UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

DESARROLLO DE UN VIDEOJUEGO MULTIJUGADOR EN PERSPECTIVA ISOMÉTRICA PARA DISPOSITIVOS MÓVILES CON BLUETOOTH

INGENIERÍA DE TELECOMUNICACIÓN

MÁLAGA, 2008 JOSÉ MANUEL MARTÍN TEJERO

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSIDAD DE MÁLAGA

Titulación: Ingeniería de Telecomunicación

Reunido el tribunal examinador en el día de la fecha, constituido por:

D./Dª. _____

D./Dª. _____

D./Dª. _____

para juzgar el Proyecto Fin de Carrera titulado:

DESARROLLO DE UN VIDEOJUEGO MULTIJUGADOR EN PERSPECTIVA ISOMÉTRICA PARA DISPOSITIVOS MÓVILES CON BLUETOOTH

del alumno D./Dª. José Manuel Martín Tejero

dirigido por D./Dª. Francisco Javier González Cañete

ACORDÓ POR _____ OTORGAR LA

CALIFICACIÓN DE _____

Y, para que conste, se extiende firmada por los componentes del tribunal, la presente diligencia

Málaga, a _____ de _____ de _____

El Presidente El Vocal El Secretario

Fdo.: _____ Fdo.: _____ Fdo.: _____

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSIDAD DE MÁLAGA

DESARROLLO DE UN VIDEOJUEGO MULTIJUGADOR EN PERSPECTIVA ISOMÉTRICA PARA DISPOSITIVOS MÓVILES CON BLUETOOTH

REALIZADO POR:

José Manuel Martín Tejero

DIRIGIDO POR:

Francisco Javier González Cañete

DEPARTAMENTO DE: Tecnología Electrónica

TITULACIÓN: Ingeniería de Telecomunicación

PALABRAS CLAVE: Aplicaciones móviles, videojuego, J2ME, XML, perspectiva isométrica, Bluetooth

RESUMEN:

El proyecto consiste en el desarrollo de una aplicación de videojuego en perspectiva isométrica para dispositivos móviles. Se trata de una aventura gráfica en la que el jugador controla a un prisionero de guerra recluido en un campo de concentración alemán durante la Segunda Guerra Mundial. Todo el campo está rodeado por vallas y muros, con soldados patrullando. El objetivo del juego es conseguir que el prisionero se fugue del campo de concentración. El jugador se servirá de diferentes objetos repartidos por el campo de concentración, así como de túneles, para alcanzar su meta. Está realizado en base a la tecnología J2ME. Contempla el diseño de un modo multijugador mediante Bluetooth. Además, la aplicación hace uso de ficheros de configuración XML donde se configura gran parte de la misma: escenarios, objetos, personajes y rutinas temporales y de movimientos en el juego.

Málaga, Septiembre de 2008

Agradecimientos

Quiero agradecer a mi tutor de proyecto fin de carrera D. Francisco Javier González Cañete por su inestimable ayuda, guía y colaboración en la realización de este proyecto, sin ella este proyecto no hubiera llegado a realizarse. Así también, quiero agradecer a todos mis amigos de la carrera, especialmente a Esther y a Pablo, por su compañía y su inestimable amistad. Agradecer a los profesores de la E. T. S. de Ingeniería de Telecomunicación de Málaga que con su dedicación me acompañaron durante estos años. Todos ellos han conformado en cierta manera como una segunda familia en este tiempo que concluye.

Índice general

Capítulo 1. Introducción

Capítulo 2. Descripción de la aplicación

Capítulo 3. Diseño lógico de la aplicación

Índice de figuras

Capítulo 1.
Introducción

1.1 Visión general

En la sociedad moderna el papel de la ingeniería es proporcionar sistemas y productos que mejoren los aspectos materiales de la vida humana, para que así la vida sea más fácil, segura y placentera.

En cuanto a la labor de la ingeniería, no debería olvidarse esta idea. Se deberían promover sistemas que llenen el ocio de las personas y hagan la vida más placentera; sistemas como los videojuegos, que, en la actualidad, proporcionan entretenimiento y evasión a millones de personas por todo el mundo.

La industria de los videojuegos ha tenido en los últimos tiempos un crecimiento deslumbrante. Los avances tecnológicos en campos como la microelectrónica, que han permitido la producción de microprocesadores más baratos y con mayores prestaciones de cómputo; el empleo generalizado de metodologías de Ingeniería Software, que, mediante la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del software, conducen a aplicaciones de alta calidad; y las tecnologías multimedia en general, que cada vez tienen mayor presencia en la sociedad de la información, entre otras bases, han creado un marco perfecto para el despegue de esta industria.

Desde el primer videojuego de la historia *Spacewar!* desarrollado por S. Russell (MIT) en 1961, hasta nuestros días, la evolución de los videojuegos ha sido constante. Hoy en día, se tienen videojuegos que aprovechan sorprendentemente las enormes capacidades gráficas de las máquinas sobre las que corren. Es más, existe una gran variedad de equipos que permiten la ejecución de videojuegos: desde los dispositivos de carácter más general, como pueden ser ordenadores, teléfonos móviles o PDA´s, hasta dispositivos de carácter más específico, como son las videoconsolas. Nombres como Wii, PlayStation o Xbox nos dan idea de la importancia económica que está alcanzando el mundo del entretenimiento basado en videojuegos. A este respecto, cabe destacar cómo la industria de los videojuegos genera en los últimos tiempos más beneficios incluso que la industria del cine.

Este proyecto se centra en un ámbito concreto del conjunto de dispositivos que permiten a los videojuegos desarrollarse: los dispositivos móviles. Cada día los dispositivos móviles presentan mayores capacidades e integran mayor número de funcionalidades de equipos de orden mayor como los ordenadores. De esta forma, no es de extrañar la presencia en ellos de videojuegos.

Por otro lado, cabe destacar que el potencial de jugadores de videojuegos para dispositivos móviles en todo el mundo se cifraba en 2006 en torno a 150 millones [1]. Y teniendo en cuenta, además, la aparición de los mercados emergentes de Asia como India o China, se estima, que se podría llegar a alcanzar la cifra de 265 millones de jugadores en 2010[1]. Esto da idea de las grandes posibilidades que presenta el campo de los videojuegos para dispositivos móviles.

No obstante, hay que resaltar que gran parte de este mercado se lo reparten los grandes proveedores de contenidos. Empresas como THQ [2], EA [3], Glu [4], Capcom, [5] Gameloft [6], etc, son las que dominan el mercado a nivel europeo. Y luego están las operadoras de telefonía que se encargan de la distribución por su red de los contenidos, y que también participan de los beneficios que generan los videojuegos. Así, no se nos debe escapar que el mercado de los videojuegos para dispositivos móviles es un mercado cada vez más competitivo, en el que grandes empresas pugnan entre sí.

A continuación, daremos una breve visión de las distintas tecnologías más conocidas que permiten el desarrollo de los videojuegos para los dispositivos móviles.

1.2 Diferentes tecnologías

Los juegos para dispositivos móviles son desarrollados utilizando tecnologías como J2ME (*Java 2 Micro Edition*) de Sun Microsystems [7], Brew (*Binary Runtime for Wireless*) de Qualcomm [8] o ExEn (*Execution Enviorement*) de In Fusio [9]. Estas plataformas constituyen capas que se colocan por encima del SO (Sistema Operativo) del dispositivo móvil. Capas que proporcionan características de programación de alto nivel, fácilmente accesibles, así como, una elevada portabilidad de las aplicaciones.

También se da la posibilidad de desarrollar estas aplicaciones directamente sobre la plataforma operativa que proporciona el teléfono móvil, como puede ser desarrollar los juegos en base a tecnologías de SO como Symbian [10]; si bien, este enfoque suele tener una limitación en cuanto a la portabilidad, por lo que el preferido por la industria es el primero, de utilización de una capa que permita alto grado de portabilidad.

Dentro del enfoque preferido por la industria, consistente en el empleo de una capa intermedia, en el desarrollo de este proyecto se optó por el uso de la plataforma de Sun Microsystems: J2ME. Sin embargo, se pueden destacar algunas características de las otras plataformas competidoras de capa intermedia.

Brew de Qualcomm se puede ver como el principal competidor de J2ME en el ámbito de desarrollo de videojuegos para dispositivos móviles. Consiste en una plataforma software que permite la ejecución de programas de juegos, de intercambio de fotos, de envío de mensajes, etc.

La principal ventaja de Brew es la portabilidad de las aplicaciones entre los dispositivos Qualcomm. Brew constituye una capa intermedia entre el nivel de aplicación y el SO del dispositivo, al igual que ocurre con J2ME. La API de Brew permite el desarrollo de aplicaciones software en C/C++ y JAVA. Brew puede verse, en cierta manera, también, como un pseudo-SO.

El principal inconveniente que presenta Brew es su *proceso software*. Existe un SDK que permite el desarrollo y la simulación de las aplicaciones. Sin embargo, no se permite como en J2ME la carga de la aplicación al dispositivo *in situ* para probarla. Ya que Brew da un control completo sobre el dispositivo móvil, las aplicaciones Brew deben ser digitalmente firmadas para que puedan ser ejecutadas en los dispositivos. Y sólo las empresas proveedoras de contenidos y los desarrolladores autenticados por Brew tienen acceso al software que produce las firmas digitales. Otro inconveniente, que se suma, es que el simulador que presenta el SDK de Brew no emula al dispositivo móvil, por lo que la depuración para dispositivos físicos resulta más complicada.

La otra competidora de J2ME, ExEn, posee también un enfoque de máquina virtual igual que J2ME. Es más, ExEn se ha diseñado buscando la compatibilidad con los *standards* J2ME: MIDP/CLDC (*Mobile Information Device Profile / Connected Limited Device Configuration*). Esta tecnología constituye un campo de trabajo diseñado y optimizado para el desarrollo de juegos, tomando como base el lenguaje Java. Además, dada su arquitectura basada en los

standards J2ME, la máquina virtual de ExEn proporciona altas prestaciones de seguridad para los dispositivos móviles sobre los que se ejecuta.

Frente a las plataformas de Brew y ExEn, se tiene la alternativa de J2ME de Sun Microsystems. En este proyecto se optó por la utilización de esta plataforma, debido a que las aplicaciones se pueden emular fácilmente en un PC durante la fase de desarrollo, utilizando emuladores proporcionados gratuitamente por Sun y siendo, además, fácil la carga de la aplicación al teléfono para realizar *tests* sobre el dispositivo físico.

Otras ventajas que se consiguen al utilizar tecnología Java en el desarrollo de aplicaciones o videojuegos son la ya comentada portabilidad y el soporte, ya que la máquina virtual de Java se incorpora casi en la totalidad de los dispositivos móviles actuales y existe una amplia documentación, de libre acceso, referente al tema por toda la red.

1.3 Tecnología JAVA

El lenguaje de programación JAVA apareció a comienzos de la década de los 90. Sun Microsystems desarrolló un lenguaje de gran robustez, que buscaba la independencia con respecto de la plataforma hardware en la que se ejecutaba. En un principio, sus objetivos se centraban en el desarrollo de software para diversos dispositivos físicos como controladores remotos o electrodomésticos. Sin embargo, estas características de robustez e independencia de plataforma llevaron a JAVA a tener un amplio desarrollo en los siguientes años a su aparición.

Las características más destacadas de JAVA son: su simpleza, diseñado para que los programadores lo pudieran aprender fácilmente, con sintaxis heredada de C/C++ y muchas de sus características de programación orientada a objetos; su robustez, el código se comprueba al precompilar y en tiempo de ejecución, proporcionando una gestión de excepciones orientada a objetos, además de una gestión interna de la memoria tanto al reservarla como liberarla (el “recolector de basura”); el enlace dinámico, un programa Java posee cierta información que facilita la resolución de accesos a objetos en tiempo de ejecución; su arquitectura neutral, un programa Java puede ser compilado en cualquier plataforma donde exista un compilador Java, de modo que el código binario resultante puede ejecutarse sobre cualquier máquina virtual Java; el que sea interpretado y de alto rendimiento, el código binario intermedio fue diseñado cuidadosamente, de manera que fuese sencillo traducirlo a código máquina nativo con un alto

rendimiento; y por último, cabe destacar, que fue diseñado siguiendo el paradigma de la programación orientada a objetos, multihilo y distribuido.

En la explosión tecnológica experimentada a partir del segundo lustro de la década de los 90 –principalmente debida a Internet- JAVA fue diversificándose. Aparecieron diferentes ediciones del lenguaje, en función de los ámbitos en los que la tecnología de Sun se iba asentando. Estos ámbitos de actuación del lenguaje fueron: J2SE (*Java 2 Standard Edition*) orientada al desarrollo de aplicaciones independientes y germen inicial de la tecnología JAVA, J2EE (*Java 2 Enterprise Edition*) enfocada al entorno empresarial, que tiene como núcleo a J2SE, y J2ME (*Java 2 Micro Edition*) orientada a la programación para pequeños dispositivos, un subconjunto de J2SE con algunas funcionalidades adicionales.

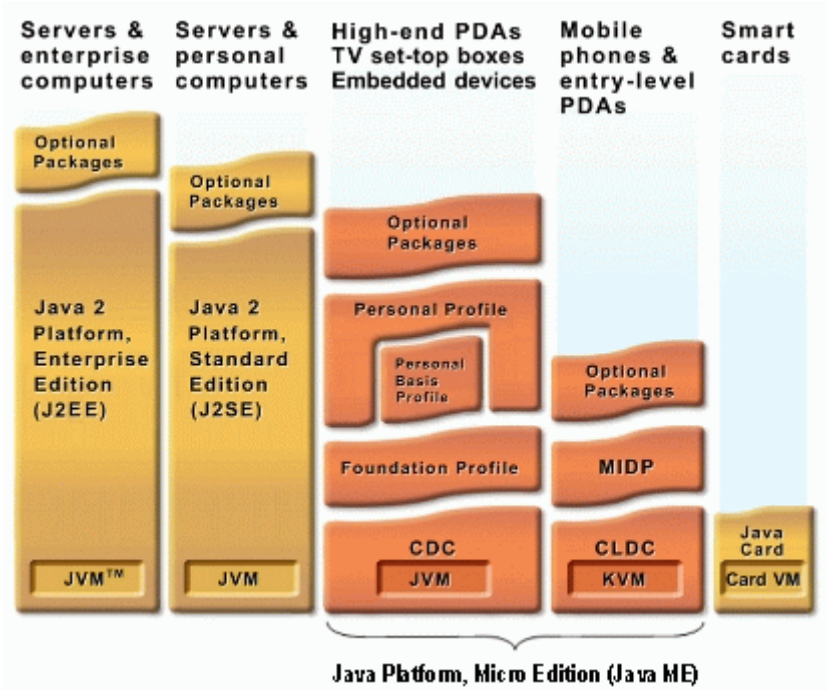


Figura 1.1 Arquitectura de la plataforma de Java.

Referirse a JAVA como un simple lenguaje de programación no se ajusta enteramente a la realidad. JAVA debe verse más como un conjunto de tecnologías que intenta abarcar todos los ámbitos de la computación, como se desprende de la **Figura 1.1**. A través de estos diferentes ámbitos se dan dos constantes que constituyen la base fundamental de la tecnología:

- El código se precompila a un código intermedio (*Bytecode*, el código máquina de la JVM) que puede ser lanzado sobre cualquier máquina virtual, independiente de la plataforma hardware en la que ésta se encuentre (véase **Figura 1.2**).

Figura 1.2 Esquema del proceso de despliegue de una aplicación Java.

- La existencia de un conjunto de API´s básicas y compartidas, las cuales constituyen una librería de componentes que proporcionan útiles recursos de programación y que junto con la máquina virtual de Java aíslan la aplicación Java del hardware y SO que subyace bajo ella como se muestra en la **Figura 1.3**.

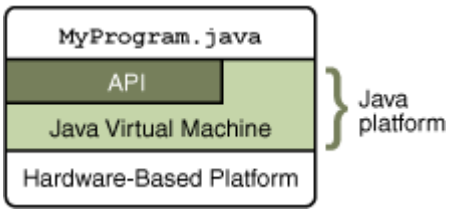


Figura 1.3 Aislamiento entre programa Java y hardware mediante API y JVM.

1.4 J2ME

El auge de todo lo relacionado con la World Wide Web e Internet produjo que gran parte del esfuerzo de los desarrolladores Java se centrara en los pilares de J2SE y J2EE, más enfocados al entorno empresarial y de intercambio de información a través de Internet, dejando un poco aparte a J2ME. Esta tendencia está cambiando debido, sobre todo, al interés creciente que las aplicaciones, como los videojuegos, para dispositivos móviles están creando.

J2ME apareció en 1999. Sun buscaba, con esta edición de su lenguaje, el despliegue de aplicaciones Java en dispositivos físicos pequeños. Se enfocó a dispositivos electrónicos con capacidades computacionales y gráficas muy reducidas, tales como teléfonos móviles, PDAs o electrodomésticos inteligentes.

Esta edición posee unos componentes básicos que la diferencian de las otras versiones, como son: el uso de una máquina virtual denominada KVM (*Kilo Virtual Machine*, debido a

que requiere sólo unos pocos Kilobytes de memoria para funcionar) en vez del uso de la JVM clásica y la inclusión de un pequeño y rápido recolector de basura.

La plataforma J2ME se divide en dos ámbitos de actuación concretos. Estos ámbitos se conocen por el nombre de configuraciones. Estas configuraciones son: la CDC (*Connected Device Configuration*) encaminada a dispositivos con una capacidad de cómputo alta y que posee una máquina virtual prácticamente similar a la de J2SE, salvo por las limitaciones gráficas y de memoria del dispositivo; y la CLDC (*Connected Limited Device Configuration*), que está orientada a dispositivos de menor capacidad que los que utilizan la CDC. En concreto, la que nos interesa para los dispositivos móviles es CLDC.

Un aspecto, a destacar en esta introducción, de los dispositivos móviles sobre los que se implanta la plataforma J2ME, una vez conocida la configuración sobre la que se desarrolla, es el SO que subyace bajo ella. En el siguiente punto se abordará una breve visión acerca de la situación actual de los SO´s para dispositivos móviles.

1.5 Sistemas Operativos Para Dispositivos Móviles

Existen diversos Sistemas Operativos para dispositivos móviles. Estos van desde los que se ocupan de dispositivos físicos de mayor peso como Tablet PC´s o PDA´s, hasta los que se encargan de dispositivos menores como teléfonos móviles, *smartphones* o Blackberries. El interés de este proyecto se centra en los dispositivos menores, concretamente, en el abanico de teléfonos móviles de alta gama que llega hasta los *smartphones* incluidos.

En este ámbito de actuación, se debe citar nombres de SO´s como Symbian, Windows Mobile [11], Palm OS [12], iPhone OS [13], en cuanto a los más relevantes. Éstos son SO´s propietarios. Frente a ellos, se tiene, también, la posibilidad de recurrir a SO´s de código abierto, basados en la plataforma Linux, como Openmoko [14], Acces Linux Platform [15] o QTopia [16]. Si bien hay que reconocer que éstos últimos tienen una relevancia menor en el mercado.

Centrándonos en los SO´s más relevantes y dejando un poco al margen las alternativas de Software Libre, se pueden destacar algunas características de los SO´s, para teléfonos móviles, más relevantes.

Symbian, participado por grandes empresas fabricantes de teléfonos móviles como Nokia, Ericcson, Panasonic, Siemens AG y Samsung. Es el SO líder en el segmento de los teléfonos móviles de alta gama o *smartphones*. Hablar de Symbian es hablar de la alternativa dada por Nokia y otros fabricantes a la irrupción en el mercado de dispositivos móviles de Microsoft con Windows Mobile. Se puede ver como el intento de los fabricantes de teléfonos móviles de controlar toda la cadena de valor que originan sus dispositivos.

Symbian corre exclusivamente sobre procesadores ARM (*Acorn or Advanced RISC Machine,* una arquitectura de procesador de 32 bits muy extendida en sistemas embebidos de bajo consumo como los teléfonos móviles) [18]. El diseño de Symbian se realizó siguiendo los principios de: la integridad y seguridad de los datos de usuario son prioridad, el tiempo del usuario no debe desperdiciarse y, sobre todo, todos los recursos en el sistema son escasos. Symbian se basa en una estructura de microkernel, optimizado para dispositivos de bajo consumo y dispositivos basados en memorias ROM. Tanto el SO como las aplicaciones Symbian siguen un modelo de diseño orientado a objetos: MVC (*Modelo de datos Vista Controlador*).

Un aspecto importante a destacar y que tiene relevancia para este proyecto es el hecho de que Symbian implementa el *standard* Java: J2ME. Por este motivo, y dada la relevancia de Symbian en el mundo de los dispositivos móviles de tipo *smartphone*, en este proyecto, se opta por tener como base al SO Symbian. Posteriormente, se concretará un poco más la base sobre la que se desarrollará la aplicación, especificándose una familia de dispositivos con el SO Symbian, sobre la que el diseño de la aplicación se centrará.

Como alternativa principal a Symbian se tiene a Windows Mobile de Microsoft. Este SO se basa en la API Win 32 de Microsoft. En Windows Mobile se incorporan las aplicaciones de escritorio típicas de Microsoft, pero adaptadas a los *smartphones*: Office Mobile, Outlook Mobile, Windows Media Player, etc. Microsoft busca que, a través de su SO, los dispositivos móviles de tipo *smartphone* trabajen como un teléfono y, además, como un dispositivo de datos.

Microsoft implantó inicialmente su SO en el segmento de los *Pocket PC´s* con y sin capacidades de teléfono móvil, abordando posteriormente a los *smartphones*. Para Microsoft un *smartphone* es un dispositivo hardware con unas características de diseño que lo diferencian de los *Pocket PC´s* como que no posee pantalla táctil, que su diseño está pensado para que pueda ser utilizado fácilmente con una sola mano y que la resolución de su pantalla es menor que la de los *Pocket PC´s*. Para este tipo de dispositivos Microsoft adaptó el inicial SO Pocket PC 2002, dando lugar a la familia de los Windows Mobile.

1.6 Fijación del entorno de trabajo

El enfoque de utilización de la capa intermedia J2ME, que aísle a la aplicación del sistema que subyace bajo ella, permite obviar las dificultades de desarrollo que implicaría el buscar una aplicación altamente *portable*, dentro de la gran variedad de fabricantes de teléfonos móviles y con los respectivos SO´s. No se debe olvidar que esta facilidad se consigue gracias a la máquina virtual de Java que cada dispositivo móvil lleva para lanzar los programas J2ME. Por lo tanto, la portabilidad de la aplicación está condicionada a la implementación que el fabricante del dispositivo móvil haga de la máquina virtual, así como, de las API´s de Sun, además de las capacidades de memoria y procesado propias del teléfono móvil.

En este sentido, aun cuando la utilización de J2ME debiera garantizar un alto grado de portabilidad de las aplicaciones, en la práctica, conviene centrarse en un grupo o familia de dispositivos móviles de un determinado fabricante, ya que como se comentó, cada fabricante realiza su propia versión de la máquina virtual y de las API´s de Sun para su dispositivo, pudiendo haber diferencias relevantes entre unas implementaciones y otras que llevaran a graves problemas de compatibilidad de la aplicación.

En este proyecto se optó por centrarse en la familia de dispositivos de la Serie 60 de Nokia –conocida como S60. La S60 de Nokia es considerada, hoy día, como una de las principales plataformas para el segmento de teléfonos móviles de alta gama o *smartphones*. Esta familia está basada en el SO Symbian. Consiste en un conjunto de librerías y aplicaciones. Sus estándares permiten el desarrollo de aplicaciones en Java MIDP (J2ME), Symbian C++ y Phyton. Además, Nokia proporciona un software de emulador, para PC, del dispositivo móvil completo, lo que facilita el desarrollo de aplicaciones. El SDK de Nokia para la Serie 60 también presenta utilidades de monitorización del dispositivo físico a través de *Bluetooth* y de visión de las trazas de las aplicaciones J2ME, que pueden resultar de gran utilidad a la hora de depurar posibles *bugs* que aparecieran en la aplicación.

Por otro lado, para realizar pruebas de la aplicación en dispositivo físico se contará con un teléfono móvil proporcionado por el DTE, un Nokia E65, perteneciente a la ya mencionada serie S60.

1.7 Objetivos principales del proyecto

El objetivo principal del proyecto es el desarrollo de un videojuego para dispositivo móvil con Bluetooth, con la particularidad de ser un videojuego en perspectiva isométrica. Para ello se toma como inspiración, y base para el desarrollo, un videojuego en perspectiva isométrica de los primeros tiempos de los ordenadores de sobremesa comerciales: *The Great Escape* [19] –desarrollado por Denton Designs [20] a mediados de la década de los 80 y que estuvo disponible para ZX Spectrum, Commodore 64, Amstrad CPC y MS-DOS. Este videojuego está basado en la película homónima, en español *La Gran Evasión*, protagonizada por *Steve McQueen* y dirigida por *John Sturges*, director de otros filmes famosos como: *Duelo de Titanes* y *Los Siete Magníficos*.

El filme se desarrolla en un campamento de prisioneros nazi donde se encuentran los mejores especialistas en escapismo del ejército aliado. Estos prisioneros aliados, sometidos a una intensa vigilancia, idearán una serie de estratagemas para evadirse de la reclusión mediante un largo túnel que los comunique con un bosque cercano, punto de salida para la salvación y libertad. El videojuego original, tomado como base, se mantiene fiel a la esencia de la película, apareciendo en él: túneles de fuga, una doble valla, barracones, etc.

En el juego existirá un personaje principal que se moverá libremente por una serie de escenarios. Este personaje principal tendrá como objetivo principal el fugarse del campo de prisioneros.

Referente a los escenarios, existirán tres tipos: el escenario principal (al que se le da el nombre de patio exterior), consistente en un recinto vallado con edificios y barracones por el que el personaje principal podrá moverse respetando siempre la colisión con los objetos y demás personajes, moviéndose en la perspectiva isométrica y estando siempre este personaje principal (al que se le da el nombre de Steve) centrado en medio de la pantalla; una serie de escenarios internos (a los que se le da el nombre de habitaciones) a los cuales el personaje principal tendrá acceso a través de puertas, en éstos el movimiento del personaje principal cambia respecto al del patio exterior, moviéndose con la capacidad de arrastrar el escenario por *scroll*; y los túneles, escenarios simplificados del juego original, pero en los que se seguirá viendo al personaje en su camino a través del túnel y en los que puede encontrar algunos obstáculos.

El personaje principal podrá recoger, para llevar consigo, y soltar, también, una serie de objetos que están desperdigados por todo el campo de prisioneros tanto en el patio exterior

como en las habitaciones. Con algunos de estos objetos, incluso, podrá interactuar con terceras entidades del campo de prisioneros, por ejemplo: existen llaves que permiten abrir puertas cerradas.

También existirán una serie de personajes con movimiento autónomo que deambularán por el patio exterior y ciertas habitaciones. Estos personajes son: los prisioneros compañeros del personaje principal, habituados a la rutina del campo de concentración y los guardias del campo. En concreto, los guardias se dividirán en dos grupos: los soldados rasos y el comandante del campo (conocido como el General), que patrullan por el campo.

En el juego existirá una rutina temporal de eventos a la que se debe adaptar el personaje principal y que los prisioneros compañeros de “Steve” dotados de cierta inteligencia respetarán.

Al juego se le dará un enfoque, además, de motor gráfico configurable basado en ficheros de configuración XML. Con esto se quiere decir que el juego en muchos de sus aspectos tanto gráficos como de lógica es configurable a través de una serie de etiquetas presentes en los ficheros de configuración XML. Aunque un motor gráfico configurable no sea un objetivo de partida del proyecto, durante la fase de desarrollo se comprobó que era la mejor forma de aproximarse a la resolución de los problemas; si bien, esto llevó a un nivel de complejidad mayor del que se esperaba en un principio, para poder incorporar toda esa lógica y configuración externas de los ficheros XML a la aplicación durante su ejecución.

Por último quiero destacar que el enfrentarse al desarrollo de un juego en perspectiva isométrica desde cero, utilizando la base de la tecnología Java J2ME, que, en principio, en cuanto al desarrollo de juegos, está esencialmente centrada para juegos planos, ha supuesto un reto de gran creatividad. Se han tenido que resolver continuos problemas referentes al modelado de datos que subyace a la aplicación, la obtención de una apariencia gráfica elegante y el traslado de la original lógica, relativamente compleja del juego tomado como inspiración, a este proyecto. Un esfuerzo continuado que ha tenido como recompensa la finalización con unos resultados muy satisfactorios, a nuestro modo de ver, de la meta de desarrollo de un videojuego en perspectiva isométrica, que nos propusimos.

