActual).data).estancias  
                .next();  
        }  
    }  
}
```



```
if (!eliminado) {
    // Por último miramos si es de otro mapa.
    Node auxN = mapas.root;
    mapas.inorden(auxN);
    auxN = mapas.next();
    while (auxN != null && eliminado == false) {
        Mapa auxMapa = (Mapa) auxN.data;
        if (auxMapa.id != ((Mapa)
mapas.search(mapaActual).data).id) {
            Node auxN2 = auxMapa.estancias.root;
            auxMapa.estancias.inorden(auxN2);
            auxN2 = auxMapa.estancias.next();
            while (auxN2 != null) {
                auxEstancia = (Estancia) auxN2.data;
                if (auxEstancia.id == id) {
                    auxMapa.estancias.delete(auxN2);
                    eliminado = true;
                }
                auxN2 = auxMapa.estancias.next();
            }
            auxN = mapas.next();
        }
    }

    //Una vez borrada la estancia, volvemos a cargar el mapa.
    if (actual == mapa)
        cargarMapa();
    midlet.pregarduar(aux);
    break;
case Utils.reSalirJuego:
    midlet.salir();
    break;
case Utils.reCambiaEscenario:
    id = Integer.parseInt(aux.info);
    descargarEscenario();
    ((Mapa) mapas.search(mapaActual).data).getEstanActual().actual = id;
    escenActual = ((Mapa) mapas.search(mapaActual).data)
        .getEscenActual();
    //Le digo que el actual es video, ya que como parteJuego es juego,
    // pues
    //al entrar en drawScreen, recargara el escenario
    actual = video;
    redibujar = true;
    break;
case Utils.reCamNombreIt:
    fin = 0;
    inicio = 0;
    while (aux.info.charAt(fin) != ';') {
        fin++;
    }
    auxItem = buscaItem(Integer.parseInt(aux.info
        .substring(inicio, fin)));
    inicio = fin + 1;
    auxItem.nombre = aux.info.substring(inicio);
    letra = -6;
    midlet.pregarduar(aux);
    break;
case Utils.reCamNombreEs:
    fin = 0;
```



```
        while (aux.info.charAt(fin) != ';') {
            fin++;
        }
        auxEstancia = buscaEstancia(Integer.parseInt(aux.info.substring(
            inicio, fin)));
        inicio = fin + 1;
        auxEstancia.nombre = aux.info.substring(inicio);
        letra = -6;
        midlet.pregarduar(aux);
        break;
    case Utils.reCaminarBlo:
        if (buscarRuta()) {
            caminando = true;
            ocupado = true;
        }
        break;
    case Utils.reCambiaVisibleIt:
        auxItem = buscaItem(Integer.parseInt(aux.info));
        if (auxItem.isVisible())
            auxItem.setVisible(false);
        else
            auxItem.setVisible(true);
        midlet.pregarduar(aux);
        break;
    case Utils.reCambiaVisibleEs:
        auxEstancia = buscaEstancia(Integer.parseInt(aux.info));
        if (auxEstancia.representa.isVisible())
            auxEstancia.representa.setVisible(false);
        else
            auxEstancia.representa.setVisible(true);
        midlet.pregarduar(aux);
        break;
    case Utils.reCombinando:
        combinando = true;
        combinado = (Item) seleccionado;
        break;
    case Utils.reCamApariencia:
        Utils.actualizarApariencia(Integer.parseInt(aux.info));
        Utils.cargarImagen(Utils.spOscarFren);
        oscar.setImage(Utils.imagen, Utils.anchoIm, Utils.altoIm);
        oscar.setFrameSequence(Utils.getFrames(Utils.spOscarFren));
        midlet.pregarduar(aux);
        break;
    }
    resultados.removeElementAt(0);
    if (resultados.size() == 0) {
        opcionSele = 0;
        Utils.cargandoPartida = false;
        if (intentaCombinar) {
            combinando = false;
            intentaCombinar = false;
            combinado = null;
            letra = -6;
        }
    }
}
```

/\*\*

\* Muestra las opciones del objeto seleccionado en la pantalla de juego



```
    * @param g
    *     objeto Graphics
    */
    public void getOpciones(Graphics g) {
        Utils.numOps = 0;
        Utils.spacing = Utils.lowFont.getHeight();
        for (int i = 0; i < seleccionado.longAcciones(); i++)
            if (seleccionado.getAccionActiva(i))
                Utils.numOps++;
        if (Utils.grupo == 0) { //grupo pequeño
            altoOpciones = Utils.lowFont.getHeight() * (Utils.numOps);
            Utils.spacing = Utils.lowFont.getHeight();
        } else {
            altoOpciones = Utils.highFont.getHeight() * (Utils.numOps);
            Utils.spacing = Utils.highFont.getHeight();
        }
        //Dibujamos el cuadrado gris
        g.setColor(0x00DDDDDD);
        int tempX = Utils.inicioPantallaX + Utils.getTamGrupoX()
            - anchoOpciones;
        int tempY = Utils.inicioPantallaY + Utils.getTamGrupoY()
            - Utils.getSpaceTextItem() - altoOpciones;
        g.fillRect(tempX, tempY, anchoOpciones, altoOpciones);
        g.setColor(Utils.highColor);
        //Ahora la flechita
        g.fillTriangle(tempX, tempY + (opcionSele * Utils.spacing)
            + Utils.spacing, tempX + 7, tempY
            + (opcionSele * Utils.spacing) + (Utils.spacing / 2), tempX,
            tempY + (opcionSele * Utils.spacing));
        //Guardamos los valores de startWidth y height en x e y
        int aux;
        aux = tempX;
        tempX = Utils.startWidth;
        Utils.startWidth = aux + ((anchoOpciones * 25) / 100);
        aux = tempY;
        tempY = Utils.startHeight;
        Utils.startHeight = aux;
        //Escribimos las opciones
        g.setColor(Utils.highColor);
        int cuentaActiva = 0;

        for (int i = 0; i < seleccionado.longAcciones(); i++) {
            if (seleccionado.getAccionActiva(i) == true) {
                g.drawString(seleccionado.getNombreAccion(i), Utils.startWidth,
                    Utils.startHeight + (cuentaActiva * Utils.spacing),
                    Graphics.TOP | Graphics.LEFT);
                cuentaActiva++;
            }
        }

        //Restablecemos el valor de StartWidth y StartHeight y spacing
        Utils.startWidth = tempX;
        Utils.startHeight = tempY;
        Utils.spacing = (Utils.lowFont.getHeight() * 85) / 100;
    }

    /**
     * Muestra Un mensaje en la pantalla de juego
    */
}
```



```
    * @param g
    *      objeto Graphics
    */
    public void getMensaje(Graphics g) {
        //Dibujamos el cuadrado gris
        g.setColor(0x00DDDDDD);
        int tempX = Utils.inicioPantallaX
            + ((Utils.getTamGrupoX() - anchoMensaje) / 2);
        int tempY = Utils.inicioPantallaY
            + ((Utils.getTamGrupoY() - Utils.getSpaceTextItem()
                - altoMensaje) / 2);
        g.fillRect(tempX, tempY, anchoMensaje, altoMensaje);

        g.setColor(Utils.highColor);
        g.setFont(Utils.lowFont);
        //Guardamos los valores de startWidth, startHeight
        //numLetras y numLineas
        int nAux;
        nAux = tempX;
        tempX = Utils.startWidth;
        Utils.startWidth = nAux + Utils.lowFont.getSize();
        nAux = tempY;
        tempY = Utils.startHeight;
        Utils.startHeight = nAux;
        nAux = Utils.spaceMen;
        int nAux2 = Utils.numLineas;

        //Modificamos algunos valores necesarios para la escritura
        Utils.spaceMen = anchoMensaje;
        switch (Utils.grupo) {
            case 0://128x128
                Utils.numLineas = altoMensaje / Utils.lowFont.getHeight() - 1;
                break;
            case 1://176x200
                Utils.numLineas = altoMensaje / Utils.lowFont.getHeight();
                break;
        }

        //Imprimimos el mensaje
        mostrarMensaje(Utils.mensaje.toString(), "CERRAR", g);

        //Restablecemos el valor de StartWidth y StartHeight
        Utils.startWidth = tempX;
        Utils.startHeight = tempY;

        if (Utils.grupo == 1)//Si el movil es de grupo grande
            g.setFont(Utils.highFont);
    }

    /**
     * Reajusta algunas variables como son el ancho y alto de la pantalla. Carga
     * el numero de letras que caben en el ancho de la pantalla, y el numero de
     * líneas que caben en el alto de la pantalla
     */
    public void ajustarVariables() {
        setFullScreenMode(true);
        irMenu = false;
        //ancho y alto de la pantalla
    }
```



los items

```
Utils.width = getWidth();
//          Espacio entre cada una de las líneas de la viñeta
Utils.spacing = (Utils.lowFont.getHeight() * 85) / 100;
if (parteJuego == juego || parteJuego == mapa) {

    //          Calcula la altura a la que se mostrará el nombre de

    Utils.startHeight = (Utils.height / 2) - (Utils.getTamGrupoY() / 2)
        + (Utils.getTamGrupoY() - Utils.getSpaceTextItem());

    Utils.startWidth = (Utils.width / 2) - (Utils.getTamGrupoX() / 2)
        + Utils.lowFont.getSize();
} else if (parteJuego == video) { //estara en Video
    //espacio que dejaremos para el texto de la viñeta
    int espacioTexto = (Utils.getTamGrupoY() * 33) / 100;
    Utils.numLineas = espacioTexto / Utils.lowFont.getHeight() - 1;
    Utils.startHeight = (Utils.height / 2) - (Utils.getTamGrupoY() / 2);
    Utils.startWidth = (Utils.width / 2) - (Utils.getTamGrupoX() / 2)
        + Utils.lowFont.getSize();
} else if (parteJuego == conversacion) {
    Utils.startHeight = Utils.inicioPantallaY + oscar.getHeight();
    Utils.numLineas = (Utils.getTamGrupoY() - oscar.getHeight())
        / Utils.lowFont.getHeight() - 1;
    if (mostrarOpciones) {
        Utils.startWidth = Utils.inicioPantallaX
            + ((oscar.getWidth() * 32) / 100);
        Utils.spaceMen = Utils.getTamGrupoX()
            - ((oscar.getWidth() * 60) / 100);
    } else {
        Utils.startWidth = Utils.inicioPantallaX;
        Utils.spaceMen = Utils.getTamGrupoX();
    }
}

}

/**
 * Devuelve el entero que identifica a una parte del juego
 *
 * @param parte
 *      Parte del juego de la que quiero saber el identificador
 * @return el identificador de la parte del juego
 */
private int getParte(String parte) {
    if (parte.equals("video"))
        return video;
    if (parte.equals("juego"))
        return juego;
    if (parte.equals("mapa"))
        return mapa;
    return -1;
}

/**
 * Haya el valor hexadecimal de un determinado color.
 *
 * @param color
 *      Color del que queremos saber su valor hexadecimal
 * @return el valor en hexadecimal de un determinado color
```



```
private int getColorHex(String color) {
    if (color.equals("Rojo"))
        return 0xFF0000;
    if (color.equals("Verde"))
        return 0x009900;
    if (color.equals("Crema"))
        return 0xFFFF99;
    if (color.equals("Amarillo"))
        return 0xD1940C;
    if (color.equals("Marrón"))
        return 0x663300;

    if (color.equals("Azul"))
        return 0x003366;
    if (color.equals("Negro"))
        return 0x000000;
    if (color.equals("Naranja"))
        return 0xFF6600;
    if (color.equals("Rosa"))
        return 0xFF00FF;
    return -1;
}

/**
 * Imprime un mensaje por pantalla
 *
 * @param mensaje
 *         Mensaje a imprimir
 * @param finMensaje
 *         Cadena de texto a poner al final del mensaje para indicar que
 *         el mensaje se ha terminado
 * @param g
 *         Objeto Graphics.
 */
public static void mostrarMensaje(String mensaje, String finMensaje,
    Graphics g) {
    System.gc();
    // inicializamos el valor de mensaje, y palabraMensaje
    Utils.mensaje.delete(0, Utils.mensaje.length());
    Utils.palabrasMensaje.removeAllElements();
    //Metemos el texto
    Utils.mensaje.append(mensaje);
    //Hayamos cada una de las palabras del mensaje
    StringBuffer aux = new StringBuffer();
    char temp;
    for (int i = 0; i < Utils.mensaje.length(); i++) {
        temp = Utils.mensaje.charAt(i);
        if (temp != ' ')
            aux.append(temp);
        else {
            Utils.palabrasMensaje.addElement(aux.toString());
            aux.delete(0, aux.length());
        }
        if (i + 1 == Utils.mensaje.length()) {
            Utils.palabrasMensaje.addElement(aux.toString());
            aux.delete(0, aux.length());
        }
    }
}
```



```
//int aux2 = 0;      aux2 almacena el numero de letras que lleva una línea
//Identifica en que línea toca imprimir
int lineaAux = 0;
int size = Utils.palabrasMensaje.size();
int tamPalabra = 0;
String palabra;
for (int i = 0; i < size; i++) {
    palabra = ((String) Utils.palabrasMensaje.elementAt(i));
    tamPalabra = palabra.length() * Utils.widthLF;
    //la ideal sería la de abajo, pero como carga demasiado pues uso la
    // anterior
    //tamPalabra=Utils.lowFont.stringWidth(palabra);
    //Si es salto de línea, imprimimos y aumentamos en una la lineaAux
    if (palabra.equals("\n")) {
        if (lineaAux >= Utils.linea
            && lineaAux <= Utils.numLineas + Utils.linea)
            g
                .drawString(
                    aux.toString(),
                    Utils.startWidth,
                    Utils.startHeight
                    +
                    ((lineaAux - Utils.linea) * Utils.spacing),
                    Graphics.TOP |
                    Graphics.LEFT);
        else if (lineaAux > Utils.numLineas + Utils.linea)
            size = 0; //para salir del bucle
            aux.delete(0, aux.length());
            //pasamos a la línea siguiente y dejamos un hueco en blanco
            // entre medias.
            lineaAux = lineaAux + 2;
            aux2 = 0;
        } else {
            if (tamPalabra > Utils.spaceMen) {
                if (lineaAux >= Utils.linea
                    && lineaAux <= Utils.numLineas +
                    Utils.linea)
                    //
                    imprimimos lo que hay previamente
                    g
                        .drawString(
                            aux.toString(),
                            Utils.startWidth,
                            Utils.startHeight
                            + ((lineaAux - Utils.linea) * Utils.spacing),
                            Graphics.TOP |
                            Graphics.LEFT);
                aux.delete(0, aux.length());
                lineaAux++;
                aux2 = 0;
                //Imprimimos la primera parte
                int noCaben = (tamPalabra - Utils.spaceMen) /
                    Utils.widthLF;
                aux.append(palabra.substring(0, palabra.length() - noCaben
                    - 1));
                if (lineaAux >= Utils.linea
                    && lineaAux <= Utils.numLineas +
                    Utils.linea)
```





```
g.drawString(
    aux.toString(),
    Utils.startWidth,
    Utils.startHeight
    + ((lineaAux - Utils.linea) * Utils.spacing),
    Graphics.TOP |
Graphics.LEFT);

aux.delete(0, aux.length());
lineaAux++;
aux2 = 0;
// Y luego el final
que no entraba lo añadimos a la palabra
//de la siguiente línea
aux.append(palabra
    .substring(palabra.length() - noCaben - 1)
    + " ");
aux2 = aux2
    + palabra.substring(palabra.length() -
noCaben - 1)
    .length() * Utils.widthLF
    + Utils.lowFont.charWidth(' ');
} else {
    //si el numero de letras de esa línea (aux2) es menor que
    // el numero
    //de letras que caben en esta pantalla, restándole además
    //el tamaño de la ultima palabra,
    if (aux2 + tamPalabra < Utils.spaceMen) {
        //Introducimos en aux, la palabra seguida de un
        espacio
        aux.append(palabra + " ");
        //actualizamos el valor de aux2
        aux2 = aux2 + tamPalabra +
Utils.lowFont.charWidth(' ');
    } else {
        //Solo imprime las filas que serán visibles, es decir
        //para no imprimir líneas que no se verían pues
        estaría
        //en la parte negra
        if (lineaAux >= Utils.linea
            && lineaAux <= Utils.numLineas
+ Utils.linea)
            g
                .drawString(
                    aux.toString(),
                    Utils.startWidth,
                    Utils.startHeight
                        + ((lineaAux - Utils.linea) * Utils.spacing),
                    Graphics.TOP | Graphics.LEFT);
            else if (lineaAux > Utils.numLineas + Utils.linea)
                size = 0; //para salir del bucle
            //al imprimir una línea borramos el contenido de aux
            aux.delete(0, aux.length());
            //aumentamos el valor de lineaAux
```



```
introducido
//aux2 vuelve a ser 0
aux2 = 0;
//y en aux metemos la palabra que no se ha

//en la línea que se acaba de imprimir, porque el
// numeroLetras
//de esa línea ya sobrepasaba los limites
aux.append(palabra + " ");
aux2 = aux2 + tamPalabra +

Utils.lowFont.charWidth(' ');

    }
}

//Cuando llegamos al final, imprimimos la línea (este como este)
if (i + 1 == Utils.palabrasMensaje.size()) {
    if (lineaAux >= Utils.linea
        && lineaAux <= Utils.numLineas + Utils.linea)
        g
            .drawString(
                aux.toString(),
                Utils.startWidth,
                Utils.startHeight
                +
                ((lineaAux - Utils.linea) * Utils.spacing),
                Graphics.TOP |
                Graphics.LEFT);
    }
}
//
// el numero de líneas que ocupa el mensaje.
Utils.numLineasMensaje = lineaAux;
//si el numero de líneas que caben en la pantalla + la línea
//en la cual se imprime la primera línea es menor que el numero
//de líneas que ocupa el mensaje (teniendo en cuenta que si el
//mensaje ocupa 7 líneas, numLineas valdrá 6), se pinta una flecha
//hacia abajo
if (Utils.numLineas + Utils.linea < Utils.numLineasMensaje) {
    g.setColor(0x00000000);
    if (Utils.grupo == 1)//Tamaño grande
        g.fillTriangle((Utils.width / 2) - 7, Utils.startHeight
            + (Utils.numLineas * Utils.spacing) + Utils.spacing
            + (Utils.spacing / 4), (Utils.width / 2),
            Utils.startHeight + (Utils.numLineas *
                Utils.spacing)
            + Utils.spacing + (Utils.spacing /
                4) + 7,
            (Utils.width / 2) + 7, Utils.startHeight
            + (Utils.numLineas *
                Utils.spacing)
            + Utils.spacing + (Utils.spacing /
                4));
    else
        g.fillTriangle((Utils.width / 2) - 7, Utils.startHeight
            + (Utils.numLineas * Utils.spacing) + Utils.spacing
            + (Utils.spacing / 2), (Utils.width / 2),
            Utils.startHeight + (Utils.numLineas *
                Utils.spacing)
            + Utils.spacing + (Utils.spacing /
                2) + 7,
```



```
Utils.spacing)
(Utls.width / 2) + 7 * (Utils.startHeight *
(Utils.numLineas
+ Utils.spacing + (Utils.spacing /
2));
    } else {
        g.drawString(finMensaje, (Utils.width - Utils.lowFont
        .stringWidth(finMensaje)) / 2, Utils.startHeight
        + (Utils.numLineas * Utils.spacing) + 2 * Utils.spacing
        - Utils.lowFont.getHeight(), Graphics.TOP | Graphics.LEFT);
    }
}

/**
 * Imprime un mensaje en la línea de textos
 *
 * @param g
 *      objeto Graphics
 * @param texto
 *      el mensaje a mostrar.
 */
private void getLineatexto(Graphics g, String texto) {
    if (combinando)
        texto = "USAR " + combinado.nombre + " CON " + texto;
    int tamPalabra = texto.length() * Utils.widthLF;

    //la ideal sería la de abajo, pero como carga demasiado pues uso la
    // anterior
    //tamPalabra=Utils.lowFont.stringWidth(texto);
    if (tamPalabra > disponible) {
        if (letra < texto.length()) {
            if (tmpAcumulado % 80 == 0)
                letra++;
        } else
            letra = -6;

        redibujar = true;
    }
    if (letra < 0) {
        if (texto.length() >= numLetras - 1)
            g.drawString(texto.substring(0, numLetras - 1),
                Utils.startWidth, Utils.startHeight, Graphics.TOP
                | Graphics.LEFT);
        else
            g.drawString(texto, Utils.startWidth, Utils.startHeight,
                Graphics.TOP | Graphics.LEFT);
    } else {
        if (texto.length() >= letra + numLetras - 1)
            g.drawString(texto.substring(letra, letra + numLetras - 1),
                Utils.startWidth, Utils.startHeight, Graphics.TOP
                | Graphics.LEFT);
        else
            g.drawString(texto.substring(letra), Utils.startWidth,
                Utils.startHeight, Graphics.TOP | Graphics.LEFT);
    }
}

/**
 * Mueve la boca del personaje que esta hablando
 */
```



```
private void moverBoca() {
    if (mostrarOpciones && tmpAcumulado % 200 == 0) {
        if (habParti)
            sprite.nextFrame();
        else
            oscar.nextFrame();
        redibujar = true;
    }
}

/**
 * Cierra la boca del personaje que esta hablando
 */
private void cerrarBoca() {
    if (habParti)
        sprite.setFrame(0);
    else
        oscar.setFrame(0);
}

/**
 * Se encarga de buscar la ruta adecuada, para llevar a nuestro persona
 * desde su posición actual, hasta el sitio de la pantalla en el que hayamos
 * pulsado.
 *
 * @return Boolean indicando si existe ruta hasta una determinada celda o
 *         no.
 */
private boolean buscarRuta() {
    int celdaPunX = getCeldaX(puntero.getX() - Utils.inicioPantallaX
        + (puntero.getWidth() / 2));
    int celdaPunY = getCeldaY(puntero.getY() - Utils.inicioPantallaY
        + (puntero.getHeight() / 2));
    int celdaOscX = getCeldaX(oscar.getX() - Utils.inicioPantallaX
        + (oscar.getWidth() - anchoCelda) / 2);
    int celdaOscY = getCeldaY(oscar.getY() - Utils.inicioPantallaY
        + (oscar.getHeight()));

    if (buscaRuta.walkability[celdaPunX + (celdaPunY * buscaRuta.mapWidth)] == false)
    {
        while (buscaRuta.walkability[celdaPunX
            + (celdaPunY * buscaRuta.mapWidth)] == false) {
            if (celdaPunY != celdaOscY) {
                if (celdaPunY < celdaOscY)
                    celdaPunY = Math.min(celdaOscY, celdaPunY + 7);
                if (celdaPunY > celdaOscY)
                    celdaPunY = Math.max(celdaOscY, celdaPunY - 7);
            } else {
                if (celdaPunX < celdaOscX)
                    celdaPunX = Math.min(celdaOscX, celdaPunX + 7);
                if (celdaPunX > celdaOscX)
                    celdaPunX = Math.max(celdaOscX, celdaPunX - 7);
            }
        }
    }

    //Solo busca una ruta si las celdas de inicio y destino son distintas.
    if (celdaPunX + (celdaPunY * buscaRuta.mapWidth) != celdaOscX
        + (celdaOscY * buscaRuta.mapWidth)) {
        if (buscaRuta.findPath(celdaOscX, celdaOscY, celdaPunX, celdaPunY)) {

```



```
        pasoActual = 0;
        return true;
    }
    return false;
}

/**
 * Nos devuelve la posición X de la celda en la que esta situada un
 * determinado píxel
 *
 * @param pixel
 *     La posición en píxeles de la que se quiere sabe a que celda
 *     corresponde
 * @return la posición X de la celda a la que está apuntando el puntero.
 */
private int getCeldaX(int pixel) {
    return pixel / anchoCelda;
}

/**
 * Nos devuelve la posición Y de la celda en la que esta situada un
 * determinado píxel
 *
 * @param pixel
 *     La posición en píxeles de la que se quiere sabe a que celda
 *     corresponde
 * @return la posición Y de la celda a la que está apuntando el puntero.
 */
private int getCeldaY(int pixel) {
    return pixel / altoCelda;
}

/**
 * Nos devuelve la posición X del píxel en el que esta situado una
 * determinada celda
 *
 * @param celda
 *     La celda de la que se quiere sabe a que píxel corresponde
 * @return el píxel X de la celda a la que está apuntando el puntero.
 */
private int getPixelX(int celda) {
    return Utils.inicioPantallaX + (celda * anchoCelda)
        - (oscar.getWidth() - anchoCelda) / 2;
}

/**
 * Nos devuelve la posición Y del píxel en el que esta situado una
 * determinada celda
 *
 * @param celda
 *     La celda de la que se quiere sabe a que píxel corresponde
 * @return el píxel Y de la celda a la que está apuntando el puntero.
 */
private int getPixelY(int celda) {
    return Utils.inicioPantallaY + (celda * altoCelda) - oscar.getHeight();
}

/**
 * Haya la Estancia cuyo identificador coincide con el parámetro que le
```



```
    * pasamos.
    * @param id
    *     Identificador de la estancia a buscar
    * @return La estancia buscado.
    */
private Estancia buscaEstancia(int id) {
    AVLTree arbol;
    Estancia auxEstancia;