Capítulo 2.
Descripción de la aplicación

2.1 Introducción

Este capítulo se centra en la descripción de la aplicación de videojuego a nivel de usuario. El objetivo principal buscado es describir la aplicación sin entrar en pormenores funcionales, de diseño o desarrollo técnico. En cierta forma, debe conformar una especie de pormenorizado manual de usuario donde se detallan todos los aspectos del juego.

Tras la lectura de este capítulo, el lector, deberá haber comprendido las múltiples facetas que presenta el juego. Facetas que van tanto desde el aspecto gráfico de la aplicación como a la lógica propia del juego. Tendrá un conocimiento amplio de los diferentes escenarios, de los personajes, de la trama. Y con este conocimiento estará en disposición de abordar el juego y seguir el desarrollo de la aventura gráfica.

No se abordarán aspectos técnicos acerca de cómo se ha hecho o cómo se ha implementado. Estos aspectos de diseño e implementación se reservarán para capítulos posteriores. En éste, se pretende, sobre todo, presentar la historia para situar al lector en un punto de partida favorable para comprender el diseño y sentido de la aplicación

2.2 Presentación de la aventura gráfica

Como ya se mencionó en la Introducción al proyecto, este videojuego toma como base, para inspiración, un videojuego en perspectiva isométrica de la década de los 80: *The Great Escape*. La intención de este proyecto no es realizar una réplica de aquel videojuego para los dispositivos móviles actuales, sino, más bien, respetando la esencia y apariencia gráfica del mismo, realizar una libre adaptación de aquel juego.

La historia del videojuego original está inspirada en la famosa película del mismo nombre -en español: *La Gran Evasión*. En ella, un grupo de prisioneros de guerra, recluidos en

uno de los campos de concentración nazi de máxima seguridad, intentará fugarse mediante la construcción de un largo y arriesgado túnel que les permita salvar la doble valla de alambre de espino que rodea todo el campo.

Los prisioneros, en el filme, tendrán que pasar por una serie de vicisitudes -los constantes registros de los barracones, el derrumbe del túnel, el recrudecimiento de las condiciones de vida en el campo, etc- que pondrán en serio aprieto su intento de fuga. Finalmente, una noche sin Luna un grupo de prisioneros, los más intrépidos y aguerridos, consiguen la evasión del campo. Pero el camino hacia la libertad sólo acaba de comenzar. Los nazis se lanzarán en su persecución por toda la región y sólo algunos afortunados conseguirán su objetivo de alcanzar la libertad.

El protagonista del filme, no es otro que *Steve McQueen*, actor de culto donde los haya, pero el film también cuenta con un elenco de actores, que acompaña a *McQueen*, digno de señalar. Actores como *James Garner*, *Richard Attemborough*, *Donald Pleasance*, *David McCallum*, *Gordon Jackson*, *James Coburn* o *Charles Bronson* nos dan idea de la importancia de la película (véase el cartel americano de la película **Figura 2.1**)

Figura 2.1 Cartel cinematográfico de *The Great Escape*.

Al personaje principal del videojuego se le refiere por su nombre: *Steve McQueen* es el hombre de acción por antonomasia. Y el jugador del videojuego desarrollado en este proyecto lo manejará para conseguir la evasión del campo de concentración nazi.

El juego original difiere ligeramente respecto de la película en cuanto al escenario principal. En la película, la acción transcurre en un recinto vallado, con torres de vigilancia y una serie de barracones donde viven los prisioneros. En el juego, a diferencia, el escenario principal lo constituye un castillo fortaleza. Éste posee un patio exterior donde se ubican los barracones de los prisioneros y que se encuentra parcialmente rodeado por una doble valla. A pesar de esta diferencia de escenario, la historia que desarrolla el juego se ajusta bastante a los detalles singulares de la película. Incluso, el juego llega a ser fidedigno a los aspectos de la fuga,

pudiéndose realizar la fuga del campo de concentración mediante túneles y la rotura de la valla de alambre de espino que rodea el campo.

Otro aspecto, a destacar, que lo diferencia de la película original es la existencia de las habitaciones interiores del castillo nazi. Por ellas, el personaje principal del videojuego, el capitán *Steve McQueen*, puede deambular a su antojo, aunque puede encontrarse algunas puertas cerradas, de forma que se las tendrá que ingeniar para abrirlas y explorar por completo el interior de la fortaleza.

Los desarrolladores del videojuego, *Denton Desings*, dieron una aventura gráfica en perspectiva isométrica 3D que en el momento de aparecer, mediados de la década de los 80, destacó por la tensa atmósfera que transmitía la historia del juego, en parte, gracias a la existencia de una rutina de actividades diaria automática que podía seguir el personaje principal. (Véase portada del juego original para *Commodore 64* en español **Figura 2.2**)

Figura 2.2 Portada del juego original para *Commodore*.

En el videojuego desarrollado en este proyecto, se toman muchos aspectos del videojuego original señalado. Principalmente, destaca su apariencia gráfica que está tomada por completo del juego original; en concreto, de la versión del juego para el emulador del *Commodore 64* para PC: *CCS64* [20].

Este emulador permite la captura de las pantallas del juego que muestra, siendo este mecanismo el utilizado para conseguir las imágenes del juego que conforman este proyecto. Se

usó también el software de retoque fotográfico Adobe PhotoShop CS para obtener y reconstruir las imágenes que sirven de base a la aplicación para realizar la apariencia gráfica.

Otros aspectos del juego original se han remozado. Aspectos como la ida al comedor del personaje principal, el ir a acostarse a dormir en la cama de su barracón, la celda de castigo o las rutinas temporales del campo. También, existen aspectos que no se han considerado, y se posponen como líneas futuras de desarrollo como es el movimiento de los guardias por las habitaciones del interior del castillo. No obstante, se ha pretendido permanecer lo más fiel posible al juego original. Esta intención se aprecia claramente en la reproducción de los escenarios del juego. Tanto el patio exterior, los barracones como las habitaciones interiores poseen un aspecto gráfico muy semejante al del juego original.

A continuación, se entrará un poco más en detalle en la descripción del videojuego desarrollado en este proyecto, empezando por las diferentes pantallas y menús que lo constituyen.

2.3 Pantallas y menús

El proyecto desarrollado cuenta con las pantallas (en J2ME se conocen como *Canvas*, nombre tomado de la clase java que permite pintar por pantalla) típicas de toda aplicación para teléfono móvil, en concreto, para el caso de los videojuegos móviles.

Existe una pantalla inicial de presentación del juego, pantallas de menús y las propias pantallas donde se desarrolla la acción del juego. En este apartado se irán mostrando las diferentes pantallas que constituyen la aplicación de videojuego, así como, la navegación que se realiza a través de ellas.

La aplicación comienza con la pantalla de presentación del videojuego. Una pantalla donde aparece la imagen de la portada del videojuego original. Esta pantalla se puede observar en la imagen de la **Figura 2.3**

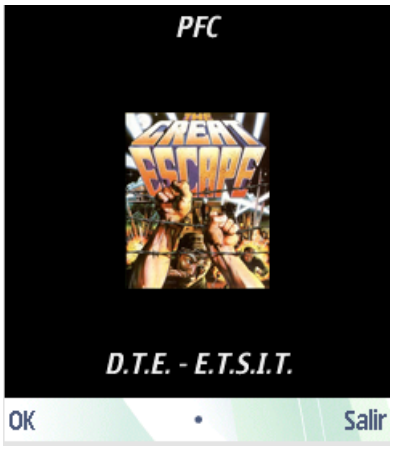


Figura 2.3 Pantalla de presentación de la aplicación.

Posee dos botones que se pueden pulsar desde teclado: el botón *OK* y el botón *Salir*. Con la pulsación del botón *OK*, la aplicación continúa su desarrollo normal, pasando a mostrar por pantalla el menú principal del juego. También hay que destacar que no sólo con pulsar el botón *OK* se pasa a mostrar dicho menú, sino, además, con la acción de pulsar cualquier otro botón diferente del botón de Salir, se consigue el mismo resultado de pasar al menú principal de la aplicación.

El botón *Salir* de la pantalla de presentación lleva a una pantalla de confirmación de salida de la aplicación. Esta pantalla se muestra en la **Figura 2.4**. En ella se pregunta al usuario si desea abandonar la aplicación. Si el usuario selecciona *Sí*, la aplicación de videojuego se cierra. En caso contrario, si el usuario selecciona *No*, entonces se sale de la pantalla de confirmación de salida, volviéndose a la pantalla de presentación previa.

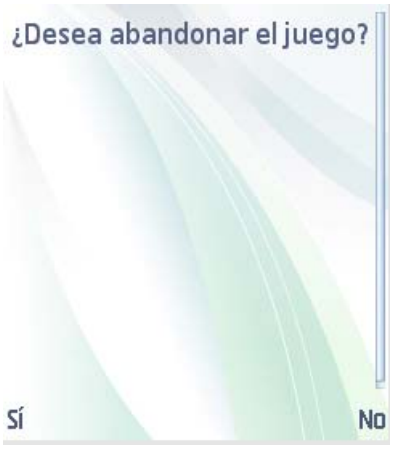


Figura 2.4 Pantalla de confirmación de salida.

A la pantalla de presentación, cuando se continúa con el juego, sigue como ya se ha mencionado la pantalla del menú principal del juego. En esta pantalla existen diversas opciones de selección. Desde ella se lanza el juego. Las opciones que se muestran son:

- Juego Nuevo
- Instrucciones
- Sonido Off/On
- Vibración Off/On
- Salir

Estas opciones se muestran en la **Figura 2.5**. Mediante las teclas de arriba y abajo del dispositivo se puede recorrer el menú. Para seleccionar una de las opciones, existen dos posibilidades, o bien, una vez situado encima de la opción elegida, pulsar el botón de *Select* que se muestra en la figura, o bien, pulsar el botón de disparo o "*intro*" que posee el dispositivo móvil, que tendrá el mismo efecto. Otro botón que aparece es el botón *Atrás*. Este botón siempre te conduce a la pantalla previa; en este caso, lleva de nuevo a la pantalla de presentación ya comentada.

Figura 2.5 Menú principal del juego.

A continuación, se irán desglosando, por orden de aparición de arriba a abajo, las diferentes opciones del menú, explicando sus acciones correspondientes.

La selección de la opción *Juego Nuevo* arranca el juego. Hasta este momento, en la ejecución de la aplicación, no se han reservado recursos para lo que es el propio juego. Seleccionando esta opción, aparecerá una pantalla donde se indica que el videojuego está cargándose en la memoria del dispositivo móvil. Esta pantalla de "cargando" se puede ver en la imagen de la **Figura 2.6**. El tiempo de carga en memoria depende de la velocidad de proceso

del dispositivo y puede variar de un dispositivo a otro. En el teléfono móvil para el cual este proyecto está optimizado, un dispositivo del fabricante Nokia, de la serie S60: Nokia E65, el tiempo está en torno a los quince segundos.

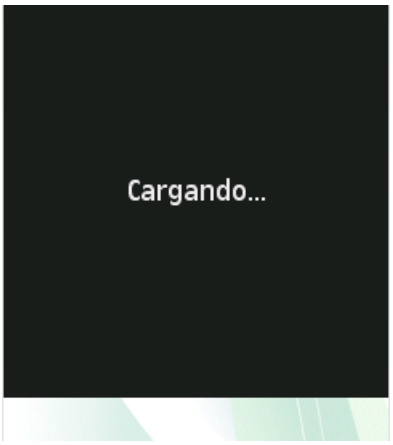


Figura 2.6 Pantalla *Cargando…* al inicio de la partida.

Transcurrido el tiempo que se lleva la aplicación cargando los mapas de los escenarios y las distintas imágenes que la constituyen, aparecerá por pantalla la imagen del juego propiamente dicho.

Una vez aparezca la primera pantalla del juego, el control de menú de la aplicación se deriva al botón *Options*, pulsándolo se desplegará el menú de control durante tiempo de juego para la aplicación. Este menú se explica un poco más adelante en este mismo apartado, una vez concluida las reseñas a las opciones del menú principal del juego que ahora nos ocupan.

La selección de la opción *Instrucciones* mostrará una pantalla de texto, donde aparecen las instrucciones para comenzar a jugar. El texto de instrucciones es una versión adaptada y remozada de las instrucciones del juego original para *Commodore 64*.

En el texto de *Instrucciones* se realiza una presentación del juego; además, se detallan la geografía del escenario principal, los distintos personajes, las características del personaje controlado por el jugador, así como diferentes aspectos de interés: la moral del personaje principal, las alarmas que pueden darse y las noticias o mensajes informativos. En la **Figura 2.7** se puede ver la pantalla de instrucciones.

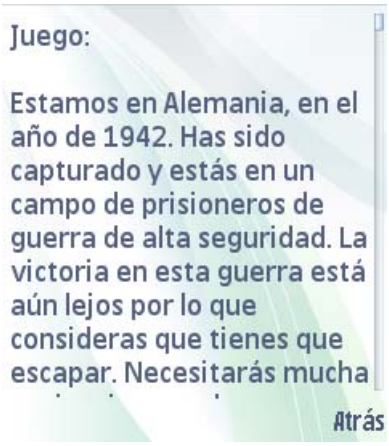


Figura 2.7 Pantalla de instrucciones.

La opción de *Sonido Off* indica que el sonido en la aplicación de videojuego está desactivado. Si se pulsa esta opción, se activará el sonido, pasando a mostrar en el acto en el menú: *Sonido On*, en vez del anterior *Sonido Off*. Si se volviera a seleccionar esta opción, se produciría el cambio a la opción de partida *Sonido Off*, desactivándose el sonido, y así sucesivamente. En la **Figura 2.8** se muestra como el menú *Sonido Off* a pasado a ser *Sonido On*, tras haberla seleccionado. La aplicación de inicio lleva el sonido desactivado.

Figura 2.8 Acción de seleccionar la opción de sonido para activarlo.

El mismo principio de actuación rige para la selección de *Vibración Off*. De partida, la vibración se encuentra desactivada. Seleccionando esta opción del menú, se activará y el menú pasará automáticamente a mostrarlo, cambiándose de nuevo, si es seleccionada otra vez, y así sucesivamente.

Por último se tiene la posibilidad de seleccionar la opción de Salir, que, al igual que en la pantalla de presentación, conduce a una pantalla de confirmación de la salida (ver **Figura 2.4**).

Siguiendo con el normal desarrollo del juego, una vez que hayamos seleccionado la opción de *Juego Nuevo*, y tras la carga de los escenarios (patio exterior y habitación donde el personaje comienza su andadura, acostado en la cama) en memoria, que se indica mediante la pantalla de *Cargando…*, el usuario de la aplicación se encontrará ante la visualización en pantalla del juego propiamente dicho. La **Figura 2.9** muestra la primera pantalla del juego.

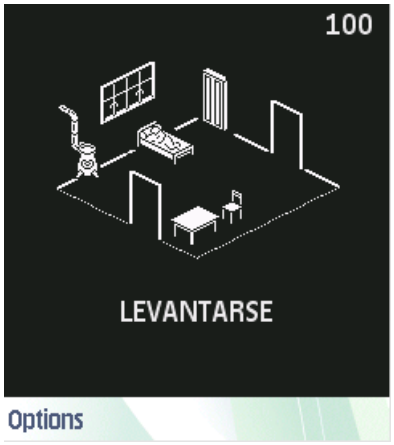


Figura 2.9 Pantalla de inicio del juego.

En el juego existen tres clases de escenarios: el escenario exterior, los escenarios interiores y los escenarios subterráneos. El escenario exterior -también llamado patio exterior, ya que comprende todo el patio del castillo fortaleza del campo de prisioneros- constituye el escenario principal del juego, y a través de él se distribuye el acceso a diferentes grupos de escenarios interiores, mediante puertas, o a los túneles, mediante sus entradas. Los escenarios interiores son las diferentes habitaciones. En éstas se tienen también puertas que enlazan con otras habitaciones o con el patio exterior, además de entradas a túneles.

En cada escenario existirá la posibilidad de acceder a un menú en tiempo de ejecución, es decir, continuándose en segundo plano (o *background*) con el desarrollo o ejecución del juego. El escenario exterior y los escenarios interiores comparten el mismo menú; en la imagen de la **Figura 2.10** se puede apreciar este menú sobre el emulador de Nokia para la serie S60. El escenario subterráneo, túnel, posee una versión reducida de dicho menú.

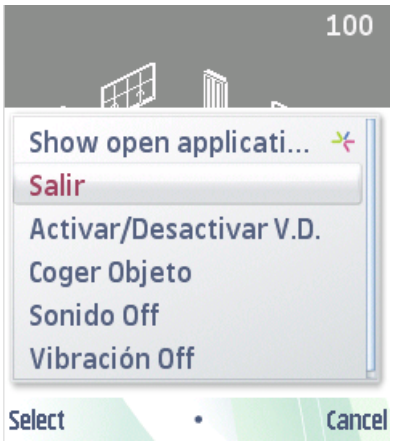


Figura 2.10 Menú en tiempo de ejecución del juego.

Cabe señalarse que en la realización sobre el dispositivo físico Nokia E65, el menú no ocupa por completo el ancho de la pantalla ni tampoco muestra la primera opción que se aprecia para el caso del emulador, *Show open applicati…* . La lista de entradas que presenta, de partida, este menú son:

- *Salir*
- *Activar/Desactivar V. D.*
- *Coger Objeto*
- *Sonido On/Off* (no apreciada en la Figura .8, debe utilizarse el cursor para visualizarla)
- *Vibración On/Off* (ídem que para el caso anterior)

Dejando para el final la opción de *Salir* por algunas peculiaridades que posee para este menú, vamos a empezar por la que le sigue: *Activar/Desactivar V.D.* Esta opción permite Activar o Desactivar Visión de Diseño. La visión de diseño es una vista del juego que se ha utilizado para la construcción de los escenarios, además de para la exigente fase de depuración de la aplicación. En la visión de diseño se muestra una cuadrícula de celdas en las que se basa la aplicación para dibujar los escenarios en perspectiva isométrica y controlar los movimientos de los personajes respetando las reglas de la perspectiva isométrica. También, aparecen una serie de marcas de colores que sirven para indicar las fronteras de colisión, así como, puntos especiales que permiten el acceso a otros escenarios, si bien los detalles de su uso se reservarán para el siguiente capítulo de la memoria.

Otra de las cosas que se muestra cuando seleccionas esta opción es el campo de visión de los guardias. Esta visión se conserva meramente a fines ilustrativos; en una versión comercial del mismo debería eliminarse. En la **Figura 2.11** se pueden apreciar dos imágenes de la mencionada visión de diseño. Una de un escenario interior, una habitación del interior del

castillo, y otra del patio exterior del castillo, donde se aprecian otros prisioneros y guardias con parte de su correspondiente campo de visión, a la vez que se muestra un mensaje por pantalla que llama al evento de formar.

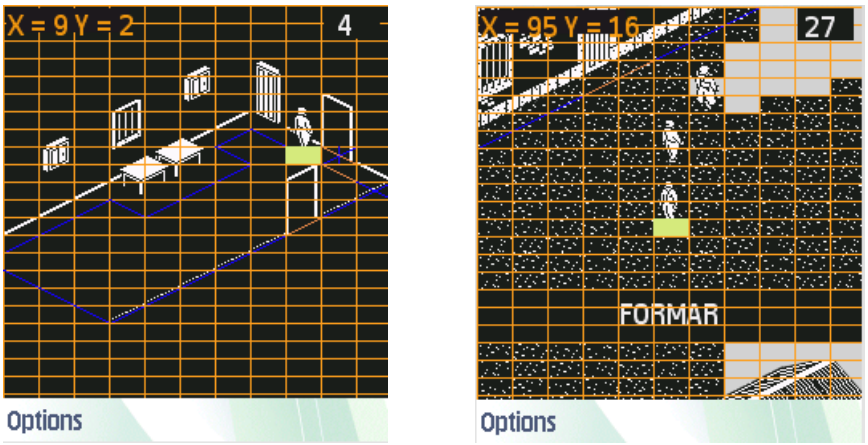


Figura 2.11 Visión de diseño del interior de una habitación y del exterior.

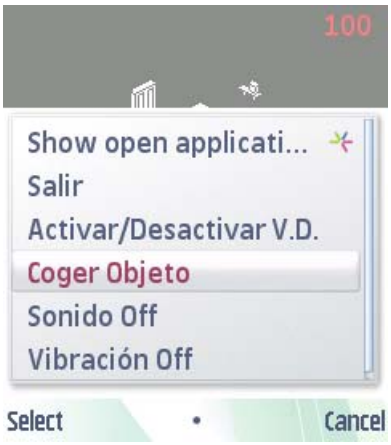
La siguiente opción que sigue a la de visión de diseño, y en la que nos detendremos un poco más a raíz de las ramificaciones que conlleva, es la de *Coger Objeto*. Con esta opción del menú, el usuario podrá recoger objetos del suelo del campo. El usuario sólo puede llevar recogidos un máximo de dos objetos. La lista de objetos del campo que puede recoger el personaje principal -*Steve*- es:

- Llave
- Pala
- Linterna
- Herramientas
- Documentos
- Radio
- Tenazas
- Bolsa
- Brújula
- Paquete de la Cruz Roja (objeto envoltorio de otros objetos y entregado por la Cruz Roja)

Más adelante en el **Apartado 2.8**, dedicado a los objetos, se detallarán un poco más éstos.

Para recoger un objeto, el usuario debe colocar al personaje encima o en el perímetro del objeto, pulsando seguidamente la opción de *Coger Objeto*. Una vez seleccionada esta opción, el objeto desaparecerá de la pantalla, mostrándose un mensaje por pantalla donde se indica que ha recogido el objeto. En la **Figura 2.12** se muestran imágenes que indican el proceso de coger objeto.

a) Colóquese al lado del objeto.



b) Seleccione *Coger Objeto*.

c) Aparece mensaje cuando recoge.

d) Puede usar o soltar el objeto.

Figura 2.12 Secuencia de acciones para recoger objeto, usarlo o volver a soltarlo.

Una vez recogido el objeto, si se pulsa de nuevo en la tecla de *Options*, aparecerán dos opciones elegibles nuevas: *Soltar <Objeto>* y *Usar <Objeto>*. Esto último también se observa en la **Figura 2.12 d)**. Si el usuario pulsa la opción de *Soltar <Objeto>*, entonces éste se suelta en el suelo siempre que haya hueco libre, es decir, que no existan otros objetos ya en esa ubicación que impidan que éste se pueda dejar en el sitio elegido. En caso de que se dé esta situación aparecerá un mensaje por pantalla donde se indica que debe cambiar de ubicación para poder soltar el objeto, ya que está sobre un objeto (véase la **Figura 2.13**).

Figura 2.13 Mensaje mostrado por pantalla al intentar soltar un objeto sobre otro.

Con la opción de *Usar <Objeto>*, el jugador puede utilizar el objeto que recogió para alguna función. Existen objetos que al seleccionar la opción de usar no tienen ningún efecto, no obstante, se mantiene en el menú dicha opción. Estos objetos son: la linterna, la radio, la bolsa y la brújula; objetos que tienen un efecto ya de por sí al ser portados por el personaje. Los restantes objetos, sí tienen una respuesta determinada a la selección de la opción *Usar <Objeto>*, pero para que se dé, el personaje principal debe estar en el ámbito definido de acción para ese objeto; por ejemplo, las tenazas sólo actuarán si el personaje se encuentra al lado de un trozo de valla, igual ocurre con las llaves: éstas sólo actuarán cuando el personaje esté al lado de una puerta cerrada y la llave sea la correspondiente que abre dicha puerta. Véase en la siguiente figura, **Figura 2.14**, un ejemplo de uso del objeto tenazas.

Figura 2.14 Ejemplo de uso del objeto tenazas.

El usuario podrá recoger y llevar consigo un máximo de dos objetos como ya se ha señalado. En el caso de que tenga ya dos objetos consigo e intentara recoger un tercero, a través de un mensaje por pantalla se le indicaría que debe soltar previamente uno de los que lleva

consigo para poder recoger aquél. En la **Figura 2.15** se muestra el menú de *Options* una vez recogidos dos objetos del campo: las Tenazas y la Llave Roja, además aparece una imagen, la de la derecha, donde se muestra el mensaje de que debe soltar al menos uno para recoger uno nuevo.

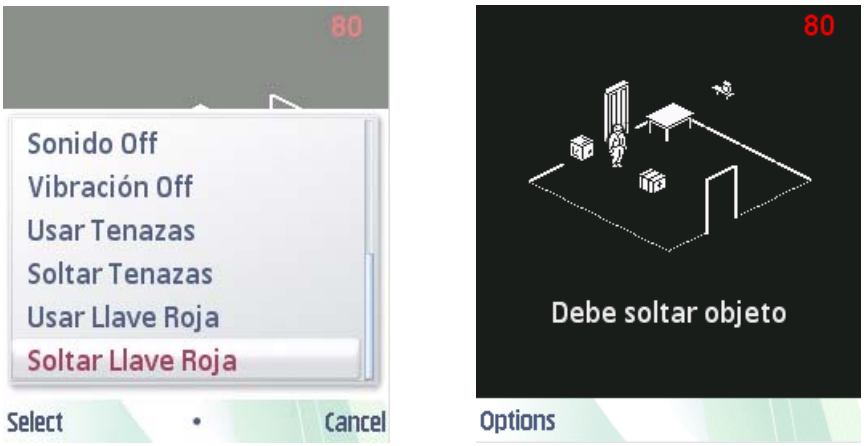
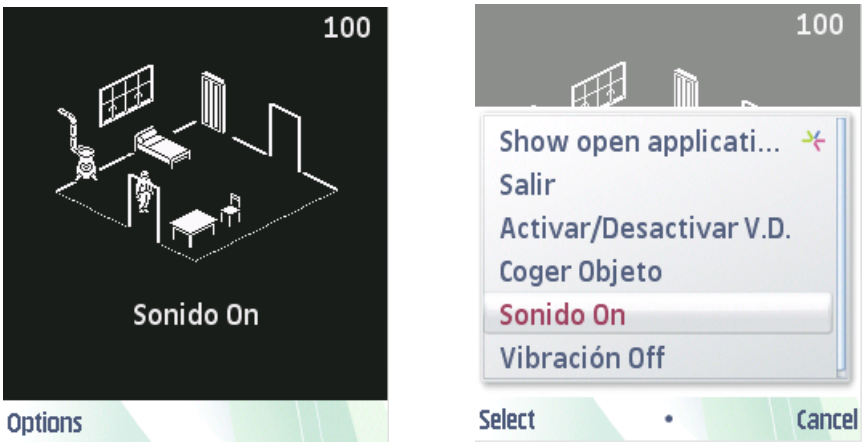


Figura 2.15 Menú de *Options* con dos objetos y mensaje mostrado al intentar recoger otro.

La opción del menú *Sonido Off/On*, sigue la misma filosofía de la opción que aparece en el menú principal. En ella se mostrará el estado en que se encuentra el sonido, es decir, si está activado o desactivado. Tras pulsarla esta opción cambiará automáticamente a su opuesta, y se volverá al juego, mostrándose por pantalla, además, un mensaje donde se indica en qué modo está el sonido. En la **Figura 2.16** se puede observar la secuencia de pantallas indicadas en este párrafo.



a) Mensaje de estado del sonido **b)** Se actualiza el menú de *Options*.

Figura 2.16 Secuencia de pasos para activar el sonido desde el menú de *Options*.

Otra opción que se tiene es la opción de *Vibración Off/On*. La forma de comportarse de esta opción es completamente idéntica a la anterior de *Sonido Off/On*. Seleccionándola, se

activará o desactivará, según el caso, la vibración en la aplicación. Y al igual que para el caso de *Sonido Off/On*, se mostrará por pantalla un mensaje indicando el estado de la vibración y se hará la actualización de dicho estado, también, en el menú de *Options*.

Por último se tiene la opción de menú de *Salir*. Se ha pospuesto esta opción de *Salir* para el final, ya que conlleva el lanzamiento de un menú. Al seleccionar la opción de *Salir* se muestra un menú de opciones donde el usuario puede elegir la opción que desee de entre las siguientes:

- *Menú juego nuevo*
- *Continuar juego*
- *Salir*

En la **Figura 2.17** se muestra el menú que se despliega una vez se selecciona esta opción.

Figura 2.17 Menú desplegable de *Salir*.

En el menú que se despliega tras seleccionar *Salir*, la selección de la opción *Menú Juego Nuevo*, conduce al menú principal del juego, ya mostrado en la **Figura 2.5** con la salvedad que, en este caso, pulsar la tecla de *Atrás* conduce al menú mostrado en la **Figura 2.17**, correspondiente a haber elegido la opción *Salir* en el menú de *Options*.

Con la opción de *Continuar Juego* se regresa al juego –el comando de *Atrás* tiene el mismo comportamiento que la opción *Continuar Juego*. Finalmente, la selección de la opción *Salir* lleva a una pantalla de confirmación para la salida como la de la **Figura 2.4**. En esta pantalla de confirmación de salida el pulsar la tecla asociada a *No* conduce al jugador, de nuevo,

al menú de partida (el menú de la **Figura 2.17**), mientras que la selección de *Sí* cierra la aplicación.

Las opciones comentadas hasta ahora del menú que se despliega durante el tiempo de juego, tras pulsar la tecla de *Options*, corresponden a los escenarios del patio exterior y de las habitaciones interiores. Los escenarios de túneles poseen un menú, cuando se pulsa la tecla de *Options*, que es una versión reducida del anterior. Las opciones de *Activar/Desactivar V.D.*, *Coger Objeto* y *Soltar <Objeto>* no aparecen. En la **Figura 2.18** se muestran dos imágenes con el menú desplegado al pulsar la tecla de *Options* en el interior de un túnel.

Figura 2.18 Menú *Options* dentro de un túnel.

Otras pantallas que son de interés son las pantallas de fin de partida. El fin de partida puede darse por dos razones: bien, el jugador consigue la evasión, en cuyo caso se muestra un mensaje de felicitación (con imagen de fondo de la liberación de los prisioneros españoles del campo de *Dachau* por los americanos), reproduciéndose si el jugador tiene activado el sonido la música del filme *La Gran Evasión*, o bien, la partida termina porque el jugador pierde, que a efectos del juego se produce cuando el jugador se queda sin moral, mostrándose una pantalla de indicación de finalización del juego (con imagen de fondo de la entrada al campo de *Auschwitz*). Estas pantallas se pueden observar en la **Figura 2.19**.

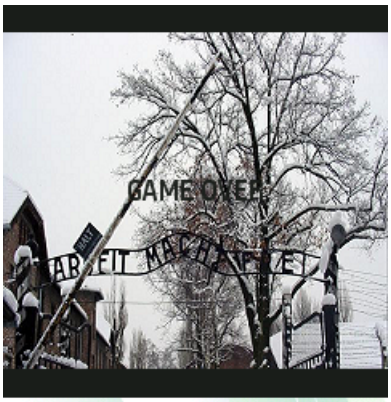


Figura 2.19 Pantallas de felicitación por la evasión y fin de partida por haber perdido.

Una vez aparezcan estas pantallas, indicándole al jugador si ha conseguido fugarse o ha perdido la partida, el usuario podrá pulsar cualquier tecla del dispositivo, tras lo cual aparecerá un menú con dos opciones a elegir: Menú juego nuevo, Salir y una tecla Atrás que lleva de nuevo a las pantallas de *¡Una Gran Evasión!* o *Game Over*. En la figura siguiente, **Figura 2.20**, se muestra la imagen del mencionado menú.

Figura 2.20 Menú mostrado tras las pantallas de fuga con éxito o juego perdido.

La selección en este menú de la opción de *Menú Juego Nuevo*, muestra por pantalla, de nuevo, el menú principal del videojuego, ya comentado -**Figura 2.5-**, la única diferencia respecto al menú principal está en que pulsando la tecla Atrás de éste lo que se hace es volver al menú de la **Figura 2.20**. La opción de *Salir* conduce a una pantalla de confirmación de cierre de la aplicación como las ya señaladas anteriormente.

2.4 Mapas de los escenarios.

Los escenarios en los que transcurre la acción del videojuego son: el patio exterior del castillo, los barracones de los prisioneros ubicados en dicho patio, las habitaciones interiores del castillo y los túneles. El patio exterior del castillo constituye el escenario principal del juego y el que sirve de unión entre los demás grupos de escenarios: habitaciones de los barracones y del interior del castillo y los túneles.

En cuanto a habitaciones existe un total de veinticinco. De ellas diecinueve son habitaciones interiores del castillo fortaleza y seis corresponden a los tres barracones, ubicados en mitad del patio exterior y que constituyen los dormitorios de los prisioneros del campo. El número de túneles presentes en el juego es dos. Estos túneles conectan habitaciones (una, un dormitorio de un barracón, el dormitorio de *Steve*, y otra una habitación interior del castillo) con diferentes sitios del patio exterior.

En la **Figura 2.21** se aprecia un mapa del escenario principal del juego: el patio exterior, junto con el interior de los tres barracones. El mapa representa un esquema fidedigno del escenario recreado en el videojuego –este mapa es una readaptación del que aparece en la revista de Micromanía dedicada al juego original [21]. Tomando como referencia este mapa, se va a realizar la descripción del escenario principal del juego: el patio exterior del castillo.

En el mapa se aprecian dos zonas especiales con sus nombres correspondientes: la zona de formar y el patio de ejercicio. La zona de formar se encuentra en la esquina noroeste del campo (tomando la longitudinal de los barracones como eje Norte-Sur y en sentido creciente la dirección del Norte). A esta zona deberán acudir los prisioneros cuando en el juego sean llamados a formar.

La otra zona que se nombra en el mapa es el patio de ejercicio. Ubicado en el Sur del campo, exterior al doble vallado y accesible a través de una doble puerta en el vallado. El patio de ejercicio es la zona de esparcimiento de los prisioneros, a la que tendrán que acudir cuando en el campo se realice un llamamiento a ejercicio. Esta zona tiene la peculiaridad de que su puerta de acceso sólo se encontrará abierta durante el tiempo asignado a realizar ejercicio; en el resto, se encontrará cerrada.

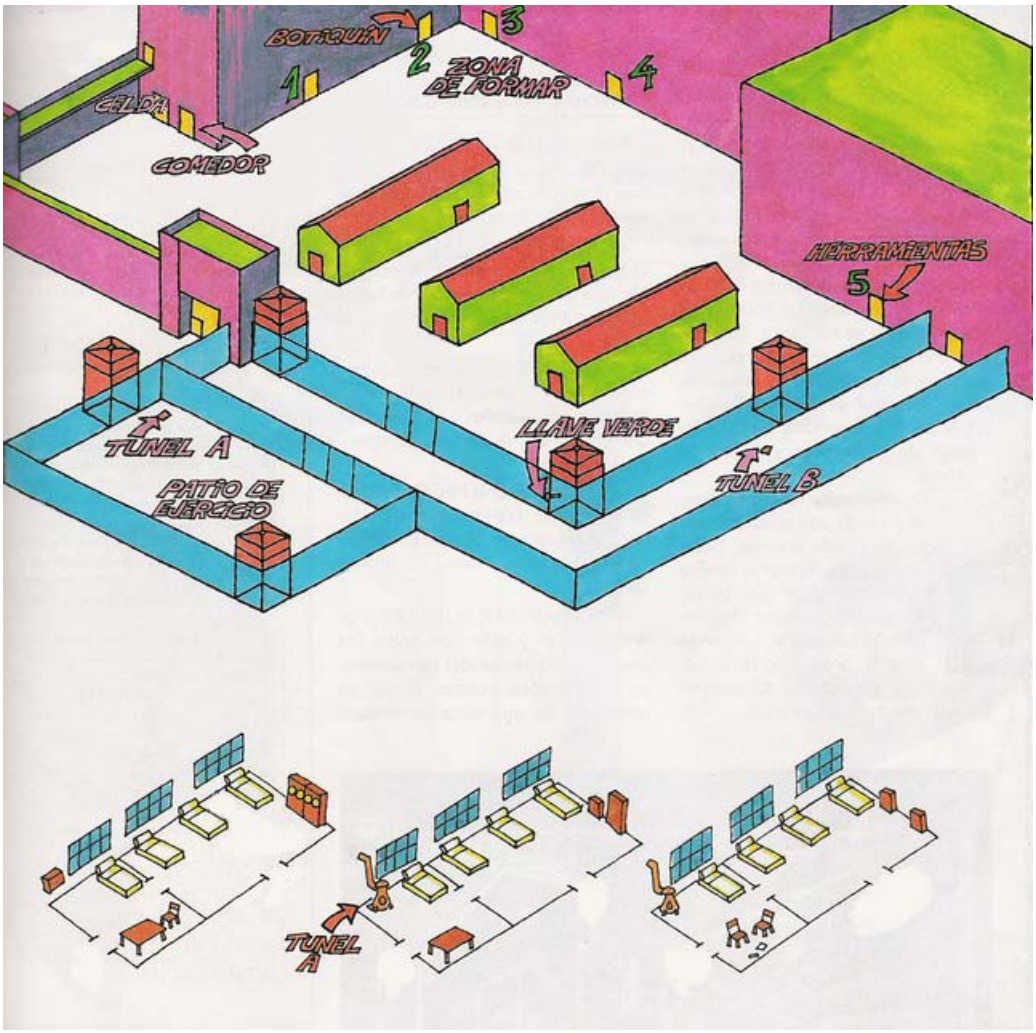


Figura 2.21 Mapa del escenario principal: patio exterior y de los tres barracones de prisioneros.

Otra zona a destacar, aunque no aparezca nombrada en el mapa, es el interior del doble vallado. Por ella patrullan dos guardias nazis del campo. En la **Figura 2.22** se tienen dos imágenes correspondientes a cada guardia patrullando su lado del doble vallado.

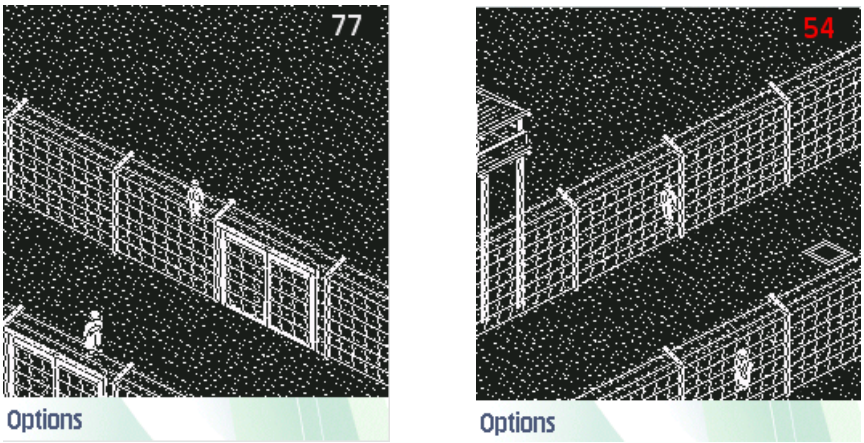


Figura 2.22 Guardias patrullando por el interior de la doble valla sur y este.

Otro accidente destacado del patio exterior lo constituyen los tres barracones de los prisioneros, cuyos interiores se aprecian en la parte inferior del mapa de la **Figura 2.21**. Respecto del mobiliario interior de las habitaciones debe señalarse que la representación mostrada en los mapas es orientativa, si bien, se ajusta en alto grado a la realidad interior. En la **Figura 2.23** se muestra una composición del interior del barracón de *Steve*, el segundo barracón. Los dos restantes barracones son iguales, sólo cambia el mobiliario y que en el barracón del Oeste se tienen las camas ocupadas permanentemente por unos prisioneros enfermos.

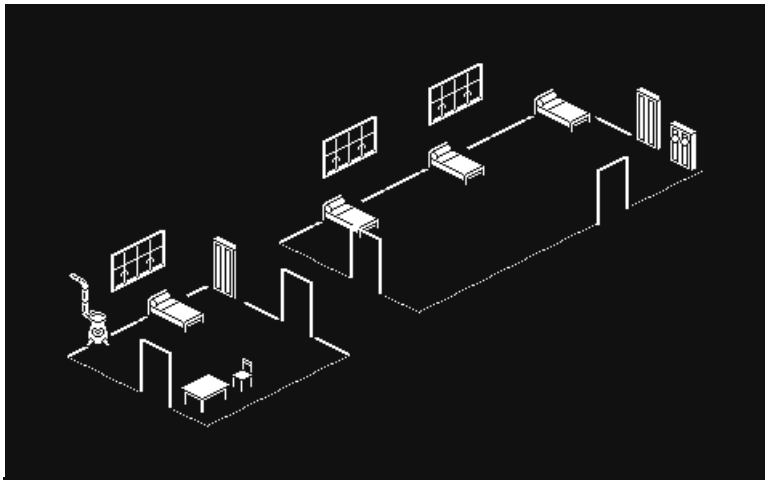


Figura 2.23 Composición del barracón de *Steve*.

En el lado Oeste existen dos puertas que dan a dos habitaciones especiales: la celda de castigo y el comedor. La **Figura 2.24** muestra estas dos puertas y en la **Figura 2.25** una composición de las dos habitaciones. La que más pega al Oeste es la correspondiente a la celda de castigo como se indica en el mapa, mientras que la otra es la que comunica con el comedor.

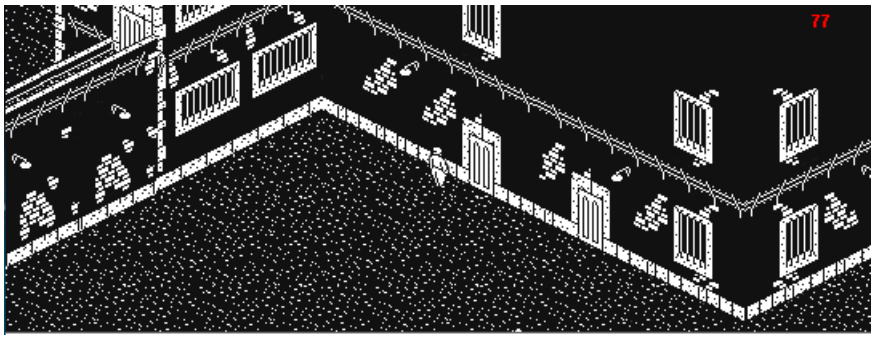


Figura 2.24 Puertas de entrada al comedor y a la celda de castigo.

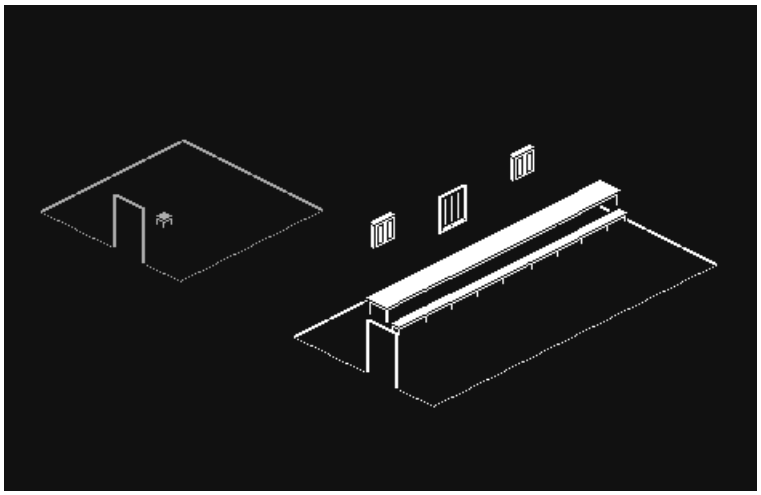


Figura 2.25 Composición con las dos habitaciones contiguas: celda y comedor.

La celda de castigo es el lugar donde se traslada al personaje principal una vez que es detenido por los guardias del campo. La puerta que da acceso a la celda de castigo está siempre cerrada, salvo cuando una vez trasladado al interior de ésta, se da permiso al jugador para que la abandone. Nótese que el color de las líneas de la celda y del banco es de un tono más azulado, esto es así puesto que su interior está a oscuras. El comedor es el lugar donde deben ir todos los prisioneros del campo cuando se llame a comer, lo constituye una habitación grande con una mesa de grandes dimensiones donde se van sentando los prisioneros.

Siguiendo por esa misma ala Oeste y subiendo un poco al Norte, nos encontramos con la puerta que en el mapa de la **Figura 2.21** tiene el distintivo de número uno. Esta puerta da a un conjunto de habitaciones interiores que comunican a través del interior del castillo con las puertas con los distintivos tres y cuatro. La puerta uno está abierta, mientras que las puertas tres y cuatro están cerradas. La puerta dos tiene al lado la leyenda de “Botiquín”, es la habitación donde se realizan las entregas de la Cruz Roja, que se comentarán más adelante -la **Figura 2.26** nos muestra su interior.

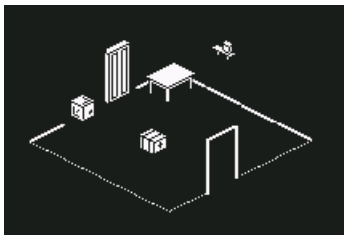


Figura 2.26 Habitación del botiquín con un paquete de la Cruz Roja.

Un hecho destacable al respecto de las cuatro puertas anteriores es la existencia de un guardia del campo que patrulla en sentido de ida y vuelta por toda la escuadra noroeste, donde

se encuentran estas puertas. También, debe señalarse, a propósito de estas puertas, que si eres descubierto por un guardia entrando o saliendo de una de ellas, automáticamente se activa la alarma de modo que cualquier guardia que te viera iría a por ti.

La última puerta con reseña numérica en el mapa de la **Figura 2.21** es la número cinco. Esta puerta da a una serie de habitaciones interiores, en concreto tres, cuya peculiaridad más destacada es que en una de ellas están las herramientas.

Un accidente topográfico del patio exterior que no se representa en el mapa de la **Figura 2.21** es el perímetro de piedras que rodea el campo allende de la doble valla y los refugios o escondites de piedras donde dejar objetos sin que puedan ser descubiertos. Este perímetro de piedras, del cual se puede apreciar una parte en la **Figura 2.27**, rodea por completo al doble vallado, surgiendo de la esquina suroeste del patio de ejercicio y rodeando el doble vallado hasta la esquina noreste.

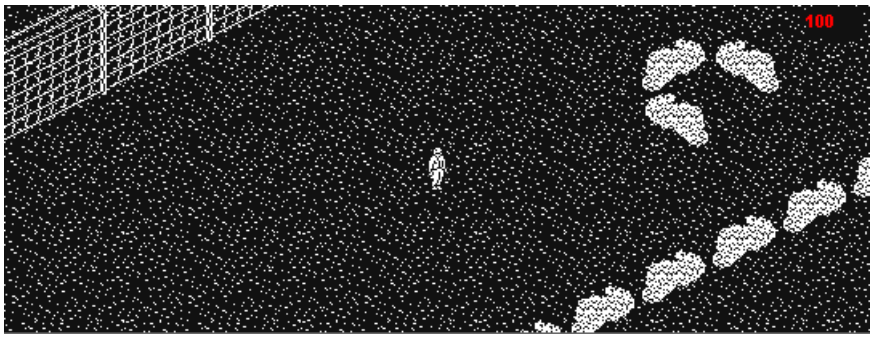


Figura 2.27 Imagen del perímetro de piedras exterior y un escondite para objetos.

Entre el doble vallado, y visto como extensión de éste, el perímetro del patio de ejercicio y el perímetro de piedras se distribuyen una serie de escondites de piedra como el que se aprecia en la **Figura 2.27**. En ellos es factible que el personaje principal del juego controlado por el jugador esconda objetos de interés para la fuga. En total existen cinco de estos escondites de piedras, repartidos: tres en lado Este del perímetro y dos en el lado Sur. Debe destacarse que las únicas vías de escape del campo de prisioneros son unas aberturas existentes en el perímetro de piedras; sólo a través de ellas puede el personaje principal conseguir la fuga del campo de prisioneros. Esta condición obliga a que la única manera de fugarse sea rompiendo el perímetro de vallado y encaminándose a una de estas aberturas. En la **Figura 2.28** puede apreciarse una de estas aberturas en el perímetro de piedras.

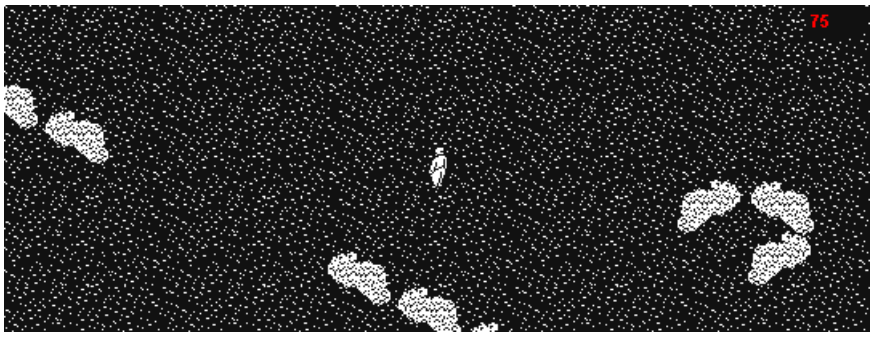


Figura 2.28 Abertura de escape en el perímetro piedras, más un refugio.

El único objeto especial que se encuentra de partida en el patio exterior es la Llave Verde. Ésta se encuentra en la esquina suroeste del campo junto a la torre de vigilancia. Esta llave abre la puerta del patio de ejercicio indicada como número cinco en el mapa de la **Figura 2.21**. Se puede ver el sitio señalado en la imagen de la **Figura 2.29**.

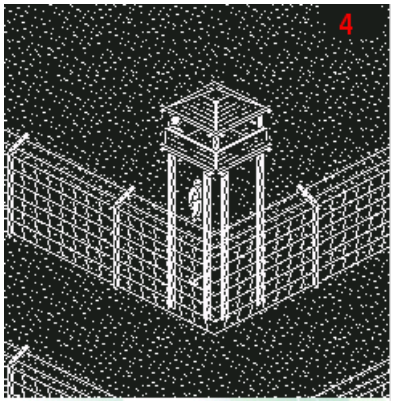


Figura 2.29 Esquina suroeste del campo –Llave Verde entre los postes de la torre.

Una vez vistos los aspectos más relevantes del mapa del patio exterior del campo, se pasará a abordar las habitaciones interiores del castillo. En la **Figura 2.30** se muestra el mapa para las habitaciones interiores del castillo fortaleza nazi donde se encuentran recluidos los prisioneros.

El mapa de las habitaciones interiores al igual que el del patio exterior es una versión adaptada al juego del que aparece en la revista Micromanía ya citada. Igual que ocurría con el mobiliario de los barracones mostrado en el mapa de la **Figura 2.21**, el mobiliario de las habitaciones mostrado en este mapa es sólo orientativo, si bien, posee un alto grado de semejanza con el real encontrado en las habitaciones. Como se percibe en el mapa existen dos grupos de habitaciones claramente diferenciados: el grupo de habitaciones constituido por los accesos a través de las puertas uno, dos, tres y cuatro, y el grupo de habitaciones al que se accede a través de la puerta cinco.

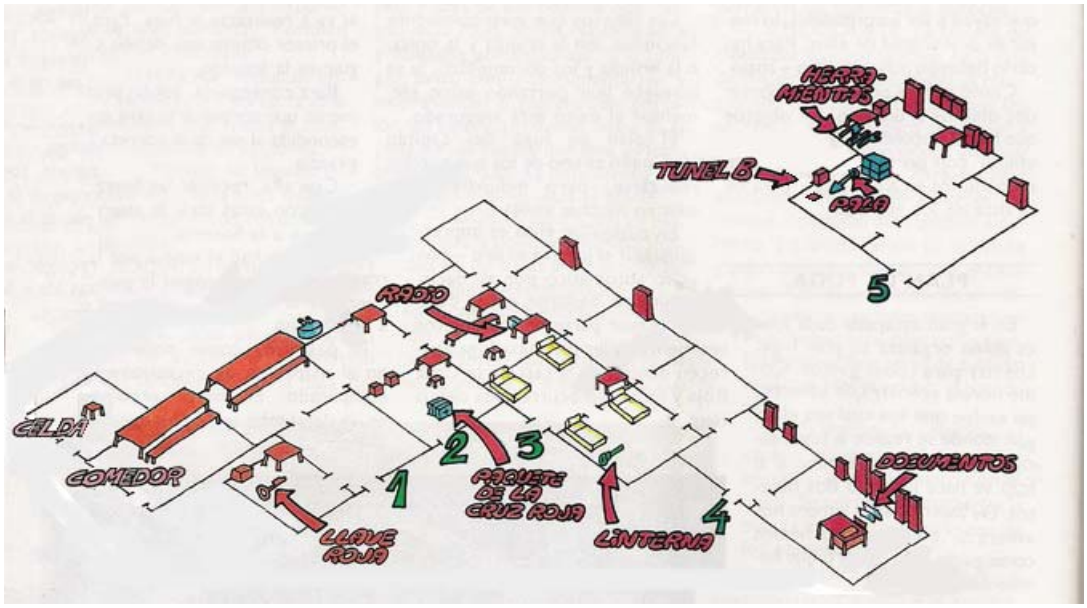


Figura 2.30 Mapa de las habitaciones interiores del castillo.

Empezando por el primer grupo señalado, se irán mostrando los aspectos y las habitaciones más destacados. En la habitación a mano izquierda de la primera a la que se accede por la puerta número uno, nos encontramos con la Llave Roja (véase la **Figura 2.31** donde se muestra esta a *Steve* en esta habitación).

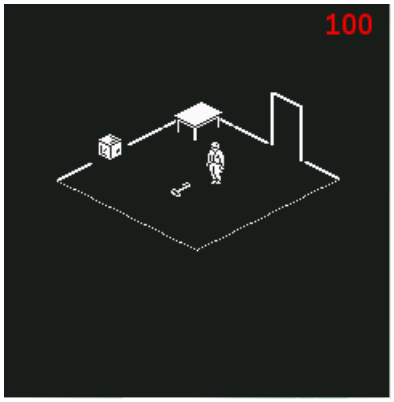


Figura 2.31 Habitación Llave Roja.

En la tercera habitación que se alcanza entrando por la puerta número uno y siguiendo la regla de desplazamiento de girar siempre a la derecha, la segunda puerta, es decir, la puerta de salida de la habitación, se encuentra cerrada. Esta puerta se abre con la Llave Roja, ya comentada. La **Figura 2.32** enseña la puerta que se encuentra cerrada.

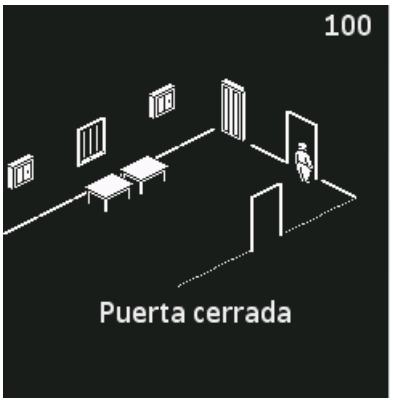


Figura 2.32 Tercera habitación interior entrando por la puerta uno del patio exterior.

Utilizando la Llave Roja, que se recogió en la habitación de la **Figura 2.31**, sobre la puerta, el jugador tendrá acceso al resto de habitaciones pertenecientes al grupo. En la segunda habitación tras cruzar la puerta cerrada nos encontraremos con una habitación donde tenemos otro objeto especial: la radio. La **Figura 2.33** capturada muestra esta habitación.

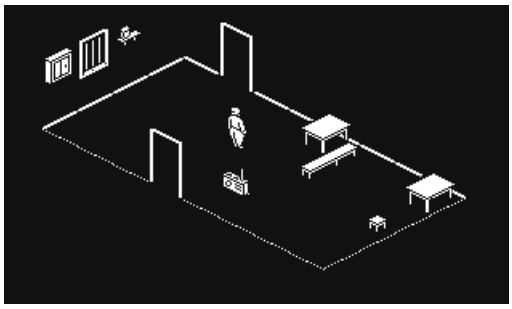


Figura 2.33 Habitación en la que se encuentra el objeto especial radio.

Tras la habitación anterior siguen cuatro habitaciones de paso; en la cuarta, nos encontraremos con una puerta a mano derecha. Tomando esta puerta, se entrará en los dormitorios de los guardias nazis del campo. En el primer dormitorio se encuentra otro objeto especial: la linterna, sin ella no se podrá entrar en los túneles. Obsérvese ésta en la **Figura 2.34**.

Figura 2.34 Dormitorio de los guardias con el objeto especial linterna.

Por último, si en vez de tomar a la derecha en la cuarta habitación, se sigue hacia delante, se entrará en la habitación que da a la puerta número cuatro del patio exterior. En esta habitación la puerta que se sigue en la misma dirección de por la que hemos venido da acceso al despacho del comandante del campo. No obstante, la puerta está cerrada y no existe llave para abrirla; de modo que para entrar en ella habrá que utilizar un objeto especial: las herramientas, que son capaces de abrir cualquier puerta del campo, pero invirtiendo para ello cierto tiempo.