    //1º Miramos si es la estancia actual
    if (seleccionado != null) {
        auxEstancia = (Estancia) seleccionado;
        if (auxEstancia.id == id)
            return auxEstancia;
    }

    //2º Miramos si es una Estancia perteneciente a el mapa actual.
    arbol = ((Mapa) mapas.search(mapaActual).data).estancias;
    auxEstancia = (Estancia) arbol.search(id).data;
    if (auxEstancia != null)
        return auxEstancia;

    //Por último miramos si es de otro mapa.
    Node auxN = mapas.root;
    mapas.inorden(auxN);
    auxN = mapas.next();
    while (auxN != null) {
        Mapa auxMapa = (Mapa) auxN.data;
        if (auxMapa.id != ((Mapa) mapas.search(mapaActual).data).id) {
            arbol = ((Mapa) mapas.search(mapaActual).data).estancias;
            auxEstancia = (Estancia) arbol.search(id).data;
            if (auxEstancia != null)
                return auxEstancia;
        }
        auxN = mapas.next();
    }

    //Si no encuentra ninguna estancia con dicho identificador
    return null;
}

/**
 * Haya el Item cuyo identificador coincide con el parámetro que le pasamos.
 */
* @param id
*     Identificador del Item a buscar
* @return El Item buscado.
*/
private Item buscaItem(int id) {
    Item auxItem;
    //Primero miramos si es el Item seleccionado.
    if (actual != conversacion && seleccionado != null) {
        auxItem = (Item) seleccionado;
        if (auxItem.id == id)
            return auxItem;
    }

    //2º miramos si es un Item perteneciente al escenario actual
    auxItem = buscaItemEsc(escenActual, id, false);
}
```



```
        if (auxItem != null)
            return auxItem;

        //3º miramos si es un Item perteneciente a la estancia actual.
        auxItem = buscaItemEst(((Mapa) mapas.search(mapaActual).data)
            .getEstanActual(), id);
        if (auxItem != null)
            return auxItem;

        //4º Miramos si es un Item perteneciente a el mapa actual.
        auxItem = buscaItemM((Mapa) mapas.search(mapaActual).data, id);
        if (auxItem != null)
            return auxItem;

        //Por último miramos si es de otro mapa.
        auxItem = buscaItemTM(id);
        if (auxItem != null)
            return auxItem;

        //En caso de no encontrarlo, devuelve null
        return null;
    }

    /**
     * Busca un Item en un escenario que se pasa por parámetro, cuyo
     * identificador coincida con el que pasamos por parámetro
     *
     * @param auxEscenario
     *         Escenario donde buscar el Item
     * @param id
     *         Identificador del Item a buscar.
     * @param comprueba
     *         Indica se comprueba el Item seleccionado o no
     * @return El item encontrado o null en caso de no encontrarlo.
     */
    private Item buscaItemEsc(Escenario auxEscenario, int id, boolean comprueba) {
        Item auxItem;
        AVLTree arbol;
        Node auxN;

        if (comprueba && !mostrarInventario && actual != conversacion
            && seleccionado != null) {
            auxItem = (Item) seleccionado;
            if (auxItem.id == id)
                return auxItem;
        }

        arbol = auxEscenario.items;
        auxN = arbol.search(id);
        if (auxN != null)
            return (Item) auxN.data;
        return null;
    }

    /**
     * Busca un Item en un estancia que se pasa por parámetro, cuyo
     * identificador coincida con el que pasamos por parámetro
     *
     * @param auxEstancia
     *         Estancia donde buscar el Item
     */
```



```
    * @param id
    * Identificador del Item a buscar.
    * @return El item encontrado o null en caso de no encontrarlo.
    */
private Item buscaItemEst(Estancia auxEstancia, int id) {
    Node auxN = auxEstancia.escenarios.root;
    Escenario auxEscenario;
    auxEstancia.escenarios.inorden(auxN);
    auxN = auxEstancia.escenarios.next();
    while (auxN != null) {
        auxEscenario = (Escenario) auxN.data;
        //Si el escenario no es el actual, que ya comprobamos antes...
        if (auxEscenario.id != escenActual.id) {
            Item auxItem = buscaItemEsc(auxEscenario, id, false);
            if (auxItem != null)
                return auxItem;
        }
        auxN = auxEstancia.escenarios.next();
    }
    return null;
}

/**
 * Busca un Item en un mapa que se pasa por parámetro, cuyo identificador
 * coincida con el que pasamos por parámetro
 */
* @param auxMapa
* Mapa donde buscar el Item
* @param id
* Identificador del Item a buscar.
* @return El item encontrado o null en caso de no encontrarlo.
*/
private Item buscaItemM(Mapa auxMapa, int id) {
    Node auxN = auxMapa.estancias.root;
    Estancia auxEstancia;
    auxMapa.estancias.inorden(auxN);
    auxN = auxMapa.estancias.next();
    while (auxN != null) {
        auxEstancia = (Estancia) auxN.data;
        // Si la estancia no es la actual, que ya comprobamos
antes...
        if (auxEstancia.id != ((Mapa) mapas.search(mapaActual).data)
            .getEstanActual().id) {
            Item auxItem = buscaItemEst(auxEstancia, id);
            if (auxItem != null)
                return auxItem;
        }
        auxN = auxMapa.estancias.next();
    }
    return null;
}

/**
 * Busca un Item en todos los mapa, cuyo identificador coincida con el que
 * pasamos por parámetro
 */
* @param id
* Identificador del Item a buscar.
* @return El item encontrado o null en caso de no encontrarlo.
*/
```





```
private Item buscaItemM(int id){
    Node auxN = mapas.root;
    Mapa auxMapa;
    mapas.inorden(auxN);
    auxN = mapas.next();
    while (auxN != null) {
        auxMapa = (Mapa) auxN.data;
        // Si el mapa no es el actual, que ya comprobamos
        antes...
        if (auxMapa.id != ((Mapa) mapas.search(mapaActual).data).id) {
            Item auxItem = buscaItemM(auxMapa, id);
            if (auxItem != null)
                return auxItem;
        }
        auxN = mapas.next();
    }
    return null;
}

/**
 * Hace avanzar un paso a Oscar, en su camino hacia el destino.
 */
private void avanzarPaso() {
    int x = getPixelX(buscaRuta.getX(buscaRuta.path[pasoActual]));
    int y = getPixelY(buscaRuta.getY(buscaRuta.path[pasoActual]));
    if (y < oscar.getY() && direccion != 1) {
        //Cargamos y colocamos el Sprite de Oscar de espaldas.
        Utils.cargarImagen(Utils.spOscarEspa);
        oscar.setImage(Utils.imagen, Utils.anchoIm, Utils.altoIm);
        oscar.setFrameSequence(Utils.getFrames(Utils.spOscarEspa));
        direccion = 1;
    } else if (y > oscar.getY() && direccion != 0) {
        //Cargamos y colocamos el Sprite de Oscar de frente.
        Utils.cargarImagen(Utils.spOscarFren);
        oscar.setImage(Utils.imagen, Utils.anchoIm, Utils.altoIm);
        oscar.setFrameSequence(Utils.getFrames(Utils.spOscarFren));
        direccion = 0;
    } else if (x > oscar.getX() && direccion != 3) {
        //Cargamos y colocamos el Sprite de Oscar hacia el lado derecho.
        Utils.cargarImagen(Utils.spOscarLado);
        oscar.setImage(Utils.imagen, Utils.anchoIm, Utils.altoIm);
        oscar.setFrameSequence(Utils.getFrames(Utils.spOscarLado));
        oscar.setTransform(Sprite.TRANS_NONE);
        direccion = 3;
    } else if (x < oscar.getX() && direccion != 2) {
        //Cargamos y colocamos el Sprite de Oscar.
        Utils.cargarImagen(Utils.spOscarLado);
        oscar.setImage(Utils.imagen, Utils.anchoIm, Utils.altoIm);
        oscar.setFrameSequence(Utils.getFrames(Utils.spOscarLado));
        oscar.setTransform(Sprite.TRANS_MIRROR);
        direccion = 2;
    }
    oscar.setPosition(x, y);
    //Cada 4 celdas de camino, avanzo un sprite
    if (pasoActual % 4 == 0)
        oscar.nextFrame();
    redibujar = true;
    if (pasoActual < buscaRuta.path_count - 1)
        pasoActual++;
}
```



```
        else {
            caminando = false;
            ocupado = false;//en el caso de reCaminarBlo
            direccion = 0;
            Utils.cargarImagen(Utils.spOscarFren);
            oscar.setImage(Utils.imagen, Utils.anchoIm, Utils.altoIm);
            oscar.setFrameSequence(Utils.getFrames(Utils.spOscarFren));
        }
    }

/**
 * Indica, atendiendo al estado actual de la partida, si se puede guardar o
 * no la partida.
 *
 * @return Si se puede guardar o no la partida.
 */
public boolean puedeGuardar() {
    if (parteJuego != video && parteJuego != conversacion
        && !mostrarMensaje)
        return true;
    return false;
}

/**
 * Guarda en la memoria del móvil, la partida en curso.
 *
 * @param g
 *      objeto Graphics necesario para llamar a drawScreen y que de
 *      esta manera se cargue la imagen de cargando
 * @throws IOException
 * @throws XmlPullParserException
 */
public void guardarPartida(Graphics g) throws XmlPullParserException,
    IOException {
    // cargamos el fondo de pantalla de guardando
    Utils.cargarImagen(Utils.imGuardando);
    sprite = null;
    //cargamos sprite con una imagen del tamaño de inventario
    Utils.cargarImagen(Utils.imGuardando);
    sprite = new Sprite(Utils.imagen, Utils.anchoIm, Utils.altoIm);
    sprite.setPosition(Utils.inicioPantallaX, Utils.inicioPantallaY);
    sprite.setFrameSequence(Utils.getFrames(Utils.imGuardando));
    sprite.paint(g);

    drawScreen(g);

    midlet.guardarDatos();
}

/**
 * Inserta un registro en el RMS que está actualmente abierto.
 *
 * @param baos
 *      Stream donde está la información a guardar
 * @throws RecordStoreNotOpenException
 * @throws InvalidRecordIDException
 * @throws RecordStoreFullException
 * @throws RecordStoreException
 */
private void insertarRegistro(ByteArrayOutputStream baos, RecordStore rs)
```



```
throws RecordStoreNotOpenException, InvalidRecordIDException,
RecordStoreFullException, RecordStoreException {
    byte[] registro = baos.toByteArray();
    rs.addRecord(registro, 0, registro.length);
    baos.reset();
}

/**
 * Establece a null todos los objetos del GameCanvas
 */
private void salir() {
    midlet = null;
    t = null;
    LM = null;
    parser = null;
    buscaRuta = null;
    mapas = null;
    escenActual = null;
    escenDetras = null;
    escenCaminable = null;
    sprite = null;
    oscar = null;
    puntero = null;
    itemsInventario = null;
    seleccionado = null;
    videosFase = null;
    converFase = null;
    participante = null;
    iniIdOp = null;
    resultados = null;
    ultimoEscenario = null;
}
}
```

## Entre2Almas

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Vector;

import javax.microedition.midlet.*;
import javax.microedition.rms.InvalidRecordIDException;
import javax.microedition.rms.RecordStore;
import javax.microedition.rms.RecordStoreException;
import javax.microedition.rms.RecordStoreFullException;
import javax.microedition.rms.RecordStoreNotFoundException;
import javax.microedition.rms.RecordStoreNotOpenException;
import javax.microedition.lcdui.*;

import org.kxml2.io.KXmlParser;
import org.xmlpull.v1.XmlPullParser;
```



```
import org.xmlpull.v1.XmlPullParserException;

/**
 *
 * Entre 2 Almas                                [http://gamelab.uniovi.es/entre2almas]
 * -----
 * PFC para la EUITIO                            [http://www.euitio.uniovi.es/]
 * Desarrollado en el GameLab    [http://gamelab.mieres.uniovi.es/]
 * Bajo licencia Freeware
 * -----
 *
 *
 *
 *
 * Clase principal del juego Entre2Almas, extiende de MIDlet y por ello esta
 * clase será la que primero se ejecute al iniciar el midlet
 *
 * @author Pedro Javier Sáez Martínez
 * @version 1.0
 */
public class Entre2Almas extends MIDlet {

    /**
     * Representa el manejador de la pantalla, y solo habrá una instancia de
     * Display por midlet
     */
    private Display display;

    /**
     * Indica si se debe mostrar el Splash Screen
     */
    private boolean isSplash = true;

    /**
     * Indica si tenemos un menú clásico (0) o avanzado (1);
     */
    public int tipoMenu;

    /**
     * La interfaz de usuario del menú, de tipo estándar con fondo de pantalla.
     * (Avanzado)
     */
    private UITerminalSTBG avanzado;

    /**
     * La interfaz de usuario del menú, de tipo estándar. (Clásico)
     */
    private UITerminalST clasico;

    /**
     * Será la base de datos del juego.
     */
    private RecordStore rs;

    /**
     * Será la base de datos temporal del juego. Se utilizará para a la hora de
     * guardar la partida por si acaso ocurriese que el móvil se queda sin
     * batería, o que se apaga de repente, o algo raro, el guardar partida se
     * hace primero en un rms temporal, y luego, cuando se ha completado, se
     * pasa al bueno.
     */
    private RecordStore rsTemp;
```



```
/**
 * Si es 0 indica que el menú activo es clásico y viceversa
 */
public int menuActivo;

/**
 * Donde se desarrollará el juego
 */
public Ejecucion gameCanvas;

/**
 * Parser utilizado para cargar los datos básicos del juego. Como son el
 * splash screen y las imágenes del menú.
 */
private KXmlParser parser;

/**
 * Guarda aquellos datos necesarios para restaurar el juego por donde estaba
 * en un futura partida.
 */
public Vector preguardado;

/**
 * Método llamado cuando se inicia la aplicación.(heredado de MIDlet)
 */
public void startApp() {
    try {
        preguardado = new Vector();
        //Recupera una instancia del Display, que será la única
        //que se usará durante toda la ejecución.
        display = Display.getDisplay(this);
        //Inicializamos el parser
        parser = new KXmlParser();
        parser.setInput(new InputStreamReader(this.getClass()
            .getResourceAsStream("/basicos.xml")));
        cargarDatos();

        //Se inicializa primero ui, porque luego es utilizado en el splash
        //además como es el primer canvas que se inicializa, pues será la
        //única clase que llamará al método Utils.grupo();
        avanzado = new UITerminalSTBG(this);
        clasico = new UITerminalST(this);

        tipoMenu = getMenuRs();
        menuActivo = tipoMenu;
        //Define a que grupo, en cuanto a tamaño de las imágenes,
        //corresponde este móvil
        cargarSplash();
    } catch (XmlPullParserException e) {
        mostrarAlerta("Entre2Almas::startApp::Problemas con el XML: " + e,
            AlertType.ERROR);
        e.printStackTrace();
    } catch (IOException e1) {
        mostrarAlerta("Entre2Almas::startApp::Error de entrada y salida: "
            + e1, AlertType.ERROR);
        e1.printStackTrace();
    }
}
```



```
/**devuelve la referencia al display
 *
 * @return Referencia al display de la aplicación.
 */
public Display getDisplay() {
    return display;
}

/**
 * Método llamado cuando se pausa el juego (heredado de MIDlet)
 */
public void pauseApp() {
}

/**
 * Método llamado cuando se destruye el juego (heredado de MIDlet)
 */
public void destroyApp(boolean unconditional) {
    System.gc();
    notifyDestroyed();
}

/**
 * Carga la pantalla de presentación
 */
private void cargarSplash() {
    if (isSplash) {
        isSplash = false;
        //Llamamos a SplashScreen
        if (tipoMenu == 0)
            new SplashScreen(this, clasico, 2000);
        else
            new SplashScreen(this, avanzado, 2000);
    }
}

/**
 * A partir de lo que el usuario elija en las opciones, cargará el menú
 * clásico o el avanzado
 */
public void menu() {
    //Entonces se supone que vamos a cargar un menú de distinto
    //formato y por tanto deberemos empezar por la primera opción
    Utils.menuPrinIdx = 0;
    if (tipoMenu == 0) {
        if (Utils.jugando)
            Utils.menuPrinIdx = 0;
        Utils.cargada = -1;
        Utils.menuAyudaIdx = 0;
        menuActivo = 0;
        Utils.parteMenu = Utils.menuPrin;
        display.setCurrent(clasico);
    } else {
        //Ajustamos las variables que han podido ser cambiadas
        //anteriormente (por ejemplo mientras jugamos)
        if (Utils.jugando) {
            avanzado.ajustarVariables();
            Utils.menuPrinIdx = 0;
        }
    }
}
```



```
meuActive = 1;
//Decimos que la próxima vez que entremos al menú, que nos
//recargue el menú (así ya saldrá lo de volver a partida)
//y además le decimos que la parte de menú en la que estamos es
//el menú principal
Utils.cargada = -1;
Utils.menuAyudaIdx = 0;
Utils.parteMenu = Utils.menuPrin;
display.setCurrent(avanzado);
    }

}

/**
 * Devuelve el tipo de menú que en ese momento esta seleccionado en las
 * opciones
 *
 * @return El tipo de menú que en ese momento esta seleccionado en las
 * opciones
 */
public String getTipoMenu() {
    if (tipoMenu == 0)
        return "Clásico";
    else
        return "Avanzado";
}

/**
 * Iniciamos la partida.
 */
public void IniciarPartida() {
    //Si ya estábamos jugando otra partida anteriormente
    if (Utils.jugando) {
        gameCanvas.salir = true;
        synchronized (gameCanvas) {
            gameCanvas.notify();
        }
    }
    Utils.jugando = true;
    //recogemos las referencias de las clases
    //el gameCanvas o la partida no se crea hasta que elegimos
    // las opciones, por eso no lo creamos en el
    //startApp como al resto.
    gameCanvas = null;
    System.gc();
    gameCanvas = new Ejecucion(this);
    gameCanvas.start();
    display.setCurrent(gameCanvas);
    //y cuando la hemos puesto en marcha, la mostramos por pantalla.
}

/**
 * Reanuda la partida, que previamente se había parado para salir al menú
 * principal
 */
public void seguirJugando() {
    //Reajustamos las variables, que han sido modificadas,
    //del GameCanvas
    gameCanvas.adjustarVariables();
    //Despertamos la Ejecución del GameCanvas
}
```



```
synchronized(gameCanvas){
    gameCanvas.notify();
}
//y cuando la hemos puesto en marcha, la mostramos por pantalla.
display.setCurrent(gameCanvas);
System.gc();
}

/**
 * salimos de la aplicación.
 */
public void salir() {
    destroyApp(true);
}

/**
 * A partir de un chorro de bytes que había en un registro, nos devuelve el
 * ultimo menú que utilizamos en la anterior partida.
 *
 * @return el tipo de menú
 */
private int getMenuRs() {
    try {
        rs = RecordStore.openRecordStore("menu", true);
        //          getNumRecords devuelve el numero de registros
        //almacenados en un determinado recordStore.
        //Si no hay ninguno significa que no hay nada guardado en la BBDD
        if (rs.getNumRecords() == 0) {
            byte[] registro;
            //usamos Streams para leer registros en el rs, así
            //almacenamos información de distinto tipo en el
            //mismo registro, y además al leer podemos distinguir
            //entre la info de un tipo u otro.
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            DataOutputStream dos = new DataOutputStream(baos);
            dos.writeInt(1);//menú avanzado
            dos.flush();
            registro = baos.toByteArray();
            //añade un registro en el recordStore, en este caso añade
            //lo que hay en el array registro desde el byte 0 hasta
            //su ultimo byte.
            rs.addRecord(registro, 0, registro.length);
            baos.close();
            dos.close();
            registro = null;
        }
        //          En el recordStore los parametros no van de 0 a N
        //sino de 1 a N
        byte[] registro = rs.getRecord(1);
        //Explicado el por que del uso de streams en
        //initRecord()
        ByteArrayInputStream bais = new ByteArrayInputStream(registro);
        DataInputStream dis = new DataInputStream(bais);
        int tipoMenu = dis.readInt();
        bais.close();
        dis.close();
        rs.closeRecordStore();
        return tipoMenu;
    } catch (Exception e) {
        try {
```





```
        rs.closeRecordStore();
    } catch (RecordStoreNotFoundException e1) {
        mostrarAlerta("Entre2Almas::getMenuRs::Error RMS: "
            + e, AlertType.ERROR);
        e1.printStackTrace();
    } catch (RecordStoreException e1) {
        mostrarAlerta("Entre2Almas::getMenuRs::Error RMS: "
            + e, AlertType.ERROR);
        e1.printStackTrace();
    }
    mostrarAlerta("Entre2Almas::getMenuRs::Error desconocido: " + e,
        AlertType.ERROR);
    return -1;
}

/**
 * Actualiza la base de datos de menú
 */
public void setMenuRs() {
    try {
        //Utilizamos streams debido a la utilidad que tienen
        //explicada en el método verRecords()
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        DataOutputStream dos = new DataOutputStream(baos);
        dos.writeInt(tipoMenu);
        dos.flush();
        //recoge todo lo almacenado a través de dos en el array
        //de bytes nreg.
        byte[] registro = baos.toByteArray();
        baos.close();
        dos.close();

        rs = RecordStore.openRecordStore("menu", true);
        rs.setRecord(1, registro, 0, registro.length);

        rs.closeRecordStore();
    } catch (Exception e) {
        System.out.println("RMS:setMenuRs::" + e);
        try {
            rs.closeRecordStore();
        } catch (RecordStoreNotFoundException e1) {
            mostrarAlerta("Entre2Almas::setMenuRs::Error RMS: " + e,
                AlertType.ERROR);
            e1.printStackTrace();
        } catch (RecordStoreException e1) {
            mostrarAlerta("Entre2Almas::setMenuRs::Error RMS: " + e,
                AlertType.ERROR);
            e1.printStackTrace();
        }
        mostrarAlerta("Entre2Almas::setMenuRs::Error desconocido: " + e,
            AlertType.ERROR);
    }
}

/**
 * Carga los datos necesarios, del archivo XML, para mostrar el menú y el
 * splash screen
 *
 * @throws XmlPullParserException
 */
```



```

    * @throws IOException parser
    *      Excepción de entrada/salida
    */
private void cargarDatos() throws XmlPullParserException, IOException {
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "Basicos");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "Imagenes");
    parser.nextTag();

    //Numero total de imagenes que tendrá el array de imagenes
    parser.require(XmlPullParser.START_TAG, null, "numImagenes");
    int numImagenes = Integer.parseInt(parser.nextText());
    Utils.imagenes = new String[numImagenes];

    parser.require(XmlPullParser.END_TAG, null, "numImagenes");
    StringBuffer temp = new StringBuffer();
    StringBuffer imagen = new StringBuffer();
    int i = 0;

    //Cargamos el array de imágenes
    while (parser.nextTag() != XmlPullParser.END_TAG) {
        if (parser.getName().equals("Imagen")) {
            parser.require(XmlPullParser.START_TAG, null, "Imagen");
            temp.append(parser.getAttributeValue(0) + ";");
            parser.nextTag();
            parser.require(XmlPullParser.START_TAG, null, "id");
            imagen.append(parser.nextText() + ";" + temp.toString());
            parser.require(XmlPullParser.END_TAG, null, "id");
            parser.nextTag();
            parser.require(XmlPullParser.START_TAG, null, "tamX");
            imagen.append(parser.nextText() + ";");
            parser.require(XmlPullParser.END_TAG, null, "tamX");
            parser.nextTag();
            parser.require(XmlPullParser.START_TAG, null, "tamY");
            imagen.append(parser.nextText());
            parser.require(XmlPullParser.END_TAG, null, "tamY");
            parser.nextTag();
            parser.require(XmlPullParser.END_TAG, null, "Imagen");
            Utils.imagenes[i++] = imagen.toString();
            temp.delete(0, temp.length());
            imagen.delete(0, imagen.length());
        }
    }
    System.gc();
    temp = null;
    imagen = null;
    parser.require(XmlPullParser.END_TAG, null, "Imagenes");
    parser.nextTag();

    //Cargamos el sprite del splash screen
    parser.require(XmlPullParser.START_TAG, null, "splash");
    Utils.splash = getDatosImagen().toString();
    parser.nextTag();
    parser.require(XmlPullParser.END_TAG, null, "splash");

    //Cargamos la imagenes del menú.
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "menu");