En el despacho del comandante se encuentra otro objeto especial: los documentos, es uno de los dos objetos que puede llevar *Steve* cuando intente la fuga, para que no sea apresado y devuelto al campo. La **Figura 2.35** muestra la puerta de entrada al despacho y el interior de éste con el objeto documentos.

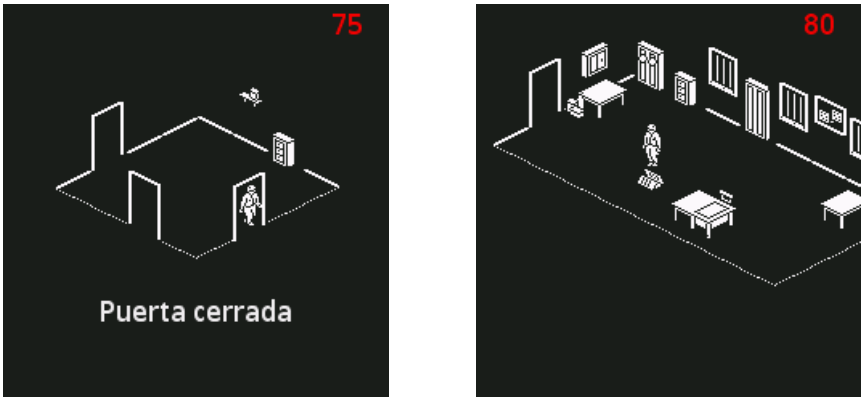


Figura 2.35 Puerta de entrada al despacho e interior de éste con el objeto documentos.

La habitación especial de botiquín o de la Cruz Roja es donde se entregan al personaje principal los paquetes de la Cruz Roja que ocultan objetos especiales, útiles para realizar la fuga. Los objetos que se entregan son: las tenazas, la bolsa y la brújula. En la siguiente figura, **Figura 2.36** se puede ver esta habitación.

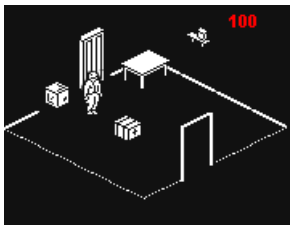


Figura 2.36 Habitación botiquín o de la Cruz Roja con objeto de la Cruz Roja entregado.

El otro grupo de habitaciones conectadas entre sí corresponde a la entrada por el patio exterior de la puerta cinco (véase mapa patio exterior **Figura 2.21** y mapa de las habitaciones

interiores **Figura 2.30**). La puerta identificada con el número cinco en los mapas está cerrada, para abrirla como ya se ha comentado hay que utilizar la Llave Verde ubicada en la esquina sureste del campo al lado de la torre de vigilancia. Usando esta llave conseguiremos entrar al interior de una habitación que nos sirve de corredor de acceso a dos más. La **Figura 2.37** muestra una composición de este grupo de habitaciones.

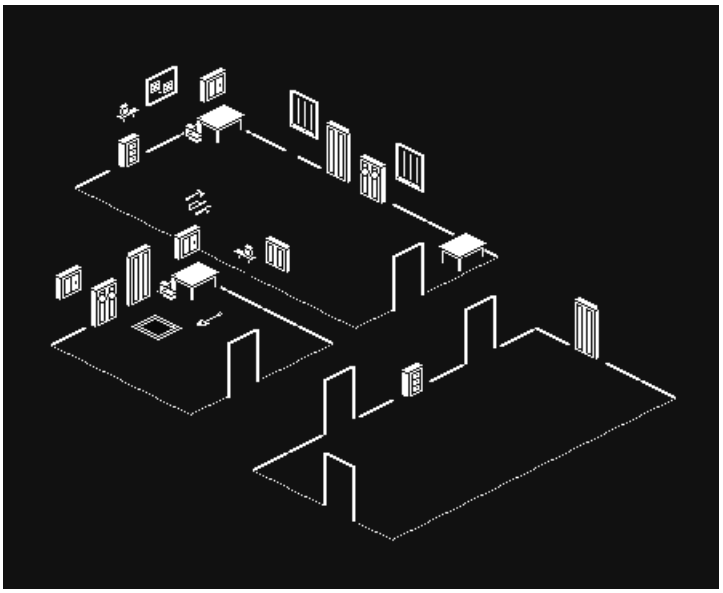


Figura 2.37 Composición del grupo de habitaciones de la puerta cinco.

Como se aprecia en la composición de la **Figura 2.37** la segunda habitación a la que da acceso la habitación corredor posee un objeto especial: se trata de las herramientas, capaces de abrir cualquier puerta cerrada y, por tanto, muy útiles para investigar los interiores del castillo. También la primera habitación posee otro objeto de gran utilidad: la pala, sin ella no podremos abrir la entrada a ciertos túneles ni remover los posibles derrumbes que nos encontremos en el interior de éstos. Debe destacarse que la puerta que da acceso a la habitación en la que se encuentra la pala está cerrada y no existe llave por lo que se deberá utilizar las herramientas que se encuentran en la habitación contigua.

Por último, para terminar este apartado, sólo queda hablar de los túneles. Existen dos túneles en el videojuego. En los mapas de la **Figura 2.21** y **Figura 2.30** se refieren por Túnel A y Túnel B. El Túnel A comunica el dormitorio de *Steve* en el segundo barracón con el patio de ejercicios. Mientras que el Túnel B comunica la primera habitación a la izquierda en el corredor al que da entrada la puerta del patio exterior cinco, con el interior del doble vallado en el lado Este. Usando la visión de diseño se puede observar dónde se encuentra la entrada de cada túnel en su respectivo escenario. Véanse las **Figuras** desde la **2.38,** a la **2.44** donde se observan los

escenarios conectados y las entradas a los túneles por una marca rectangular en el suelo de color azul.

- Túnel A:

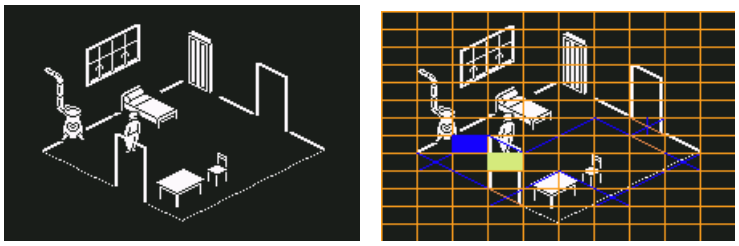


Figura 2.38 Entrada túnel desde dormitorio *Steve*.

Debe destacarse que para entrar en un túnel se hace necesario que *Steve* lleve consigo el objeto linterna. Si este objeto no fuera uno de los dos objetos que puede llevar con él, la aplicación no le dejaría entrar en el túnel, mostrando un mensaje por pantalla donde le indica que no puede entrar en el túnel. La **Figura 2.39** muestra este mensaje.

Figura 2.39 Mensaje de no puede entrar en túnel por no llevar linterna.

La otra entrada de este túnel como se indicó se encuentra en el patio de ejercicio. Para identificarla mejor se dibuja en su ubicación una salida de alcantarillado. En la **Figura 2.40** podemos apreciarla.

Cuando la salida del túnel se encuentra bloqueada por un derrumbe, como se aprecia en la imagen **a)** de la **Figura 2.41**, *Steve* debe utilizar la pala para removerlo; en este caso aparecerá un mensaje en el que se indica que *Steve* está picando piedra, véase la imagen **b)** de la **Figura 2.41**.

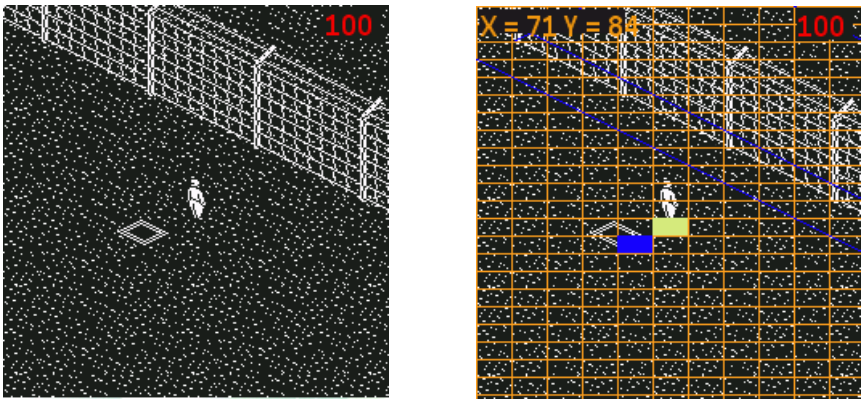
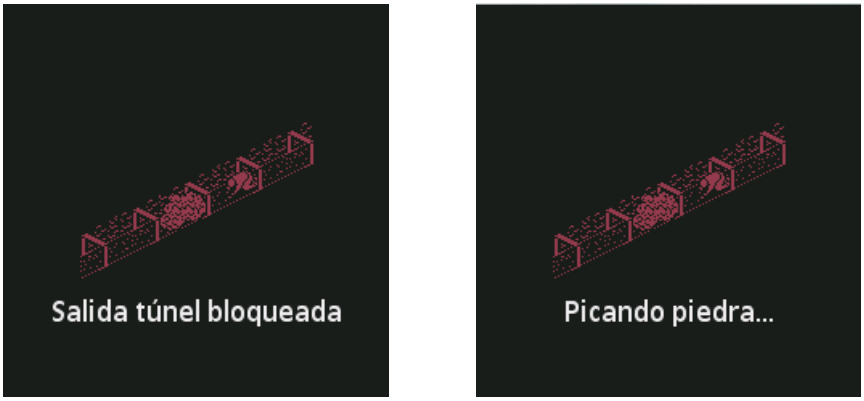


Figura 2.40 Entrada al Túnel A desde el patio de ejercicio.



a) Salida túnel bloqueada. **b)** Usando la pala para remover.

Figura 2.41 Imagen **a)** salida del túnel bloqueada por derrumbe y **b)** uso pala en túnel.

- Túnel B:

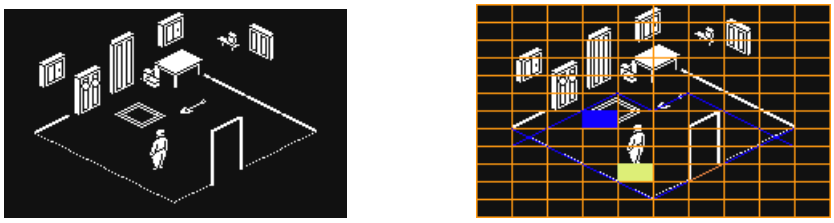


Figura 2.42 Entrada al túnel desde habitación contigua a la de las herramientas.

Debe señalarse que la entrada mostrada en la **Figura 2.42** está bloqueada- este hecho se le comunicará al jugador a través de un mensaje por pantalla cuando pase por encima de la entrada-, por lo que debe utilizarse la pala que se encuentra en esta habitación para abrirla. En la siguiente figura se muestra lo anterior, **Figura 2.43**.

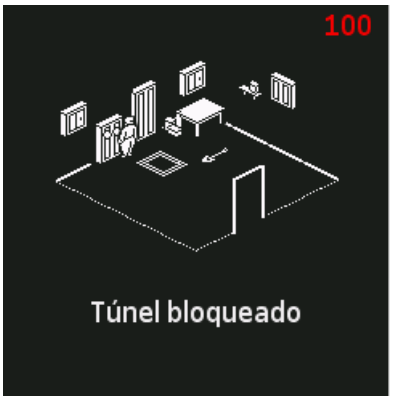


Figura 2.43 *Steve* intentando entrar en túnel bloqueado.

La otra entrada del Túnel B se encuentra en el interior de la doble valla en el lado Este. La **Figura 2.44** muestra esa entrada.

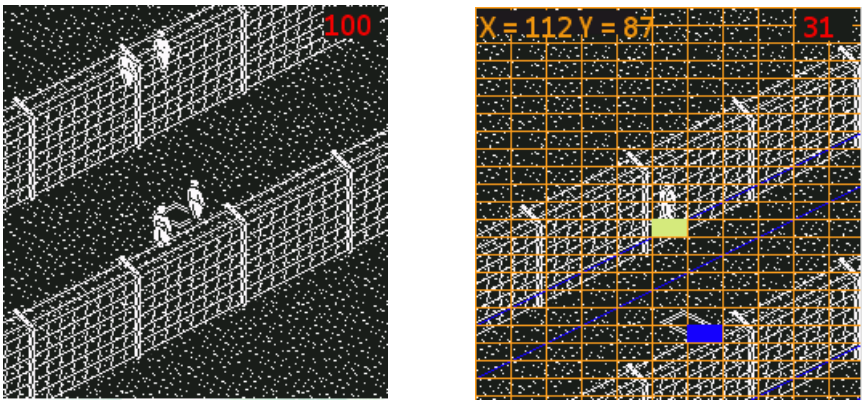


Figura 2.44 Entrada Túnel B en el interior de la doble valla.

2.5 Personajes

Existen tres tipos de personajes en la aplicación de videojuego desarrollada: el comandante del campo de concentración –también conocido como "*El General*"-, los soldados nazis que patrullan por el campo y los prisioneros de guerra, tipo al que pertenece el personaje principal, *Steve McQueen*.

- *Comandante:*

En la **Figura 2.45** se aprecia el *sprite* que constituye a este personaje. Es el guardia del campo de mayor graduación. Existe un único comandante en el campo de concentración. Una de sus características, y lo que lo hace especialmente peligroso, es que se trata del tipo de guardia

con mayor campo de visión, por este motivo *Steve* debe tratar de evitarlo cuando esté realizando algún intento de fuga.

Figura 2.45 *Sprite* del comandante.

Otra de sus características consiste en realizar los recuentos de prisioneros. Cuando se realiza un llamamiento a formar, este personaje acude a la zona de formación a pasar revista. Igualmente cuando se hace el llamamiento a comer, este personaje acude a la puerta del comedor permaneciendo frente a ella para controlar el acceso al mismo. Y, finalmente, cuando se llama a realizar ejercicios, acude al patio de ejercicio permaneciendo en él hasta que se cumple el tiempo asignado a los prisioneros para que estén en esa zona. El resto del tiempo se pasea por el patio exterior vigilando que todo esté en condiciones y que no se esté produciendo nada extraño.

- *Soldados:*

Existen cinco soldados, que junto con el comandante del campo constituyen la dotación de guardias del campo de concentración. En la **Figura 2.46** se puede ver el *sprite* de este tipo de personaje.

Figura 2.46 *Sprite* de los soldados.

Los diferentes soldados se distribuyen en diversas tareas de patrullaje por el campo. Tres de ellos, tienen un itinerario de patrulla fijo: dos por el interior del doble vallado, estando cada uno en uno de los lados de este vallado patrullándolo en camino de ida y vuelta, y el tercero, patrullando la esquina noroeste de la pared del castillo, en la zona de las puertas que van del número uno al cuatro, según el mapa de escenario de la **Figura 2.21**.

Los otros dos restantes tienen movimientos más libres. En general se encontrarán patrullando por el patio exterior libremente, si bien, cuando se realice llamamiento a formar o comer, éstos irán en un primer momento a la zona donde se desarrolla la acción del llamamiento para después reanudar su patrulla normal por la totalidad del patio. Cuando se realiza el llamamiento a ejercicio, estos dos soldados acompañan al comandante al patio de ejercicio; mientras uno de ellos se diferencia realizando una patrulla de ida y vuelta por el perímetro de la

valla Sur y Este, el comandante y el otro soldado se mueven aleatoriamente por el patio de ejercicio.

- *Prisioneros:*

Son seis los compañeros de desdichas de *Steve*; número que se corresponde con el total de camas de los dormitorios grandes de dos barracones. Estos personajes cumplen estrictamente sin desviarse un ápice la rutina temporal de un día en el campo y pueden servir al jugador poco iniciado para ir descubriendo las diferentes zonas del campo de prisioneros. En la **Figura 2.47** se muestra el *sprite* de los prisioneros.

Figura 2.47 *Sprite* de los prisioneros.

El personaje principal del juego controlado por el jugador, *Steve McQueen*, pertenece a este tipo de personaje.

2.6 Guión temporal: Un día en el campo de prisioneros.

El juego comenzará con *Steve McQueen* acostado en su dormitorio. *Steve* aparecerá tumbado en la única cama de ese dormitorio, como se muestra en la **Figura 2.48**. Éste se encuentra en uno de los tres barracones del campo -en concreto, el que está ubicado en mitad del patio exterior del castillo fortaleza. Sonará la campana -ésta siempre suena cuando ocurre un evento temporal que se muestre por pantalla-, indicándole que debe levantarse y prepararse para empezar un nuevo día en el campo de prisioneros de guerra.

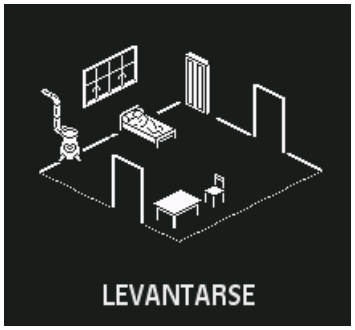


Figura 2.48 *Steve* acostado en su dormitorio, e indicándole la aplicación que se levante.

Este dormitorio será su refugio dentro del recinto vallado. En él, podrá esconder los objetos que vaya encontrando por el campo y que puedan servirle de ayuda para la fuga, sin que los soldados nazis den con ellos en los registros que realizan todos los días al amanecer. En la figura mostrada a continuación –**Figura 2.49**- se ve a *Steve* en su dormitorio con tres objetos escondidos -para que queden escondidos basta con que los deje en el suelo del dormitorio.

Figura 2.49 *Steve* en su dormitorio con tres objetos escondidos.

En la habitación de al lado del barracón, se encontrará otro dormitorio con tres camas (véase la **Figura 2.50**). En ellas han estado durmiendo tres prisioneros compañeros de penalidades de *Steve*, que ya han abandonado el dormitorio cuando sonó la campana y se encuentran pululando por el exterior de los barracones. Todos los barracones constan de dos dormitorios, al igual que éste, y dos puertas exteriores que dan, respectivamente, a cada dormitorio: una orientada al Sur y otra al Este.

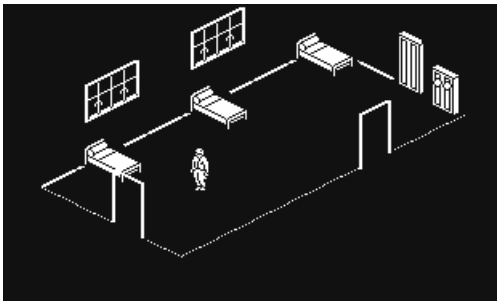


Figura 2.50 *McQueen* en el dormitorio de los compañeros de barracón.

En el barracón del Oeste, se encuentran los prisioneros que están más enfermos. Éstos permanecen todo el día en la cama sin fuerzas para moverse, véase el dormitorio principal de dicho barracón mostrado en la **Figura 2.51**. Al otro lado, en el barracón del Este, *Steve* tiene tres compañeros más que todavía tienen fuerza. Entre los prisioneros del barracón de *Steve* y del barracón del Este, en total, suman 7, los prisioneros, contando a *Steve*, que pueden todavía moverse.

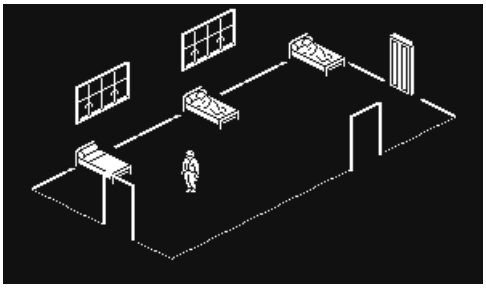


Figura 2.51 *Steve* en el dormitorio principal del barracón del oeste.

En la **Figura 2.52** se observa en el centro de la imagen el barracón de *Steve*, y parte de los otros dos barracones, el del Este y el del Oeste.

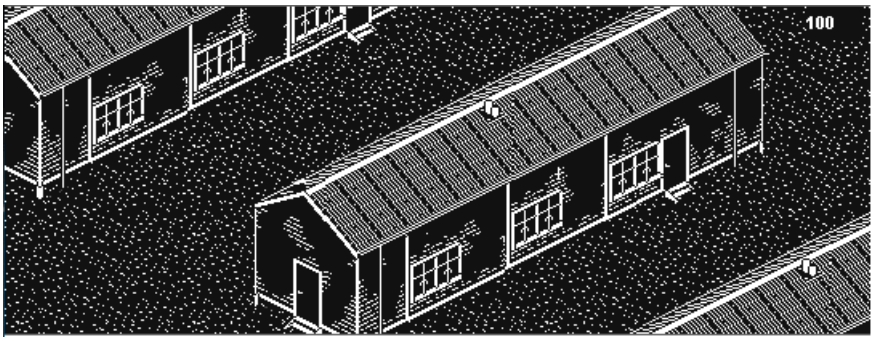


Figura 2.52 Barracón de *McQueen* en el centro de la imagen y parte de los dos restantes.

El espíritu de rebeldía de los compañeros de *Steve* ha sido subyugado por la atmósfera opresiva del campo y vagan por éste completamente desmoralizados, cumpliendo a rajatabla las estrictas normas de conducta que ha impuesto el comandante del campo, un despiadado esbirro de *Hitler*, miembro de la *Wehrmacht* y obsesionado por la seguridad de su campo de prisioneros, al que se le conoce por el sobrenombre de "El General". *Steve* se encuentra solo, ninguno de sus compañeros tiene la fuerza de ánimo suficiente para ayudarle en su intentona de fuga. Ellos tienen bastante con arrastrarse por el campo de concentración y sobrevivir. Pero *Steve* no se resigna, y pondrá todo de su parte para conseguir evadirse de aquel "agujero".

Steve podrá salir de la cama por el lado que no pega a la pared y moverse libremente por su dormitorio, como se muestra en la imagen de la **Figura 2.53**. Lo normal será que abandone el cuarto por la puerta Sur del mismo. La puerta Norte da al otro dormitorio del barracón.

Una vez cruce el umbral de la puerta Sur de su dormitorio, saldrá al exterior y verá la luz de un nuevo día sin esperanza en el campo de prisioneros. Tras levantarse, tendrá un tiempo de distensión para moverse por las zonas permitidas del campo de concentración. Pero este

breve *impasse* durará poco tiempo. En seguida, la atmósfera opresiva del campo se hará patente con un llamamiento a formar.

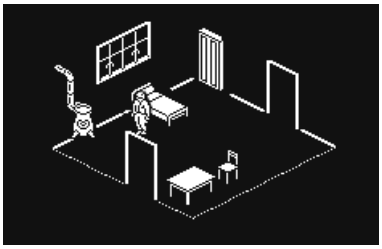


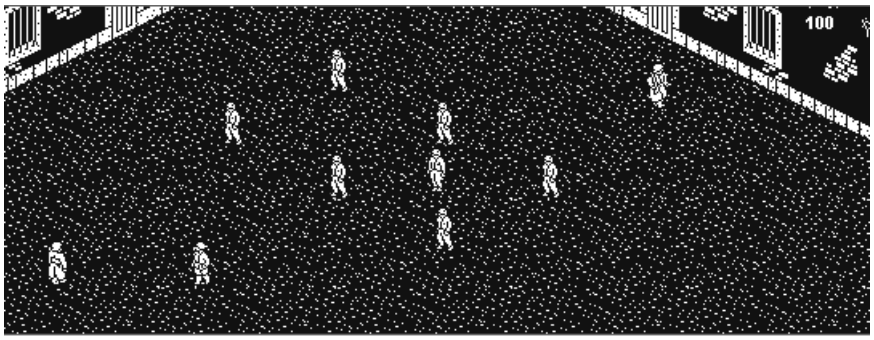
Figura 2.53 *McQueen* en el lado de su cama que no pega a la pared.

Steve y el resto de prisioneros del campo que pueden moverse tendrán un tiempo para ir a formar a una región determinada del campo. La zona de formación se encuentra en la esquina noroeste (Ver **Figura 2.54** y **Figura 2.55**).

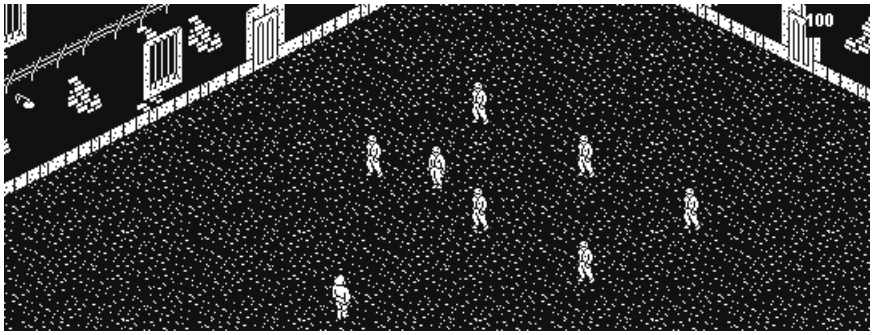
Figura 2.54 Esquina noroeste, zona de formación.

Transcurrido un tiempo, "El General", que ya se encontrará en la zona especificada, llamará a la formación de prisioneros a permanecer firmes y pasará revista a todos los prisioneros, girándose hacia ellos (ver secuencia de imágenes de la **Figura 2.55**). Si *Steve* no consiguiera llegar a la zona de formación antes de esta orden dada a los prisioneros, se encontraría en peligro de ser detenido por los soldados y el propio comandante que pasa revista, como se ve en la **Figura 2.55 b)**. (En el juego se comunica que está en peligro de ser detenido si es visto por un soldado poniendo a rojo el indicador de moral del jugador que aparece en la esquina superior derecha.) Al final de esta cita se hace recuento de prisioneros. A este respecto, si *Steve* faltara a 2 citas de recuento de las que hay a lo largo de la rutina del campo, sería considerado prisionero en rebeldía y estaría en peligro de ser detenido por los guardias del campo en cualquier momento.

.



a) Zona de formación con los prisioneros formando antes de que el comandante de la orden de firmes.



b) Zona de formación con *Steve*, sus compañeros prisioneros y "El General" después de haber dado la orden de firmes y girado hacia ellos para pasar revista.

Figura 2.55 Imágenes de la zona de formación antes y después de la orden de firmes.

Tras el recuento realizado en la zona de formación, en el campo se hace el llamamiento a comer. Los prisioneros tienen una única comida al día. Esto entra dentro de la estrategia de desgaste psicológico del despiadado comandante responsable del campo

Los prisioneros se dirigirán tras terminar el tiempo de formación al comedor. Al comedor se accede por una puerta del castillo situada en la zona oeste del campo, cercana al muro oeste (puerta que se muestra en la **Figura 2.56**).

Al igual que en la formación, el personaje principal, *Steve*, tendrá un tiempo para llegar al comedor, si se entretuviera y no llegara a tiempo, estaría en peligro de ser apresado por los guardias del campo.

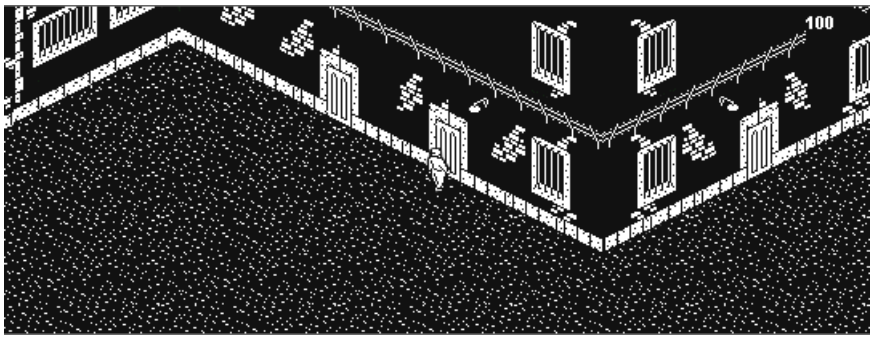
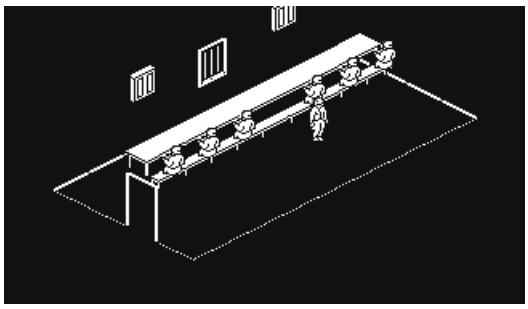
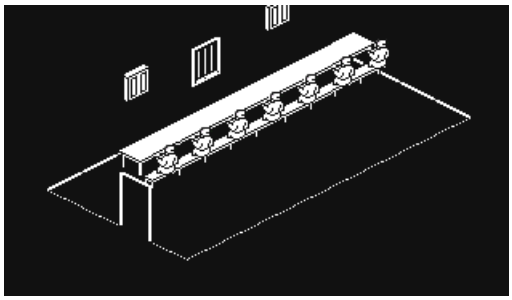


Figura 2.56 Puerta de entrada al comedor, la puerta a su izquierda es la de la celda.

La habitación del comedor consta de una larga mesa donde se van sentando los prisioneros a medida que van llegando al mismo, como se desprende de la **Figura 2.57 a)** y **b)**. No existe orden preestablecido de asiento de forma que *Steve* puede elegir donde sentarse, e, incluso, puede levantarse de la mesa cuando desee y moverse por la sala, siempre que no la abandone no suscitará la sospecha del comandante ni de sus secuaces, que merodearán por las cercanías del comedor.



a) *Steve* levantado y sus compañeros sentados a la mesa del comedor.



b) *Steve* y sus compañeros sentados a la mesa del comedor.

Figura 2.57 Imágenes de la habitación de comedor.

"El General" haciendo patente su obsesión por la seguridad del campo permanecerá en la puerta del comedor (como se muestra en la imagen de la **Figura 2.58**), para, una vez cumplido el tiempo asignado de comida, hacer recuento de los prisioneros. *Steve* debe tratar de

no faltar demasiado a las citas de recuento porque, si no, se encontraría en una situación difícil con todos los soldados nazis del campo buscándole y estrechándole el cerco.

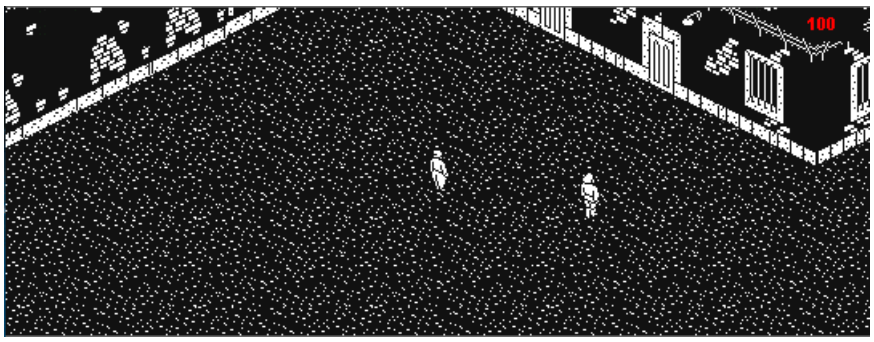
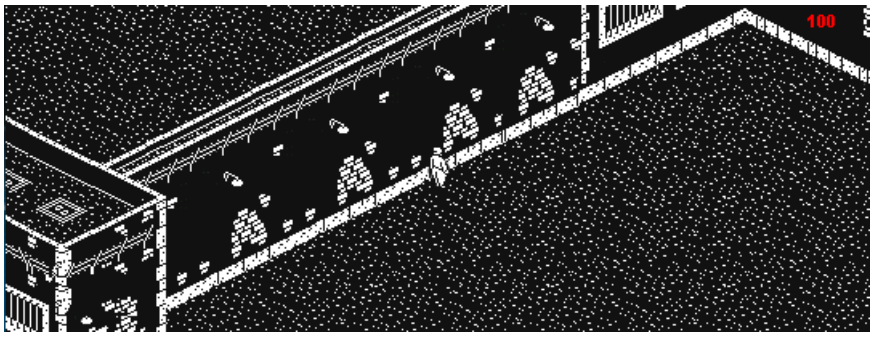


Figura 2.58 *Steve* colocado *sigilosamente* detrás de "El General" en la guardia.

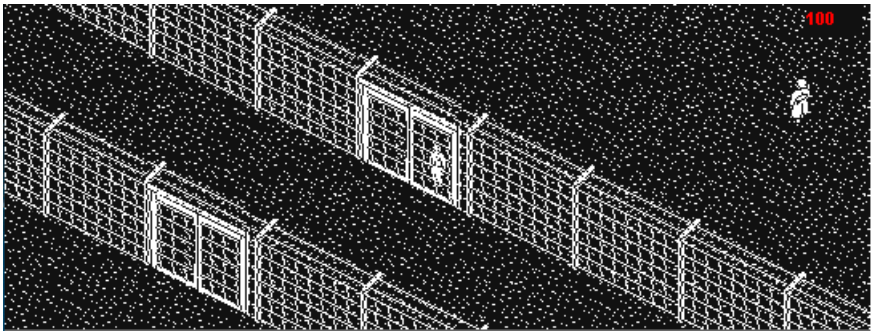
A continuación, tras la comida los prisioneros tienen un tiempo en el que pueden dar un paseo por el patio de la fortaleza. En este tiempo de libre movimiento, los compañeros de *Steve* aprovechan para andar un poco e intentar olvidar su reclusión forzosa. *Steve* puede aprovechar para investigar el campo de concentración, ver los puntos débiles de la vigilancia e ir descubriendo los distintos objetos que se hayan escondidos por él y que pueden servirle para conseguir su objetivo de fugarse.

No obstante, *Steve* debe andarse con cuidado. En el patio del castillo fortaleza existen algunas zonas prohibidas para los prisioneros, a las cuales no deben acercarse, bajo el riesgo de ser detenidos y conducidos a la celda de castigo, pero la osadía de *Steve*, sin duda, le conducirá a infringir estas normas.

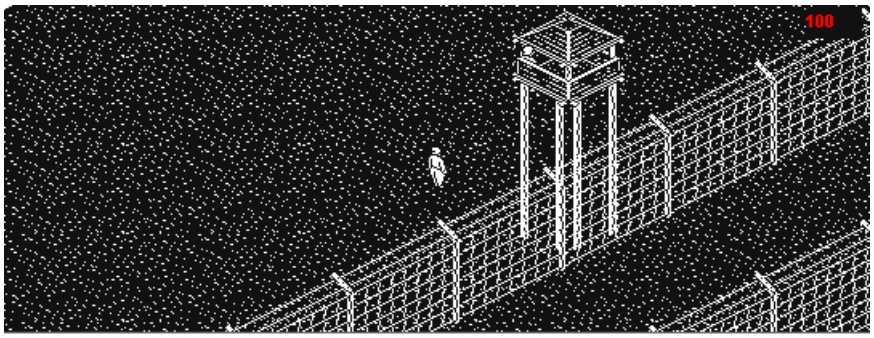
Las zonas prohibidas en el patio exterior del castillo son la franja de unos ocho pasos que pega a la doble valla en el Este, el acercarse demasiado a la valla Sur que da al patio de ejercicios, otra franja de unos cinco pasos en el muro Oeste y el ser descubierto cruzando las puertas de la intendencia nazi que se encuentran en la pared Norte y Oeste. En la secuencia de imágenes de la **Figura 2.59** se observan las distintas zonas señaladas; no obstante, se remite al lector a un apartado posterior dedicado a las zonas prohibidas. Si *Steve* fuera descubierto en algunas de estas zonas prohibidas por un soldado del campo, éste le seguiría conminándolo a que la abandonara, de modo que si *Steve* insistiera en permanecer en dicha zona, al darle alcance el guardia, lo detendría, llevándolo a la celda de castigo.



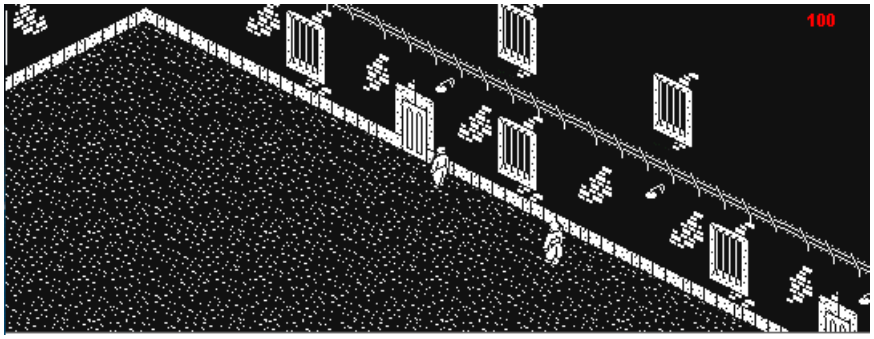
a) Zona prohibida del muro este.



b) Zona prohibida de la valla sur.



c) Zona prohibida de la valla este.

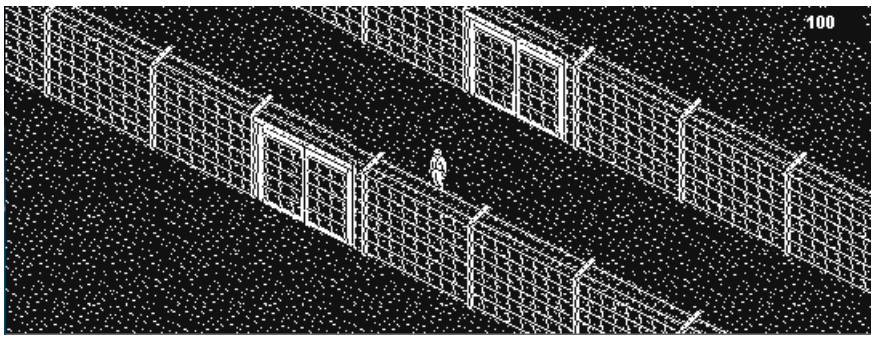


d) Zona prohibida puertas de intendencia al norte del patio, con un soldado de patrulla.

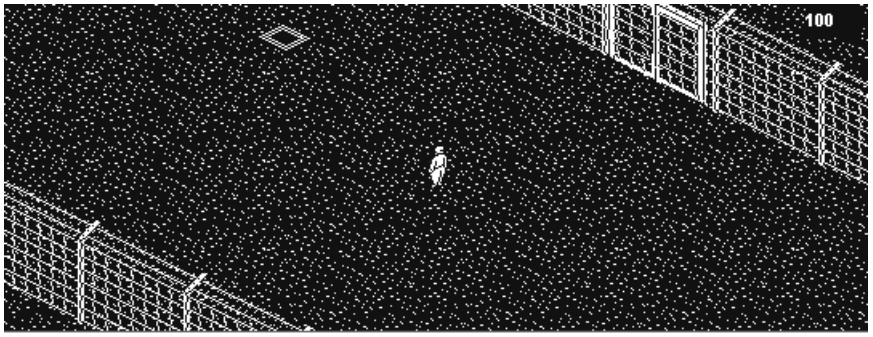
Figura 2.59 Imágenes **a)**, **b)**, **c)** y **d)** de zonas prohibidas definidas en el patio exterior.

En la rutina del campo al tiempo de libre movimiento anterior le sigue el período de ejercicio. Una vez se realiza el llamamiento para ejercicio, la puerta de la doble valla que da al campo de ejercicios (en la **Figura 2.60** se muestra la entrada por la doble valla y el campo de

ejercicios propiamente) por el sur se abrirá y los prisioneros se dirigirán a él para realizar un poco de actividad física.



a) *Steve* cruzando la doble valla, para acceder al patio de ejercicios.



b) *McQueen* en el centro del patio de ejercicios.

Figura 2.60 Imágenes **a)** entrada en la doble valla y **b)** el patio de ejercicios.

Steve tendrá un tiempo para ir al campo de ejercicios, transcurrido éste, si no se hallara en el recinto del campo de ejercicios se encontraría, al igual que pasaba para el tiempo de formar al amanecer y de comer, en peligro de ser detenido por los guardias del campo de concentración, con lo que de desmoralizante tiene el verse en la celda de castigo, sin luz y sólo con el sustento de pan y agua, (véase la celda de castigo en la **Figura 2.61**).

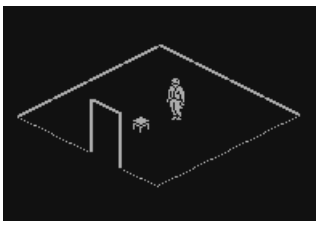


Figura 2.61 *Steve* en la celda de castigo.

Cuando se cumpla el tiempo asignado para hacer ejercicio físico en el campo de ejercicios, se realizará otra vez recuento de prisioneros. Durante el tiempo de estar en el campo de ejercicio “El General”, estará dentro de éste, observando a los prisioneros, como se observa

en la **Figura 2.62**. Los prisioneros no deben acercarse a la valla del campo de ejercicio, siendo detenidos si esto se produjera, ya que en este punto del campo de concentración sólo hay un vallado simple, al contrario de lo que ocurre en el resto del campo, circundado por un doble vallado con soldados nazis patrullando por su interior. Para aumentar la seguridad en el campo de ejercicio el comandante del campo ha establecido que un soldado patrulle el perímetro más débil de valla mientras los prisioneros estén en él haciendo ejercicio y que otro soldado vigile la puerta de la doble valla que le sirve de entrada.

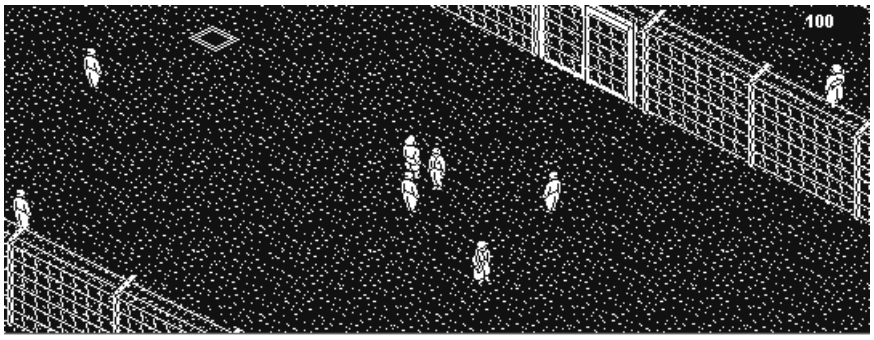


Figura 2.62 *Steve* en el centro del patio de ejercicios.

El siguiente evento que se comunica a los prisioneros en la rutina diaria del campo es el de abandonar el campo de ejercicios. A los prisioneros se les concede un intervalo de tiempo en el que deben abandonar el campo de ejercicios; si transcurrido este tiempo *Steve* no abandonara el campo de ejercicios podría ser detenido por los guardias que se quedan en el campo de ejercicios comprobando que todos los prisioneros lo hayan abandonado. Los doblegados compañeros prisioneros de *Steve* abandonarán el campo de ejercicios sin oponer resistencia, sólo *Steve* puede intentar alguna resistencia *en pro* de su objetivo de fuga.

Al evento de abandonar el campo de ejercicios sigue un tiempo de libre movimiento por el patio exterior, igual que el que se tiene al amanecer y levantarse *Steve*, y al terminar el tiempo de comer. A este período de distensión y de libre movimiento por el campo sigue, otra vez, otro llamamiento a formar en la esquina noroeste del patio, idéntico al que se produce poco después de amanecer. En este tiempo de formar se hace el último recuento de prisioneros en el campo, antes de que se les envíe de vuelta a sus barracones para dormir.

Tras la última formación en la esquina noroeste del campo, los prisioneros tendrán que irse a dormir a sus respectivos barracones. Cada uno de los compañeros de *Steve* se dirigirá a la puerta de su dormitorio, en el barracón asignado, cruzándola y acostándose en su cama para dormir (véase figura siguiente, **Figura 2.63**). *Steve* deberá dirigirse también a su dormitorio en

el segundo barracón y acostarse en su cama, ya que al poco tiempo de indicárseles a los prisioneros que se vayan a dormir, se iniciará el toque de queda de la noche.

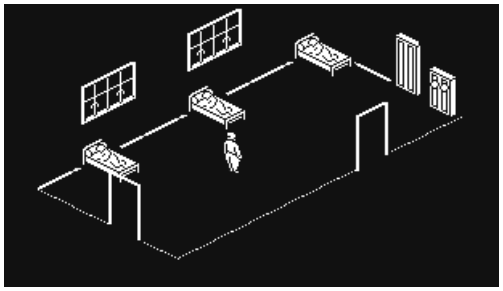
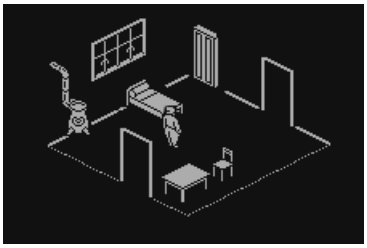


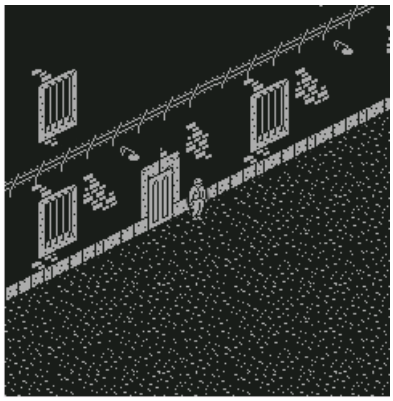
Figura 2.63 *Steve* en el dormitorio de sus compañeros una vez éstos se han acostado.

Poco antes de anochecer se inicia el toque de queda nocturno en el campo de prisioneros, quedando totalmente prohibido el moverse por el exterior de los barracones. Si *Steve* fuera descubierto en el exterior de su barracón sería arrestado por los guardias que patrullan por el campo por la noche.

Los prisioneros son mandados a dormir poco antes del anochecer. Cuando la noche cae sobre el campo la oscuridad se apodera de todo, incluso, en el interior del castillo y de los barracones, mal iluminados para evitar bombardeos nocturnos, esa oscuridad se hace patente (ver las imágenes de la **Figura 2.64**).



a) Dormitorio de *Steve* de noche, mal iluminado para evitar los bombardeos.



b) El capitán *McQueen* de noche ante la puerta de entrada primera de la pared oeste.

Figura 2.64 Imágenes nocturnas **a)** habitación interior y **b)** exterior.

La noche siempre es un buen momento para allanar el camino a la fuga. De esta idea es consciente el capitán *Steve*, y tratará de sacarle el máximo de partido para sus intereses y conseguir dejar atrás ese campo claustrofóbico donde el azaroso destino de la guerra le condujo. Mientras sus compañeros duermen tranquilamente en las camas de sus barracones *Steve* pondrá en marcha sus planes de fuga.

A la noche sigue el amanecer. Este será un período temporal en el que los prisioneros pueden moverse libremente por el patio exterior. Tras cada nuevo día los soldados nazis que patrullan por el campo realizarán una revisión exhaustiva del mismo, de modo que todo lo que no se ajuste al estado normal del campo será rectificado. Los objetos abandonados por *Steve* en zonas no seguras serán devueltos a su ubicación original, las puertas que *Steve* abriera a lo largo del día en su ir y venir por el campo se cerrarán, y los posibles desperfectos en la valla producidos por algún intento de fuga se repararán.

A modo de resumen del guión temporal mostrado, en el siguiente capítulo, se indicarán los diferentes eventos temporales, por los que transcurre la acción de un día en el campo de prisioneros para el capitán *Steve McQueen* donde para cada uno se muestra en detalle si se tiene que realizar alguna acción especial por parte de *Steve* y los demás personajes del juego. La intención es clarificar un poco más, si cabe, el eje de tiempo en el que se desarrolla la acción de un día en el videojuego desarrollado, ya que en lo visto hasta ahora se ha hecho más bien un relato continuo de lo que acontece en el día.

2.7 Lista de los eventos temporales

Lista pormenorizada de los eventos temporales en los que se desenvuelve la acción del juego:

- *Amanecer:*

  Steve podrá moverse libremente en el patio exterior por las zonas no prohibidas.

- *Levantarse:*

Si es el primer día *Steve*, aparece acostado en la cama de su dormitorio en el barracón central. Deberá levantarse de la cama y podrá moverse libremente en el patio exterior por las zonas no prohibidas.

- *Formar(en la mañana):*

  Steve deberá acudir a la zona de formación en la esquina noroeste del campo y permanecer allí hasta que el comandante pase revista.

- *Firmes (correspondiente al formar de la mañana):*

  Steve deberá permanecer en la zona de formación mientras no se indique otro evento por pantalla. Si estuviera fuera de dicha zona, corre el riesgo de ser detenido por los soldados del campo si lo ven. Al final de este evento se hace el control de citas para *Steve*. Si en ese momento *Steve* no estuviera en la zona de formación se le contaría que ha faltado a una cita, de las dos que se le permiten para poner en alerta a todos los guardias del campo. Si no se encuentra en la zona de formación puede intentar llegar hasta ella, aún cuando lo persigan, ya que si llegara a ella sin ser detenido lo dejarían en paz.

- *Comer:*

  Los prisioneros acudirán a la puerta del comedor y se introducirán en él, ocupando puestos correlativos según orden de llegada a la mesa del mismo. Transcurrido un tiempo *Steve* deberá estar en el interior del comedor, si no, corre el riesgo de ser detenido por los guardias del campo. "*El General*" se colocará en la puerta de entrada controlando una posible llegada de *Steve* a destiempo. Al final del tiempo de este evento, sólo si *Steve* no está en el interior del comedor se le contará como una falta a recuento.

- *Libre (que sigue a comer):*

  Ídem que para amanecer y levantarse, el personaje principal, controlado por el jugador, *Steve*, podrá moverse libremente por las zonas no prohibidas del patio exterior.

- *Ejercicio:*

  Steve deberá acudir al patio de ejercicios con el resto de prisioneros -la puerta del campo ejercicios sólo permanece abierta durante el tiempo de *ejercicio* y *fin ejercicio*, en el resto de eventos permanecerá cerrada. Tendrá un tiempo para llegar a él; una vez transcurrido, si no se encontrara en él, estaría en

riesgo de ser detenido. Al final de ejercicio se realiza control de asistencia a cita para *Steve*, como en los eventos de *formar* y *comer*.

- *Fin Ejercicio:*

  Una vez concluido el tiempo de ejercicio físico, se tendrá un tiempo para abandonar el patio de ejercicio a través de la puerta en la doble valla. Si no se abandonara el campo de ejercicio los soldados que patrullan por su interior podrían detener a *Steve*.

- *Libre (que sigue a fin ejercicio):*

  Ídem que para el otro *libre*, sólo que un soldado se queda patrullando por el patio de ejercicio para cerciorarse de que no ha quedado ningún prisionero en él.

- *Formar (antes que anochezca):*

  Ídem que para el otro *formar*, el que ocurre poco después que amanezca.

- *Firmes (antes que anochezca):*

  Ídem que para el otro *firmes*, el que ocurre poco después que amanezca.

- *Dormir:*

  Los prisioneros se dirigirán a la puerta de su dormitorio en el barracón correspondiente de los dos que habitan prisioneros que pueden moverse y se introducirán en él acostándose en las camas. *Steve* tendrá un tiempo para ir a su dormitorio en el segundo barracón, transcurrido éste, si se encontrara en el exterior, en el patio, correría el riesgo de ser apresado.

- *Noche:*

  En este tiempo se hará de noche en el castillo fortaleza. La noche se percibe tanto en el patio exterior como en el interior de las habitaciones. Todo el campo de prisioneros salvo el barracón de *Steve* es zona prohibida. Por la noche se da el toque de queda. Si *Steve* se encontrara en el exterior de su barracón, correría el riesgo de ser detenido, por los soldados y el comandante que patrullan por el exterior.

Una vez terminado el último evento temporal de la lista, se pasará a estar de nuevo en el primero, es decir, en *amanecer*. La única ruptura a esta continuidad en la sucesión de los eventos temporales de la lista sucede cuando *Steve* es detenido. Entonces, *Steve* pasa a estar en una celda oscura, sin luz, aislado, hasta que en el amanecer de un nuevo día –al jugador se le indicará que puede salir de la celda y que se trata de un nuevo día- coincidiendo con el evento temporal de la lista *levantarse*, le permiten que abandone la celda de castigo, véanse las imágenes de la figura siguiente, **Figura 2.65**.

a) Imagen de *Steve* en el momento de ser introducido en la celda de castigo.

b) Imagen de *Stevie* en la celda de castigo en el momento en que se le comunica que puede salir al exterior, una vez terminado su castigo.

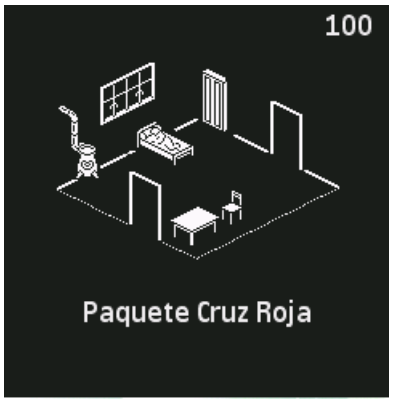
Figura 2.65 Situación en la celda de castigo.

2.8 Objetos en el campo de prisioneros

Por todo el campo de prisioneros se encuentran repartidos una serie de objetos que pueden servir de ayuda para la fuga. Estos objetos van desde llaves (existen dos llaves: la Llave Verde y la Llave Roja), una pala, unas herramientas, una linterna, unos documentos y una radio.

Además de estos objetos, los cuales deberá ir descubriendo *Steve* y colocando en un lugar seguro como su cuarto para que no sean descubiertos, existen otra serie de objetos que se irán recibiendo a través de la Cruz Roja.

Estos objetos son: una brújula, unas tenazas y una bolsa. Los objetos de la Cruz Roja serán entregados todas las mañanas, véase la imagen con el mensaje que lo indica en la **Figura 2.66**, al poco tiempo de que se dé la orden en el campo de levantarse a los prisioneros. Estos objetos vendrán empaquetados, pudiéndolos guardar *Steve* de esta forma, o desempaquetándolos -operación con la que se extraerá el objeto que viene oculto en el paquete de la Cruz Roja. Es el botiquín donde se recibirán estos paquetes de la Cruz Roja, la puerta tres de la pared Oeste del campo, fíjese en las imágenes donde se muestra de la **Figura 2.66**.



a) Imagen en la que se muestra mensaje de paquete de la Cruz Roja entregado.

b) Imagen de la puerta de entrada al botiquín de la Cruz Roja.

Figura 2.66 Imágenes mensaje Cruz Roja, entrada del botiquín de la Cruz Roja.

Si durante el registro que todos los amaneceres realizan los guardias del campo se descubriera un objeto de la cruz roja en un lugar no seguro, éste sería requisado. No obstante, la cruz roja volvería a enviar los objetos requisados en los siguientes días, así hasta que todos los

objetos se hubieran entregado, momento en el que ya no volverían a enviar más objetos, salvo que se requisaran por los guardias del campo en los registros que realizan al amanecer.

Para que la fuga tenga éxito el capitán *Steve* deberá portar consigo la brújula y uno de estos dos objetos: la bolsa o los documentos. *McQueen* sólo puede llevar en cada momento consigo un máximo de dos objetos, por lo que deberá tener una cuidadosa preparación de la fuga. A este respecto, en el exterior del vallado del campo existen unos escondrijos de roca, ya comentados en un apartado anterior, donde puede ocultar objetos (véase la **Figura 2.27** donde se muestran). Estos objetos escondidos entre las rocas del exterior al vallado no serán vistos por los soldados cuando se haga el registro del campo y de sus alrededores al amanecer de cada día.

A continuación, se detallará de forma más pormenorizada los diferentes objetos señalados en este apartado.

Lista de objetos especiales del campo de prisioneros, los cuales *Steve* puede recoger y servirse de ellos:

-  Llave: este objeto especial sirve para abrir puertas. Una llave concreta sólo abre una puerta. Para abrir una puerta *Steve* debe estar situado justo en frente de ésta y en el menú *Options*, seleccionar la entrada *Usar Llave* …. Existen dos llaves, ya comentadas en el apartado de explicación de los mapas: la Llave Verde y la Llave Roja (véase el **Apartado 2.4**, conjuntamente con las **Figuras 2.21**, **2.30** y **2.31**). En la imagen de la **Figura 2.31**, se muestra este objeto, en concreto se trata de la Llave Roja.

-  Pala: sirve para abrir las entradas a los túneles bloqueadas y en el interior de éstos remover los derrumbes que impiden al jugador salir por el otro extremo del túnel. Para usarla el jugador debe elegir en el menú de *Options* la opción Usar Pala. Sólo existe una pala en todo el campo de prisioneros (véase el **Apartado 2.4**, conjuntamente con la **Figura 2.30** y también la composición de la **Figura 2.37**).