```



```
parser.nextTag();
//Primero la del menú principal
parser.require(XmlPullParser.START_TAG, null, "menuPrin");
Utils.titulo = parser.getAttributeValue(0);
Utils.imMenuPrin = getDatosImagen().toString();
//cargamos las opciones del menú principal
cargaOpMenuPrin();
parser.require(XmlPullParser.END_TAG, null, "menuPrin");
parser.nextTag();

//Luego la imagen de opciones
parser.require(XmlPullParser.START_TAG, null, "menuOpciones");
Utils.imOpciones = getDatosImagen().toString();
parser.nextTag();
//y su opción tipo de menú.
parser.require(XmlPullParser.START_TAG, null, "tipoMenu");
Utils.tiposMenuTxt = parser.getAttributeValue(0);
parser.nextTag();
parser.require(XmlPullParser.START_TAG, null, "clasico");
Utils.tiposMenu[0] = parser.nextText();
parser.require(XmlPullParser.END_TAG, null, "clasico");
parser.nextTag();
parser.require(XmlPullParser.START_TAG, null, "avanzado");
Utils.tiposMenu[1] = parser.nextText();
parser.require(XmlPullParser.END_TAG, null, "avanzado");
parser.nextTag();
parser.require(XmlPullParser.END_TAG, null, "tipoMenu");
parser.nextTag();
parser.require(XmlPullParser.END_TAG, null, "menuOpciones");
parser.nextTag();

//Luego la imagen de memoria
parser.require(XmlPullParser.START_TAG, null, "memoria");
Utils.imMemoria = getDatosImagen().toString();
// cargamos las opciones del menú principal
cargaOpMemo();
parser.require(XmlPullParser.END_TAG, null, "memoria");
parser.nextTag();

//Imagen de ayuda
parser.require(XmlPullParser.START_TAG, null, "ayuda");
Utils.imAyuda = getDatosImagen().toString();
// cargamos las opciones y el contenido de sus sub-ayudas
cargaOpAyuda();
parser.require(XmlPullParser.END_TAG, null, "ayuda");
parser.nextTag();

//Imagen de acerca de...
parser.require(XmlPullParser.START_TAG, null, "acerca");
Utils.imAcerca = getDatosImagen().toString();
parser.nextTag();
parser.require(XmlPullParser.START_TAG, null, "contenido");
Utils.conteAcerca = parser.nextText();
parser.require(XmlPullParser.END_TAG, null, "contenido");
parser.nextTag();
parser.require(XmlPullParser.END_TAG, null, "acerca");
parser.nextTag();
parser.require(XmlPullParser.END_TAG, null, "menu");
parser.nextTag();
```



```
//Sprite de cargando
parser.require(XmlPullParser.START_TAG, null, "cargando");
Utils.spCargando = getDatosImagen().toString();
parser.nextTag();
parser.require(XmlPullParser.END_TAG, null, "cargando");
parser.nextTag();

//          Sprite de guardando
parser.require(XmlPullParser.START_TAG, null, "guardando");
Utils.imGuardando = getDatosImagen().toString();
parser.nextTag();
parser.require(XmlPullParser.END_TAG, null, "guardando");
parser.nextTag();

//Sprite de puntero
parser.require(XmlPullParser.START_TAG, null, "puntero");
Utils.spPuntero = getDatosImagen().toString();
parser.nextTag();
parser.require(XmlPullParser.END_TAG, null, "puntero");
parser.nextTag();

//Imagen del puntero que comprueba el area caminable
parser.require(XmlPullParser.START_TAG, null, "punCami");
Utils.imPunCami = getDatosImagen().toString();
parser.nextTag();
parser.require(XmlPullParser.END_TAG, null, "punCami");

//Imagen temporal para cargar el inventario
parser.nextTag();
parser.require(XmlPullParser.START_TAG, null, "imTmpInv");
Utils.imTmpInv = getDatosImagen().toString();
parser.nextTag();
parser.require(XmlPullParser.END_TAG, null, "imTmpInv");

//Cargamos los datos del protagonista.
parser.nextTag();
parser.require(XmlPullParser.START_TAG, null, "apariencias");
parser.nextTag();
parser.require(XmlPullParser.START_TAG, null, "numApariencias");
//          Cargamos el array de apariencias
Utils.apariencias = new String[Integer.parseInt(parser.nextText()) * 4];
parser.require(XmlPullParser.END_TAG, null, "numApariencias");
while (parser.nextTag() != XmlPullParser.END_TAG) {
    cargarApariencia();
}
parser.require(XmlPullParser.END_TAG, null, "apariencias");
//una vez cargadas, activamos una cualquiera
Utils.actualizarApariencia(0);

parser.nextTag();
parser.require(XmlPullParser.END_TAG, null, "Basicos");
parser = null;
}

/**
 * Recupera, a partir de un archivo XML, los datos descriptivos de una
 * imagen
 *
 * @throws IOException
```



```

    * @throws XmlPullParserException
    * @throws IOException
    */
public StringBuffer getDatosImagen() throws XmlPullParserException,
    IOException {
    StringBuffer imagen = new StringBuffer();
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "id");
    imagen.append(parser.nextText() + ";");
    parser.require(XmlPullParser.END_TAG, null, "id");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "frameIni");
    imagen.append(parser.nextText() + ";");
    parser.require(XmlPullParser.END_TAG, null, "frameIni");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "numFrames");
    imagen.append(parser.nextText());
    parser.require(XmlPullParser.END_TAG, null, "numFrames");
    System.gc();
    return imagen;
}

/**
 * Carga del archivo basicos.xml, las nombre de las opciones del Menú
 * Principal.
 *
 * @throws XmlPullParserException
 * @throws IOException
 */
private void cargaOpMenuPrin() throws XmlPullParserException, IOException {
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "opciones");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "opSegJug");
    Utils.opMenuPrin[0] = parser.getAttributeValue(0);
    parser.nextTag();
    parser.require(XmlPullParser.END_TAG, null, "opSegJug");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "opNuevaPartida");
    Utils.opMenuPrin[1] = parser.getAttributeValue(0);
    parser.nextTag();
    parser.require(XmlPullParser.END_TAG, null, "opNuevaPartida");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "opMemoria");
    Utils.opMenuPrin[2] = parser.getAttributeValue(0);
    parser.nextTag();
    parser.require(XmlPullParser.END_TAG, null, "opMemoria");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "opOpciones");
    Utils.opMenuPrin[3] = parser.getAttributeValue(0);
    parser.nextTag();
    parser.require(XmlPullParser.END_TAG, null, "opOpciones");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "opAyuda");
    Utils.opMenuPrin[4] = parser.getAttributeValue(0);
    parser.nextTag();
    parser.require(XmlPullParser.END_TAG, null, "opAyuda");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "opAcerca");

```



```
Utils.opMenuPrin[5] = parser.getAttributeValue(0);
parser.nextTag();
parser.require(XmlPullParser.END_TAG, null, "opAcerca");
parser.nextTag();
parser.require(XmlPullParser.END_TAG, null, "opciones");
parser.nextTag();
}

/**
 * Carga del archivo basicos.xml, las nombre de las opciones del Menú
 * Memoria.
 *
 * @throws XmlPullParserException
 * @throws IOException
 */
private void cargaOpMemo() throws XmlPullParserException, IOException {
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "opciones");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "guardar");
    Utils.opMenuMemo[0] = parser.getAttributeValue(0);
    parser.nextTag();
    parser.require(XmlPullParser.END_TAG, null, "guardar");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "cargar");
    Utils.opMenuMemo[1] = parser.getAttributeValue(0);
    parser.nextTag();
    parser.require(XmlPullParser.END_TAG, null, "cargar");
    parser.nextTag();
    parser.require(XmlPullParser.END_TAG, null, "opciones");
    parser.nextTag();
}

/**
 * Carga del archivo basicos.xml, las nombre de las opciones del Menú Ayuda.
 * Además también cargará el contenido de cada una de esta sub-ayudas.
 *
 * @throws XmlPullParserException
 * @throws IOException
 */
private void cargaOpAyuda() throws XmlPullParserException, IOException {
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "opciones");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "tematica");
    Utils.opMenuAyuda[0] = parser.getAttributeValue(0);
    Utils.txtSubAyuda[0] = parser.nextText();
    parser.require(XmlPullParser.END_TAG, null, "tematica");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "teclasGenerales");
    Utils.opMenuAyuda[1] = parser.getAttributeValue(0);
    Utils.txtSubAyuda[1] = parser.nextText();
    parser.require(XmlPullParser.END_TAG, null, "teclasGenerales");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "enVideo");
    Utils.opMenuAyuda[2] = parser.getAttributeValue(0);
    Utils.txtSubAyuda[2] = parser.nextText();
    parser.require(XmlPullParser.END_TAG, null, "enVideo");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "enInventario");
}
```



```
Utils.opMenuAyuda[3] = parser.getAttributeValue(0);
Utils.txtSubAyuda[3] = parser.nextText();
parser.require(XmlPullParser.END_TAG, null, "enInventario");
parser.nextTag();
parser.require(XmlPullParser.START_TAG, null, "enMapa");
Utils.opMenuAyuda[4] = parser.getAttributeValue(0);
Utils.txtSubAyuda[4] = parser.nextText();
parser.require(XmlPullParser.END_TAG, null, "enMapa");
parser.nextTag();
parser.require(XmlPullParser.START_TAG, null, "conversando");
Utils.opMenuAyuda[5] = parser.getAttributeValue(0);
Utils.txtSubAyuda[5] = parser.nextText();
parser.require(XmlPullParser.END_TAG, null, "conversando");
parser.nextTag();
parser.require(XmlPullParser.END_TAG, null, "opciones");
parser.nextTag();
}

/**
 * Carga una apariencia de nuestro personaje principal
 *
 * @throws XmlPullParserException
 * @throws IOException
 */
public void cargarApariencia() throws XmlPullParserException, IOException {
    parser.require(XmlPullParser.START_TAG, null, "apariencia");
    int id = Integer.parseInt(parser.getAttributeValue(0));
    id = id * 4;
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "lado");
    Utils.apariencias[id] = getDatosImagen().toString();
    id++;
    parser.nextTag();
    parser.require(XmlPullParser.END_TAG, null, "lado");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "espalda");
    Utils.apariencias[id] = getDatosImagen().toString();
    id++;
    parser.nextTag();
    parser.require(XmlPullParser.END_TAG, null, "espalda");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "frente");
    Utils.apariencias[id] = getDatosImagen().toString();
    id++;
    parser.nextTag();
    parser.require(XmlPullParser.END_TAG, null, "frente");
    parser.nextTag();
    parser.require(XmlPullParser.START_TAG, null, "cara");
    Utils.apariencias[id] = getDatosImagen().toString();
    parser.nextTag();
    parser.require(XmlPullParser.END_TAG, null, "cara");
    parser.nextTag();
    parser.require(XmlPullParser.END_TAG, null, "apariencia");
}

/**
 * Guardará en preguardado los resultados necesarios para restablecer el
 * juego al momento de la última partida guardada.
 *
 * @param res
```



```
    */      Resultado a almacenar.
public void preguardar(Resultado res) {
    //no se pueden meter entero en un objeto Vector
    preguardado.addElement(res);
}

/**
 * Guarda, los datos necesarios de la partida actual, en la base de datos.
 *
 */
public void guardarDatos() {
    try {
        //Por si acaso ocurriese que el móvil se queda sin batería, o que
        // se
        //apaga de repente, o algo raro, el guardar partida se hace primero
        // en
        //un rms temporal, y luego, cuando se ha completado, se pasa al
        // bueno.
        rsTemp = RecordStore.openRecordStore("temporal", true);

        //                                usamos Streams para leer registros en el rs, así
        //almacenamos información de distinto tipo en el
        //mismo registro, y además al leer podemos distinguir
        //entre la info de un tipo u otro.
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        DataOutputStream dos = new DataOutputStream(baos);

        //                                Fase actual
        dos.writeInt(gameCanvas.nFase);
        //                                Mapa actual
        dos.writeInt(gameCanvas.mapaActual);
        //                                Estancia actual
        dos.writeInt(((Mapa) (gameCanvas.mapas
                                .search(gameCanvas.mapaActual).data)).activa);
        //Escenario actual
        dos
                                .writeInt(((Mapa) (gameCanvas.mapas
                                .search(gameCanvas.mapaActual).data))
                                .getEstanActual().actual);

        //                                Parte de juego
        dos.writeInt(gameCanvas.parteJuego);
        insertarRegistro(baos);
        Resultado res;

        //guardamos los resultados preguardados.
        for (int i = 0; i < preguardado.size(); i++) {
            res = (Resultado) preguardado.elementAt(i);
            dos.writeInt(res.id);
            dos.writeUTF(res.info);
            insertarRegistro(baos);
        }
        dos.flush();
        baos.close();
        dos.close();

        //borra el contenido actual de la base de datos buena
        rs = RecordStore.openRecordStore("datosGuardados", true);
        //si la base de datos contiene alguna otra partida guardada,
        // entonces borramos
    }
}
```





```
//dicha partida guardada. Más bien sus registros
if (rs.getNumRecords() != 0) {
    rs.closeRecordStore();
    RecordStore.deleteRecordStore("datosGuardados");
    rs = RecordStore.openRecordStore("datosGuardados", true);
}

//sobre escribe, la base de datos temporal, sobre la buen
byte[] registro;
for (int i = 1; i <= rsTemp.getNumRecords(); i++) {
    registro = rsTemp.getRecord(i);
    rs.addRecord(registro, 0, registro.length);
}

rsTemp.closeRecordStore();
rs.closeRecordStore();
RecordStore.deleteRecordStore("temporal");

Utils.guardandoPartida = false;
} catch (Exception e) {
    mostrarAlerta("Entre2Almas::guardarDatos::Error desconocido: " + e,
        AlertType.ERROR);
    e.printStackTrace();
}
}

/**
 * Inserta un registro en el RMS que está actualmente abierto.
 *
 * @param baos
 *      Stream donde está la información a guardar
 * @throws RecordStoreNotOpenException
 * @throws InvalidRecordIDException
 * @throws RecordStoreFullException
 * @throws RecordStoreException
 */
private void insertarRegistro(ByteOutputStream baos)
    throws RecordStoreNotOpenException, InvalidRecordIDException,
        RecordStoreFullException, RecordStoreException {
    byte[] registro = baos.toByteArray();
    rsTemp.addRecord(registro, 0, registro.length);
    baos.reset();
}

/**
 * A la hora de cargar una partida, la cual fue guardada anteriormente, este
 * método nos devuelve los datos esenciales de la carga. (nFase,mapaActual,
 * escenario y estancia actual y parteJuego.
 *
 * @return Un array de bytes que contiene todos los datos esenciales
 */
public byte[] cargaDatosEsenciales() {
    byte[] registro = null;

    //Borramos todos los datos que estén preguardados.
    preguardado.removeAllElements();

    try {
        rs = RecordStore.openRecordStore("datosGuardados", true);
        // En el recordStore los parámetros no van de 0 a N
    }
}
```



```
//sino de 1 a N
registro = rs.getRecord(1);
rs.closeRecordStore();
return registro;
} catch (Exception e) {
    System.out.println(e);
    try {
        rs.closeRecordStore();
    } catch (RecordStoreNotFoundException e1) {
        mostrarAlerta("Entre2Almas::cargarDatosEsenciales::Error RMS: "
            + e, AlertType.ERROR);
        e1.printStackTrace();
    } catch (RecordStoreException e1) {
        mostrarAlerta("Entre2Almas::cargarDatosEsenciales::Error RMS: "
            + e, AlertType.ERROR);
        e1.printStackTrace();
    }
}
mostrarAlerta(
    "Entre2Almas::cargarDatosEsenciales::Error desconocido: "
        + e, AlertType.ERROR);
return registro;
}
}

/**
 * A la hora de cargar una partida, la cual fue guardada anteriormente, este
 * método carga los resultados que hay que ejecutar para que la partida se
 * quede en el mismo estado que cuando se guardo.
 */
public void cargarResultados() {
    try {
        rs = RecordStore.openRecordStore("datosGuardados", true);
        byte[] registro;
        ByteArrayInputStream bais = null;
        DataInputStream dis = null;
        Resultado res;

        //En el recordStore los parámetros no van de 0 a N
        //sino de 1 a N
        for (int i = 2; i <= rs.getNumRecords(); i++) {
            registro = rs.getRecord(i);
            bais = new ByteArrayInputStream(registro);
            dis = new DataInputStream(bais);
            res = new Resultado(dis.readInt(), dis.readUTF());
            gameCanvas.resultados.addElement(res);
            bais.reset();
            dis.reset();
        }

        bais.close();
        dis.close();
        rs.closeRecordStore();
    } catch (Exception e) {
        try {
            rs.closeRecordStore();
        } catch (RecordStoreNotFoundException e1) {
            mostrarAlerta("Entre2Almas::cargarResultados::Error RMS: " + e,
                AlertType.ERROR);
            e1.printStackTrace();
        } catch (RecordStoreException e1) {

```



```
        mostrarAlerta("Entre2Almas::cargarResultados::Error RMS: " + e,
        AlertType.ERROR);
        e1.printStackTrace();
    }
    mostrarAlerta("Entre2Almas::cargarResultados::Error desconocido: "
    + e, AlertType.ERROR);
}

/**
 * Indica, atendiendo al estado actual de la partida, si se puede guardar o
 * no la partida.
 *
 * @return Si se puede guardar o no la partida.
 */
public boolean puedeGuardar() {
    return gameCanvas.puedeGuardar();
}

/**
 * Indica si hay alguna partida guardada.
 *
 * @return Indica si hay alguna partida guardada.
 */
public boolean hayGuardada() {
    boolean hay = false;
    try {
        rs = RecordStore.openRecordStore("datosGuardados", true);
        if (rs.getNumRecords() != 0)
            hay = true;
        rs.closeRecordStore();

        return hay;
    } catch (RecordStoreFullException e) {
        mostrarAlerta("Entre2Almas::hayGuardada::Error RMS: " + e,
        AlertType.ERROR);
        e.printStackTrace();
    } catch (RecordStoreNotFoundException e) {
        mostrarAlerta("Entre2Almas::hayGuardada::Error RMS: " + e,
        AlertType.ERROR);
        e.printStackTrace();
    } catch (RecordStoreException e) {
        mostrarAlerta("Entre2Almas::hayGuardada::Error RMS: " + e,
        AlertType.ERROR);
        e.printStackTrace();
    }
    return hay;
}

/**
 * Muestra una alerta por pantalla
 *
 * @param texto
 *     El texto que contendrá la alerta.
 * @param tipo
 *     El tipo de alerta a mostrar
 */
public void mostrarAlerta(String texto, AlertType tipo) {
    Alert alr = new Alert("Info", texto, null, tipo);
    alr.setTimeout(Alert.FOREVER);
}
```



```
} display.setCurrent(alr);  
}
```

## Escenario

```
import javax.microedition.lcdui.game.Sprite;  
  
/**  
 * Entre 2 Almas [http://gamelab.uniovi.es/entre2almas]  
 * -----  
 * PFC para la EUITIO [http://www.euitio.uniovi.es/]  
 * Desarrollado en el GameLab [http://gamelab.mieres.uniovi.es/]  
 * Bajo licencia Freeware  
 * -----  
 * 2005-2006  
 *  
 *  
 * Un objeto Estancia, puede estar formado de objetos Escenario. Implementa la  
 * interfaz Comparable  
 *  
 * @author Pedro Javier Sáez Martínez  
 * @version 1.0  
 */  
public class Escenario implements Comparable {  
  
    /**  
     * Identificador del escenario  
     */  
    int id;  
  
    /**  
     * Nombre del escenario.  
     */  
    String nombre;  
  
    /**  
     * Posición inicial X, en la que empieza el personaje principal en el  
     * escenario  
     */  
    int posInicialX;  
  
    /**  
     * Posición inicial Y, en la que empieza el personaje principal en el  
     * escenario  
     */  
    int posInicialY;  
  
    /**  
     * Imagen que representará al Escenario. La imagen de cada escenario estará  
     * formado por 3 frames, el 1º será la representación del escenario que se  
     * verá por pantalla (delante del personaje), el 2º es el escenario que se  
     * ve por detrás del personaje y el 3º es el área caminable del escenario  
     */  
    public Sprite representa;
```



```
/** Indices de, entre las imágenes que forman el sprite representa, las
 * imágenes que corresponden a la parte de atrás, a la de delante y a la
 * caminable.
 */
int imDelante;

int imDetras;

int imCaminable;

/**
 * Indica si un escenario, tiene alguna parte del escenario, la cual está
 * por delante del personaje En caso de que esté a true, el escenario tendrá
 * 3 frames: - 1º para lo que se ve delante - 2º para lo que se ve detrás -
 * 3º el área caminable En caso contrario solo tendrá 2: - 1º la parte del
 * escenario que se ve detrás. - 2º el área caminable.
 */
boolean delante;

/**
 * Items que va a haber en el escenario
 */
AVLTree items;

/**
 * Constructor de Escenario
 *
 * @param id
 *         Identificador del Escenario
 * @param nombre
 *         Nombre del escenario
 * @param imagen
 *         identificador de Utils que define la imagen que representa
 *         este escenario
 * @param posInicialX
 *         Posición inicial X, en la que empieza el personaje principal
 *         en el escenario
 * @param posInicialY
 *         Posición inicial Y, en la que empieza el personaje principal
 *         en el escenario
 * @param items
 *         Items que va a haber en el escenario
 * @param delante
 *         Indica si un escenario, tiene alguna parte del escenario, la
 *         cual está por delante del personaje
 */
public Escenario(int id, String nombre, String imagen, int posInicialX,
                 int posInicialY, AVLTree items, boolean delante) {
    this.id = id;
    this.nombre = nombre;
    // cargamos la representación del escenario
    Utils.cargarImagen(imagen);
    representa = new Sprite(Utils.imagen, Utils.ancholm, Utils.altoIm);
    representa.setFrameSequence(Utils.getFrames(imagen));
    representa.setPosition(Utils.inicioPantallaX, Utils.inicioPantallaY);

    this.posInicialX = posInicialX;
    this.posInicialY = posInicialY;
```



```
this.items = items;
this.delante = delante;
if (delante) {
    imDelante = 0;
    imDetras = 1;
    imCaminable = 2;
} else {
    imDelante = -1;
    imDetras = 0;
    imCaminable = 1;
}
}

/**
 * @return Devuelve el posInicialX.
 */
public int getPosInicialX() {
    if (Utils.grupo == 1)//si es de 176x200
        return (posInicialX * Utils.aumentaX) / 1000;
    return posInicialX;
}

/**
 * @return Devuelve el posInicialY.
 */
public int getPosInicialY() {
    if (Utils.grupo == 1)//si es de 176x200
        return (posInicialY * Utils.aumentaY) / 1000;
    return posInicialY;
}

/**
 * Devuelve la clave del comparable
 *
 * @return La clave del comparable identificador del comparable.
 */
public int getKey() {
    return id;
}
}
```

## Estancia

```
import java.util.Vector;
import javax.microedition.lcdui.game.Sprite;

/**
 * Entre 2 Almas [http://gamelab.uniovi.es/entre2almas]
 * -----
 * PFC para la EUITIO [http://www.euitio.uniovi.es/]
 * Desarrollado en el GameLab [http://gamelab.mieres.uniovi.es/]
 * Bajo licencia Freeware
 * -----
 *
 * 2005-2006
 *
 */
```



```
‡ Un objeto Mapa, tendrá objetos Estancia. Implementa la interfaz Comparable y
* Opcionable.
*
* @author Pedro Javier Sáez Martínez
* @version 1.0
*/
public class Estancia implements Opcionable, Comparable {

    /**
     * Identificador de la Estancia
     */
    int id;

    /**
     * Escenarios disponibles para una determinada estancia
     */
    public AVLTree escenarios;

    /**
     * Nombre de la estancia
     */
    String nombre;

    /**
     * Indica el id del escenario activo actual.
     */
    public int actual;

    /**
     * Imagen que representará a la estancia en el mapa
     */
    public Sprite representa;

    /**
     * Posición X en la que la imagen que representa esta estancia, se colocará
     * en el mapa
     */
    public int posX;

    /**
     * Posición Y en la que la imagen que representa esta estancia, se colocará
     * en el mapa
     */
    public int posY;

    /**
     * Array que indica que opciones está o no disponibles
     */
    public Opcion[] acciones;

    /**
     * @param id
     *      Identificador de la estancia.
     * @param imagen
     *      identificador de Utils que define la imagen que representa
     *      esta estancia
     * @param nombre
     *      Nombre de la estancia
     * @param actual
```



```
    * Escenario activo actualmente
    * @param posX
    * posición X del sprite de esta estancia en el mapa
    * @param posY
    * posición Y del sprite de esta estancia en el mapa
    * @param acciones
    * Array que indica que opciones tendrá o no disponibles el Item
    * @param escenarios
    * Escenarios que forman parte de la Estancia.
    * @param visible
    * Indica si la estancia será visible en el mapa
    */
public Estancia(int id, String imagen, String nombre, int actual, int posX,
                int posY, Vector acciones, AVLTree escenarios, boolean visible) {
    this.id = id;
    this.escenarios = escenarios;
    this.nombre = nombre;
    this.actual = actual;
    if (Utils.grupo == 1) { //si el grupo es 176x200
        posX = (posX * Utils.aumentaX) / 1000;
        posY = (posY * Utils.aumentaY) / 1000;
    }
    this.posX = posX;
    this.posY = posY;
    // Inicializamos a null las opciones disponibles.
    this.acciones = new Opcion[acciones.size()];
    acciones.copyInto(this.acciones);
    // cargamos la representación de la estancia
    Utils.cargarImagen(imagen);
    representa = new Sprite(Utils.imagen, Utils.anchoIm, Utils.altoIm);
    representa.setPosition(Utils.inicioPantallaX + posX,
                          Utils.inicioPantallaY + posY);
    representa.setFrameSequence(Utils.getFrames(imagen));
    this.representa.setVisible(visible);
}

/**
 * Devuelve el escenario que actualmente esta activo en esta estancia
 *
 * @return el escenario activo para una estancia
 */
public Escenario getEscenActual() {
    return (Escenario) escenarios.search(actual).data;
}

/**
 * Indica si una determinada acción tiene o no alguna consecuencia
 *
 * @param opcionSele
 * La opción seleccionada en el juego
 * @return boolean indicando si esa opción o acción tiene alguna
 * consecuencia si es true es que no hay consecuencias, y viceversa.
 */
public boolean getConseActivaAcc(int opcionSele) {
    return acciones[opcionSele].consecuencias.isEmpty();
}

/**
 * Establece la consecuencia activa para una determinada acción.
 */
```





```
    * @param id
    *     Identificador de la opción a modificar.
    * @param valor
    *     el nuevo valor de conseActiva
    */
    public void setConseActivaAcc(String id, int valor) {
        Opcion auxOp = getOpcion(id);
        auxOp.conseActiva = valor;
    }

    /**
     * Devuelve la consecuencia para una determinada acción
     *
     * @param opcionSele
     *     La opción seleccionada en el juego
     * @return Vector con la consecuencia activa para una determinada acción
     */
    public Vector getConsecuenciaAcc(int opcionSele) {

        return (Vector) acciones[opcionSele].consecuencias
            .elementAt(acciones[opcionSele].conseActiva);
    }

    /**
     * El nombre de una determinada acción.
     *
     * @param opcionSele
     *     La opción seleccionada en el juego
     * @return nombre de la acción.
     */
    public String getNombreAccion(int opcionSele) {
        return acciones[opcionSele].texto;
    }

    /**
     * Devuelve el numero de acciones disponibles para un Seleccionable
     *
     * @return el numero de acciones disponibles para un Seleccionable
     */
    public int longAcciones() {
        return acciones.length;
    }

    /**
     * Establece si una acción esta activa o no
     *
     * @param opcionSele
     *     La opción seleccionada en el juego
     * @param valor
     *     boolean indicando si está activa o no.
     */
    public void setAccionActiva(int opcionSele, boolean valor) {
        acciones[opcionSele].activa = valor;
    }

    /**
     * Indica si una determinada acción esta activa o no.
     *
     * @param opcionSele
```



```

    * La opción seleccionada en el juego
    * @return boolean indicando si la opción está o no activa.
    */
    public boolean getAccionActiva(int opcionSele) {
        return acciones[opcionSele].activa;
    }

    /**
     * Devuelve la clave del comparable
     *
     * @return La clave del comparable identificador del comparable.
     */
    public int getKey() {
        return id;
    }

    /**
     * Devuelve la opción cuyo id coincide con el que se pasa como parámetro
     *
     * @param id
     *         String con el identificador de la opción
     * @return La opción cuyo id coincide con el de la opción.
     */
    public Opcion getOpcion(String id) {
        for (int i = 0; i < longAcciones(); i++) {
            if (acciones[i] != null) {
                if (acciones[i].id.equals(id)) {
                    return acciones[i];
                }
            }
        }
        return null;
    }
}