-  Linterna: sin ella en el juego no se permite la entrada en los túneles. *Steve* debe llevarla consigo en el interior del túnel. Al igual que ocurre con la pala la primera vez que se introduzca en el túnel para poder abordar los derrumbes que bloqueen la salida. Sólo existe una linterna en el campo, la que se encuentra en el segundo dormitorio de

los guardias (véase el **Apartado 2.4**, conjuntamente con la **Figura 2.30** y la **Figura 2.34**).

-  Herramientas: con ellas el jugador puede abrir cualquier puerta cerrada en el campo de prisioneros, para ello deberá invertir un cierto tiempo. El modo de uso es igual que el de las llaves. Una vez colocado el personaje principal *Steve* en la puerta, el jugador deberá seleccionar Usar Herramientas, tras lo cual aparecerá un mensaje donde se indica que se está forzando la puerta. Durante el tiempo que dura la operación de forzar la puerta con las herramientas, *Steve*, quedará parado delante de la puerta, de modo que si es visto por algún guardia, éste se ira a detenerlo; motivo por el que cuando se utilicen las herramientas en el patio del castillo, el jugador debe tener el cuidado de que no haya cerca ningún guardia que pueda verle en algún momento. En la **Figura 2.67,** se muestra una imagen de uso de las herramientas. Existe un único objeto herramientas (véase el **Apartado 2.4**, conjuntamente con la composición de la **Figura 2.37**)

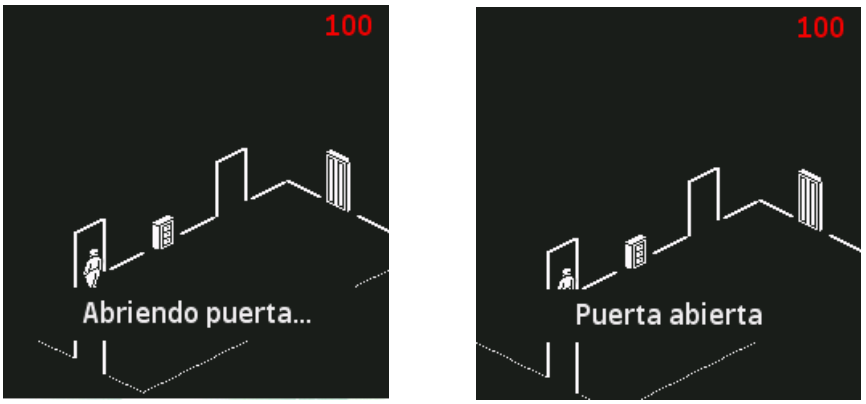


Figura 2.67 Imagen donde se muestra el uso de las herramientas.

-  Documentos: este objeto se encuentra en el despacho del comandante del campo y junto con la brújula constituye una de las combinaciones de objetos especiales que debe llevar *Steve*, cuando se escape del campo de prisioneros para que la fuga tenga éxito (véase el **Apartado 2.4**, conjuntamente con la **Figura 2.30** y la **Figura 2.35**).

-  Radio: este objeto actualmente no tiene ninguna utilidad, se incorpora para que pueda ser contemplado en la implementación de las líneas futuras del proyecto (véase el **Apartado 2.4**, conjuntamente con la **Figura 2.30** y la **Figura 2.33**).

-  Tenazas: objeto entregado por la Cruz Roja en un paquete (para los objetos entregados por la Cruz Roja véase el **Apartado 2.4**, conjuntamente con la **Figura 2.30** y la **Figura 2.36**). Este objeto es indispensable para realizar la fuga, ya que en cualquiera de las opciones de fuga existentes que tome el jugador al final siempre tendrá que romper al menos una vez la valla de alambre de espino que rodea el campo. En la **Figura 2.14** se muestra un ejemplo de uso de las tenazas.

-  Bolsa: otro objeto entregado por la Cruz Roja y que conjuntamente con la brújula forma el otro tándem de objetos que debe llevar *Steve* para conseguir la fuga.

-  Brújula: el último objeto que entrega la Cruz Roja e indispensable para la fuga, ya que si se quiere que la fuga tenga éxito, *Steve* debe llevarla consigo conjuntamente con la bolsa o los documentos.

-  Paquete Cruz Roja: se trata de un objeto envoltorio donde se entregan los objetos de la Cruz Roja, al usarlo a través del menú *Options* con la opción Usar Paquete <*Objeto*>, el paquete se abre y se deposita su contenido en el suelo (véase al respecto la **Figura 2.68**).

Figura 2.68 Imágenes donde se muestra el uso de un paquete de la Cruz Roja.

2.9 Zonas prohibidas.

En este apartado se indicarán las distintas zonas prohibidas. Lo primero que debe señalarse es que todas las habitaciones interiores del juego, salvo las del segundo barracón, el barracón de *Steve* y la habitación de comedor, están prohibidas. Por tanto, en todas ellas aparecerá el indicador de moral en rojo. La segunda cosa a señalar serían las zonas prohibidas del patio del castillo; para ello se va a utilizar el mapa ya visto en el apartado dedicado a los mapas de los escenarios. En la **Figura 2.69** se muestran las zonas prohibidas en rojo, en un momento normal del juego. Destáquese de la figura que las zonas con un color rojo más claro representan zonas prohibidas de anchura de una celda.

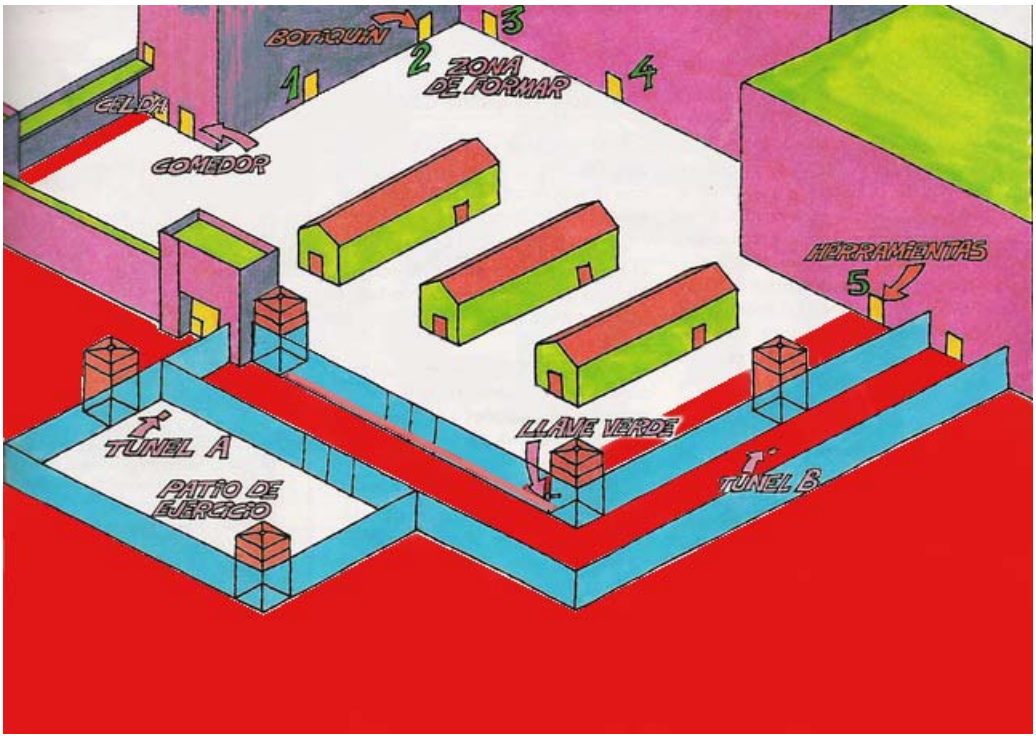


Figura 2.69 Zonas prohibidas en color rojo y líneas rosas: momento normal en el campo.

Dependiendo del evento temporal que esté en curso en el juego las zonas prohibidas van cambiando para amoldarse a las exigencias del guión. Cuando se hace el llamamiento a firmes, tras llamar a formar, el campo de prisioneros se vuelve entero una zona prohibida salvo la región de formar, donde debe encontrarse *Steve* si no quiere ser detenido.

En los eventos temporales: comer, dormir y noche ocurre otro tanto, pero en este caso es todo el campo el que se convierte en zona prohibida, sólo quedando la posibilidad de que *Steve* permanezca en el comedor, si se trata de comer, o en su barracón, si se trata de dormir o noche,

ya que el resto del campo estará considerado como zona prohibida para *Steve* durante esos eventos.

La situación de las zonas prohibidas adquiere una distribución especial durante el llamamiento a realizar ejercicio. Durante el tiempo que tienen los personajes para ir al patio de ejercicios se habilita un pasillo a través de la entrada en la doble valla, quedando el patio de ejercicio como zona también habilitada. En la **Figura 2.70** se observa sobre el mapa de partida lo anteriormente comentado.

Una vez transcurrido el tiempo asignado para que los personajes accedan al patio de ejercicios, todo lo que es el patio exterior se convertirá en zona prohibida, quedando como única zona habilitada para que pueda estar *Steve*: el patio de ejercicio.

Al finalizar el tiempo de ejercicio se volverán a habilitar las zonas indicadas en la **Figura 2.70** para que el personaje pueda abandonar el patio de ejercicio. A este período de tiempo corresponde el evento temporal Fin Ejercicio.

Cuando el evento temporal anterior, Fin Ejercicio, concluya la situación del campo de prisioneros en cuanto a las zonas prohibidas volverá a la representada en la **Figura 2.69**.

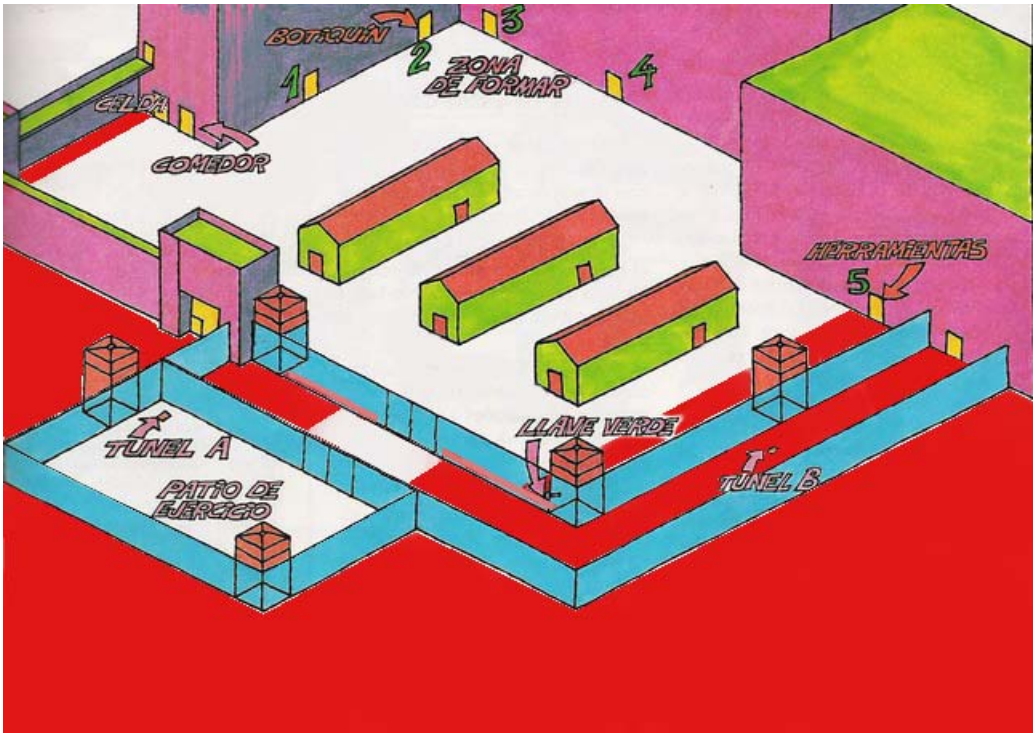


Figura 2.70 Zonas prohibidas durante el tiempo asignado para acceder al patio de ejercicio.

Capítulo 3.
Diseño lógico de la aplicación

3.1 Introducción

En este capítulo se abordarán los detalles de diseño a alto nivel de la aplicación. Tras el capítulo anterior donde se expone la descripción de la aplicación de videojuego, este capítulo profundizará en los detalles lógicos que permiten el desarrollo del mismo. Los detalles de implementación en el lenguaje de programación Java, tales como: los diagramas de clases, las relaciones entre ellas, los métodos, etc …, se reservarán para el siguiente capítulo.

Una idea importante que se abordará es la forma de codificación de la información en el videojuego: información que lo define. Gran parte de la lógica del videojuego se encuentra en unos ficheros de configuración XML que son leídos por la aplicación Java para generar el juego. Dentro de este aspecto importante del juego, que son los ficheros de configuración XML, se entrará a describir su estructura, así como las diferentes etiquetas y atributos que los constituyen.

Esta configurabilidad a través de ficheros externos XML de gran parte de la lógica de la aplicación, llevó a una mayor complejidad a la hora de la programación y desarrollo, ya que muchos aspectos de la lógica del juego son tratados desde un punto de vista de generalización. Este hecho, sin duda, constituyó un reto de gran esfuerzo y trabajo, junto con el desarrollo de las restantes facetas del juego como: los motores gráficos y de movimiento, la estructura *multithread* interna de la aplicación y los controladores de situaciones concretas para el juego, del tipo de llevar a dormir o a comer a los personajes secundarios.

3.2 Estructura de la aplicación

Como ya se ha comentado en la introducción, gran parte de la lógica del juego se encuentra en ficheros de configuración XML. Estos ficheros son leídos por la aplicación Java, la cual, utilizando también cierta información contenida en su propio código –esta información es

de una proporción mucho menor comparada con la que proporcionan los ficheros XML-, construirá el juego.

La **Figura 3.1** muestra un esquema donde se indica la estructura de motor gráfico configurable mediante ficheros XML de la aplicación de videojuego para el teléfono móvil.

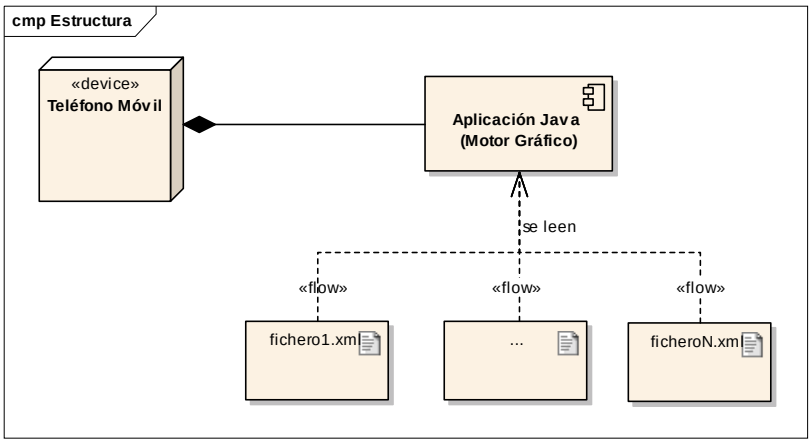


Figura 3.1 Estructura de motor gráfico configurable, en parte, mediante ficheros XML.

Como se aprecia en la figura anterior, existe un flujo de información desde los ficheros XML a la aplicación Java, mediante la lectura por parte de la aplicación de los ficheros, conformando gracias a esa información el videojuego que se muestra por el dispositivo móvil.

Existen diversos tipos de ficheros de configuración, cada uno dedicado a un ámbito concreto de la lógica de la aplicación. Estos ficheros son los siguientes:

- *mapadibujoexterior.xml*: se encarga de especificar el dibujo, por pantalla del escenario patio exterior del castillo, así como de la ubicación de los diferentes objetos de partida deseados en el patio exterior.
- *mapafronteraexterior.xml*: especifica las fronteras de colisión con los contornos del escenario del patio exterior del castillo: paredes, muros, barracones, vallas, etc…
- *mapapuertasexterior.xml*: indica las diferentes puertas que comunican con escenarios interiores –habitaciones-, en el escenario del patio exterior del castillo fortaleza.
- *tuneles.xml*: se especifican los túneles existentes en el campo de prisioneros, indicándose la ubicación de las dos entradas –en qué escenarios se encuentra cada una: habitación interior o patio exterior-, si la entrada está bloqueada por derrumbe

o no, el sentido de entrada al túnel desde el escenario superior patio exterior o habitación.

- *personajes.xml*: en este XML se codifican los diferentes personajes que existen controlados por la aplicación, así como la respuesta particular a determinados eventos temporales si así se fijara.
- *tiempo.xml*: fichero que codifica toda la información temporal de los distintos eventos temporales que se darán en la aplicación, junto con las respuestas de los diferentes tipos de personajes a dichos eventos.
- *zonasprohibidas.xml*: en este fichero se definen las diferentes zonas prohibidas y cómo cambian conforme el hilo temporal se desarrolla.
- *cruzroja.xml*: en él se codifica la información de los objetos a entregar, junto con el momento del día en que se realiza, por la Cruz Roja en la habitación de Botiquín.
- *ficheroshabitacionesinteriores.xml*: existen un total de veinticinco habitaciones; cada una lleva el nombre único de su habitación por nombre de fichero, si bien todas comparten la estructura de XML de *ficherohabitacionesinteriores*, donde se especifican: el dibujo de la habitación, los objetos que se encuentran en ella de partida, las fronteras de colisión con los límites de la habitación y mobiliario, las puertas que enlazan con otras habitaciones o el patio exterior.

Estos ficheros codifican prácticamente la totalidad de la información que define el videojuego, salvo algunos pequeños detalles que han quedado embebidos en el código. Durante las intensas pruebas que se llevaron a cabo para construir la compleja lógica del juego junto con los escenarios, la idea de tener un motor gráfico configurable a través de recursos externos, como son los ficheros XML, surgió como una necesidad imprescindible para llevar a buen fin este proyecto.

En cierta manera, estos ficheros XML con su estructura, junto con sus etiquetas, salvando siempre las distancias, conforman una especie de lenguaje de etiquetas que permite la construcción de un juego en perspectiva isométrica 3D de características similares a las del actual: un escenario principal, como es en este caso el patio exterior del castillo, una serie de escenarios internos como son las habitaciones, aunque no existe una limitación para ellos a ser de clase habitación, a los que se accede y que se comunican entre sí y con el patio exterior mediante unas marcas especiales, puertas, y otros escenarios especiales: los túneles.

3.3 Motor gráfico

Desde un punto de vista de alto nivel, la aplicación Java desarrollada puede dividirse en una serie de bloques funcionales. La **Figura 3.2** muestra un esquema de alto nivel del motor gráfico.

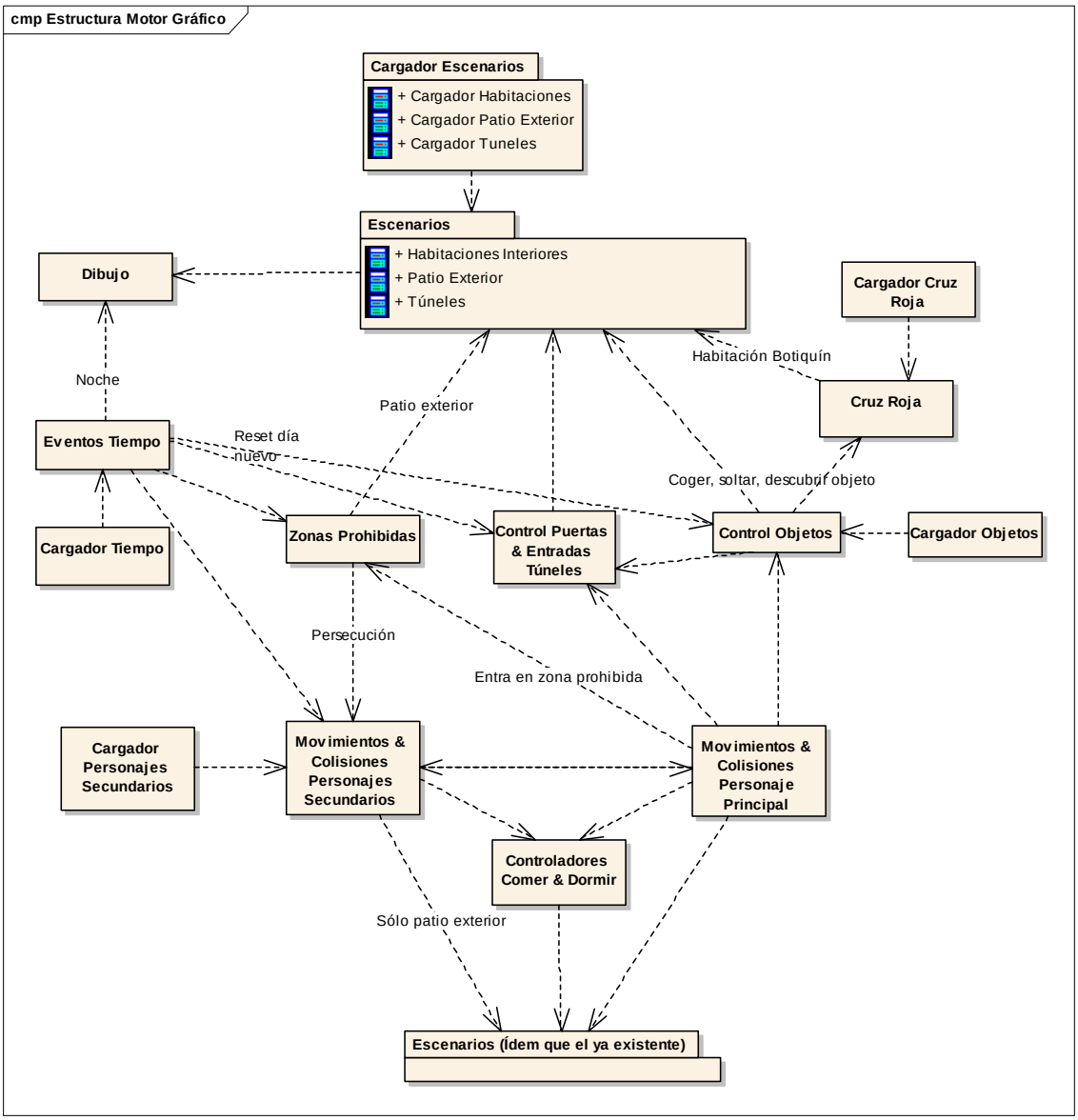


Figura 3.2 Abstracción de alto nivel de la estructura del motor gráfico.

A continuación se va a dar una descripción funcional de cada uno de los bloques del esquema. Se pretende dar al lector una visión introductoria de cada bloque abstracto, para que así pueda tener un concepto global del motor gráfico.

En el bloque de *Eventos Tiempo* se encierra conceptualmente todo el desarrollo y control temporal del videojuego. Como ya se ha comentado en el capítulo anterior, en el juego existe una intensa rutina de eventos temporales que se van sucediendo. Estos eventos rigen la vida de los personajes del campo. Según el evento en curso, cada personaje, controlado por la aplicación tendrá que realizar una secuencia de movimientos. Además, determina el tipo de zonas prohibidas que se encuentran en activo, ya que dependiendo del evento temporal se darán unas zonas u otras.

En el bloque de *Dibujo* se engloba toda la funcionalidad de la aplicación encaminada a realizar los gráficos del videojuego que se muestran por pantalla. Los gráficos mostrados van a depender del escenario en el que se encuentre el personaje principal, que puede ser: patio exterior, habitaciones interiores o túneles.

La entidad *Cruz Roja* se encarga de la gestión de entrega de los objetos especiales de la Cruz Roja. La entrega se realiza poco después del evento temporal *levantarse*, aunque esta relación es más una imposición lógica externa y configurable –este aspecto se entenderá mejor cuando se vea la explicación asociada al fichero *cruzroja.xml*. También, depende del bloque de *Control Objetos*, ya que éste le comunicará cuándo un objeto de la Cruz Roja ha sido descubierto abandonado en una zona no segura, y, por tanto, puede ser entregado de nuevo en la habitación de Botiquín.

La entidad *Zonas Prohibidas* gestiona todo lo relacionado con las zonas prohibidas. Aspectos como si el personaje principal se encuentra en una de ellas, habilitación para los guardias para iniciar persecución si ven al personaje principal dentro de una zona prohibida, la variación de las zonas prohibidas según el evento temporal en curso, etc.

El bloque *Movimientos & Colisiones Personajes Secundarios* abstrae todo el control de los personajes secundarios autómatas: el comandante, los soldados, los prisioneros. Cada personaje posee una rutina de movimientos que varía según el evento temporal en curso. También, en el caso de los guardias del campo, el que éstos vean al personaje principal en una zona prohibida o haciendo algo indebido como romper la valla, forzar una puerta o entrar/salir de las puertas de intendencia del castillo (puertas número uno, dos, tres y cuatro, véase el mapa de la **Figura 2.21** en el **Capítulo 2**) produce el efecto de iniciar la persecución. Un aspecto importante del que se encarga este bloque son las colisiones y su resolución. Los personajes secundarios pueden colisionar con el escenario (patio exterior), con otros personajes secundarios y con el personaje principal. Este módulo debe resolverlas de forma que no se produzcan situaciones extrañas y el movimiento resulte natural.

Movimientos & Colisiones Personaje Principal: este bloque agrupa a nivel lógico todo el control de movimientos y colisiones en el que interviene el personaje principal. En cuanto a las posibilidades de movimientos surge una gran cantidad de variantes en el ámbito de cercanía con colisión fija del escenario –este aspecto se explicará detalladamente más adelante en el apartado dedicado a la movilidad del personaje principal.

En la entidad de *Túneles* se abstrae todo el control de los túneles, lo que debe mostrarse por pantalla a través de la entidad *Dibujo* y todas las particularidades que lo relacionado con los túneles posea; por ejemplo, si se encuentra una salida bloqueada por derrumbe, los movimientos dentro del túnel del personaje principal, etc.

El *Patio Exterior* engloba todo lo relacionado con el escenario principal del videojuego. Por él deambulan los personajes secundarios, así como el personaje principal controlado por el usuario. Los personajes secundarios están, en sus movimientos, circunscritos a este escenario; si bien, respecto a esto último, se tiene la excepción de las acciones que realizan los prisioneros para sentarse a la mesa del comedor y para acostarse en las camas de sus respectivos dormitorios.

La entidad abstracta de *Habitaciones Interiores*, como su propio nombre indica, abstrae todo lo relacionado con las habitaciones interiores del juego. Es importante destacar que, a diferencia de los túneles, en las habitaciones sí está permitido coger y soltar objetos, igual que en el patio exterior. El personaje principal en su deambular por los escenarios es el que, a través de las puertas que vaya tomando, determine las habitaciones que se lanzan. En estas habitaciones interiores se presentan aspectos de la movilidad ampliados como son la posibilidad de que exista *scroll* de pantalla, el alineamiento entre puerta de entrada y de salida en la misma diagonal, aspectos todos que se irán aclarando a lo largo del capítulo.

El bloque *Control Objetos* representa toda la funcionalidad de manipulación de objetos especiales que existe en la aplicación: coger, soltar, usar, el que sean descubiertos abandonados en el campo y devueltos a su ubicación inicial en el reset de un día nuevo, etc.

Control Puertas & Entradas Túneles representa la funcionalidad asociada con las puertas y las entradas de los túneles. Funciones como abrir puertas cuando se usan las llaves o las herramientas, crear puertas a través de la valla cuando se usan las tenazas, abrir entradas a los túneles con la pala, además de volver a cerrar las puertas y agujero-puerta en valla cuando se da el reset de un nuevo día, etc.

Controladores Comer & Dormir se encargan de gestionar todo lo relacionado con las actuaciones de los personajes cuando éstos se van a dormir y a comer. Gestionan dónde se sienta cada personaje en la mesa del comedor, por su orden de llegada, y la cama que ocupan los personajes en sus dormitorios cuando es tiempo de dormir. Las acciones de las que se ocupan son tanto de los personajes secundarios como del personaje principal. Estos controladores se basan en parte en información contenida en el fichero XML *personajes.xml*, así como, en aspectos de configuración que se han quedado embebidos en el código como puede ser toda la gestión de sentarse y levantarse de la mesa del comedor o el ir a dormir para el personaje principal.

Los *Cargadores* –englobando en este nombre a todos los que aparecen en la **Figura 3.2-** son los encargados de extraer la información de configuración específica de los ficheros XML y trasladarla a cada bloque funcional correspondiente.

3.4 Estructura multithread o multihilo

Este proyecto hace un gran uso de los *threads* o hilos Java. Los programas *multithreaded* o *multihilo* parten de la idea de multitarea a su nivel más bajo: programas individuales que darán la impresión de llevar a cabo varias labores a la vez. Cada una de estas labores recibe el nombre de *thread* o *hilo*. Los programas capaces de ejecutar más de un *thread* a la vez reciben el nombre de *mulithreaded* o *multihilo*. Se puede pensar que cada *thread* se está ejecutando en un contexto separado, cada uno con su propia CPU, registros, memoria, etcétera.

La aplicación constará a lo largo de su tiempo de ejecución de varios hilos. El primer hilo que se ejecuta en la aplicación proviene de la clase de arranque en J2ME. Esta clase es una clase que hereda de la superclase abstracta MIDlet y que obliga, como tal, a implementar unos métodos –estos métodos son el de arranque (*startApp*), pausa (*pauseApp*) y destrucción (*destroyApp*) de la aplicación.

Generalmente, la forma de proceder es lanzar otro hilo de ejecución desde el método de arranque (*startApp*) para que tome y lleve el control y el peso de la aplicación. En este proyecto, el nombre de ese hilo primario es la clase *MotorGráficoExteriorCanvasThread*. Desde este hilo, se realiza al comienzo de su vida lo que es toda la carga de los ficheros de configuración XML, salvo los ficheros correspondientes a la descripción de las habitaciones que se cargan

según vaya accediendo a las. Se cargan por orden los ficheros XML: *zonasprohibidas.xml*, *mapafronteraexterior.xml*, *mapadibujoexterior.xml*, *personajes.xml*, *mapapuertasexterior.xml*, *tuneles.xml*, *cruzroja.xml* y *tiempo.xml*. Es de destacar que una vez terminada la carga de los ficheros XML señalados y que sean fijados parámetros de control de dibujo como el número de celdas que caben a lo ancho y a lo alto en la pantalla, entre otros; si en el fichero de configuración *mapafronteraexterior.xml* se especificó que el personaje principal comienza su andadura acostado en la cama de su dormitorio, como es el caso normal, entonces, este hilo principal se suspenderá a través del *monitor* de un objeto Java reservado para esta función y lanzará un hilo nuevo que constituirá la habitación interior en la que se encuentra *Steve* acostado de inicio. Este hilo nuevo se implementa en la clase *MotorGraficoInteriorCanvasThread*. Esta clase, al instanciarse, se encarga de controlar la representación en los escenarios de las habitaciones interiores. Para ello, carga el fichero XML de configuración donde se define la habitación en curso. El hilo principal *MotorGraficoExteriorCanvasThread* permanecerá en suspensión mientras el control esté en manos de un hilo secundario como *MotorGraficoInteriorCanvasThread*.

Una vez que el personaje principal regrese al patio exterior del castillo, atravesando alguna puerta que le lleva a él; la lógica despertará al hilo principal, *MotorGraficoExteriorCanvasThread*, para que retome el control de la representación por pantalla.

Si, por lo contrario, en el fichero *mapafronteraexterior.xml* no se indica que empiece desde la habitación, sino que aparezca directamente en el patio exterior en alguna celda especificada de la matriz (esto último de gran utilidad a la hora de realizar pruebas en el escenario principal patio exterior), el hilo de *MotorGraficoExteriorCanvasThread* continuará con la ejecución normal del juego.

En su movilidad por el patio exterior el personaje principal puede acceder a través de una marca de puerta a un escenario de habitación interior. Cuando ésto se produzca, el hilo principal, *MotorGraficoExteriorCanvasThread*, se suspende, lanzándose el hilo de *MotorGraficoInteriorCanvasThread* que controla la representación de la habitación interior, tras cargar el fichero XML de la habitación correspondiente. El objeto en que se instancia el *MotorGraficoExteriorCanvasThread*, además, contendrá un atributo donde se guarde la referencia al objeto instancia del *MotorGraficoInteriorCanvasThread* en curso.

Si en su movimiento por la habitación el personaje principal tomara una puerta que le llevase a otra habitación, entonces lo que ocurriría es que se destruiría el hilo de

MotorGraficoInteriorCanvasThread anterior y se lanzaría otro hilo nuevo de *MotorGraficoInteriorCanvasThread*, cargando en éste la nueva habitación y actualizándose la referencia en el atributo del objeto *MotorGraficoExteriorCanvasThread*. De esta forma, sólo habrá un hilo de *MotorGraficoInteriorCanvasThread* a la vez que el hilo principal en suspenso.

Igual ocurre con los escenarios especiales de los túneles. Cuando el personaje en su movimiento por el patio exterior encuentra la entrada a un túnel, accediendo a él, en este caso, se lanza un hilo Java nuevo llamado *TunelCanvasThread*, que se encargará de controlar la representación en el escenario túnel. Acto seguido de lanzarse este hilo, el hilo principal de *MotorGraficoExteriorCanvasThread* entrará en suspensión, como ocurría con los hilos asociados a las habitaciones interiores. Además, en el objeto de instancia de la clase *MotorGraficoExteriorCanvasThread* se actualizará un atributo para referencia reservado a estos hilos –las referencias a los hilos secundarios son de utilidad a la hora de controlar el cierre o el reinicio de la aplicación. Si el personaje principal sale por el otro extremo del túnel a una habitación, entonces se cargará esta habitación con su correspondiente hilo, destruyéndose el hilo de *TunelCanvasThread* del que provenía el personaje. De igual forma, si, estando el personaje principal en una habitación interior, entrara éste en un túnel, entonces el hilo de la habitación se destruiría y se lanzaría un hilo de *TunelCanvasThread* que controle la escena del túnel, con su referencia actualizada en el objeto del hilo principal. En cualquier caso, una vez que se regrese al escenario principal, patio exterior, estos hilos secundarios de los que se puede provenir se destruirán y se despertará el hilo principal del patio exterior para que controle la acción.

Cabe destacarse que cada *CanvasThread* de los señalados, posee un *game loop* (bucle del juego) donde se realiza todo el control del escenario tanto de movimientos como de representación gráfica. Esta arquitectura de múltiples hilos rompe un poco con el esquema tradicional de los juegos Java en la bibliografía. Tradicionalmente los juegos Java se construyen en base a un único hilo con un *game loop* desde el que se controlan todos los aspectos del juego. Si bien, el uso de múltiples *threads* puede suponer un incremento en el coste de la aplicación en términos de memoria, además de las complicaciones que pueden derivar de su manejo –motivo por el cual en la bibliografía siempre se presenta un único hilo-, para un juego donde existen múltiples escenarios con características diferentes y complejas, como entidad de encapsulamiento para el código, los *threads* pueden resultar de gran utilidad, aparte de proporcionar una solución elegante, que en el caso de los móviles de gama media-alta como son los celulares de Nokia de la serie S60, su empleo, con cuidado, no supone una excesiva carga para el sistema.

Aparte de los hilos ya mencionados, existen dos hilos más de características especiales que se suman a los anteriores. Se trata de los hilos temporales: uno el que controla los sucesivos eventos temporales que se dan y el otro el que controla en qué momento deben entregarse los objetos de la Cruz Roja.

Cuando se carga el fichero XML *tiempo.xml* al comienzo del hilo principal por parte de su controlador (en la **Figura 3.2** este controlador se engloba dentro del bloque funcional de *EventosTiempo*) se lanza un hilo de planificación (en segundo plano y no accesible al programador) que irá proporcionando los diferentes eventos de tiempo. Este hilo de planificación recibe el nombre en el API de J2ME de clase *Timer*. Con ella podemos planificar en el tiempo cuándo se van a presentar los diferentes eventos a través de una clase *envoltorio* que hereda de *Timer*, además de otra clase heredera de *TimerTask*, que controla la acción a realizar cuando salte cada evento en el *Timer* (ambas desarrolladas en este proyecto) –estas clases *envoltorio* reciben el nombre de: *Tiempo* y *EventoTiempo*, respectivamente.

Al igual que con *tiempo.xml*, cuando se cargue *cruzroja.xml* su controlador se encargará de lanzar otro *Timer* con la planificación de cada cuánto tiempo debe entregarse un objeto en la habitación de botiquín por la Cruz Roja –en este caso las clases herederas de *Timer* y *TimerTask* son: *TimerCruzRoja* y *TimerTaskCruzRoja*. En la **Figura 3.3** se muestra un esquema para los hilos que aparecen en la aplicación tras arrancar el juego y regresando desde la habitación de dormitorio de Steve al patio exterior, directamente tras el arranque.

Hay un momento donde estos *Timer's* rompen su continuidad en el tiempo, y es cuando el personaje principal es detenido, trasladado a la celda de castigo y se le informa que ha pasado un nuevo día. En este momento, los *Timer's* del controlador de tiempo y del controlador de la cruz roja, son destruidos y vueltos a relanzar para que exista sincronización entre el tiempo que marcaban y el instante en que se encuentra el juego tras amanecer un nuevo día desde la celda de castigo. El hecho de que se tengan que destruir y volver a relanzar estos *Timer's* proviene de que una vez que se configura el objeto J2ME que instancia la clase *Timer*, éste permanece inalterable en su eje de tiempo. J2ME no proporciona ningún método para realizar desplazamientos en el eje de tiempo de un objeto *Timer*, ni tampoco proporciona forma de acceder al hilo que va asociado con el *Timer*, éste se encuentra en segundo plano y no es accesible de ninguna forma en J2ME.

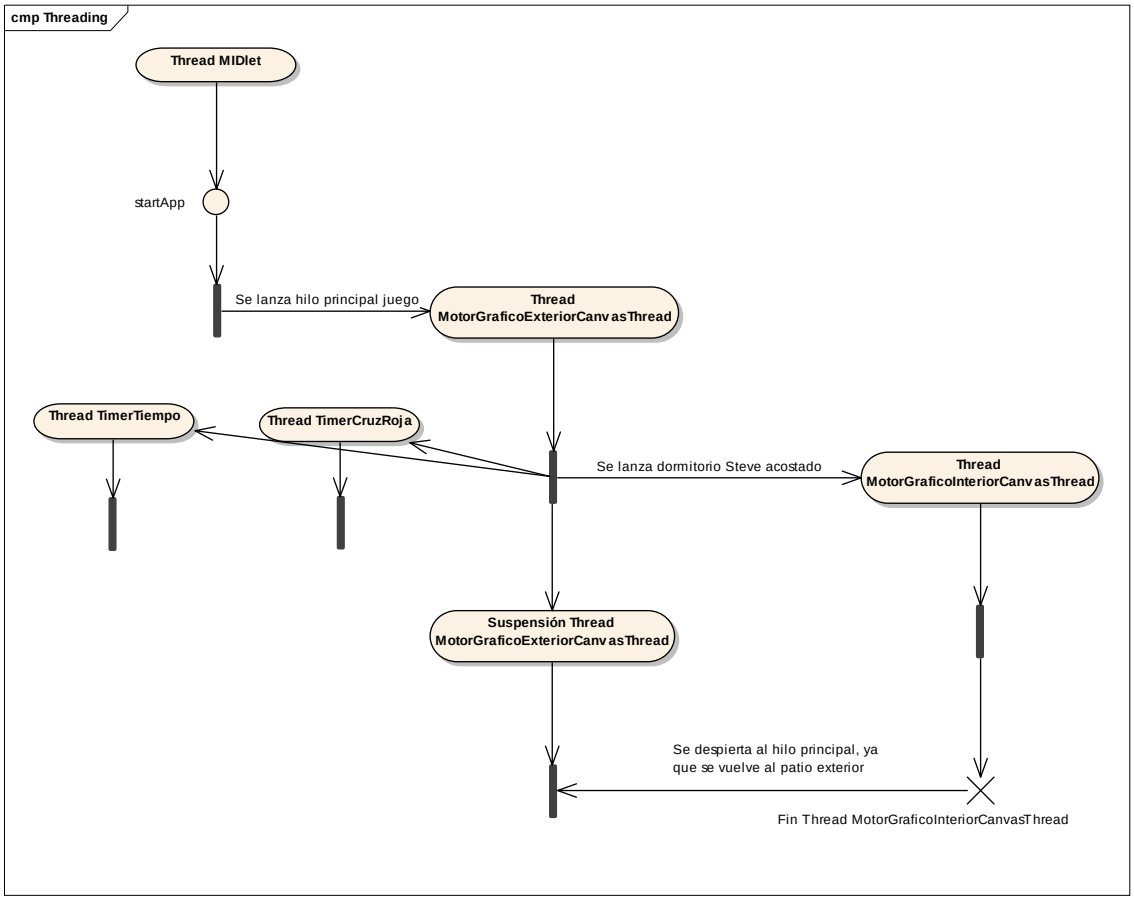


Figura 3.3 Esquema multihilo de arranque del juego.

Por último y relacionado con esta idea de los hilos, se debe destacar el comportamiento especial que conllevan los comandos Java. En J2ME, los comandos, teclas que tienen una función especial, como los que se despliegan en la aplicación cuando seleccionas la tecla de *Opciones*, tienen asociado un método, *commandAction*, que posee el código de la acción a realizar por ese comando; pues en J2ME, este *commandAction* que puede poseer cualquier clase que implemente la interfaz de la API de Java: *CommandListener*, tiene a todos los efectos asociado una especie de "*hilo de ejecución diferente*" al de ejecución normal, por lo que cada vez que se use un comando, además aparecerá otra especie de *"hilo de ejecución diferente"* adicional para realizar la acción del comando.

3.5 Perspectiva isométrica

También conocida como "perspectiva ¾", constituye una representación visual de un objeto tridimensional en dos dimensiones, en la que los tres ejes espaciales definen ángulos de 120º, y las dimensiones de la realidad se miden en una misma escala sobre cada uno de ellos. La

isometría es una de las formas de proyección utilizadas en dibujo técnico que tiene la ventaja de permitir la representación a escala, y la desventaja de no reflejar la disminución aparente de tamaño -proporcional a la distancia- que percibe el ojo humano. En la **Figura 3.4** se muestran los ejes de la perspectiva isométrica junto con la representación en ésta de un cubo.

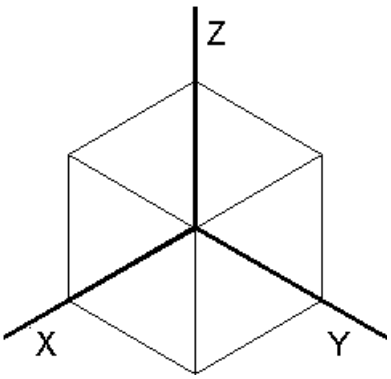


Figura 3.4 Ejes de la perspectiva isométrica, junto con la representación de un cubo.

En lo que respecta a la perspectiva isométrica para el juego, resulta de interés el ángulo de elevación de los ejes X e Y respecto de la horizontal. En la perspectiva isométrica este ángulo es de 30º. No obstante, en muchos juegos de perspectiva isométrica se utiliza una aproximación: a fin de evitar el *pixelado*, en algunos casos, se lleva la proyección a un sistema 2:1, es decir, a una inclinación de 26,5º (arctg 0,5) en lugar de 30º, que no corresponde a una proyección isométrica propiamente dicha, sino *"dimétrica"*. La proyección *dimétrica* se usa, sobre todo, para representar objetos más largos que anchos y altos, hecho que entronca con la naturaleza de los juegos isométricos de grandes escenarios como el que nos atañe. Pues bien, en el juego original se utiliza la aproximación anterior señalada, por lo que no se puede hablar de una perspectiva isométrica propiamente dicha. La **Figura 3.5** muestra los ejes de la perspectiva *dimétrica* utilizada en el juego.

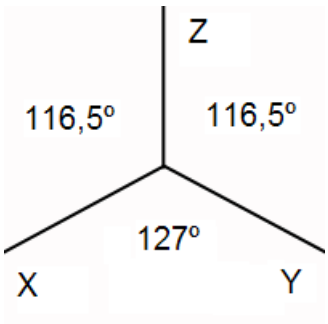


Figura 3.5 Ejes de la perspectiva *dimétrica* utilizada en el juego.

En la aplicación de videojuego desarrollado se sigue con la aproximación a la perspectiva isométrica, sistema 2:1, del juego original. El juego se construye en base a una matriz de celdas, donde cada celda posee el doble de ancho que de alto; hecho que conduce a un ángulo de elevación de los ejes X e Y de 26,5º -como ya se ha comentado. En la **Figura 3.6** se puede ver esta matriz de celdas en la que se basa la construcción del juego.

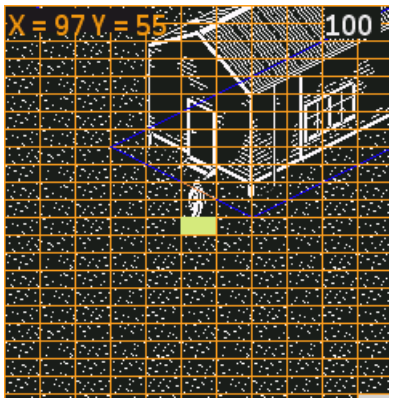


Figura 3.6 Imagen de la matriz de celdas sobre la que se basa la construcción del juego.

El tamaño de las celdas, concretamente la diagonal de éstas, define el tamaño del paso de los personajes, ya que en un juego isométrico el desplazamiento se realiza siguiendo los ejes X e Y, en definitiva, las diagonales de las celdas; a un tamaño menor de celda, mayor sensación de continuidad en los movimientos de los personajes. Por otro lado, el tamaño de las celdas limita el número de ellas que se mostrará por pantalla. Un elevado número de celdas –y, por tanto, un tamaño de celda pequeño- ralentiza la operación en el juego, ya que cada celda que sea mostrada por pantalla, deberá ser procesada para comprobar si posee algún elemento que se deba representar.

De estos dos factores limitantes surge un compromiso: menor tamaño de celda implica paso de los personajes más continuos, pero un mayor número de celdas mostradas por pantalla, y por tanto, una ralentización en el procesado. Y viceversa, mayor tamaño de celda implica menor número de celdas por pantalla y, por tanto, mayor velocidad de procesado, pero un paso de los personajes más discontinuo.

Otro factor limitante a la hora de fijar el tamaño de la celda es el dibujo de los escenarios. Si la aplicación hubiera partido de un diseño gráfico propio no habría habido problema, ya que la representación de los diferentes elementos que conforman los escenarios se habría realizado una vez fijado este tamaño de celda y se habría ajustado a él. Pero esto no ocurrió así, este proyecto toma como base para la apariencia gráfica -como ya se comentó- las capturas del juego original para el emulador de *Commodore 64* de PC: CCS64. Hay que

destacar que este hecho jugó un papel importante en la fijación del tamaño de la celda de la matriz también, puesto que debía elegirse un tamaño de celda tal que la horizontal, es decir, el ancho de los bloques de dibujo en que se basa el juego original, y que se iban a tomar para esta aplicación, fueran aproximadamente múltiplos del ancho de la celda, para así facilitar la representación en base a la matriz de celdas de estos dibujos. No obstante, este hecho de buscar un ancho de celda del cual fueran aproximadamente múltiplos los anchos de bloque de los dibujos del juego original, no evitó una etapa de fuerte procesado de las capturas para adaptarlas.

Para fijar este tamaño de celda se hicieron diversas pruebas viendo los resultados de movilidad del personaje, principalmente la sensación de continuidad y fluidez, de velocidad de desarrollo de la acción, básicamente la velocidad de procesado para mostrar una pantalla, y de encaje de los bloques de dibujo. Tras utilizar diversos tamaños de celda, finalmente, se optó por fijarla a veintidós píxeles de ancho frente a once por alto.

3.6 Matrices del juego

La construcción del juego se basa en una serie de matrices que conforman los diferentes aspectos de los escenarios –concretamente, se trata de los escenarios: patio exterior y las habitaciones interiores, ya que los túneles se basan en otro principio de construcción. Se puede decir que por cada escenario (patio exterior y cada una de las habitaciones interiores) existirán tres matrices; una por cada uno de los aspectos fundamentales que definen el escenario: las fronteras de colisión, los elementos de dibujo y los enlaces (puertas y entradas) a otros escenarios, aspectos contenidos en los ficheros de configuración XML. Estas tres matrices se conocen como *matriz de fronteras*, *matriz de dibujo* y *matriz de puertas*, respectivamente. Estas matrices serán matrices Java de tipo *short*, para disminuir su peso en memoria.

Las matrices se cargan por unos cargadores a la aplicación mediante la lectura de los ficheros XML donde se definen; estos cargadores además se encargan de cargar en la aplicación otra información útil. Las matrices tendrán, para cada escenario concreto, idénticas dimensiones –siendo las dimensiones de cada escenario definidas en el XML del escenario. Las matrices del escenario principal, patio exterior, se mantienen cargadas en memoria durante todo el tiempo que el juego esté cargado en memoria. Por otro lado, las matrices de las habitaciones interiores sólo están cargadas en memoria durante el tiempo que se está en esa habitación, en concreto, es

toda la habitación la que está cargada en memoria sólo durante el tiempo que ésta se muestra por pantalla igual que ocurre para los túneles.

Para el patio exterior se tiene que la *matriz de fronteras* de colisión se define en el XML *mapafronteraexterior.xml*, la *matriz de dibujo* del escenario en el XML *mapadibujoexterior.xml* y la *matriz de puertas* de enlace con otros puntos del escenario o las habitaciones interiores, en el fichero *mapapuertasexterior.xml*.

En cambio para los escenarios de las habitaciones interiores, la información de las tres matrices está incluida en el fichero que define por completo la habitación. Esto es así, ya que de partida se consideró que el escenario principal sería de unas dimensiones bastante mayores que el resto, por lo que se vio conveniente repartir la información de configuración entre varios ficheros XML para que resultara más manejable. Por el contrario, las habitaciones serían de unas dimensiones más reducidas por lo que sería factible incluir toda la información que define las matrices en un solo fichero XML para las habitaciones.

Más adelante, cuando se aborde la descripción de los ficheros XML de configuración se profundizará en la forma de definirlas. La idea importante que se debe extraer de este apartado es la correspondencia entre la matriz de celdas que se muestra en la visión de diseño con las matrices que definen los escenarios -patio exterior y las habitaciones interiores. Puede visualizarse como si se tratara, cada una de las matrices, de mapas de celdas superpuestos en planos paralelos, cada uno ocupándose de un aspecto: las fronteras, el dibujo, las puertas; los personajes se mueven por esas celdas, viéndose su movimiento afectado por las distintas marcas que se encuentran en las matrices señaladas.

3.7 Dibujo de escenarios

Los gráficos del juego se construyen en base a tres planos de dibujo superpuestos –lo que se conoce en la industria como un *z-buffer* de tres niveles. Con estos tres planos de dibujo se consiguen los efectos de profundidad que caracterizan al juego. Además, se utiliza una matriz de celdas –*matriz de dibujo*- para controlar la ubicación de los diferentes bloques de dibujo para el escenario.

El primer plano que se dibuja es el plano que va a servir de fondo, en éste se dibujan el suelo, las paredes y objetos de mobiliario sobre los que van a proyectar sombra los personajes que se mueven libremente. La **Figura 3.7** se observa un ejemplo de dicho efecto.

El siguiente plano en dibujarse es el plano de los personajes; en él se dibujan todos los personajes. Dado el ancho de los *frames* de los personajes quince píxeles –frente a los once píxeles de ancho de la celda- y el hecho que éstos se dibujan en la vertical central de la celda que ocupan, recorriendo la matriz por filas de arriba hacia abajo, la representación resulta correcta, no produciéndose ningún efecto indeseado y respetándose las sombras que los personajes por debajo podrían proyectar sobre los que se encontraran en filas superiores. La **Figura 3.8** muestra el dibujo de los personajes en el plano intermedio.

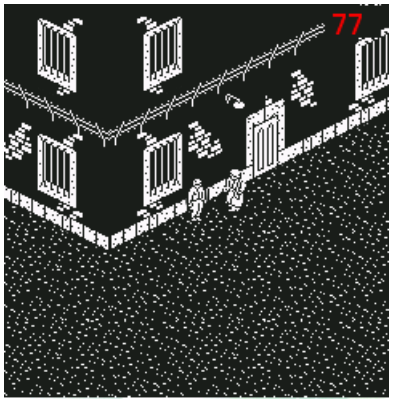


Figura 3.7 Plano uno: fondo sobre el que los personajes se dibujan.

Por último está el tercer y último plano de dibujo, el cual hace sombra a todos los anteriores. Este plano sirve para conseguir los efectos de profundidad del lado Oeste de los barracones, la esquina Noreste, el muro Sur del campo y el vallado y las torres de vigilancia. Gracias a este plano, sobre los personajes se proyecta la sombra de los edificios y el vallado, desapareciendo tras ellos todo el tiempo que su movimiento lo lleve tras esa sombra. En la **Figura 3.9** se muestran los diferentes casos señalados de dibujo en el plano superior.

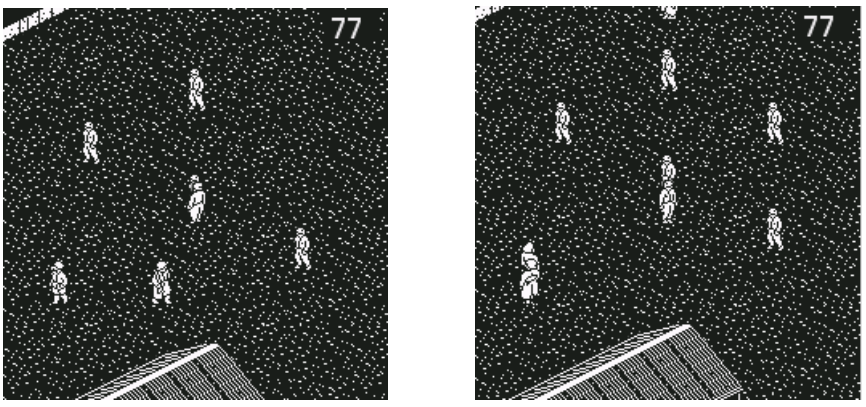
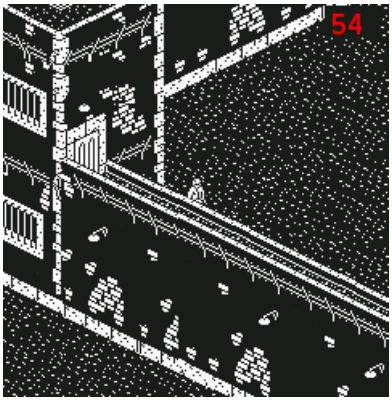
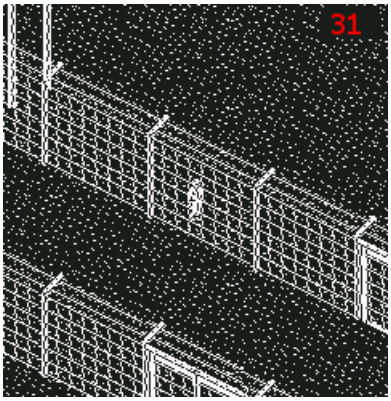


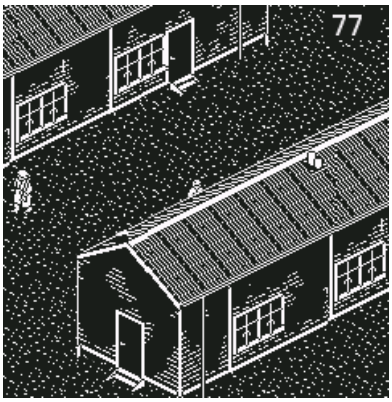
Figura 3.8 Imágenes de ejemplo de algunos personajes y sombras entre ellos, plano dos.



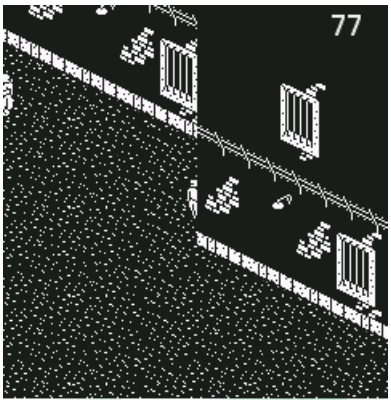
a) Muro Sur del campo.



b) Vallado.



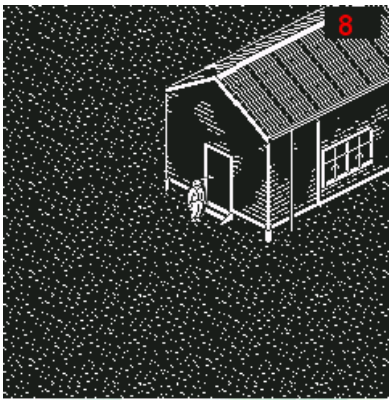
c) Barracones.