```

## Item

```

import java.util.Vector;

import javax.microedition.lcdui.game.Sprite;

/**
 * Entre 2 Almas [http://gamelab.uniovi.es/entre2almas]
 * -----
 * PFC para la EUITIO [http://www.euitio.uniovi.es/]
 * Desarrollado en el GameLab [http://gamelab.mieres.uniovi.es/]
 * Bajo licencia Freeware
 * -----
 *
 * 2005-2006
 *
 *
 * Un objeto de tipo Item, formará parte de otro objeto de tipo Escenario.
 * Implementa la interfaz Comparable y Opcionable.
 *
 * @author Pedro Javier Sáez Martínez
 * @version 1.0

```



```
*/
public class Item extends Sprite implements Opcionable, Comparable {

    /**
     * Nombre del Item, el cual será mostrado cuando el puntero este encima del
     * Item.
     */
    String nombre;

    /**
     * indica si el Item es animado.
     */
    boolean animado;

    /**
     * Tiempo que debe transcurrir para que un item animado, cambie a su
     * siguiente frame.
     */
    int tmpAnima;

    /**
     * Sprite que almacena unos frames especiales del Item, como puede ser la
     * imagen que un objeto tendrá en el inventario, o los frames de los
     * personajes en las conversaciones.
     */
    String imagenEspecial;

    /**
     * Array que indica que opciones está o no disponibles
     */
    public Opcion[] acciones;

    /**
     * Las posibles combinaciones que este Item tiene con otros items.
     */
    public AVLTree combinaciones;

    /**
     * Identificador del Item
     */
    int id;

    /**
     * Indica si el Item se vera delante o detrás del personaje principal, es
     * decir, que si el personaje y el item están en la misma posición, este
     * atributo indica quien se mostrará delante. Y si el Item se verá delante o
     * detrás del escenario Si posición vale... - 0 --> El item estará por
     * detrás del escenario (Objetos no cogibles) - 1 --> El item estará entre
     * el escenario y el personaje principal - 2 --> El item estará por delante
     * del personaje principal
     */
    int posicion;

    /**
     * Constructor de la clase. Se usa cuando el Item es animado.
     *
     * @param nombre
     *         Nombre del Item.
     * @param image
     *         identificador de Utils que define la imagen que representa
     */
}
```



```

* @param esteItem
* @param posX
*     posición X del sprite de este Item
* @param posY
*     posición Y del sprite de este Item
* @param visible
*     indica si el Sprite es visible
* @param animado
*     indica si el Item es animado.
* @param tmpAnima
*     Tiempo que debe transcurrir para que un item animado, cambie a
*     su siguiente frame.
* @param imagenEspecial
*     identificador de Utils que define el Sprite especial del Item
* @param acciones
*     Array que indica que opciones tendrá o no disponibles el Item
* @param posicion
*     Indica como de visible será el Item
* @param id
*     identificador que tendrá el Item
* @param combinable
*     Indica si un Item es combinable con otros Items.
* @param combinaciones
*     Las posibles combinaciones que este Item tiene con otros
*     items.
*/
public Item(String nombre, String image, int posX, int posY,
            boolean visible, boolean animado, int tmpAnima,
            String imagenEspecial, Vector acciones, int posicion, int id,
            boolean combinable, AVLTree combinaciones) {
    // cargamos el sprite
    super(Utils.imagen, Utils.anchoIm, Utils.altoIm);
    this.setFrameSequence(Utils.getFrames(image));
    if (Utils.grupo == 1) { //si el grupo es 176x200
        posX = (posX * Utils.aumentaX) / 1000;
        posY = (posY * Utils.aumentaY) / 1000;
    }
    this.setPosition(Utils.inicioPantallaX + posX, Utils.inicioPantallaY
                    + posY);
    this.setVisible(visible);
    this.nombre = nombre;
    //Inicializamos a null las opciones disponibles.
    this.acciones = new Opcion[acciones.size()];
    acciones.copyInto(this.acciones);
    this.animado = animado;
    this.posicion = posicion;
    this.id = id;
    this.tmpAnima = tmpAnima;
    this.imagenEspecial = imagenEspecial;
    if (combinable) {
        this.combinaciones = combinaciones;
    }
}

/**
 * Indica si una determinada acción tiene o no alguna consecuencia
 */
* @param opcionSele
*     La opción seleccionada en el juego

```



```

    * @return boolean indicando si esa opción o acción tiene alguna
    consecuencia si es true es que no hay consecuencias, y viceversa.
    */
    public boolean getConseActivaAcc(int opcionSele) {

        return acciones[opcionSele].consecuencias.isEmpty();

    }

    /**
     * Establece la consecuencia activa para una determinada acción.
     *
     * @param id
     *      Identificador de la opción a modificar.
     * @param valor
     *      el nuevo valor de conseActiva
     */
    public void setConseActivaAcc(String id, int valor) {
        Opcion auxOp = getOpcion(id);
        auxOp.conseActiva = valor;
    }

    /**
     * Devuelve la consecuencia para una determinada acción
     *
     * @param opcionSele
     *      La opción seleccionada en el juego
     * @return Vector con la consecuencia activa para una determinada acción
     */
    public Vector getConsecuenciaAcc(int opcionSele) {

        return (Vector) acciones[opcionSele].consecuencias
            .elementAt(acciones[opcionSele].conseActiva);

    }

    /**
     * El nombre de una determinada acción.
     *
     * @param opcionSele
     *      La opción seleccionada en el juego
     * @return nombre de la acción.
     */
    public String getNombreAccion(int opcionSele) {
        return acciones[opcionSele].texto;
    }

    /**
     * Devuelve el numero de acciones disponibles para un Seleccionable
     *
     * @return el numero de acciones disponibles para un Seleccionable
     */
    public int longAcciones() {
        return acciones.length;
    }

    /**
     * Establece si una acción esta activa o no
     *
     * @param opcionSele
     *      La opción seleccionada en el juego
     * @param valor

```



```
    */ boolean indicando si está activa o no.
    public void setAccionActiva(int opcionSele, boolean valor) {
        acciones[opcionSele].activa = valor;
    }

    /**
     * Indica si una determinada acción esta activa o no.
     *
     * @param opcionSele
     *     La opción seleccionada en el juego
     * @return boolean indicando si la opción está o no activa.
     */
    public boolean getAccionActiva(int opcionSele) {
        return acciones[opcionSele].activa;
    }

    /**
     * Indica si una determinada combinación tiene o no alguna consecuencia
     *
     * @param indice
     *     Índice de la combinación seleccionada en el juego
     * @return boolean indicando si esa combinación tiene alguna consecuencia si
     *     es true es que no hay consecuencias, y viceversa.
     */
    public boolean getConseActivaCom(int indice) {

        return ((Combinacion) combinaciones.search(indice).data).consecuencias
            .isEmpty();
    }

    /**
     * Establece la consecuencia activa para una determinada combinación.
     *
     * @param id
     *     Id de la combinación seleccionada en el juego
     * @param valor
     *     el nuevo valor de conseActiva
     */
    public void setConseActivaCom(int id, int valor) {
        ((Combinacion) combinaciones.search(id).data).conseActiva = valor;
    }

    /**
     * Devuelve la consecuencia para una determinada combinación
     *
     * @param id
     *     id de la combinación seleccionada en el juego
     * @return Vector con la consecuencia activa para una determinada
     *     combinación
     */
    public Vector getConsecuenciaCom(int id) {
        Combinacion aux = (Combinacion) combinaciones.search(id).data;
        return (Vector) aux.consecuencias.elementAt(aux.conseActiva);
    }

    /**
     * Establece si una combinación esta activa o no
     *
     * @param id
```



```
    * @param id Id de la combinación seleccionada en el juego
    * boolean indicando si está activa o no.
    */
    public void setComActiva(int id, boolean valor) {
        ((Combinacion) combinaciones.search(id).data).activa = valor;
    }

    /**
     * Indica si una determinada combinación esta activa o no.
     *
     * @param id
     *      Id de la combinación seleccionada en el juego
     * @return boolean indicando si la combinación está o no activa.
     */
    public boolean getComActiva(int id) {
        return ((Combinacion) combinaciones.search(id).data).activa;
    }

    /**
     * Devuelve la clave del comparable
     *
     * @return La clave del comparable identificador del comparable.
     */
    public int getKey() {
        return id;
    }

    /**
     * Devuelve la opción cuyo id coincide con el que se pasa como parametro
     *
     * @param id
     *      String con el identificador de la opción
     * @return La opción cuyo id coincide con el de la opción.
     */
    public Opcion getOpcion(String id) {
        for (int i = 0; i < longAcciones(); i++) {
            if (acciones[i] != null) {
                if (acciones[i].id.equals(id)) {
                    return acciones[i];
                }
            }
        }
        return null;
    }
}
```

## Mapa

```
import javax.microedition.lcdui.game.Sprite;

/**
 * Entre 2 Almas [http://gamelab.uniovi.es/entre2almas]
 * -----
 * PFC para la EUITIO [http://www.euitio.uniovi.es/]
```



```

* Desarrollado en el GameLab [http://gamelab.mieres.uniovi.es/]
* Juego licencia Freeware
* -----
*
*
*
*
* El juego tendrá al menos un objeto de tipo Mapa. Implementa la interfaz
* Comparable.
*
* @author Pedro Javier Sáez Martínez
* @version 1.0
*/
public class Mapa implements Comparable {

    /**
     * Identificador del Mapa
     */
    int id;

    /**
     * Arbol AVL con las estancias activas del mapa
     */
    public AVLTree estancias;

    /**
     * Variable que indica el id de la estancia está activa
     */
    public int activa;

    /**
     * Imagen de fondo del mapa
     */
    Sprite representa;

    /**
     * Constructor del Mapa
     *
     * @param id
     *         identificador del mapa.
     * @param representa
     *         Imagen de fondo del mapa
     * @param estancias
     *         Estancias activas del mapa
     * @param activa
     *         Variable que indica cual de las estancias está activa
     */
    public Mapa(int id, String representa, AVLTree estancias, int activa) {
        this.id = id;
        this.estancias = estancias;
        this.activa = activa;
        //cargamos la imagen de fondo del mapa
        Utils.cargarImagen(representa);
        this.representa = new Sprite(Utils.imagen, Utils.anchaIm, Utils.altoIm);
        this.representa.setPosition(Utils.inicioPantallaX,
            Utils.inicioPantallaY);
        this.representa.setFrameSequence(Utils.getFrames(representa));
    }

    /**
     * Devuelve la estancia actual

```





```
    * @return la estancia que actualmente tiene que ser mostrado
    */
    public Estancia getEstanActual() {
        return (Estancia) estancias.search(activa).data;
    }

    /**
     * Devuelve el escenario a mostrar
     *
     * @return el escenario que actualmente tiene que ser mostrado
     */
    public Escenario getEscenActual() {
        return this.getEstanActual().getEscenActual();
    }

    /**
     * Devuelve la clave del comparable
     *
     * @return La clave del comparable identificador del comparable.
     */
    public int getKey() {
        return id;
    }
}
```

## Node

```
/**
 * Class for a node of a binary search tree.
 */
public class Node {

    /** The data stored in the node. */
    public Comparable data;

    /** The node's parent. */
    public Node parent;

    /** The node's balanceFactor. */
    public int bf;

    /** The node's left child. */
    public Node left;

    /** The node's right child. */
    public Node right;

    /**
     * Initializes a node with the key and the data and makes other pointers
     * null. The Balance Factor is initialized to be 0.
     *
     * @param data
     *      Data to save in the node.
     */
    public Node(Comparable data) {
```



```
this.data = data;
this.parent = null;
this.left = null;
this.right = null;
this.bf = 0;
} //Node

/**
 * Performs a left rotation of the subtree rooted at Node a, and returns the
 * new root. The pointers and balance factors are updated as specified in
 * the AVL handout.
 */

public Node rotateLeft() {
    Node b = this.right;
    Node a = b.parent;

    a.right = b.left;
    if (a.right != null)
        a.right.parent = a;

    b.left = a;
    if (b.left != null)
        a.parent = b;

    if (b.bf == +1) {
        a.bf = 0;
        b.bf = 0;
    } else { //b.bf == 0
        a.bf = +1;
        b.bf = -1;
    }

    return b;
} //rotateLeft

/**
 * Performs a right rotation of the subtree rooted at Node a, and returns
 * the new root. The pointers and balance factors are updated as specified
 * in the AVL handout.
 */

public Node rotateRight() {
    Node b = this.left;
    Node a = b.parent;

    a.left = b.right;
    if (a.left != null)
        a.left.parent = a;

    b.right = a;
    if (b.right != null)
        a.parent = b;

    if (b.bf == -1) {
        a.bf = 0;
        b.bf = 0;
    } else { //b.bf == 0
        a.bf = -1;
    }
}
```



```
        }        b.bf = +1;

        return b;

    } //rotateRight

/**
 * Performs a double right left rotation of the subtree rooted at Node a,
 * and returns the new root. The pointers and balance factors are updated as
 * specified in the AVL handout.
 */

public Node rotateRightLeft() {
    Node b = this.right;
    Node a = b.parent;
    Node c = b.left;

    a.right = c.left;
    if (a.right != null)
        a.right.parent = a;

    b.left = c.right;
    if (b.left != null)
        b.left.parent = b;

    //Updates the balance factors differently depending on
    //whether node c had a balance factor of +1 or -1
    if (c.bf == -1) {
        a.bf = 0;
        b.bf = +1;
    } else if (c.bf == 0) {
        a.bf = 0;
        b.bf = 0;
    } else {
        a.bf = -1;
        b.bf = 0;
    }

    c.left = a;
    a.parent = c;

    c.right = b;
    b.parent = c;

    c.bf = 0;

    return c;

} //rotateRightLeft

/**
 * Performs a double left right rotation of the subtree rooted at Node a,
 * and returns the new root. The pointers and balance factors are updated as
 * specified in the AVL handout.
 */

public Node rotateLeftRight() {
    Node b = this.left;
    Node a = b.parent;
    Node c = b.right;
```



```
a.left = c.right;
if (a.left != null)
    a.left.parent = a;

b.right = c.left;
if (b.right != null)
    b.right.parent = b;

//Updates the balance factors differently depending on
//whether node c had a balance factor of +1 or -1
if (c.bf == -1) {
    b.bf = 0;
    a.bf = +1;
} else if (c.bf == 0) {
    b.bf = 0;
    a.bf = 0;
} else {
    b.bf = -1;
    a.bf = 0;
}

c.left = b;
b.parent = c;

c.right = a;
a.parent = c;

c.bf = 0;

return c;

} //rotateLeftRight

} //Node class
```

## Opcion

```
import java.util.Vector;

/**
 * Entre 2 Almas [http://gamelab.uniovi.es/entre2almas]
 * -----
 * PFC para la EUITIO [http://www.euitio.uniovi.es/]
 * Desarrollado en el GameLab [http://gamelab.mieres.uniovi.es/]
 * Bajo licencia Freeware
 * -----
 *
 * 2005-2006
 *
 *
 * El programa podrá mostrar opciones de muchos objetos. Esas opciones serán
 * objetos del tipo Opción
 *
 * @author Pedro Javier Sáez Martínez
```



```
*/@version 1.0
public class Opcion {

    /**
     * Indica si está opción está activa para un determinado Objeto
     */
    boolean activa;

    /**
     * Texto de la opción
     */
    String texto;

    /**
     * Identificador de la opción.
     */
    String id;

    /**
     * Consecuencias de una determinada opción. En este vector se almacenarán
     * vectores de objetos de la clase Resultado. Un Objeto puede tener
     * consecuencias distintas en momentos distintos, entonces aquí almaceno
     * todas las consecuencias que puede tener y para cada consecuencia
     * almacenamos el Vector de Resultados, en el orden en que ocurrirán. Una
     * vez que una consecuencia ya no se necesita, pues se borra del array
     */
    Vector consecuencias;

    /**
     * Indica la consecuencia que está activa, de entre las disponibles.
     */
    int conseActiva;

    /**
     * Constructor de la clase Opción
     *
     * @param activa
     *         indica si está opción está activa para un determinado Objeto
     * @param texto
     *         texto de la opción
     * @param id
     *         Identificador de la opción
     * @param consecuencias
     *         Consecuencias de una determinada opción
     */
    public Opcion(boolean activa, String texto, String id, Vector consecuencias) {
        this.activa = activa;
        this.texto = texto;
        this.id = id;
        this.consecuencias = consecuencias;
        conseActiva = 0;
    }
}
```



## Opcionable

```
import java.util.Vector;

/**
 * Entre 2 Almas [http://gamelab.uniovi.es/entre2almas]
 * -----
 * PFC para la EUITIO [http://www.euitio.uniovi.es/]
 * Desarrollado en el GameLab [http://gamelab.mieres.uniovi.es/]
 * Bajo licencia Freeware
 * -----
 *
 * 2005-2006
 *
 * Aquellas clases cuyos objetos pueden disponer de opciones para mostrar,
 * deberán implementar la interfaz Opcionable
 *
 * @author Pedro Javier Sáez Martínez
 * @version 1.0
 */

public interface Opcionable {

    /**
     * Devuelve la opción cuyo id coincide con el que se pasa como parámetro
     *
     * @param id
     *      String con el identificador de la opción
     * @return La opción cuyo id coincide con el de la opción.
     */
    public Opcion getOpcion(String id);

    /**
     * Indica si una determinada acción tiene o no alguna consecuencia
     *
     * @param opcionSele
     *      La opción seleccionada en el juego
     * @return boolean indicando si esa opción o acción tiene alguna
     *      consecuencia si es true es que no hay consecuencias, y viceversa.
     */
    public boolean getConseActivaAcc(int opcionSele);

    /**
     * Devuelve la consecuencia para una determinada acción
     *
     * @param opcionSele
     *      La opción seleccionada en el juego
     * @return Vector con la consecuencia activa para una determinada acción
     */
    public Vector getConsecuenciaAcc(int opcionSele);

    /**
     * Establece la consecuencia activa para una determinada acción.
     *
     * @param id
     *      Identificador de la opción a modificar.
     * @param valor
     */
}
```



```
    // el nuevo valor de conseActiva
    public void setConseActivaAcc(String id, int valor);

    /**
     * El nombre de una determinada acción.
     *
     * @param opcionSele
     *        La opción seleccionada en el juego
     * @return nombre de la acción.
     */
    public String getNombreAccion(int opcionSele);

    /**
     * Devuelve el numero de acciones disponibles para un Seleccionable
     *
     * @return el numero de acciones disponibles para un Seleccionable
     */
    public int longAcciones();

    /**
     * Establece si una acción esta activa o no
     *
     * @param opcionSele
     *        La opción seleccionada en el juego
     * @param valor
     *        boolean indicando si está activa o no.
     */
    public void setAccionActiva(int opcionSele, boolean valor);

    /**
     * Indica si una acción esta activa o no
     *
     * @param opcionSele
     *        La opción seleccionada en el juego
     * @return boolean indicando si está activa o no
     */
    public boolean getAccionActiva(int opcionSele);
}
```

## PathFinding

```
// [ ----- ]
//[ A* - Algorithm for J2ME ]
//[ ----- ]
//
//[ ----- ]
//[ code written by Sebastian Lehmann ]
//[ This is a simple version of the A*-Algorithm! ]
//[ ----- ]
//[ note that my english is not the very best ]
//[ because I'm from Germany :-) ]
//[ ----- ]
//
//IMPORTANT NOTES:
//While in other A* codes the coordinates of any point
```



```
//is given as a X and Y coordinate here there is the Y
//coordinate multiplied with the width and added to the
//X coordinate. This way decreases the use of memory which
//is in most cases a problem on mobile phones!
//Use functions getIndex(x,y), getX(index) and getY(index)
//to get the coordinates whether as an index value (for
//the use of array index) or as X / Y coords (for the use
//to calculate with coords).
//
//This algorithm doesn't exactly equal A* because I
//don't use the "closed list". But that would only incre-
//ase the use of memory... I think that my way - without
//using closed list - is a bit slower in the speed of
//finding paths. But the memory is - as said - a big
//problem by using A* on mobile applications, so there
//is no other way...

public class PathFinding{
    // maximum map matrix size (needed to initialize arrays)
    int mapWidthMax = 88; // I M P O R T A N T ! ! !
    int mapHeightMax = 100; // Change to max value of map size!!!

    // real map matrix size (needed to calculate coords and paths)
    int mapWidth; //the real size of the map
    int mapHeight;

    // the maximum index value of any array (used for FOR-LOOPS)
    int imax = mapWidthMax*mapHeightMax/8;
    // array to store whether a point is walkable or not
    // NOTE: set values from outside this class or add a function in this class to
    // do this job!
    boolean walkability[] = new boolean[mapWidthMax*mapHeightMax];
    // array to store all paths which are possibly the shortest (used during the
    // calculation)
    // You may ask: Why is the size (mapwidth*mapheight/8)??? => the count of
    // possible paths accepted (because of memory problems on mobile phones...)
    // IMPORTANT: an array field value represents the time needed to walk along the
    // path!
    // NOTE: THIS IS USUALLY CALLED "OPEN LIST" IN ANOTHER A*-CODES!
    int paths[] = new int[imax];

    // the currently count of stored paths in the array paths[] (used during the
    // calculation)
    // You may ask: Why is the count initialized with 1? => this path is the path
    // with the only field equalizing the START field
    int paths_count = 1;

    // the index of paths[] which represents the currently shortest path (the
    // shortest time) (used during the calculation)
    int shortest_path = 0;

    // the coordinate (index-based) of the last field of any PATH (!) (used during
    // the calculation)
    int paths_coord[] = new int[imax];

    // the direction of the parent field of any FIELD (!) (used during the
    // calculation)
    // IF -1: THIS FIELD HAS NO PARENT FIELD (ISN'T INCLUDED IN ANY PATH)
    byte parent[] = new byte[mapWidthMax*mapHeightMax];
```





```
byte direction_down = 64;
byte direction_left = 4;
byte direction_right = 16;

// the guessed way (time) of any path (used during the calculation)
int paths_guessed_way[] = new int[imax];

// used to exit the loop when the path was found
boolean path_found = false;

// the returned array containing the shortest path
int path[] = new int[imax];
// the count:
int path_count;
// time needed to walk along the whole path:
int path_total_length = 0;

//OPTIONAL: try different heuristic methods:
//manhattan: is faster, but gives you not in every case the shortest path
//real: takes more time but gives you in EVERY CASE the shortest path (if there
// is one...)
int manhattan = 1;
int real = 2;
int real_diagonalvalue = 14; //if 14, then "real" finds as said the shortest
// path.
//if 20, then "real" equals "manhattan" heuristic.
//if between, then it's a userdefined heuristic.
int heuristic = manhattan;

public PathFinding(int width, int height){
    // the values width and height represent the size of the map matrix
    mapWidth = width;
    mapHeight = height;
}

public boolean findPath(int startx, int starty, int targetx, int targety){
    // this function is the main part:
    // it calculates the shortest (fastest) path to walk from START to TARGET
    // STARTX / STARTY: the start coordinates
    // TARGETX / TARGETY: the target (end) coordinates

    // RETURN VALUE: TRUE, IF THERE IS A POSSIBLE PATH; OTHERWISE
    FALSE.

    if (startx < 0 || starty < 0 || startx > mapWidth-1 || starty > mapHeight-1 ||
        targetx < 0 || targety < 0 || targetx > mapWidth-1 || targety > mapHeight-1)
        return false;

    // RESET THE VARIABLES
    path_found = false;
    // clear parent array (fill it with byte values -1)
    for (int i1 = 0; i1 <= mapWidth*mapHeight-1; i1++)
        parent[i1] = (byte)-1;
    // init the paths: the 1st path (and only path) is the path with the only field
    // equalizing the START field
    paths_count = 1;
```



```
// it is the shortest path (because it's the only one...)
shortest_path = 0;
// the distance between START and this path is of course zero:
paths[shortest_path] = 0;
// now set the START coord to the array field:
paths_coord[shortest_path] = getIndex(startx,starty);
// its parents is itself:
parent[paths_coord[shortest_path]] = (byte)0;

// calculate the guessed way from the START to TARGET: (CALLD "H COST" IN
// ANOTHER A*-CODES)
int xoffset = Math.abs(getX(paths_coord[0]) - targetx);
int yoffset = Math.abs(getY(paths_coord[0]) - targety);

if (heuristic == manhattan){
    //MANHATTAN DISTANCE (ONLY NON-DIAGONAL:
    paths_guessed_way[0] = (xoffset+yoffset)*10;
}
if (heuristic == real){
    //REAL DISTANCE (DIAGONAL & NON-DIAGONAL):
    if (xoffset > yoffset)
        paths_guessed_way[0] = (xoffset-yoffset)*10 + yoffset*real_diagonalvalue;
    else
        paths_guessed_way[0] = (yoffset-xoffset)*10 + xoffset*real_diagonalvalue;
}
paths_guessed_way[0] += paths[0];

// START SEARCHING FOR THE SHORTEST PATH :

// Loop the search while the end of the shortest path doesn't equal TARGET.
// If there isn't any possible way to walk, the loop will be exited.
while (!path_found){

    ///////////////////////////////////
    //  S T E P 1 //
    ///////////////////////////////////

    //=> CALCULATE THE GUESSED WAY OF ANY PATH (CALLD "F COST" IN
    ANOTHER
    // A*-CODES)
    /*
    * for (int i1 = 0; i1 <= paths_count-1; i1++) { // calculate the guessed way
    * from the end of any path to TARGET: (CALLD "H COST" IN ANOTHER A*-
    CODES)

    * xoffset = Math.abs(getX(paths_coord[i1]) - targetx); yoffset =
    * Math.abs(getY(paths_coord[i1]) - targety); if (heuristic == manhattan) {
    * //MANHATTAN DISTANCE (ONLY NON-DIAGONAL: paths_guessed_way[i1]
    =
    * (xoffset+yoffset)*10; } if (heuristic == real) { //REAL DISTANCE (DIAGONAL &
    * NON-DIAGONAL): if (xoffset > yoffset) paths_guessed_way[i1] =
    * (xoffset-yoffset)*10 + yoffset*real_diagonalvalue; else
    * paths_guessed_way[i1] = (yoffset-xoffset)*10 + xoffset*real_diagonalvalue; }
    * // and add to this way the way from START to the end of any path: (CALLD
    * "G COST" IN ANOTHER A*-CODES)
    *
    * paths_guessed_way[i1] += paths[i1]; }
    */
```



```
ANOTHER

////////////////////
//  S T E P 2 //
////////////////////

//=> FIND OUT THE CURRENTLY SHORTEST PATH ("LOWEST F SCORE" IN
// A*-CODES)

shortest_path = 0;

for (int i1 = 0; i1 <= paths_count-1; i1++){
    // is this way (-> i1) shorter than the shortest path so replace it:
    if (paths_guessed_way[i1] < paths_guessed_way[shortest_path])
        shortest_path = i1;
}

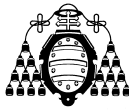
////////////////////
//  S T E P 3 //
////////////////////

//=> SEARCH FOR NEW PATHS AND ADD THEM TO THE ARRAYS ("OPEN
LIST" IN ANOTHER
// A*-CODES)

// try if the current field is positioned at the border of the map,
// and if true then only try the fields inside the map -
// otherwise there would be thrown a ArrayIndexOutOfBoundsException!
int i1min; int i1max; int i2min; int i2max;
if (getX(paths_coord[shortest_path]) <= 0)
    i1min = 0;
else
    i1min = -1;
if (getX(paths_coord[shortest_path]) >= mapWidth-1)
    i1max = 0;
else
    i1max = 1;
if (getY(paths_coord[shortest_path]) <= 0)
    i2min = 0;
else
    i2min = -1;
if (getY(paths_coord[shortest_path]) >= mapHeight-1)
    i2max = 0;
else
    i2max = 1;

// this for-loops try any field around the end of the shortest path
for (int i1 = i1min; i1 <= i1max; i1++){ // XOFFSET
    for (int i2 = i2min; i2 <= i2max; i2++){ // YOFFSET
        int thisx = getX(paths_coord[shortest_path])+i1;
        int thisy = getY(paths_coord[shortest_path])+i2;

        // if the field is neither itself nor START nor unwalkable ...
        if ( !(i1 == 0 && i2 == 0) ) && !(getIndex(thisx,thisy) == getIndex(startx,starty))
        && walkability[getIndex(thisx,thisy)] {
            // calc time needed to walk from old to new field:
```



```
int t_needed = 0;
t_needed += 10;
if (i2 != 0)
    t_needed += 10;
if (t_needed == 20)
    t_needed = 14;

// if this field is already included in another path (IS IN OPEN OR
// CLOSED LIST),
// then check which path is the shorter one:
if (parent[getIndex(thisx,thisy)] > 0) {

    // what's the path# which already exists? => i3
    int i3;
    boolean not_on_closed_list = false;
    search:
    for (i3 = 0; i3 <= paths_count-1; i3++){
        if (paths_coord[i3] == getIndex(thisx,thisy)){
            not_on_closed_list = true;
            break search;
        }
    }

    //if it isn't on closed list then replace the old field with the new
    // one:
    if (not_on_closed_list){
        if (paths[shortest_path] + t_needed < paths[i3]) {
            // set its length (time needed to walk) (G COST)
            paths[i3] = paths[shortest_path] + t_needed;

            // set the new guessed way (F COST)
            xoffset = Math.abs(thisx - targetx);
            yoffset = Math.abs(thisy - targety);

            if (heuristic == manhattan){
                //MANHATTAN DISTANCE (ONLY NON-DIAGONAL:
                paths_guessed_way[i3] = (xoffset+yoffset)*10;
            }
            if (heuristic == real) {
                //REAL DISTANCE (DIAGONAL & NON-DIAGONAL):
                if (xoffset > yoffset)
                    paths_guessed_way[i3] = (xoffset-yoffset)*10 + yoffset*real_diagonalvalue;
                else
                    paths_guessed_way[i3] = (yoffset-xoffset)*10 + xoffset*real_diagonalvalue;
            }
            paths_guessed_way[i3] += paths[i3];

            // set the parent direction of the new field
            parent[paths_coord[i3]] = 0;
            if (i1==1)
                parent[paths_coord[i3]] += direction_left;
            if (i1==-1)
                parent[paths_coord[i3]] += direction_right;
            if (i2==1)
                parent[paths_coord[i3]] += direction_up;
            if (i2==-1)
                parent[paths_coord[i3]] += direction_down;
```



```
    }}
  }
  else { // if it's not already on open list, then add it:
    // add it to the arrays (OPEN LIST)
    paths_coord[paths_count] = getIndex(thisx,thisy);

    // set its length (time needed to walk) (G COST)
    paths[paths_count] = paths[shortest_path] + t_needed;

    // set its guessed way (F COST)
    xoffset = Math.abs(thisx - targetx);
    yoffset = Math.abs(thisy - targety);
    if (heuristic == manhattan){
      //MANHATTAN DISTANCE (ONLY NON-DIAGONAL:
      paths_guessed_way[paths_count] = (xoffset+yoffset)*10;
    }
    if (heuristic == real) {
      //REAL DISTANCE (DIAGONAL & NON-DIAGONAL):
      if (xoffset > yoffset)
        paths_guessed_way[paths_count] = (xoffset-yoffset)*10 +
yoffset*real_diagonalvalue;
      else
        paths_guessed_way[paths_count] = (yoffset-xoffset)*10 +
xoffset*real_diagonalvalue;
    }

    paths_guessed_way[paths_count] += paths[paths_count];

    // set the parent direction of the new field
    parent[paths_coord[paths_count]] = 0;
    if (i1==1)
      parent[paths_coord[paths_count]] += direction_left;
    if (i1==-1)
      parent[paths_coord[paths_count]] += direction_right;
    if (i2==1)
      parent[paths_coord[paths_count]] += direction_up;
    if (i2==-1)
      parent[paths_coord[paths_count]] += direction_down;

    // increase the count of paths in the open list
    paths_count++;
  }
}
}

//////////
//  S T E P 5 //
//////////
```



```
//=> DELETE THE PATH FROM OPEN LIST
if (!path_found) {

    // IF THIS PATH HAS REACHED THE TARGET THEN EXIT LOOP
    if (paths_coord[shortest_path] == getIndex(targetx,targety)) {
        path_found = true;
        //zeit fALr den weg
        path_total_length = paths[shortest_path];
    }

    // CAUTION: IF THE PATH IS THE ONLY ONE => THEN THERE IS NO
    POSSIBLE PATH !!!
    // => EXIT SEARCH
    if (paths_count == 1)
        break;

    // delete the old shortest path from the OPEN LIST:
    for (int i3 = shortest_path+1; i3 <= paths_count-1; i3++) {
        //shift any entry (entry# = i3) to the index before:
        paths[i3-1] = paths[i3];
        paths_coord[i3-1] = paths_coord[i3];
        paths_guessed_way[i3-1] = paths_guessed_way[i3];
    }
    paths_count--;
}

if (!path_found)
    return (false);
else{

    // AFTER THE SEARCH:

    // follow the parent's history backwards

    int fx = targetx; // the backwards followed coordinates
    int fy = targety;
    int n = 0; // the number of backwards followed points

    // DIESE WHILE-SCHLEIFE STELLT DIE RAaCKVERFOLGUNG DES PFADES
    DAR.

    // MANCHMAL ERGIBT DIE WHILE-SCHLEIFE EINE ENDLOSSCHLEIFE!
    // IN DIESEM FALL EINFACH AUSKOMMENTIEREN ( /*...*/ )

    //loop until the path follow reached START
    while(!(fx == startx && fy == starty)) {
        if (parent[getIndex(fx,fy)] == direction_left )
            { addPoint(n,fx,fy); fx--; }
        else if(parent[getIndex(fx,fy)] == direction_left + direction_up )
            { addPoint(n,fx,fy); fx--; fy--; }
        else if(parent[getIndex(fx,fy)] == direction_up )
            { addPoint(n,fx,fy); fy--; }
        else if(parent[getIndex(fx,fy)] == direction_right + direction_up )
            { addPoint(n,fx,fy); fx++; fy--; }
        else if(parent[getIndex(fx,fy)] == direction_right )
```



```
        { addPoint(n,fx,fy); fx++; } == direction_right + direction_down )
        { addPoint(n,fx,fy); fx++; fy++; }
        else if(parent[getIndex(fx,fy)] == direction_down )
        { addPoint(n,fx,fy); fy++; }
        else if(parent[getIndex(fx,fy)] == direction_left + direction_down )
        { addPoint(n,fx,fy); fx--; fy++; }
        n++;
    }

    //path[] nach vorne "holen"
    for (int i1 = 0; i1 <= n-1; i1++){
        path[i1] = path[imax - n + i1];
    }

    //anzahl pfadpunkte
    path_count = n;

    return (true);
}

}

int getIndex(int x, int y){
    return (y * mapWidth + x);
}

int getX(int index){
    return (index % mapWidth);
}

int getY(int index){
    return ((index - getX(index)) / mapWidth);
}

void addPoint(int n, int x, int y){

    path[imax - n - 1] = getIndex(x,y);

    // to find errors, uncomment this line to list the path points
    //System.out.println("x:" + x + " y:" + y + " direction:" +
    // parent[getIndex(x,y)]);
}

}
```

## Resultado

```
/**
 * Entre 2 Almas [http://gamelab.uniovi.es/entre2almas]
 * -----
 * PFC para la EUITIO [http://www.euitio.uniovi.es/]
 * Desarrollado en el GameLab [http://gamelab.mieres.uniovi.es/]
 * Bajo licencia Freeware
```



```
⌘ ----- 2005-2006
*
*
* Un objeto de tipo Opción, tendrá unas consecuencias, y esas consecuencias
* tendrán objetos de tipo Resultado.
*
* @author Pedro Javier Sáez Martínez
* @version 1.0
*/
public class Resultado {

    /**
     * Identificador del resultado a realizar (ejemplo Utils.reMensaje es un
     * numero que aquí servirá como id)
     */
    int id;

    /**
     * Aquí se almacenarán, en los casos en que sea necesario, el texto del
     * resultado a realizar, por ejemplo un mensaje
     */
    String info;

    /**
     * Constructor de la clase Resultado
     *
     * @param id
     *         Identificador del resultado a realizar
     * @param info
     *         información añadida un resultado
     */
    public Resultado(int id, String info) {
        this.id = id;
        this.info = info;
    }
}
```

## SplashScreen

```
import java.util.Timer;
import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.Sprite;

/**
 * Entre 2 Almas [http://gamelab.uniovi.es/entre2almas]
 * -----
 * PFC para la EUITIO [http://www.euitio.uniovi.es/]
 * Desarrollado en el GameLab [http://gamelab.mieres.uniovi.es/]
 * Bajo licencia Freeware
 * -----
 *
 * 2005-2006
 *
 *
 * La pantalla de presentación. Hereda de Canvas.
```





```
⌘ @author Pedro Javier Sáez Martínez
* @version 1.0
*/
public final class SplashScreen extends Canvas {
    /**
     * Manejador de la pantalla
     */
    private Display display;

    /**
     * Objeto con capacidad para ser situado en la pantalla (display), y
     * representa a lo siguiente que aparecerá en la pantalla cuando desaparezca
     * se quite el splash
     */
    private Displayable next;

    /**
     * Controlará el tiempo de ejecución
     */
    private Timer timer;

    /**
     * Sprite que contiene todos los frames del Splash screen
     */
    private Sprite sprite;

    /**
     * Tiempo que se verá cada frame del sprite
     */
    private int dismissTime;

    /**
     * Numero de frames que tiene el sprite
     */
    private static int frames;

    /**
     * Constructor del SplashScreen
     *
     * @param midlet
     *         Midlet del juego.
     * @param next
     *         siguiente pantalla a mostrar una vez que pase el splash
     * @param dismissTime
     *         tiempo que se mostrará cada frame
     */
    public SplashScreen(Entre2Almas midlet, Displayable next, int dismissTime) {
        timer = new Timer();
        this.display = midlet.getDisplay();
        setFullScreenMode(true);
        this.next = next;

        SplashScreen.frames = Utils.getNumFrames(Utils.splash);

        //cargamos los frames del splash screen
        Utils.cargarImagen(Utils.splash);
        sprite = new Sprite(Utils.imagen, Utils.anchorIm, Utils.altoIm);
        sprite.setFrameSequence(Utils.getFrames(Utils.splash));
    }
}
```



```
this.dismissTime = dismissTime;
//situamos el frame del sprite en el centro de la pantalla
sprite.setPosition((getWidth() / 2) - (sprite.getWidth() / 2),
                    (getHeight() / 2) - (sprite.getHeight() / 2));
sprite.setVisible(true);
//al mostrar este canvas en el display salta el metodo showNotify
display.setCurrent(this);

}

/**
 * Comprueba si después de mostrar una imagen del splash, un determinado
 * tiempo, todavía quedan imágenes por mostrar o no
 */
@param splashScreen
 * el splash screen que estamos mostrando
*/
public static void access(SplashScreen splashScreen) {

    if (splashScreen.sprite.getFrame() == frames - 1)
        splashScreen.dismiss();
    else {
        splashScreen.cont();
    }
}

/**
 * Continúa mostrando la siguiente imagen que le corresponde al splash
 * screen
 */
private void cont() {
    sprite.nextFrame();
    repaint();
    showNotify();
}

/**
 * Anula el timer que controlaba el tiempo que se mostraba cada splash y
 * muestra la siguiente pantalla (el menú)
 */
private void dismiss() {
    timer.cancel();
    setFullScreenMode(false);
    display.setCurrent(next);
}

/**
 * Método llamado cuando una tecla es presionada.
 */
protected void keyPressed(int keyCode) {
    dismiss();
}

/**
 * Pinta la pantalla
 */
protected void paint(Graphics g) {
    int width = getWidth();
    int height = getHeight();
    g.setColor(0x00FFFFFF); // white
}
```



```

        g.fillRect(0,0,width,height);
        g.setColor(Color.PINK);
        g.drawRect(1, 1, width - 3, height - 3);

        sprite.paint(g);
    }

    /**
     * Método llamado si un puntero es pulsado
     */
    protected void pointerPressed(int x, int y) {
        dismiss();
    }

    /**
     * Este método salta inmediatamente después de que el Canvas sea visible en
     * la pantalla.
     */
    protected void showNotify() {
        if (dismissTime > 0)
            //schedule hará new Countdown(this) cuando pase el tiempo
            // "dismissTime"
            timer.schedule(new Countdown(this), dismissTime);
    }
}

```

## UIFactory

```

import javax.microedition.lcdui.Graphics;

/**
 * Entre 2 Almas [http://gamelab.uniovi.es/entre2almas]
 * -----
 * PFC para la EUITIO [http://www.euitio.uniovi.es/]
 * Desarrollado en el GameLab [http://gamelab.mieres.uniovi.es/]
 * Bajo licencia Freeware
 * -----
 *
 *
 *
 *
 * Interfaz que define que métodos deberá usar un menú. Es un patrón factory, el
 * cual permite adaptar más fácilmente la interfaz a los distintos tipo de
 * móviles.
 *
 * @author Pedro Javier Sáez Martínez
 * @version 1.0
 */
public interface UIFactory {

    /**
     * Dibuja la pantalla del menú principal
     *
     * @param g
     *      Objeto Graphics
     */
    public abstract void getMenuPrincipal(Graphics g);
}

```



```
/**
 * Muestra las opciones que hay dentro de "Memoria"
 *
 * @param g
 *       Objeto Graphics
 */
public abstract void getPantallaMemoria(Graphics g);

/**
 * Entrar en el menú de opciones
 *
 * @param g
 *       Objeto Graphics
 */
public abstract void getPantallaOpciones(Graphics g);

/**
 * Dibuja el contenido de la "Ayuda"
 *
 * @param g
 *       Objeto Graphics
 */
public abstract void getPantallaAyuda(Graphics g);

/**
 * Dibuja el contenido de las distintas opciones de la ayuda
 *
 * @param g
 *       Objeto Graphics
 */
public void getPantallaSubAyuda(Graphics g);

/**
 * Muestra el contenido de la "Acerca de..."
 *
 * @param g
 *       Objeto Graphics
 */
public abstract void getPantallaAcerca(Graphics g);
}
```

## UITerminalST

```
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Canvas;
import javax.microedition.lcdui.Choice;
import javax.microedition.lcdui.ChoiceGroup;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
```



```
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.List;
import javax.microedition.lcdui.StringItem;

/**
 * Entre 2 Almas [http://gamelab.uniovi.es/entre2almas]
 * -----
 * PFC para la EUITIO [http://www.euitio.uniovi.es/]
 * Desarrollado en el GameLab [http://gamelab.mieres.uniovi.es/]
 * Bajo licencia Freeware
 * -----
 *
 * 2005-2006
 *
 *
 * Es el menú de alto nivel, los cuales se adaptan a las cualidades de los
 * propios móviles.
 *
 * @author Pedro Javier Sáez Martínez
 * @version 1.0
 */
public class UITerminalST extends Canvas implements UIFactory, CommandListener {

    /**
     * Midlet del juego
     */
    private Entre2Almas midlet;

    /**
     * Manejador de la pantalla
     */
    private Display display;

    /**
     * Comando volver, el cual tendrá prioridad 0. Se utiliza en la pantalla de
     * ayuda, de opciones y de memoria
     */
    private Command volverCommand = new Command("Volver", Command.BACK, 0);

    /**
     * Comando Salir, el cual tendrá prioridad 0. Se utiliza en el menú principal
     */
    private Command salirCommand = new Command("Salir", Command.EXIT, 0);

    /**
     * Comando aceptar, el cual tendrá prioridad 1. Se utiliza en la pantalla de
     * opciones
     */
    private Command aceptarCommand = new Command("Aceptar", Command.BACK, 1);

    /**
     * Lista que será utilizada para mostrar las opciones de los distintos
     * menús.
     */
    List opciones;

    /**
     * Formulario en que se mostrarán StringItems y demás.
     */
    Form formulario;
```



```
/**ChoiceGroup utilizado en la pantalla de opciones.
 */
ChoiceGroup opcionOpciones;

/**
 * Mensaje que se mostrará en la ayuda y en el acerca de... Un StringItem va
 * dentro de un Form
 */
StringItem mensaje;

/**
 * Constructor de la clase que representa el interfaz clásico
 */
public UITerminalST(Entre2Almas midlet) {
    this.midlet = midlet;
    this.display = midlet.getDisplay();
    opciones = new List(" ", Choice.IMPLICIT);
    formulario = new Form(" ");
    String[] estados = { Utils.tiposMenu[0], Utils.tiposMenu[1] };
    opcionOpciones = new ChoiceGroup(Utils.tiposMenuTxt, List.EXCLUSIVE,
        estados, null);
    opcionOpciones.setLayout(ChoiceGroup.LAYOUT_NEWLINE_BEFORE);
    opcionOpciones.setSelectedIndex(midlet.tipoMenu, true);
    mensaje = new StringItem(null, " ");
}

/**
 * Dibuja la pantalla del menú principal
 *
 * @param g
 *      objeto Graphics
 */
public void getMenuPrincipal(Graphics g) {
    Utils.cargada = Utils.menuPrin;
    //Vaciamos el contenido de opciones
    opciones.deleteAll();
    opciones.setTitle(Utils.titulo);
    //Añadimos las opciones disponibles a la lista.
    if (Utils.jugando)
        //Si estamos jugando, activamos la opción de seguir jugando,
        //la cual está situada en la posición 0 de el array opMenuPrin
        opciones.append(Utils.opMenuPrin[0], null);

    //añadimos el resto de opciones
    for (int i = 1; i < Utils.opMenuAyuda.length; i++)
        opciones.append(Utils.opMenuPrin[i], null);

    opciones.removeCommand(volverCommand);
    opciones.removeCommand(acceptCommand);
    //      Añadimos a la pantalla los comandos elegir y salir.
    //opciones.addCommand(selectCommand);
    opciones.addCommand(salirCommand);
    int i = Utils.menuPrinIdx;
    if (Utils.jugando)
        opciones.setSelectedIndex(Utils.menuPrinIdx, true);
    else {
        if (Utils.menuPrinIdx > 0)
            opciones.setSelectedIndex(Utils.menuPrinIdx - 1, true);
    }
}
```



```
opciones.setCommandListener(this);
display.setCurrent(opciones);
}

/**
 * Muestra las opciones que hay dentro de "Memoria"
 *
 * @param g
 *       Objeto Graphics
 */
public void getPantallaMemoria(Graphics g) {
    Utils.cargada = Utils.memoria;
    //vaciamos el contenido de la lista opciones
    opciones.deleteAll();
    //El titulo coincidirá con el nombre que tiene en el menú principal
    opciones.setTitle(Utils.opMenuPrin[Utils.menuPrinIdx]);
    opciones.removeCommand(salirCommand);
    //opciones.removeCommand(selectCommand);
    opciones.addCommand(volverCommand);
    //opciones.addCommand(selectCommand);
    opciones.setCommandListener(this);
    //establecemos las opciones de la pantalla memoria
    if (Utils.jugando)
        opciones.append(Utils.opMenuMemo[0], null);
    opciones.append(Utils.opMenuMemo[1], null);
    display.setCurrent(opciones);
}

/**
 * Dibuja el contenido de la pantalla "Opciones"
 *
 * @param g
 *       Objeto Graphics
 */
public void getPantallaOpciones(Graphics g) {
    Utils.cargada = Utils.opcion;
    formulario.deleteAll();
    // El titulo coincidirá con el nombre que tiene en el menú principal
    formulario.setTitle(Utils.opMenuPrin[Utils.menuPrinIdx]);
    opcionOpciones.setSelectedIndex(midlet.tipoMenu, true);
    formulario.append(opcionOpciones);
    formulario.removeCommand(volverCommand);
    formulario.addCommand(acceptCommand);
    formulario.setCommandListener(this);
    display.setCurrent(formulario);
}

/**
 * Dibuja el contenido de la "Ayuda"
 *
 * @param g
 *       Objeto Graphics
 */
public void getPantallaAyuda(Graphics g) {
    Utils.cargada = Utils.ayuda;
    //Vaciamos el contenido de opciones
    opciones.deleteAll();
    // El titulo coincidirá con el nombre que tiene en el menú principal
    opciones.setTitle(Utils.opMenuPrin[Utils.menuPrinIdx]);
    //Añadimos las opciones disponibles a la lista.
```



```
for (int i = 0; i < Utils.opMenuAyuda.length; i++) {
    opciones.append(Utils.opMenuAyuda[i], null);
    opciones.removeCommand(salirCommand);
    opciones.addCommand(volverCommand);
    opciones.setSelectedIndex(Utils.menuAyudaIdx, true);
    opciones.setCommandListener(this);
    display.setCurrent(opciones);
}

/**
 * Dibuja el contenido de la "Ayuda"
 *
 * @param g
 *      Objeto Graphics
 */
public void getPantallaSubAyuda(Graphics g) {
    Utils.cargada = Utils.parteMenu;
    formulario.deleteAll();
    //Dependiendo de la opción de ayuda en la que nos encontramos, le
    // ponemos
    //uno u otro título y uno u otro texto.
    formulario.setTitle(Utils.opMenuAyuda[Utils.menuAyudaIdx]);
    mensaje.setText(Utils.txtSubAyuda[Utils.menuAyudaIdx]);

    formulario.append(mensaje);
    formulario.removeCommand(acceptCommand);
    formulario.addCommand(volverCommand);
    formulario.setCommandListener(this);
    display.setCurrent(formulario);
}

/**
 * Muestra el contenido de la "Acerca de..."
 *
 * @param g
 *      Objeto Graphics
 */
public void getPantallaAcerca(Graphics g) {
    Utils.cargada = Utils.acerca;
    formulario.deleteAll();
    formulario.setTitle(Utils.opMenuPrin[Utils.menuPrinIdx]);
    //StringItem que se mostrará en el form y que contiene
    //lo que mostraremos en la ayuda
    mensaje.setText(Utils.conteAcerca);
    formulario.append(mensaje);
    formulario.removeCommand(acceptCommand);
    formulario.addCommand(volverCommand);
    formulario.setCommandListener(this);
    display.setCurrent(formulario);
}

/**
 * Cuando se pulsa un comando, viene a este método.
 */
public void commandAction(Command c, Displayable d) {
    switch (Utils.parteMenu) {
        case Utils.menuPrin:
            if (c == salirCommand) {
                midlet.salir();
            } else { //El único comando que queda es el botón central de select

```





```
int aux = opciones.getSelectedIndex();
if (!Utils.jugando)
    aux = aux + 1;
switch (aux) {
case 0://Seguir Jugando
    midlet.seguirJugando();
    break;
case 1://Nueva Partida
    midlet.IniciarPartida();
    break;
case 2://Memoria
    Utils.menuPrinIdx = 2;
    Utils.parteMenu = Utils.memoria;
    break;
case 3://Opciones
    Utils.menuPrinIdx = 3;
    Utils.parteMenu = Utils.opcion;
    break;
case 4://Ayuda
    Utils.menuPrinIdx = 4;
    Utils.parteMenu = Utils.ayuda;
    break;
case 5://Acerca de...
    Utils.menuPrinIdx = 5;
    Utils.parteMenu = Utils.acerca;
    break;
}
//si no pones delante el display.setCurrent(this) no tira
//el repaint(), ya que en ese momento el display tendrá como
//objeto Displayable un List y el repaint solo llama al
//paint si el display tiene como Displayable un canvas.
if (aux > 1) { //Si no quiere ni seguir partida ni nueva partida
    display.setCurrent(this);
    repaint();
}
}

break;
case Utils.memoria:
    if (c == volverCommand) {
        Utils.parteMenu = Utils.menuPrin;
        display.setCurrent(this);
        repaint();
    } else {
        int aux = opciones.getSelectedIndex();
        if (!Utils.jugando)
            aux = aux + 1;
        switch (aux) {
        case 0://Guardar Partida
            if (midlet.puedeGuardar()) {
                Utils.guardandoPartida = true;
                midlet.seguirJugando();
            } else
                midlet
                    .mostrarAlerta(
                        "Mientras en la
partida estés en un video, "

+ "en una conversación, o se este mostrando un mensaje, no se podrá guardar ")
    }
}
```



```
+ "la partida", AlertType.INFO); break;
    case 1://Cargar Partida
        if (midlet.hayGuardada()) {
            Utils.cargandoPartida = true;
            midlet.IniciarPartida();
        } else
            midlet
                .mostrarAlerta(
                    "No hay ninguna
partida guardada, la cual se pueda cargar.",
                    AlertType.INFO);
            break;
        }
    }
    break;
case Utils.ayuda:
    if (c == volverCommand)
        Utils.parteMenu = Utils.menuPrin;
    else { //El único comando que queda es el botón central de select
        int aux = opciones.getSelectedIndex();
        switch (aux) {
            case 0:
                Utils.menuAyudaIdx = 0;
                Utils.parteMenu = Utils.ayudaTematica;
                break;
            case 1:
                Utils.menuAyudaIdx = 1;
                Utils.parteMenu = Utils.ayudaTeclas;
                break;
            case 2:
                Utils.menuAyudaIdx = 2;
                Utils.parteMenu = Utils.ayudaVideo;
                break;
            case 3:
                Utils.menuAyudaIdx = 3;
                Utils.parteMenu = Utils.ayudaInventario;
                break;
            case 4:
                Utils.menuAyudaIdx = 4;
                Utils.parteMenu = Utils.ayudaMapa;
                break;
            case 5:
                Utils.menuAyudaIdx = 5;
                Utils.parteMenu = Utils.ayudaConversacion;
                break;
        }
    }
    display.setCurrent(this);
    repaint();
    break;
case Utils.acerca:
    if (c == volverCommand) {
        Utils.parteMenu = Utils.menuPrin;
        display.setCurrent(this);
        repaint();
    }
}
```



```
        break;
    case Utils.opcion:
        if (c == acceptCommand) {
            midlet.tipoMenu = opcionOpciones.getSelectedIndex();
            Utils.parteMenu = Utils.menuPrin;
            midlet.setMenuRs();
            midlet.menu();
        }
        break;
    default:
        if (c == volverCommand) {
            Utils.parteMenu = Utils.ayuda;
            display.setCurrent(this);
            repaint();
        }
        break;
    }
}

/**
 * Método llamado por el display cuando pone en pantalla este Canvas
 */
protected void paint(Graphics g) {
    if (Utils.cargada != Utils.parteMenu) {
        switch (Utils.parteMenu) {
            case Utils.menuPrin://Menú principal
                getMenuPrincipal(g);
                break;
            case Utils.memoria://Memoria:
                getPantallaMemoria(g);
                break;
            case Utils.ayuda://Ayuda
                getPantallaAyuda(g);
                break;
            case Utils.acerca://Acerca de...
                getPantallaAcerca(g);
                break;
            case Utils.opcion://opciones
                getPantallaOpciones(g);
                break;
            default:
                getPantallaSubAyuda(g);
                break;
        }
    }
}
```

## UITerminalSTBG

```
import java.util.Vector;

import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Canvas;
import javax.microedition.lcdui.Command;
```



```
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.game.Sprite;

/**
 * Entre 2 Almas [http://gamelab.uniovi.es/entre2almas]
 * -----
 * PFC para la EUITIO [http://www.euitio.uniovi.es/]
 * Desarrollado en el GameLab [http://gamelab.mieres.uniovi.es/]
 * Bajo licencia Freeware
 * -----
 *
 * 2005-2006
 *
 *
 * Menú de bajo nivel, es decir, que será igual en todos los móviles, pero es
 * más colorido, más bonito, más vistoso
 *
 * @author Pedro Javier Sáez Martínez
 * @version 1.0
 */
public class UITerminalSTBG extends Canvas implements UIFactory,
    CommandListener {

    /**
     * Midlet del juego
     */
    private Entre2Almas midlet;

    /**
     * sprite que albergará las imágenes de fondo.
     */
    private Sprite sprite;

    /**
     * Comando volver, el cual tendrá prioridad 0.Se utiliza en la pantalla de
     * ayuda, de opciones y de memoria
     */
    private Command volverCommand = new Command("Volver", Command.BACK, 0);

    /**
     * Comando Elegir, el cual tendrá prioridad 1.Se utiliza en el menú
     * principal
     */
    private Command selectCommand = new Command("Elegir", Command.OK, 1);

    /**
     * Comando Salir, el cual tendrá prioridad 0.Se utiliza en el menú principal
     */
    private Command salirCommand = new Command("Salir", Command.EXIT, 0);

    /**
     * Comando aceptar, el cual tendrá prioridad 1.Se utiliza en la pantalla de
     * opciones
     */
    private Command acceptCommand = new Command("Aceptar", Command.BACK, 1);

    /**
     * Indica cual de las opciones del menú opciones está seleccionada.
     */
}
```



```
private int menuOpIdx;
/**
 * Indica cual de las opciones del menú memoria está seleccionada.
 */
private int menuMemIdx;

/**
 * Opciones de los menús
 */
private Vector opciones;

public UITerminalSTBG(Entre2Almas midlet) {
    //Ponemos la pantalla a modo completo. Hay que poner esto antes
    //de saber en que grupo está el móvil, porque si no está
    //a fullScreen la zona de comandos como una parte de la pantalla
    //lo cual hace que getHeight() sea menor que el valor absoluto real
    setFullScreenMode(true);

    //Obtenemos el grupo al que pertenece el móvil.
    Utils.getGrupo(getWidth(), getHeight());

    ajustarVariables();
    this.midlet = midlet;

    Utils.mensaje = new StringBuffer();
    Utils.palabrasMensaje = new Vector();

    opciones = new Vector();

    Utils.cargada = -1;
    Utils.parteMenu = Utils.menuPrin;

    // Se establece como opción seleccionada la primera del menú
    Utils.menuPrinIdx = 0;

    //cargamos el fondo de pantalla del menú principal
    Utils.cargarImagen(Utils.imMenuPrin);
    sprite = new Sprite(Utils.imagen, Utils.anchoIm, Utils.altoIm);
    sprite.setFrameSequence(Utils.getFrames(Utils.imMenuPrin));

    // situamos el frame del sprite en el centro de la pantalla
    sprite.setPosition((Utils.width / 2) - (sprite.getWidth() / 2),
        (Utils.height / 2) - (sprite.getHeight() / 2));
    sprite.setVisible(true);

    //Establecemos el CommandListener
    setCommandListener(this);
}

/**
 * Dibuja la pantalla del menú principal
 *
 * @param g
 *      objeto Graphics
 */
public void getMenuPrincipal(Graphics g) {
    sprite.paint(g);
}
```



```
g.setFont(Utils.lowFont);
g.setColor(Utils.lowColor);
int size = opciones.size();
for (int i = 0; i < size; i++) {
    //Si la opción esta seleccionada, utilizamos highFont, highColor
    if (i == Utils.menuPrinIdx) {
        g.setFont(Utils.highFont);

        g.setColor(Utils.highColor);
        g
            .drawString((String) opciones.elementAt(i),
                (Utils.width - Utils.highFont
                    .stringWidth((String) opciones
                        .elementAt(i))) / 2,
                    Utils.startHeight
                        + (i *
                            Utils.highFont.getHeight())
                        + Utils.spacing,
                    Graphics.TOP
                        |
                    Graphics.LEFT);

        g.setFont(Utils.lowFont);
        g.setColor(Utils.lowColor);
    } else {
        //si la opción no esta seleccionada, utilizamos
        //lowFonto y lowColor
        g
            .drawString((String) opciones.elementAt(i),
                (Utils.width - Utils.lowFont
                    .stringWidth((String) opciones
                        .elementAt(i))) / 2,
                    Utils.startHeight
                        + (i *
                            Utils.highFont.getHeight())
                        + Utils.spacing,
                    Graphics.TOP
                        |
                    Graphics.LEFT);
    }
}

/**
 * Muestra las opciones que hay dentro de "Memoria"
 */
public void getPantallaMemoria(Graphics g) {
    sprite.paint(g);

    g.setFont(Utils.lowFont);
    g.setColor(Utils.lowColor);
    int size = opciones.size();
    for (int i = 0; i < size; i++) {
        //Si la opción esta seleccionada, utilizamos highFont, highColor
        if (i == menuMemIdx) {
            g.setFont(Utils.highFont);
```



```
        g.setColor(Utils.highColor);
        g
            .drawString((String) opciones.elementAt(i),
                        (Utils.width - Utils.highFont

        .stringWidth((String) opciones

        .elementAt(i))) / 2,

                                Utils.startHeight
                                + (i *
Utils.highFont.getHeight())
                                + Utils.spacing,
Graphics.TOP
                                |
Graphics.LEFT);

        g.setFont(Utils.lowFont);
        g.setColor(Utils.lowColor);

    } else {
        //si la opción no esta seleccionada, utilizamos
        //lowFonto y lowColor
        g
            .drawString((String) opciones.elementAt(i),
                        (Utils.width - Utils.lowFont

        .stringWidth((String) opciones

        .elementAt(i))) / 2,

                                Utils.startHeight
                                + (i *
Utils.highFont.getHeight())
                                + Utils.spacing,
Graphics.TOP
                                |
Graphics.LEFT);
    }
}

/**
 * Muestra la pantalla de guardando
 */
public void getPantallaGuardando(Graphics g) {
    if (Utils.guardandoPartida)
        sprite.paint(g);
    else
        Utils.parteMenu = Utils.memoria;
}

/**
 * Dibuja el contenido de la "Ayuda"
 *
 * @param g
 *         Objeto Graphics
 */
public void getPantallaAyuda(Graphics g) {
    sprite.paint(g);

    g.setFont(Utils.lowFont);
```



```
g.setColor(Utils.lowColor);
int size = opciones.size();
for (int i = 0; i < size; i++) {
    //Si la opción esta seleccionada, utilizamos highFont, highColor
    if (i == Utils.menuAyudaIdx) {
        g.setFont(Utils.highFont);

        g.setColor(Utils.highColor);
        g
            .drawString((String) opciones.elementAt(i),
                (Utils.width - Utils.highFont
                    .stringWidth((String) opciones
                        .elementAt(i))) / 2,
                    Utils.startHeight
                        + (i *
                            Utils.highFont.getHeight())
                        + Utils.spacing,
                    Graphics.TOP
                        |
                    Graphics.LEFT);

        g.setFont(Utils.lowFont);
        g.setColor(Utils.lowColor);

    } else {
        //si la opción no esta seleccionada, utilizamos
        //lowFonto y lowColor
        g
            .drawString((String) opciones.elementAt(i),
                (Utils.width - Utils.lowFont
                    .stringWidth((String) opciones
                        .elementAt(i))) / 2,
                    Utils.startHeight
                        + (i *
                            Utils.highFont.getHeight())
                        + Utils.spacing,
                    Graphics.TOP
                        |
                    Graphics.LEFT);
    }
}

/**
 * Muestra el contenido de la "Acerca de..."
 */
public void getPantallaAcerca(Graphics g) {
    System.gc();
    sprite.paint(g);

    g.setFont(Utils.lowFont);
    g.setColor(Utils.highColor);

    //aux2 almacena el numero de letras que lleva una línea
    int aux2 = 0;
    //Identifica en que línea toca imprimir
    int lineaAux = 0;
```





```
//Ira almacenando el contenido de una línea
StringBuffer aux = new StringBuffer();
int size = Utils.palabrasMensaje.size();
int tamPalabra = 0;
String palabra;
for (int i = 0; i < size; i++) {
    palabra = ((String) Utils.palabrasMensaje.elementAt(i));
    tamPalabra = palabra.length() * Utils.widthLF;
    //la ideal sería la de abajo, pero como carga demasiado pues uso la
    // anterior
    //tamPalabra=Utils.lowFont.stringWidth(palabra);
    //Si es salto de línea, imprimimos y aumentamos en una la lineaAux
    if (palabra.equals("\n")) {
        if (lineaAux >= Utils.linea
            && lineaAux <= Utils.numLineas + Utils.linea)
            g
                .drawString(
                    aux.toString(),
                    Utils.startWidth,
                    Utils.startHeight
                    +
                    ((lineaAux - Utils.linea) * Utils.spacing),
                    Graphics.TOP |
                    Graphics.LEFT);
        else if (lineaAux > Utils.numLineas + Utils.linea)
            size = 0; //para salir del bucle
            aux.delete(0, aux.length());
            lineaAux = lineaAux + 2;
            aux2 = 0;
        } else {
            if (tamPalabra > Utils.spaceMen) {
                // imprimimos lo
                que hay previamente
                g
                    .drawString(
                        aux.toString(),
                        Utils.startWidth,
                        Utils.startHeight
                        +
                        ((lineaAux - Utils.linea) * Utils.spacing),
                        Graphics.TOP |
                        Graphics.LEFT);
                aux.delete(0, aux.length());
                lineaAux++;
                aux2 = 0;
                //Imprimimos la primera parte
                int noCaben = (tamPalabra - Utils.spaceMen) /
                    Utils.widthLF;
                aux.append(palabra.substring(0, palabra.length() - noCaben
                    - 1));
                g
                    .drawString(
                        aux.toString(),
                        Utils.startWidth,
                        Utils.startHeight
                        +
                        ((lineaAux - Utils.linea) * Utils.spacing),
                        Graphics.TOP |
                        Graphics.LEFT);
                aux.delete(0, aux.length());
```



```

        lineaAux++;
        aux2 = 0;
        //Y luego el final que no entraba lo añadimos a la palabra
        //de la siguiente línea
        aux.append(palabra
                    .substring(palabra.length() - noCaben - 1)
                    + " ");
        aux2 = aux2
        + palabra.substring(palabra.length() -
                             noCaben - 1)
                             .length() * Utils.widthLF
        + Utils.lowFont.charWidth(' ');
    } else {
        //si el numero de letras de esa línea (aux2) es menor que
        // el numero
        //de letras que caben en esta pantalla, restándole además
        //el tamaño de la ultima palabra,
        if (aux2 + tamPalabra < Utils.spaceMen) {
            //Introducimos en aux, la palabra seguida de un
            espacio
            aux.append(palabra + " ");
            //actualizamos el valor de aux2
            aux2 = aux2 + tamPalabra +
            Utils.lowFont.charWidth(' ');
        } else {
            //Solo imprime las filas que serán visibles, es decir
            //para no imprimir líneas que no se verían pues
            estaría
            //en la parte negra
            if (lineaAux >= Utils.linea
                && lineaAux <= Utils.numLineas
                + Utils.linea)
            g
            .drawString(
                aux.toString(),
                Utils.startWidth,
                Utils.startHeight
                + ((lineaAux - Utils.linea) * Utils.spacing),
                Graphics.TOP | Graphics.LEFT);
            else if (lineaAux > Utils.numLineas + Utils.linea)
                size = 0; //para salir del bucle
            //al imprimir una línea borramos el contenido de aux
            aux.delete(0, aux.length());
            //aumentamos el valor de lineaAux
            lineaAux++;
            //aux2 vuelve a ser 0
            aux2 = 0;
            //y en aux metemos la palabra que no se ha
            introducido
            //en la línea que se acaba de imprimir, porque el
            // numeroLetras
            //de esa línea ya sobrepasaba los limites
            aux.append(palabra + " ");
            aux2 = aux2 + tamPalabra +
            Utils.lowFont.charWidth(' ');
        }
    }
}
```



```
        }
    }
}
//Cuando llegamos al final, imprimimos la línea (este como este)
if (i + 1 == Utils.palabrasMensaje.size()) {
    if (lineaAux >= Utils.linea
        && lineaAux <= Utils.numLineas + Utils.linea)
    {
        g
            .drawString(
                aux.toString(),
                Utils.startWidth,
                Utils.startHeight
                +
                ((lineaAux - Utils.linea) * Utils.spacing),
                Graphics.TOP |
                Graphics.LEFT);
    }
}
//el numero de líneas que ocupa el mensaje.
Utils.numLineasMensaje = lineaAux;
//si el numero de líneas que caben en la pantalla + la línea
//en la cual se imprime la primera línea es menor que el numero
//de líneas que ocupa el mensaje (teniendo en cuenta que si el
//mensaje ocupa 7 líneas, numLineas valdrá 6), se pinta una flecha
//hacia abajo
if (Utils.numLineas + Utils.linea < Utils.numLineasMensaje) {
    g.setColor(0x00000000);
    g
        .fillTriangle((Utils.width / 2) - 7,
            Utils.startHeight
            + (Utils.numLineas *
                Utils.spacing)
            + Utils.spacing,
            (Utils.width / 2),
            Utils.startHeight
            + (Utils.numLineas *
                Utils.spacing)
            + Utils.spacing + 7,
            (Utils.width / 2) + 7,
            Utils.startHeight
            + (Utils.numLineas *
                Utils.spacing)
            + Utils.spacing);
}

}

/**
 * Dibuja el contenido de la pantalla "Opciones"
 *
 * @param g
 *      Objeto Graphics
 */
public void getPantallaOpciones(Graphics g) {
    sprite.paint(g);

    g.setFont(Utils.lowFont);
    g.setColor(Utils.lowColor);
    int size = opciones.size();
```



```

        for (int i = 0; i < size; i++) {
            //Si la opción esta seleccionada, utilizamos highFont, highColor
            if (i == menuOpIdx) {
                g.setFont(Utils.highFont);

                g.setColor(Utils.highColor);
                g.drawString((String) opciones.elementAt(i), Utils.startWidth,
                    Utils.startHeight + (i * Utils.highFont.getHeight())
                    + Utils.spacing, Graphics.TOP |
Graphics.LEFT);

                g.setFont(Utils.lowFont);
                g.setColor(Utils.lowColor);

            } else {
                //si la opción no esta seleccionada, utilizamos
                //lowFonto y lowColor
                g.drawString((String) opciones.elementAt(i), Utils.startWidth,
                    Utils.startHeight + (i * Utils.highFont.getHeight())
                    + Utils.spacing, Graphics.TOP |
Graphics.LEFT);
            }
        }
    }

/**
 * Dibuja el contenido de las distintas opciones de la ayuda
 *
 * @param g
 *      Objeto Graphics
 */
public void getPantallaSubAyuda(Graphics g) {
    System.gc();
    sprite.paint(g);

    g.setFont(Utils.lowFont);
    g.setColor(Utils.highColor);

    //aux2 almacena el numero de letras que lleva una línea
    int aux2 = 0;
    //Identifica en que línea toca imprimir
    int lineaAux = 0;
    //Ira almacenando el contenido de una línea
    StringBuffer aux = new StringBuffer();
    int size = Utils.palabrasMensaje.size();
    int tamPalabra = 0;
    String palabra;
    for (int i = 0; i < size; i++) {
        palabra = ((String) Utils.palabrasMensaje.elementAt(i));
        tamPalabra = palabra.length() * Utils.widthLF;
        //la ideal sería la de abajo, pero como carga demasiado pues uso la
        // anterior
        //tamPalabra=Utils.lowFont.stringWidth(palabra);
        //Si es salto de línea, imprimimos y aumentamos en una la lineaAux
        if (palabra.equals("\n")) {
            if (lineaAux >= Utils.linea
                && lineaAux <= Utils.numLineas + Utils.linea)
                g
                    .drawString(
                        aux.toString(),
                        Utils.startWidth,

```



```
Utils.startHeight +
((lineaAux - Utils.linea) * Utils.spacing),
Graphics.TOP |
Graphics.LEFT);
else if (lineaAux > Utils.numLineas + Utils.linea)
    size = 0; // para salir del bucle
aux.delete(0, aux.length());
// pasamos a la línea siguiente y
dejamos un hueco en blanco
// entre medias.
lineaAux = lineaAux + 2;
aux2 = 0;
} else {
    if (tamPalabra > Utils.spaceMen) {
        // imprimimos lo
        que hay previamente
        g
        .drawString(
            aux.toString(),
            Utils.startWidth,
            Utils.startHeight
            +
            ((lineaAux - Utils.linea) * Utils.spacing),
            Graphics.TOP |
            Graphics.LEFT);
        aux.delete(0, aux.length());
        lineaAux++;
        aux2 = 0;
        // Imprimimos la primera parte
        int noCaben = (tamPalabra - Utils.spaceMen) /
        aux.append(palabra.substring(0, palabra.length() - noCaben
            - 1));
        g
        .drawString(
            aux.toString(),
            Utils.startWidth,
            Utils.startHeight
            +
            ((lineaAux - Utils.linea) * Utils.spacing),
            Graphics.TOP |
            Graphics.LEFT);
        aux.delete(0, aux.length());
        lineaAux++;
        aux2 = 0;
        // Y luego el final que no entraba lo añadimos a la palabra
        // de la siguiente línea
        aux.append(palabra
            .substring(palabra.length() - noCaben - 1)
            + " ");
        aux2 = aux2
            + palabra.substring(palabra.length() -
            noCaben - 1)
            .length() * Utils.widthLF
            + Utils.lowFont.charWidth(' ');
    } else {
        // si el numero de letras de esa línea (aux2) es menor que
        // el numero
        // de letras que caben en esta pantalla, restándole además
```



```

//el tamaño de la última palabra
if (aux2 + tamPalabra < Utils.spaceMen) {
    //Introducimos en aux, la palabra seguida de un
    espacio
    aux.append(palabra + " ");
    //actualizamos el valor de aux2
    aux2 = aux2 + tamPalabra +

Utils.lowFont.charWidth(' ');
} else {
    //Solo imprime las filas que serán visibles, es decir
    //para no imprimir líneas que no se verían pues
    estaría
    //en la parte negra
    if (lineaAux >= Utils.linea
        && lineaAux <= Utils.numLineas
+ Utils.linea)

        g
        .drawString(

            aux.toString(),
            Utils.startWidth,
            Utils.startHeight
            + ((lineaAux - Utils.linea) * Utils.spacing),
            Graphics.TOP | Graphics.LEFT);

    else if (lineaAux > Utils.numLineas + Utils.linea)
        size = 0; //para salir del bucle
    //al imprimir una línea borramos el contenido de aux
    aux.delete(0, aux.length());
    //aumentamos el valor de lineaAux
    lineaAux++;
    //aux2 vuelve a ser 0
    aux2 = 0;
    //y en aux metemos la palabra que no se ha
    introducido
    //en la línea que se acaba de imprimir, porque el
    // numeroLetras
    //de esa línea ya sobrepasaba los límites
    aux.append(palabra + " ");
    aux2 = aux2 + tamPalabra +

Utils.lowFont.charWidth(' ');

}
}
}
//Cuando llegamos al final, imprimimos la línea (este como este)
if (i + 1 == Utils.palabrasMensaje.size()) {
    if (lineaAux >= Utils.linea
        && lineaAux <= Utils.numLineas + Utils.linea)

        g
        .drawString(
            aux.toString(),
            Utils.startWidth,
            Utils.startHeight
            +
            ((lineaAux - Utils.linea) * Utils.spacing),

```



```
Graphics.LEFT);      }
    }
    //el numero de líneas que ocupa el mensaje.
    Utils.numLineasMensaje = lineaAux;
    //si el numero de líneas que caben en la pantalla + la línea
    //en la cual se imprime la primera línea es menor que el numero
    //de líneas que ocupa el mensaje (teniendo en cuenta que si el
    //mensaje ocupa 7 líneas, numLineas valdrá 6), se pinta una flecha
    //hacia abajo
    if (Utils.numLineas + Utils.linea < Utils.numLineasMensaje) {
        g.setColor(0x00000000);
        g
            .fillTriangle((Utils.width / 2) - 7,
                Utils.startHeight
                    + (Utils.numLineas *
                        Utils.spacing)
                    + Utils.spacing,
                (Utils.width / 2),
                    Utils.startHeight
                        + (Utils.numLineas *
                            Utils.spacing)
                        + Utils.spacing + 7,
                (Utils.width / 2) + 7,
                    Utils.startHeight
                        + (Utils.numLineas *
                            Utils.spacing)
                        + Utils.spacing);
    }
}

/**
 * Decide que y cuando cargar en cada momento
 *
 * @param g
 *      Objeto Graphics
 * @param pantalla
 *      pantalla que se desea cargar (ejemplo ayuda, opciones...)
 */
private void cargar(Graphics g, int pantalla) {
    switch (pantalla) {
        case Utils.menuPrin://pantalla del menú principal
            Utils.cargada = Utils.menuPrin;

            Utils.linea = 0;
            removeCommand(acceptCommand);
            removeCommand(volverCommand);
            addCommand(salirCommand);
            addCommand(selectCommand);

            //cargamos el fondo de pantalla del menú principal
            Utils.cargarImagen(Utils.imMenuPrin);
            sprite.setImage(Utils.imagen, Utils.anchoIm, Utils.altoIm);
            sprite.setFrameSequence(Utils.getFrames(Utils.imMenuPrin));

            //vaciamos el vector opciones
            opciones.removeAllElements();
    }
}
```



lista.

```
//                                Añadimos las opciones disponibles a la
if (Utils.jugando)
    //Si estamos jugando, activamos la opción de seguir jugando,
    //la cual está situada en la posición 0 de el array opMenuPrin
    opciones.addElement(Utils.opMenuPrin[0]);

//añadimos el resto de opciones
for (int i = 1; i < Utils.opMenuAyuda.length; i++)
    opciones.addElement(Utils.opMenuPrin[i]);
//espacio entre las opciones del menú principal
Utils.spacing = Utils.highFont.getHeight() / 2;

//Calcula la altura a la que se muestra la primera de las
//opciones del menú
//Teniendo en cuenta que hay n-1 opciones en minúsculas
//1 opción en mayúsculas, y que luego habrá n-1 espacios
Utils.startHeight = (Utils.lowFont.getHeight() * (opciones.size() - 1))
    + (Utils.highFont.getHeight())
    + ((opciones.size() - 1) * Utils.spacing);
Utils.startHeight = (Utils.height - Utils.startHeight) / 2;

g.setColor(0x00000000);
g.fillRect(0, 0, Utils.width, Utils.height);

getMenuPrincipal(g);
break;
case Utils.memoria://pantalla de Memoria
    Utils.cargada = Utils.memoria;
    Utils.linea = 0;
    menuMemIdx = 0;
    removeCommand(salirCommand);
    addCommand(volverCommand);
    addCommand(selectCommand);

//                                cargamos el fondo de pantalla de la
memoria

Utils.cargarImagen(Utils.imMemoria);
sprite.setImage(Utils.imagen, Utils.anchoIm, Utils.altoIm);
sprite.setFrameSequence(Utils.getFrames(Utils.imMemoria));

//vaciamos el vector opciones
opciones.removeAllElements();
//establecemos las opciones de Memoria
if (Utils.jugando)
    opciones.addElement(Utils.opMenuMemo[0]);
opciones.addElement(Utils.opMenuMemo[1]);
//espacio entre las opciones del menú principal
Utils.spacing = Utils.highFont.getHeight() / 2;

//Calcula la altura a la que se muestra la primera de las
//opciones del menú
//Teniendo en cuenta que hay n-1 opciones en minúsculas
//1 opción en mayúsculas, y que luego habrá n-1 espacios
Utils.startHeight = (Utils.lowFont.getHeight() * (opciones.size() - 1))
    + (Utils.highFont.getHeight())
    + ((opciones.size() - 1) * Utils.spacing);
Utils.startHeight = (Utils.height - Utils.startHeight) / 2;

g.setColor(0x00000000);
```





```
g.fillRect(0, 0, Utils.width, Utils.height);
getPantallaMemoria(g);
break;
case Utils.ayuda://Ayuda
    Utils.cargada = Utils.ayuda;
    Utils.linea = 0;
    removeCommand(salirCommand);
    addCommand(volverCommand);
    addCommand(selectCommand);

    //                                cargamos el fondo de pantalla de la ayuda
    Utils.cargarImagen(Utils.imAyuda);
    sprite.setImage(Utils.imagen, Utils.anchoIm, Utils.altoIm);
    sprite.setFrameSequence(Utils.getFrames(Utils.imAyuda));

    //vaciamos el vector opciones
    opciones.removeAllElements();
    //Añadimos las opciones disponibles a la lista.
    for (int i = 0; i < Utils.opMenuAyuda.length; i++)
        opciones.addElement(Utils.opMenuAyuda[i]);
    //espacio entre las opciones del menú principal
    Utils.spacing = Utils.highFont.getHeight() / 2;

    //Calcula la altura a la que se muestra la primera de las
    //opciones del menú
    //Teniendo en cuenta que hay n-1 opciones en minúsculas
    //1 opción en mayúsculas, y que luego habrá n-1 espacios
    Utils.startHeight = (Utils.lowFont.getHeight() * (opciones.size() - 1))
        + (Utils.highFont.getHeight())
        + ((opciones.size() - 1) * Utils.spacing);
    Utils.startHeight = (Utils.height - Utils.startHeight) / 2;

    g.setColor(0x00000000);
    g.fillRect(0, 0, Utils.width, Utils.height);

    getPantallaAyuda(g);
    break;
case Utils.acerca://Acerca de...
    Utils.cargada = Utils.acerca;
    Utils.linea = 0;
    removeCommand(selectCommand);
    removeCommand(salirCommand);
    addCommand(volverCommand);

    //                                cargamos el fondo de pantalla del menú
principal
    Utils.cargarImagen(Utils.imAcerca);
    sprite.setImage(Utils.imagen, Utils.anchoIm, Utils.altoIm);
    sprite.setFrameSequence(Utils.getFrames(Utils.imAcerca));

    //Espacio entre cada una de las líneas de la ayuda.
    Utils.spacing = Utils.lowFont.getHeight();

    //                                Calcula la altura y anchura a la que empieza
el acerca de...
    if (Utils.height <= sprite.getHeight())
        Utils.startHeight = 0;
    else
        Utils.startHeight = (Utils.height / 2)
```



```
- (sprite.getHeight() / 2);
if (Utils.width <= sprite.getWidth())
    Utils.startWidth = Utils.lowFont.getSize();
else
    Utils.startWidth = (Utils.width / 2) - (sprite.getWidth() / 2)
        + Utils.lowFont.getSize();

//inicializamos el valor de mensaje, y palabraMensaje
Utils.mensaje.delete(0, Utils.mensaje.length());
Utils.palabrasMensaje.removeAllElements();

Utils.mensaje.append(Utils.conteAcerca);
//Hayamos cada una de las palabras del mensaje
StringBuffer aux2 = new StringBuffer();
for (int i = 0; i < Utils.mensaje.length(); i++) {
    char temp = Utils.mensaje.charAt(i);
    if (temp != ' ')
        aux2.append(temp);
    else {
        Utils.palabrasMensaje.addElement(aux2.toString());
        aux2.delete(0, aux2.length());
    }
    if (i + 1 == Utils.mensaje.length()) {
        Utils.palabrasMensaje.addElement(aux2.toString());
        aux2.delete(0, aux2.length());
    }
}

g.setColor(0x00000000);
g.fillRect(0, 0, Utils.width, Utils.height);
getPantallaAcerca(g);
break;
case Utils.opcion://opciones
    Utils.cargada = Utils.opcion;
    Utils.linea = 0;
    menuOpIdx = 0;
    removeCommand(salirCommand);
    addCommand(acceptCommand);
    addCommand(selectCommand);

    //                                cargamos el fondo de pantalla del menú

    Utils.cargarImagen(Utils.imOpciones);
    sprite.setImage(Utils.imagen, Utils.anchoIm, Utils.altoIm);
    sprite.setFrameSequence(Utils.getFrames(Utils.imOpciones));

    menuOpIdx = 0;
    //vaciamos el vector opciones
    opciones.removeAllElements();
    //establecemos las opciones
    opciones.addElement(Utils.tiposMenuTxt + ": "
        + midlet.getTipoMenu());
    //espacio entre las opciones
    Utils.spacing = Utils.highFont.getHeight() / 2;

    //Calcula la altura a la que se muestra la primera de las
    //opciones
    //Teniendo en cuenta que hay n-1 opciones en minúsculas
    //1 opción en mayúsculas, y que luego habrá n-1 espacios
```

principal



mismo

la ayuda.

```

Utils.startHeight = ((Utils.lowFont.getHeight() * (opciones.size() - 1))
    + ((opciones.size() - 1) * Utils.spacing);
Utils.startHeight = (Utils.height - Utils.startHeight) / 2;
Utils.startWidth = (Utils.width / 2) - (sprite.getWidth() / 2)
    + Utils.highFont.getSize();

g.setColor(0x00000000);
g.fillRect(0, 0, Utils.width, Utils.height);

getPantallaOpciones(g);
break;

default://En caso de que estemos en alguna de las opciones de la ayuda
//      para las pantallas de ayuda, en todas se hace lo
//
//      aunque sea otro tipo de ayuda, lo que se consigue
//haciendo esto es que piense que cualquier tipo ya está cargado
Utils.cargada = pantalla;
Utils.linea = 0;
removeCommand(selectCommand);
removeCommand(salirCommand);
addCommand(volverCommand);

//Espacio entre cada una de las líneas de la ayuda.
Utils.spacing = Utils.lowFont.getHeight();

//      Calcula la altura y anchura a la que empieza

if (Utils.height <= sprite.getHeight())
    Utils.startHeight = 0;
else
    Utils.startHeight = (Utils.height / 2)
        - (sprite.getHeight() / 2);

if (Utils.width <= sprite.getWidth())
    Utils.startWidth = Utils.lowFont.getSize();
else
    Utils.startWidth = (Utils.width / 2) - (sprite.getWidth() / 2)
        + Utils.lowFont.getSize();

//inicializamos el valor de mensaje, y palabraMensaje
Utils.mensaje.delete(0, Utils.mensaje.length());
Utils.palabrasMensaje.removeAllElements();

if (pantalla == Utils.ayudaTematica)
    Utils.mensaje.append(Utils.txtSubAyuda[0]);
else if (pantalla == Utils.ayudaTeclas)
    Utils.mensaje.append(Utils.txtSubAyuda[1]);
else if (pantalla == Utils.ayudaVideo)
    Utils.mensaje.append(Utils.txtSubAyuda[2]);
else if (pantalla == Utils.ayudaInventario)
    Utils.mensaje.append(Utils.txtSubAyuda[3]);
else if (pantalla == Utils.ayudaMapa)
    Utils.mensaje.append(Utils.txtSubAyuda[4]);
else if (pantalla == Utils.ayudaConversacion)
    Utils.mensaje.append(Utils.txtSubAyuda[5]);
//Hayamos cada una de las palabras del mensaje
StringBuffer palabras = new StringBuffer();
for (int i = 0; i < Utils.mensaje.length(); i++) {
    char temp = Utils.mensaje.charAt(i);

```



```
        if (temp != palabras.charAt(i))
            palabras.append(temp);
        else {
            Utils.palabrasMensaje.addElement(palabras.toString());
            palabras.delete(0, palabras.length());
        }
        if (i + 1 == Utils.mensaje.length()) {
            Utils.palabrasMensaje.addElement(palabras.toString());
            palabras.delete(0, palabras.length());
        }
    }

    g.setColor(0x00000000);
    g.fillRect(0, 0, Utils.width, Utils.height);
    getPantallaSubAyuda(g);
    break;
}

/**
 * Método llamado por el display cuando pone en pantalla este Canvas
 */
protected void paint(Graphics g) {
    g.setColor(0x00000000);
    g.fillRect(0, 0, getWidth(), getHeight());
    switch (Utils.parteMenu) {
        case Utils.menuPrin://Menú principal
            if (Utils.cargada != Utils.menuPrin && midlet.tipoMenu == 1) {
                cargar(g, Utils.menuPrin);
            } else
                getMenuPrincipal(g);
            break;
        case Utils.memoria://Memoria:
            if (Utils.cargada != Utils.memoria)
                cargar(g, Utils.memoria);
            else
                getPantallaMemoria(g);
            break;
        case Utils.ayuda://Ayuda
            if (Utils.cargada != Utils.ayuda)
                cargar(g, Utils.ayuda);
            else
                getPantallaAyuda(g);
            break;
        case Utils.acerca://Acerca de...
            if (Utils.cargada != Utils.acerca)
                cargar(g, Utils.acerca);
            else
                getPantallaAcerca(g);
            break;
        case Utils.opcion://opciones
            if (Utils.cargada != Utils.opcion)
                cargar(g, Utils.opcion);
            else
                getPantallaOpciones(g);
            break;
        default://En caso de que estemos en alguna de las opciones de la ayuda
            if (Utils.cargada != Utils.parteMenu)
                cargar(g, Utils.parteMenu);
            else
    }
```



```
        break;    getPantallaSubAyuda(g);

    }
    repaint();
}

/**
 * Captura la pulsación de teclas en el menú
 */
protected void keyPressed(int code) {
    switch (Utils.parteMenu) {
        case Utils.menuPrin:
            if (getGameAction(code) == Canvas.UP && Utils.menuPrinIdx - 1 >= 0)
                Utils.menuPrinIdx--;
            else if (getGameAction(code) == Canvas.DOWN
                && Utils.menuPrinIdx + 1 < opciones.size()) {
                Utils.menuPrinIdx++;
            } else if (getGameAction(code) == Canvas.FIRE) {
                if (!Utils.jugando)
                    Utils.menuPrinIdx++;
                switch (Utils.menuPrinIdx) {
                    case 0://Seguir Jugando
                        midlet.seguirJugando();
                        break;
                    case 1://Nueva Partida
                        midlet.IniciarPartida();
                        break;
                    case 2://Memoria
                        Utils.parteMenu = Utils.memoria;
                        break;
                    case 3://Opciones
                        Utils.parteMenu = Utils.opcion;
                        break;
                    case 4://Ayuda
                        Utils.parteMenu = Utils.ayuda;
                        break;
                    case 5://Acerca de...
                        Utils.parteMenu = Utils.acerca;
                        break;
                }
                if (!Utils.jugando)
                    Utils.menuPrinIdx--;
            }
            break;
        case Utils.memoria:
            if (getGameAction(code) == Canvas.UP && menuMemIdx - 1 >= 0)
                menuMemIdx--;
            else if (getGameAction(code) == Canvas.DOWN
                && menuMemIdx + 1 < opciones.size()) {
                menuMemIdx++;
            } else if (getGameAction(code) == Canvas.FIRE) {
                if (!Utils.jugando)
                    menuMemIdx++;
                switch (menuMemIdx) {
                    case 0://Guardar Partida
                        if (midlet.puedeGuardar()) {
                            Utils.guardandoPartida = true;
                            midlet.seguirJugando();
                        } else

```

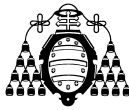


```
midlet
.mostrarAlerta(
    "Mientras en la
partida estés en un video, "

    + "en una conversación, o se este mostrando un mensaje, no se podrá guardar "

    + "la partida", AlertType.INFO);
        break;
    case 1://Cargar Partida
        if (midlet.hayGuardada()) {
            Utils.cargandoPartida = true;
            midlet.IniciarPartida();
        } else
            midlet
                .mostrarAlerta(
                    "No hay ninguna
partida guardada, la cual se pueda cargar.",

                    AlertType.INFO);
                        break;
                    }
                if (!Utils.jugando)
                    menuMemIdx--;
            }
        break;
    case Utils.ayuda:
        if (getGameAction(code) == Canvas.UP && Utils.menuAyudaIdx - 1 >= 0) {
            Utils.menuAyudaIdx--;
        } else if (getGameAction(code) == Canvas.DOWN
            && Utils.menuAyudaIdx + 1 < opciones.size()) {
            Utils.menuAyudaIdx++;
        } else if (getGameAction(code) == Canvas.FIRE) {
            switch (Utils.menuAyudaIdx) {
                case 0://Tematica del juego
                    Utils.parteMenu = Utils.ayudaTematica;
                    break;
                case 1://Teclas generales
                    Utils.parteMenu = Utils.ayudaTeclas;
                    break;
                case 2://En un video
                    Utils.parteMenu = Utils.ayudaVideo;
                    break;
                case 3://En el inventario
                    Utils.parteMenu = Utils.ayudaInventario;
                    break;
                case 4://En el mapa
                    Utils.parteMenu = Utils.ayudaMapa;
                    break;
                case 5://En una conversacion
                    Utils.parteMenu = Utils.ayudaConversacion;
                    break;
            }
        }
        break;
    case Utils.acerca:
        if (getGameAction(code) == Canvas.UP && Utils.linea > 0)
            Utils.linea--;
        else if (getGameAction(code) == Canvas.DOWN
```



```
Utils.numLineas)
    && Utils.numLineasMensaje - Utils.linea >
    Utils.linea++;

    break;
case Utils.opcion://opciones
    if (getGameAction(code) == Canvas.UP && menuOpIdx - 1 >= 0)
        menuOpIdx--;
    else if (getGameAction(code) == Canvas.DOWN
        && menuOpIdx + 1 < opciones.size()) {
        menuOpIdx++;
    } else if (getGameAction(code) == Canvas.FIRE
        || getGameAction(code) == Canvas.RIGHT
        || getGameAction(code) == Canvas.LEFT) {
        switch (menuOpIdx) {
        case 0://Tipo Menu
            switch (menuOpIdx) {
            case 0://Tipo Menu
                if (midlet.tipoMenu == 0)
                    midlet.tipoMenu = 1;
                else
                    midlet.tipoMenu = 0;
                //Actualizamos las opciones a mostrar
                opciones.removeElementAt(0);
                opciones.addElement(Utils.tiposMenuTxt + ": "
                    + midlet.getTipoMenu());
                break;
            }
            break;
        }

        //repaint();
    }
    break;
default://En caso de que estemos en alguna de las opciones de la ayuda
    if (getGameAction(code) == Canvas.UP && Utils.linea > 0) {
        Utils.linea--;
    } else if (getGameAction(code) == Canvas.DOWN
        && Utils.numLineasMensaje - Utils.linea >
Utils.numLineas) {
        Utils.linea++;
    }
    break;
}

}

/**
 * Cuando se pulsa un comando, viene a este metodo.
 */
public void commandAction(Command c, Displayable d) {
    switch (Utils.parteMenu) {
    case Utils.menuPrin://Menú principal
        if (c == selectCommand) {
            if (!Utils.jugando)
                Utils.menuPrinIdx++;
            switch (Utils.menuPrinIdx) {
            case 0://Seguir Jugando
                midlet.seguirJugando();
                break;
            case 1://Nueva Partida
```



```
midlet.IniciarPartida();
break;
case 2://Memoria
    Utils.parteMenu = Utils.memoria;
    break;
case 3://Opciones
    Utils.parteMenu = Utils.opcion;
    break;
case 4://Ayuda
    Utils.parteMenu = Utils.ayuda;
    break;
case 5://Acerca de...
    Utils.parteMenu = Utils.acerca;
    break;
}
if (!Utils.jugando)
    Utils.menuPrinIdx--;
} else if (c == salirCommand) {
    midlet.salir();
}

break;
case Utils.memoria://Memoria
    if (c == selectCommand) {
        if (!Utils.jugando)
            menuMemIdx++;
        switch (menuMemIdx) {
            case 0://Guardar Partida
                if (midlet.puedeGuardar()) {
                    Utils.guardandoPartida = true;
                    midlet.seguirJugando();
                } else
                    midlet
                        .mostrarAlerta(
partida estés en un video, "
                        "Mientras en la
+ "en una conversación, o se este mostrando un mensaje, no se podrá guardar "
+ "la partida", AlertType.INFO);
                    break;
            case 1://Cargar Partida
                if (midlet.hayGuardada()) {
                    Utils.cargandoPartida = true;
                    midlet.IniciarPartida();
                } else
                    midlet
                        .mostrarAlerta(
partida guardada, la cual se pueda cargar.",
                        "No hay ninguna
AlertType.INFO);
                    break;
                }
            if (!Utils.jugando)
                menuMemIdx--;
        }
    } else if (c == volverCommand)
        Utils.parteMenu = Utils.menuPrin;
    break;
```





```
case Utils.ayuda://Ayuda
    if (c == selectCommand) {
        switch (Utils.menuAyudaIdx) {
            case 0://Tematica del juego
                Utils.parteMenu = Utils.ayudaTematica;
                break;
            case 1://Teclas generales
                Utils.parteMenu = Utils.ayudaTeclas;
                break;
            case 2://En un video
                Utils.parteMenu = Utils.ayudaVideo;
                break;
            case 3://En el inventario
                Utils.parteMenu = Utils.ayudaInventario;
                break;
            case 4://En el mapa
                Utils.parteMenu = Utils.ayudaMapa;
                break;
            case 5://En una conversacion
                Utils.parteMenu = Utils.ayudaConversacion;
                break;
        }

    } else if (c == volverCommand) {
        Utils.parteMenu = Utils.menuPrin;
        Utils.menuAyudaIdx = 0;
    }
    break;
case Utils.acerca://Acerca de
    if (c == volverCommand)
        Utils.parteMenu = Utils.menuPrin;
    break;
case Utils.opcion://opciones
    if (c == selectCommand) {
        switch (menuOpIdx) {
            case 0://Tipo Menu
                if (midlet.tipoMenu == 0)
                    midlet.tipoMenu = 1;
                else
                    midlet.tipoMenu = 0;
                //Actualizamos las opciones a mostrar
                opciones.removeElementAt(0);
                opciones.addElement(Utils.tiposMenuTxt + ": "
                    + midlet.getTipoMenu());
                break;
        }
    } else if (c == acceptCommand) {
        Utils.parteMenu = Utils.menuPrin;
        if (midlet.menuActivo == midlet.tipoMenu)
            repaint();
        else {
            midlet.setMenuRs();
            midlet.menu();
        }
    }
    break;
default://En caso de que estemos en alguna de las opciones de la ayuda
    if (c == volverCommand)
        Utils.parteMenu = Utils.ayuda;
    break;
```



```
    }  
}  
  
/**  
 * Reajusta algunas variables como son el ancho y alto de la pantalla. Carga  
 * el numero de letras que caben en el ancho de la pantalla, y el numero de  
 * líneas que caben en el alto de la pantalla  
 */  
public void ajustarVariables() {  
    setFullScreenMode(false);  
  
    Utils.width = getWidth();  
    Utils.height = getHeight();  
  
    //                cargamos el numero de letras que puede tener este grupo en una  
    // pantalla  
    Utils.spaceMen = Utils.getTamGrupoX();  
    switch (Utils.grupo) {  
    case 0://128x128  
        if (Utils.height < Utils.getTamGrupoY())  
            Utils.numLineas = Utils.height / Utils.lowFont.getHeight() - 2;  
        else  
            Utils.numLineas = Utils.getTamGrupoY()  
                / Utils.lowFont.getHeight() - 2;  
        break;  
    case 1://176x200  
        if (Utils.height < Utils.getTamGrupoY())  
            Utils.numLineas = Utils.height / Utils.lowFont.getHeight() - 1;  
        else  
            Utils.numLineas = Utils.getTamGrupoY()  
                / Utils.lowFont.getHeight() - 1;  
        break;  
    }  
}  
}
```

## Utils

```
import java.io.IOException;  
import java.util.Vector;  
import javax.microedition.lcdui.Font;  
import javax.microedition.lcdui.Image;  
  
/**  
 * Entre 2 Almas [http://gamelab.uniovi.es/entre2almas]  
 * -----  
 * PFC para la EUITIO [http://www.euitio.uniovi.es/]  
 * Desarrollado en el GameLab [http://gamelab.mieres.uniovi.es/]  
 * Bajo licencia Freeware  
 * -----  
 *  
 * 2005-2006  
 *
```



```
‡ Clase estática que se encargará de mantener algunas variables comunes a todas
* las clases, y a realizar tareas repetitivas.
*
* @author Pedro Javier Sáez Martínez
* @version 1.0
*/
public class Utils {

    /**
     * Habrá dos tamaños de imágenes, dependiendo del tamaño de pantalla del
     * móvil. Así tendremos 2 grupos: El de 128x128 --> Aquellos móviles con
     * pantallas iguales o superiores a 128x128 --> Su valor int es 0 El de
     * 176x200 --> Aquellos móviles con pantallas iguales o superiores a 176x200
     * --> Su valor int es 1
     */
    static int grupo;

    /**
     * Solo habrá una imagen cargada en cada momento, cada imagen puede contener
     * los frames de muchos sprites, lo cual hace que sean muy grandes y por eso
     * solo se mantiene una cargada en memoria.
     */
    static Image imagen = null;

    /**
     * Además también se guardan sus datos de alto y ancho de cada uno de los
     * frames que la forman.
     */
    static int anchoIm, altoIm = 0;

    /**
     * Y también el nombre que tiene la imagen, para así saber cual tenemos
     * creada y así puede que nos encontremos con casos en los que no es
     * necesarios cargar la imagen porque ya esta cargada
     */
    static StringBuffer nomImagen = new StringBuffer();

    //Array de imágenes que contiene descripciones de imágenes cuyos parámetros
    // son:
    //Antes del primer punto y coma se indicará la ruta de la imagen
    //después se indicará el ancho de cada uno de los frames de la imagen
    //y por ultimo se indicará el alto de cada uno de los frames de la imagen

    /**
     * Array de imágenes que contiene descripciones de imágenes cuyos parámetros
     * son: Antes del primer punto y coma se indicará la ruta de la imagen
     * después se indicará el ancho de cada uno de los frames de la imagen y por
     * ultimo se indicará el alto de cada uno de los frames de la imagen
     */
    static String[] imagenes;

    //imágenes necesarias para el juego.
    static String splash, imMenuPrin;

    static String imOpciones;

    static String imMemoria;

    static String imAyuda;
```



```
static String imAcerca;

static String spCargando;

static String imGuardando;

static String spPuntero;

static String imPunCami;

static String spOscarLado;

static String spOscarEspa;

static String spOscarFren;

static String spOscarCara;

static String imTmpInv;

static String[] apariencias;

// *****//
//
//                                     //
//          Variables estáticas para la formación //
//          de mensajes por pantalla //
//***** //
/**
 * Fuentes que vamos a utilizar en el menú
 */
final static Font lowFont = Font.getFont(Font.FACE_MONOSPACE,
Font.STYLE_PLAIN, Font.SIZE_MEDIUM);

final static Font highFont = Font.getFont(Font.FACE_MONOSPACE,
Font.STYLE_BOLD, Font.SIZE_LARGE);

/**
 * Ancho de lowFont
 */
final static int widthLF = Utils.lowFont.charWidth('a');

/**
 * Establecemos el color de las letras del menú
 */
final static int lowColor = 0xFFFFFFFF;// letras No Iluminada

final static int highColor = 0x00000000;// letra iluminada

/**
 * Altura y anchura a la que se comienzan a imprimir las letras
 */
static int startHeight;

static int startWidth;

/**
 * Marca el espacio que se deja para el nombre de los items en la pantalla
 * de juego, para los móviles de grupo pequeño
```



```
*/
static final int spaceTextItemPeq = 14;

/**
 * Marca el espacio que se deja para el nombre de los items en la pantalla
 * de juego, para los móviles de grupo grande
 */
static final int spaceTextItemGra = 22;

/**
 * Delimita el espacio entre líneas de caracteres
 */
static int spacing;

/**
 * Espacio que se deja para imprimir el mensaje
 */
static int spaceMen;

/**
 * Numero de líneas que caben en una pantalla para el grupo al que pertenece
 * el móvil
 */
static int numLineas;

/**
 * De entre las líneas que forman un mensaje, este atributo define cual se
 * mostrará primero por pantalla
 */
static int linea = 0;

/**
 * Mensaje a imprimir
 */
static StringBuffer mensaje;

/**
 * Numero de líneas que ocupa el mensaje a imprimir
 */
static int numLineasMensaje;

/**
 * las palabras que forman el mensaje a imprimir
 */
static Vector palabrasMensaje;

/**
 * Ancho de la pantalla
 */
static int width;

/**
 * Alto de la pantalla
 */
static int height;

/**
 * Marca el píxel, a partir del cual empieza la coordenada X del cuadro de
 * 128x128 o de 176x200, y este cuadro centrado estará en el móvil. Para
 * establecer su valor se utiliza el primer sprite de tamaño 128x128 o
```



```
    * 176x200 (según grupo) y se aplica la siguiente formula (Utils.width / 2) -  
    * (sprite.getWidth() / 2)  
    */  
    static int inicioPantallaX;  
  
    /**  
    * Marca el píxel, a partir del cual empieza la coordenada X del cuadro de  
    * 128x128 o de 176x200, y este cuadro centrado estará en el móvil. Para  
    * establecer su valor se utiliza el primer sprite de tamaño 128x128 o  
    * 176x200 (según grupo) y se aplica la siguiente formula (Utils.height / 2) -  
    * (sprite.getHeight() / 2)  
    */  
    static int inicioPantallaY;  
  
    //*****  
    //  
    //  
    // Variables estáticas de los menús //  
    //***** //  
    /**  
    * Indica si el juego está en ejecución, lo cual significa que debemos  
    * mostrar el menú con la opción Seguir Jugando.  
    */  
    static boolean jugando = false;  
  
    /**  
    * Indica si se esta cargando una partida guardada con anterioridad.  
    */  
    static boolean cargandoPartida = false;  
  
    /**  
    * Indica si se esta guardando una partida.  
    */  
    static boolean guardandoPartida = false;  
  
    /**  
    * Indica cual de las opciones del menú principal está seleccionada.  
    */  
    static int menuPrinIdx;  
  
    /**  
    * Indica cual de las opciones del menú ayuda está seleccionada.  
    */  
    static int menuAyudaIdx;  
  
    /**  
    * Parte del menú en el que nos encontramos  
    */  
    public static int parteMenu;  
  
    /**  
    * Almacena el identificador de la pantalla que está actualmente guardada  
    */  
    public static int cargada;  
  
    /**  
    * Cada pantalla estará identificada con un entero  
    */  
    public static final int menuPrin = 0;
```



```
public static final int memoria = 1;
public static final int ayuda = 2;

public static final int acerca = 3;

public static final int opcion = 4;

public static final int ayudaTematica = 5;

public static final int ayudaTeclas = 6;

public static final int ayudaVideo = 7;

public static final int ayudaInventario = 8;

public static final int ayudaMapa = 9;

public static final int ayudaConversacion = 10;

public static String titulo;

/**
 * Array que almacenará las opciones del menú principal.
 */
public static String opMenuPrin[] = new String[6];

/**
 * Array que almacenará las opciones del menú memoria.
 */
public static String opMenuMemo[] = new String[2];

/**
 * Array que almacenará las opciones del menú ayuda.
 */
public static String opMenuAyuda[] = new String[6];

/**
 * Array que almacenará el texto de cada una de las sub-ayudas.
 */
public static String txtSubAyuda[] = new String[6];

/**
 * Almacenará el contenido de acerca de...
 */
public static String conteAcerca;

/**
 * Almacena el nombre de los tipos de menú que puede haber - En la posición
 * 0 estará el nombre para el menú de alto nivel, el cual está adaptado a la
 * interfaz nativa de cada móvil. - En la posición 1 estará el nombre para
 * el menú de bajo nivel, es decir, que será igual en todos los móviles,
 * pero es más colorido, más bonito, más vistoso, pero no adaptado a la
 * interfaz nativa de cada móvil
 */
public static String tiposMenu[] = new String[2];

/**
 * Nombre de la opción elegir tipo de menú
 */
```



```
public static String tiposMenuTxt;
//*****
//
//
//          Variables estáticas de la lógica //
//          del juego //
//*****
public static final int reSalir = 0;

public static final int reMensaje = 1;

public static final int reAddInvent = 2;

public static final int reDelInvent = 3;

public static final int reVideo = 4;

public static final int reCombinando = 5;

public static final int reMapa = 6;

public static final int reIrEstancia = 7;

public static final int reConversacion = 8;

public static final int reRespuesta = 9;

public static final int reRespuOscar = 10;

public static final int reJuego = 11;

public static final int reCargaOpSup = 12;

public static final int reCargaOpInf = 13;

public static final int reQuitarOp = 14;

public static final int reQuitarOpYSup = 15;

public static final int reCamActivaOpIt = 16;

public static final int reCamActivaOpEs = 17;

public static final int reCamActivaOpCon = 18;

public static final int reCamFase = 19;

public static final int reEliminarItem = 20;

public static final int reEliminarEstan = 21;

public static final int reCamNombreIt = 22;

public static final int reCamNombreEs = 23;

public static final int reCaminarBlo = 24;

public static final int reSalirJuego = 25;
```





```
public static final int reCamConsecuCom = 26;
public static final int reCamConsecuCon = 27;

public static final int reCamActivaCom = 28;

public static final int reCambiaEscenario = 29;

public static final int reCamConsecuIt = 30;

public static final int reCamConsecuEs = 31;

public static final int reCambiaVisibleIt = 32;

public static final int reCambiaVisibleEs = 33;

public static final int reCaminar = 34;

public static final int reCamMapa = 35;

public static final int reCamApariencia = 36;

public static final int reAddInventSin = 37;

public static int numOps;

public static final int aumentaX = 1375;

public static final int aumentaY = 1631;

/**
 * No se podrá hacer instancias de la clase.
 */
private Utils() {
}

/**
 * Establece el grupo nal que pertenece un móvil, si a los que usaran
 * imágenes con tamaño de 128x128 o a los que usarán las de 176x200
 *
 * @param ancho
 *     Ancho de la pantalla del móvil
 * @param alto
 *     Alto de la pantalla del móvil
 */
static public void getGrupo(int ancho, int alto) {
    if (ancho >= 128 && ancho < 176 && alto >= 128 && alto < 200)
        grupo = 0;
    else if (ancho >= 176 && alto >= 200)
        grupo = 1;
}

/**
 * Nos muestra el ancho de pantalla que le corresponde al grupo
 *
 * @return el ancho de pantalla para el grupo
 */
static public int getTamGrupoX() {
    if (grupo == 0)
        return 128;
```



```
}        return 176;//si es de grupo 1...

/**
 * Nos muestra el alto de pantalla que le corresponde al grupo
 *
 * @return el alto de pantalla para el grupo
 */
static public int getTamGrupoY() {
    if (grupo == 0)
        return 128;
    return 200;//si es de grupo 1...
}

/**
 * Este método devolverá la secuencia de frames que forman el objeto pedido.
 *
 * @param peticion
 *        el sprite que se pide.
 * @return secuencia de frames que forman el objeto pedido
 */
static public int[] getFrames(String peticion) {
    int numFrames = getNumFrames(peticion);
    int[] frames = new int[numFrames];
    int valorInicial = getFrameInicial(peticion);
    for (int i = 0; i < numFrames; i++)
        frames[i] = valorInicial + i;
    return frames;
}

/**
 * Devuelve el frame inicial de un Sprite
 *
 * @param peticion
 *        el Sprite del que se quiere sacar el frame inicial
 * @return el frame inicial de un SPrte
 */
static public int getFrameInicial(String peticion) {
    int numPunCom = 0;
    //Utilizamos StringBuffer para evitar trabajo extra al recolector
    // de basura
    StringBuffer aux = new StringBuffer();
    for (int i = 0; numPunCom < 2; i++) {
        if (peticion.charAt(i) == ';')
            numPunCom++;
        else if (numPunCom == 1)
            aux.append(peticion.charAt(i));
    }
    //Con el toString transformamos el StringBuffer en una cadena de
    //caracteres.
    return (int) Integer.parseInt(aux.toString());
}

/**
 * Devuelve el numero de frames para una petición determinada
 *
 * @param peticion
 *        Sprite del que se quieren saber los frames
 * @return el numero de frames que tiene ese Sprite
 */
```



```
static public int getNumFrames(String peticion) {
    int numPunCom = 0;
    int aux = 0;
    for (int i = 0; numPunCom < 2; i++) {
        if (peticion.charAt(i) == ';')
            numPunCom++;
        aux = i;
    }
    return (int) Integer.parseInt(peticion.substring(aux + 1));
}

/**
 * Devuelve los datos de la imagen en la que está almacenada la petición o
 * Sprite
 *
 * @param peticion
 *      Sprite del que se quiere saber la imagen
 * @return los datos de la imagen en el que esta el Sprite
 */
static private String getDatosImagen(String peticion) {
    int auxId = 0;
    while (peticion.charAt(auxId) != ';') {
        auxId++;
    }
    auxId = (int) Integer.parseInt(peticion.substring(0, auxId));
    for (int i = 0; i < imagenes.length; i++) {
        if (getId(imagenes[i]) == auxId) {
            auxId = i;
            i = imagenes.length;
        }
    }
    return imagenes[auxId + grupo];
}

/**
 * Devuelve el identificador de una imagen
 *
 * @param datosImagen
 *      datos de la imagen de los que se quiere extraer el
 *      identificador
 * @return identificador la imagen
 */
static private int getId(String datosImagen) {
    int aux = 0;
    while (datosImagen.charAt(aux) != ';') {
        aux++;
    }
    return Integer.parseInt(datosImagen.substring(0, aux));
}

/**
 * Devuelve la ruta de una imagen
 *
 * @param nombreImagen
 *      datos de la imagen de los que se quiere extraer la ruta
 * @return ruta de directorios donde está la imagen
 */
static private String getRuta(String nombreImagen) {
    int numPunCom = 0;
```



```
//Utilizamos StringBuffer para evitar trabajo extra al recolector
// de basura
StringBuffer aux = new StringBuffer();
for (int i = 0; numPunCom < 2; i++) {
    if (nombreImagen.charAt(i) == ';')
        numPunCom++;
    else if (numPunCom == 1)
        aux.append(nombreImagen.charAt(i));
}
//Con el toString transformamos el StringBuffer en una cadena de
//caracteres.
return aux.toString();
}

/**
 * Devuelve el ancho de una cada uno de los frame de una imagen
 *
 * @param nombreImagen
 *      Datos de la imagen.
 * @return EL ancho de cada uno de los frames de una imagen
 */
static private int getAncho(String nombreImagen) {
    int numPunCom = 0;
    //Utilizamos StringBuffer para evitar trabajo extra al recolector
    // de basura
    StringBuffer aux = new StringBuffer();
    for (int i = 0; numPunCom < 3; i++) {
        if (nombreImagen.charAt(i) == ';')
            numPunCom++;
        else if (numPunCom == 2)
            aux.append(nombreImagen.charAt(i));
    }
    //Con el toString transformamos el StringBuffer en una cadena de
    //caracteres.
    return Integer.parseInt(aux.toString());
}

/**
 * Devuelve el alto de una cada uno de los frame de una imagen
 *
 * @param nombreImagen
 *      Datos de la imagen.
 * @return EL alto de cada uno de los frames de una imagen
 */
static private int getAlto(String nombreImagen) {
    int numPunCom = 0;
    int aux = 0;
    for (int i = 0; numPunCom < 3; i++) {
        if (nombreImagen.charAt(i) == ';')
            numPunCom++;
        aux = i;
    }
    return (int) Integer.parseInt(nombreImagen.substring(aux + 1));
}

/**
 * Carga los datos mas importante de una imagen en la que está el Sprite
 * solicitado
 */
```



```

    * @param nomSprite
    *     nombre del sprite cuya imagen el la que está contenido se
    *     quiere cargar
    */
static public void cargarImagen(String nomSprite) {
    String datosImagen = getDatosImagen(nomSprite);
    if (!Utils.creada(datosImagen)) {
        nomImagen.delete(0, nomImagen.length());
        nomImagen.append(datosImagen);
        imagen = createImage(Utils.getRuta(datosImagen));
        anchoIm = getAncho(datosImagen);
        altoIm = getAlto(datosImagen);
    }
}

/**
 * Indica si una imagen (que se necesita cargar para un sprite) ya esta
 * creada. Esto puede pasar porque puede ser que el sprite que se cargo
 * anteriormente a este, estaba contenido en la misma imagen.
 */
* @param nombre
*     Nombre de la imagen que se quiere cargar
* @return si esta creada o no
*/
static private boolean creada(String nombre) {
    boolean creada = false;

    if (nomImagen.toString().equals(nombre)) {
        creada = true;
    }
    return creada;
}

/**
 * Método que devuelve un objeto Image a partir de una imagen PNG, la cual
 * tiene que estar en la carpeta res.
 */
* @param nombre
*     Nombre de la imagen en la carpeta res
* @return Image con la imagen ya creada.
*/
static private Image createImage(String nombre) {
    try {
        return Image.createImage(nombre);
    } catch (IOException e) {
        System.out.println("error de entrada y salida");
    }
    System.gc();
    return null;
}

/**
 * Devuelve el espacio reservado para el nombre de los Items en la pantalla
 * de juego, teniendo en cuenta si el móvil es de un grupo u otro
 */
* @return El espacio reservado para el nombre de los Items en la pantalla
*     de juego.
*/
static public int getSpaceTextItem() {
    if (grupo == 0)

```



```
        else    return spaceTextItemPeq;
               return spaceTextItemGra;
    }

    /**
     * Actualiza a la apariencia activa del personaje principal.
     *
     * @param activa
     *      La apariencia activa del personaje principal
     */
    static public void actualizarApariencia(int activa) {
        activa = activa * 4;
        Utils.spOscarLado = Utils.apariencias[activa];
        activa++;
        Utils.spOscarEspa = Utils.apariencias[activa];
        activa++;
        Utils.spOscarFren = Utils.apariencias[activa];
        activa++;
        Utils.spOscarCara = Utils.apariencias[activa];
    }

    /**
     * Devuelve el identificador que corresponde a un determina resultado por
     * ejemplo a Salir le corresponde reSalir
     *
     * @param resultado
     *      String con el nombre del resultado.
     * @return Identificador que corresponde a una determinado resultado
     */
    public static int getIdResultado(String resultado) {
        if (resultado.equals("Salir"))
            return reSalir;
        if (resultado.equalsIgnoreCase("Mensaje"))
            return reMensaje;
        if (resultado.equalsIgnoreCase("Video"))
            return reVideo;
        if (resultado.equalsIgnoreCase("Mapa"))
            return reMapa;
        if (resultado.equalsIgnoreCase("Conversacion"))
            return reConversacion;
        if (resultado.equalsIgnoreCase("Juego"))
            return reJuego;
        if (resultado.equalsIgnoreCase("Respuesta"))
            return reRespuesta;
        if (resultado.equalsIgnoreCase("RespuOscar"))
            return reRespuOscar;
        if (resultado.equalsIgnoreCase("cargaOpSup"))
            return reCargaOpSup;
        if (resultado.equalsIgnoreCase("cargaOpInf"))
            return reCargaOpInf;
        if (resultado.equalsIgnoreCase("quitarOp"))
            return reQuitarOp;
        if (resultado.equalsIgnoreCase("quitarOpYSup"))
            return reQuitarOpYSup;
        if (resultado.equalsIgnoreCase("salirJuego"))
            return reSalirJuego;
        if (resultado.equalsIgnoreCase("Caminar"))
            return reCaminar;
        if (resultado.equalsIgnoreCase("CaminarBlo"))
```



```
return reCamivarBl;  
if (resultado.equalsIgnoreCase("CambiaEscenario"))  
    return reCambiaEscenario;  
if (resultado.equalsIgnoreCase("camMapa"))  
    return reCamMapa;  
if (resultado.equalsIgnoreCase("irEstancia"))  
    return reIrEstancia;  
if (resultado.equalsIgnoreCase("combinando"))  
    return reCombinando;  
if (resultado.equalsIgnoreCase("camFase"))  
    return reCamFase;  
if (resultado.equalsIgnoreCase("addInventario"))  
    return reAddInvent;  
if (resultado.equalsIgnoreCase("addInventarioSin"))  
    return reAddInventSin;  
if (resultado.equalsIgnoreCase("delInventario"))  
    return reDellInvent;  
if (resultado.equalsIgnoreCase("camActivaOpIt"))  
    return reCamActivaOpIt;  
if (resultado.equalsIgnoreCase("camActivaOpEs"))  
    return reCamActivaOpEs;  
if (resultado.equalsIgnoreCase("camActivaOpCon"))  
    return reCamActivaOpCon;  
if (resultado.equalsIgnoreCase("eliminarItem"))  
    return reEliminarItem;  
if (resultado.equalsIgnoreCase("eliminarEstan"))  
    return reEliminarEstan;  
if (resultado.equalsIgnoreCase("camNombreIt"))  
    return reCamNombreIt;  
if (resultado.equalsIgnoreCase("camNombreEs"))  
    return reCamNombreEs;  
if (resultado.equalsIgnoreCase("cambiaVisibleIt"))  
    return reCambiaVisibleIt;  
if (resultado.equalsIgnoreCase("cambiaVisibleEs"))  
    return reCambiaVisibleEs;  
if (resultado.equalsIgnoreCase("camConsecuIt"))  
    return reCamConsecuIt;  
if (resultado.equalsIgnoreCase("camConsecuEs"))  
    return reCamConsecuEs;  
if (resultado.equalsIgnoreCase("camConsecuCom"))  
    return reCamConsecuCom;  
if (resultado.equalsIgnoreCase("camConsecuCon"))  
    return reCamConsecuCon;  
if (resultado.equalsIgnoreCase("camActivaCom"))  
    return reCamActivaCom;  
if (resultado.equalsIgnoreCase("camApariencia"))  
    return reCamApariencia;  
return -1;  
}  
}
```

## Video

```
import java.util.Vector;
```



```
/** Entre 2 Almas                                     [http://gamelab.uniovi.es/entre2almas]
 * -----
 * PFC para la EUITIO                                [http://www.euitio.uniovi.es/]
 * Desarrollado en el GameLab                         [http://gamelab.mieres.uniovi.es/]
 * Bajo licencia Freeware
 * -----
 *                                                     2005-2006
 *
 *
 * Clase Video. Implementa el interfaz Comparable.
 *
 * @author Pedro Javier Sáez Martínez
 * @version 1.0
 */
public class Video implements Comparable {

    /**
     * Identificador del Video
     */
    int id;

    /**
     * Nombre del conjunto de viñetas al que hay que llamar.
     */
    String vinyeta;

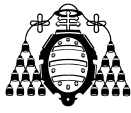
    /**
     * Texto a mostrar en cada una de las viñetas
     */
    Object[] texto;

    /**
     * Indica a la parte de juego a la que se pasa después del video
     */
    int sigParte;

    /**
     * COnstructor de la clase Video
     *
     * @param id
     *         Identificador del video.
     * @param vinyeta
     *         conjunto de viñetas al que hay que llamar.
     * @param texto
     *         Texto a mostrar en cada una de las viñetas
     * @param sigParte
     *         parte de juego a la que se pasa después del video
     */
    public Video(int id, String vinyeta, Vector texto, int sigParte) {
        this.id = id;
        this.texto = new Object[texto.size()];
        this.vinyeta = vinyeta;
        this.sigParte = sigParte;
        texto.copyInto(this.texto);
    }

    /**
     * Devuelve la clave del comparable
     */
}
```





```
*/@return La clave del comparable identificador del comparable.  
public int getKey() {  
    return id;  
}  
  
}
```