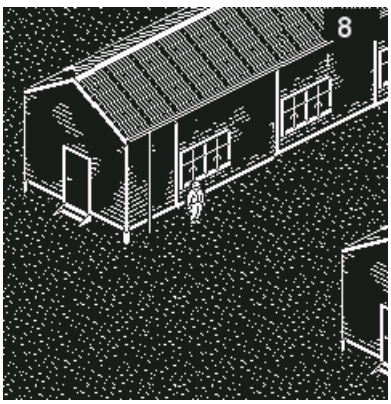


d) Esquina Noreste.

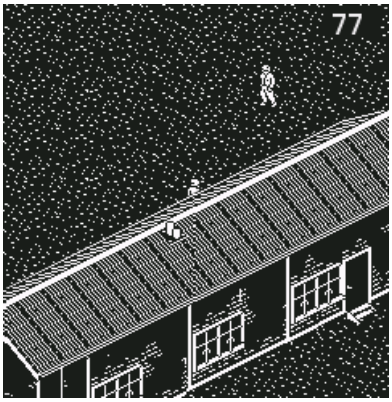
Figura 3.9 Imágenes de ejemplo de dibujo de plano tres.

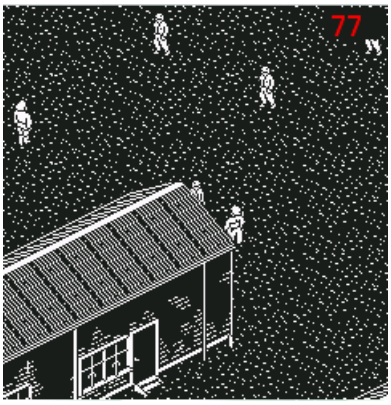
Hay entidades de dibujo del campo que pertenecen a dos planos de dibujo diferente, concretamente, a los planos que se han dado en llamar uno y tres. Estas entidades son los barracones, la esquina Noreste y la torre de vigilancia pegada a la entrada al campo. Veamos los efectos de los que se habla detenidamente. La **Figura 3.10** muestra el efecto señalado para el barracón.





a) Caras Sur y Este dibujadas en el plano uno. (**Figura 3.10**)

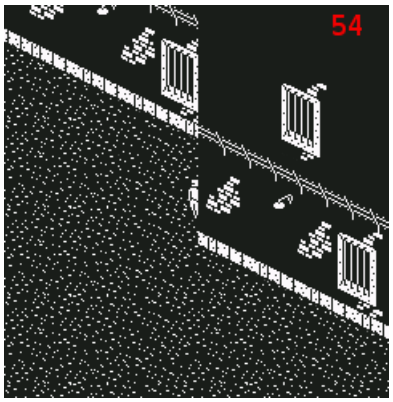


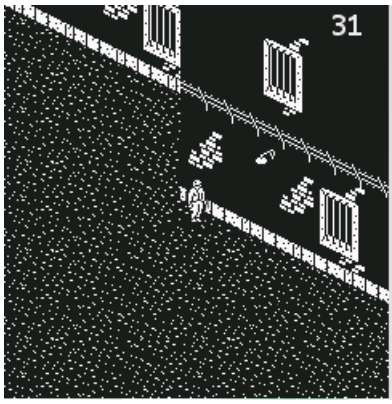


b) Caras Norte y Oeste del barracón dibujadas en plano tres.

Figura 3.10 Variación de planos de dibujo del barracón.

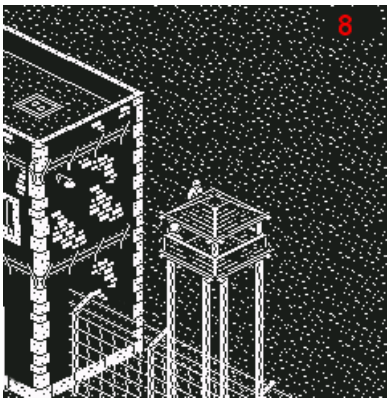
La **Figura 3.11** muestra los efectos de dibujo en plano uno y dos para la esquina Noroeste y la **Figura 3.12** los efectos para la torre de vigilancia.

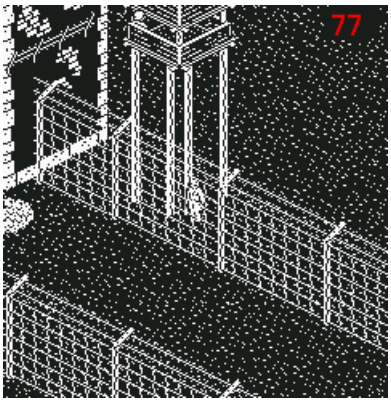




a) Dibujo esquina sobre plano tres. **b)** Dibujo esquina sobre plano uno.

Figura 3.11 Variación de los planos de dibujo para la esquina Noreste.





a) Dibujo torre sobre plano tres **b)** Dibujo torre sobre plano uno.

Figura 3.12 Efectos de dibujo plano uno y tres de la torre.

Estos efectos se consiguen partiendo la imagen en dos mitades, y asignado a cada una el plano correspondiente de dibujo: uno o tres. Un aspecto importante de esta técnica, supone el desplazamiento que una de las partes tiene que sufrir, desplazamiento que debe conservar la posición inicial de esta parte del dibujo. La solución que se siguió consistió en ampliar la imagen que se iba a desplazar en alto igual al de una celda y en esta imagen desplazar la parte de dibujo en sentido contrario tantos píxeles como el alto de la celda. A continuación, la marca para la parte desplazada se baja una celda, en vertical sólo. La **Figura 3.13** muestra una composición con las partes en que se dividen las imágenes.

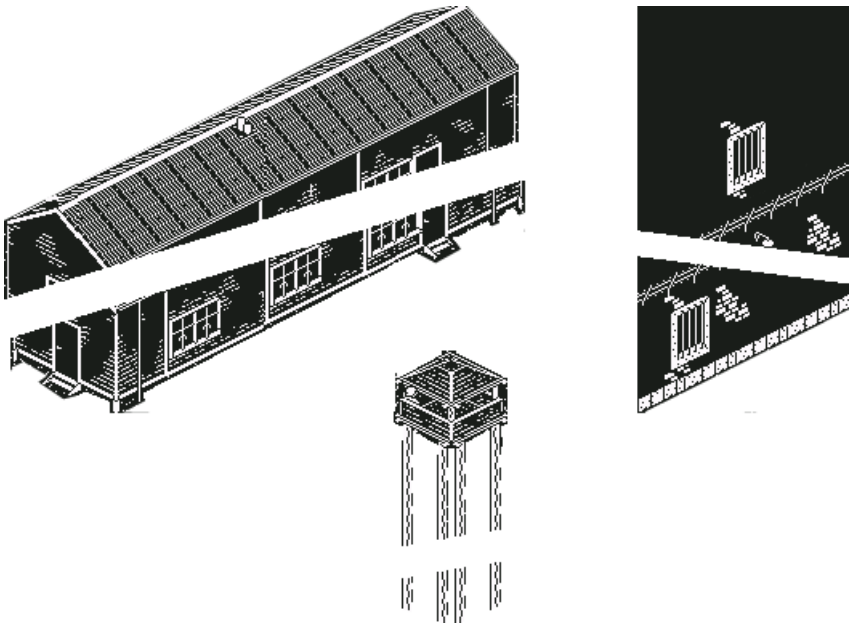


Figura 3.13 Partes utilizadas para poder dibujar la entidad en dos planos: uno y tres.

Debe destacarse que la construcción de los escenarios se hace en base a bloques de dibujo que se van repitiendo y combinando entre sí. Para ello, se siguen determinadas diagonales sobre la *matriz de dibujo*. Estas diagonales se especifican en los ficheros XML mediante etiquetas dedicadas, en el apartado dedicado a éstos XML se profundizará en este tema. En estas diagonales se van introduciendo de forma equiespaciada marcas en la matriz de dibujo que serán leídas por la aplicación para insertar en la posición de la celda donde se haya la marca, el dibujo, según las características que especificarán los atributos de la etiqueta del fichero XML de configuración. La **Figura 3.14** muestra un ejemplo de la repetición de bloques de dibujo para construir un escenario.

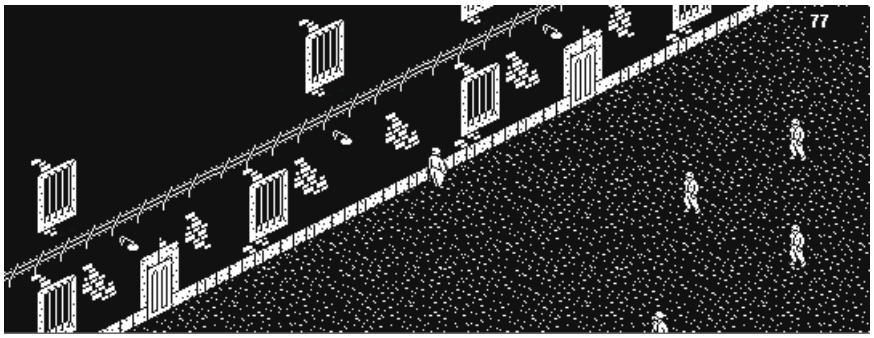


Figura 3.14 Siguiendo la diagonal de la pared se observa repetición bloque pared con puerta.

Esta construcción del escenario en base a diagonales, los ejes X e Y de la proyección, en definitiva, se consigue gracias al empleo de transparencias en las imágenes de cada bloque, lo que permite que la superposición de éstas dé como resultado una apariencia normal. En la **Figura 3.15** las zonas de *cuadrados* azules representan la transparencia para la imagen. Esta transparencia se consigue tras un procesado de la imagen capturada del juego original en el programa Adobe Photoshop CS, recortando la zona que se quiera que sea transparente.

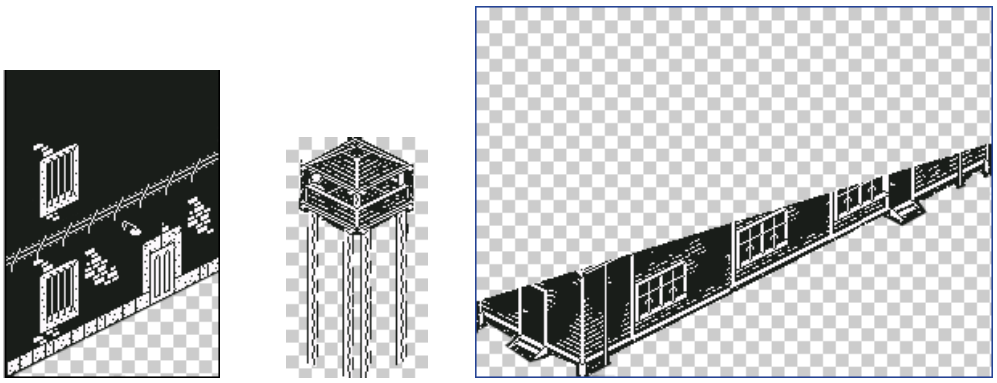
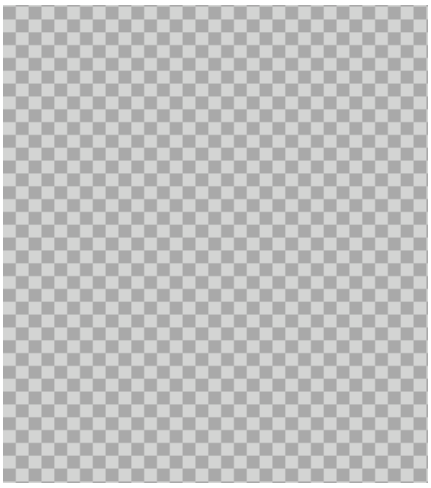
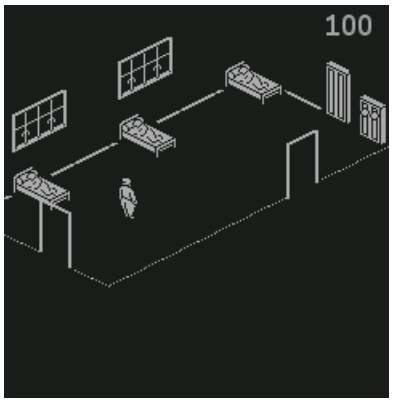


Figura 3.15 Zonas de *cuadrados* azules equivalentes a transparencia en la imagen.

Otro efecto que debe destacarse es el de la noche en el campo de prisioneros. Cuando se produce el evento Noche en el juego se hace de noche en todo el campo dentro y fuera de él. Este efecto se consigue pintando por pantalla, antes de toda la composición, un lienzo de imagen con opacidad reducida de color azul oscuro. La **Figura 3.16 a)** muestra el lienzo para realizar el efecto de sombra de la noche, mientras que la **Figura 3.16 b)** muestra un ejemplo de dicho efecto sobre el juego.



a) Lienzo con opacidad reducida para generar el efecto de la noche.



b) Imagen de ejemplo del efecto del lienzo noche en el juego.

Figura 3.16 Lienzo empleado para conseguir el efecto noche y un ejemplo.

3.8 Movilidad personaje principal

El personaje principal sigue las reglas de desplazamiento de la perspectiva isométrica, es decir, a lo largo de los ejes X e Y de la proyección. El desplazamiento está discretizado, moviéndose los pies del personaje desde el centro de una celda, por la diagonal, al centro de otra celda contigua en la matriz.

La tecla de movimiento hacia arriba (ARR) desplaza al personaje en el eje Y negativo, mientras que la tecla hacia abajo (ABA) lo desplaza en sentido contrario, eje Y positivo. Igual ocurre para la tecla de movimiento hacia derecha (DER), sólo que en este caso se desplaza a través del eje X negativo, mientras que para su opuesta, la tecla izquierda (IZQ), el desplazamiento se realiza en sentido eje X positivo –véase la **Figura 3.17**.

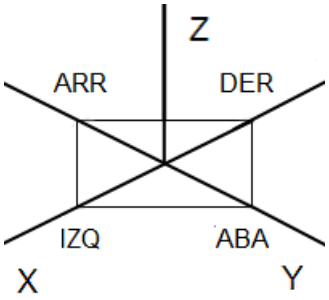


Figura 3.17 Movimientos y tecla correspondiente desde el centro de una celda.

Hay que destacar que los personajes, tanto los secundarios como el principal, ocupan siempre una celda de la matriz utilizada para construir el juego. Esta celda, además, no se puede compartir; si un personaje intenta ocupar una celda en la que se encuentra otro, entonces se produce un conflicto de colisión, de modo que la aplicación resuelve para que no se produzcan situaciones extrañas –a este respecto el algoritmo que resuelve para el personaje principal es sencillo: si la celda a la que se quiere mover está ocupada, se mantiene al personaje en la cela de partida, y se actualiza el movimiento del paso del personaje; por otro lado, la resolución para el movimiento de los personajes secundarios es más compleja, ésta se verá en el apartado dedicado a la movilidad de los personajes secundarios.

El dibujo de los personajes en la aplicación se realiza de forma que los pies de los mismos ocupen siempre, aproximadamente, el centro de la celda en la que se encuentran (véase la **Figura 3.18** donde se aprecia lo anterior).

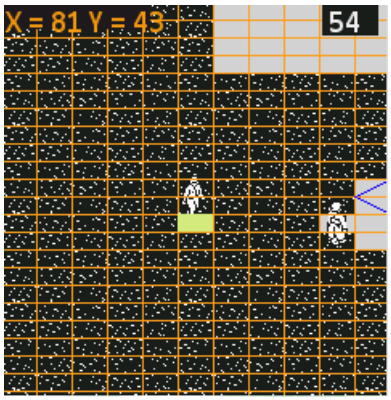


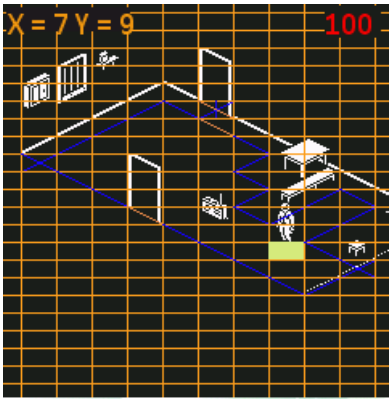
Figura 3.18 Posición de personajes con los pies sobre la celda que ocupan.

Otra importante característica de la movilidad del personaje principal es su posición respecto de la pantalla. En el escenario del patio exterior, el personaje principal ocupa el centro de la pantalla, quedando fijo en ella; al realizar desplazamientos la “ventana” del patio que se

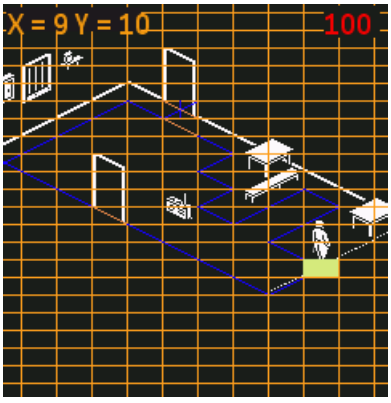
muestra, es decir, la pantalla en sí, se desplaza con el personaje, como si se tratara de una cámara que lo siguiera y tuviera siempre, al personaje, en el centro de la imagen.

En la movilidad del personaje resulta de gran importancia fijar la celda que ocupa, en definitiva, establecer cuál es la celda en la que se encuentran los pies del personaje. Para ello, la aplicación, a partir de la altura y ancho del *frame* del personaje junto con la ubicación central del personaje en la pantalla, obtiene la posición de referencia de la pantalla -"ventana"- sobre el escenario; de modo que para el personaje principal, centrado en ella, sus pies queden en la celda que ocupa. Una vez fijada la celda de los pies del personaje en este escenario, puesto que los desplazamientos son siempre en base a celda, no hay que volver a recalcularla, basta con desplazar la ventana en una celda en la misma dirección que el personaje. Este establecimiento de la posición de referencia de la pantalla se realiza siempre que, viniendo de una habitación o túnel, se sale al patio exterior. El motivo es que en estos escenarios, el tipo de desplazamiento del personaje principal cambia.

Para las habitaciones, la movilidad del personaje principal es diferente. En este escenario el personaje se puede mover libremente por la pantalla, es decir, no está fijo en el centro de la misma. El personaje se mueve de celda en celda del escenario habitación y una vez que llega al borde de la pantalla del dispositivo realiza *scroll* o arrastre de la pantalla para seguir moviéndose –véanse las imágenes de la **Figura 3.19**. Al igual que en el patio exterior el personaje ocupa una celda concreta, la que se encuentra bajo sus pies. En este caso el algoritmo para dibujar al personaje principal resulta más sencillo, ya que sólo hay que saber la celda en la que se encuentra el personaje y dibujar al personaje con los pies sobre ella.



a) Movimiento libre.



b) *Scroll* de pantalla.

Figura 3.19 Imágenes donde se muestra el movimiento libre y el *scroll* de pantalla.

El *scroll* o arrastre de pantalla se realiza siempre que el personaje principal alcance una posición tal que entre éste y el borde de la pantalla del dispositivo la anchura sea de una celda -

las partes que arrastran pantalla en cada caso, debido a la naturaleza del desplazamiento en isometría, son pies y/o cabeza. El *scroll* va a depender del tamaño de la pantalla del dispositivo, así como del tamaño del escenario interior. En habitaciones pequeñas, donde el personaje no alcance una distancia inferior a una celda respecto del borde de la pantalla, el *scroll* no se hará efectivo, en las etiquetas del XML para dichas habitaciones pequeñas se podrá desactivar.

La movilidad del personaje principal cambia también en lo referente a los escenarios túneles. En éstos no existen la variedad de movimientos que posee el personaje en los dos anteriores. Sólo tiene dos opciones, según sentido de la marcha a través del túnel. Hay que destacar que el sentido de movimiento con el que entramos en el túnel desde uno de los escenarios anteriores: habitaciones o patio exterior, es el sentido de salida que se conserva dentro de éste. Su opuesto es el que hace que se cambie el sentido de la marcha –en realidad, lo que cambia es la imagen que se muestra por pantalla, imagen estática en la que aparece el personaje principal en sentido contrario al que tenía. Una vez dentro del túnel, el usuario puede cambiar el sentido pulsando la tecla de movimiento opuesta a la que le hizo entrar en el túnel, e, incluso, puede volver a retomar el sentido que le conduce a la salida. En la **Figura 3.20** se pueden observar los dos sentidos de movimiento.

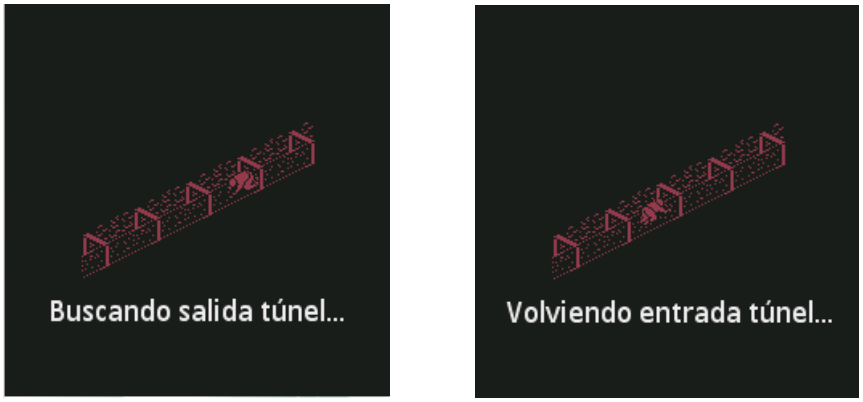


Figura 3.20 Los dos sentidos de movimiento dentro del túnel.

Volviendo sobre la movilidad en los escenarios: patio exterior y habitaciones, hay que destacar las posibilidades de colisiones que el personaje principal posee con las fronteras que demarcan muros, paredes, vallas y demás objetos de mobiliario, susceptibles de producir una colisión con él.

El personaje principal en su desplazamiento por el escenario con una dirección y sentido fijos puede encontrarse con las fronteras de algún elemento del escenario comentado en el párrafo anterior. Se pueden dar diferentes casos dependiendo de la diagonal que sigue en el escenario:

- Colisión con celda que impide el paso: en la diagonal de movimiento que sigue el personaje principal existe una celda que contiene una marca de frontera que le impide el paso. En este caso, el personaje se desplazará hasta esa celda y una vez sobre ella se detendrá; el seguir insistiendo en esa dirección y sentido lo único que conseguirá será que se actualice el *frame* del personaje con sucesivos pasos. La **Figura 3.21** muestra un ejemplo de este tipo de colisión con las fronteras del escenario.

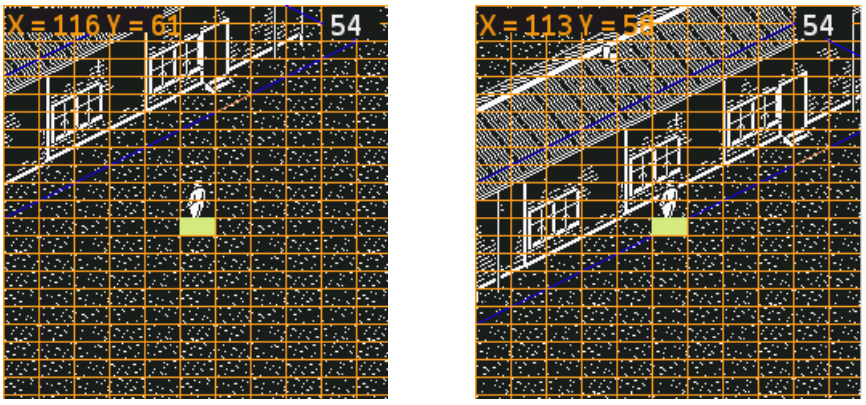


Figura 3.21 Secuencia de imágenes de colisión con frontera en la misma diagonal de movimiento.

- Colisión con la unión de dos celdas que poseen marca que impide el paso en el sentido de la marcha del personaje: en la diagonal de desplazamiento que sigue el personaje no existe celda con marca, pero sí nos encontramos con dos celdas contiguas por esquina con marcas que le impedirían el paso. Esto se da en llamar colisión con *intersticio* que impide el paso del personaje. Véase la secuencia de imágenes de la **Figura 3.22**. En este caso cuando el personaje llegue la *intersticio* se frenará su marcha al igual que en la colisión previa, ya comentada.

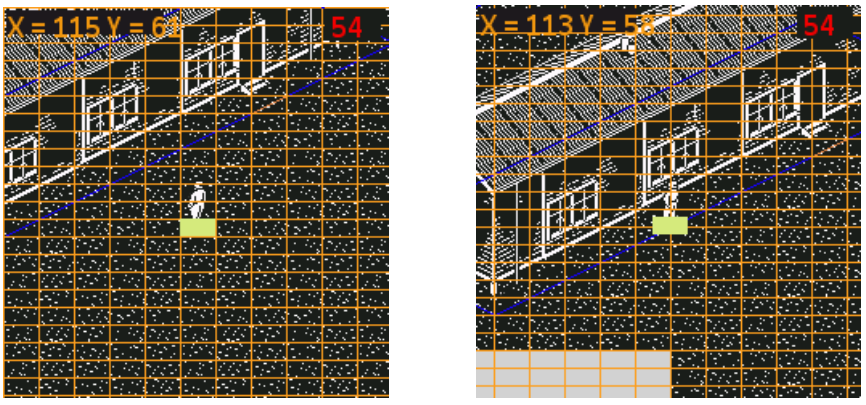


Figura 3.22 Colisión con *intersticio* de dos celdas frontera.

- Colisión esquina formada por dos celdas que poseen marcas de frontera perpendiculares: existen dos tipos de esquinas, las esquinas *hacia dentro* y las esquinas *hacia fuera*. En el primer tipo, esquina *hacia dentro*, el personaje principal se desplaza por una diagonal que contiene marca frontera, pero la dirección y sentido del desplazamiento son válidos. Al llegar a la intersección de las dos celdas con marcas perpendiculares el personaje se frena por la esquina –véase la secuencia de imágenes de la **Figura 3.23**. En el segundo tipo de esquina tenemos, la situación contraria al ser esta *hacia fuera* el personaje seguirá la marcha normal –obsérvese la secuencia de imágenes de la **Figura 3.24**.

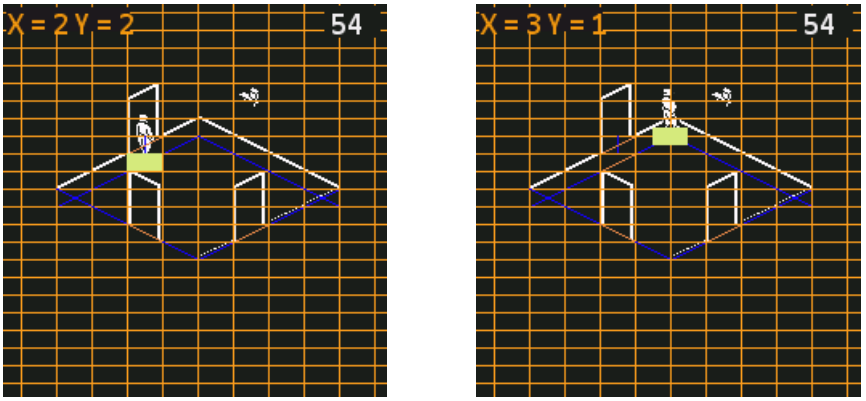


Figura 3.23 Colisión con esquina *hacia dentro*.

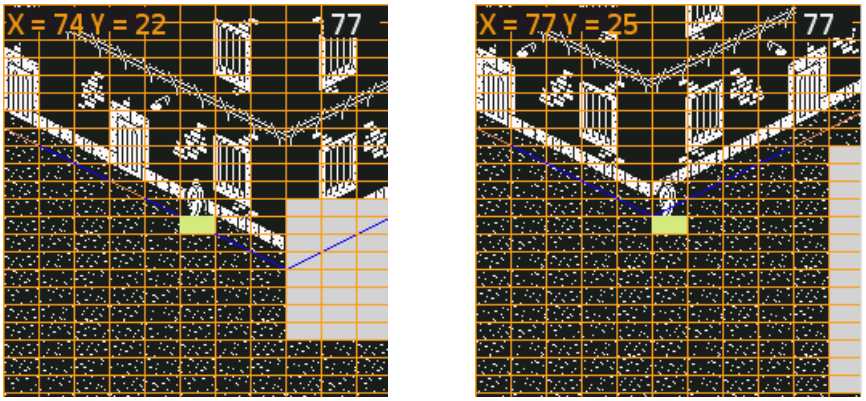


Figura 3.24 Movimiento a través de esquina *hacia fuera*.

- Colisión con esquina formada por marca aspa: en este caso el personaje se mueve por una diagonal generalmente con marcas frontera pero con una dirección y sentido que le permiten moverse, encontrándose con una celda que posee una marca en aspa que lo frena. La **Figura 3.25** muestra la secuencia de imágenes que ejemplifican lo indicado.

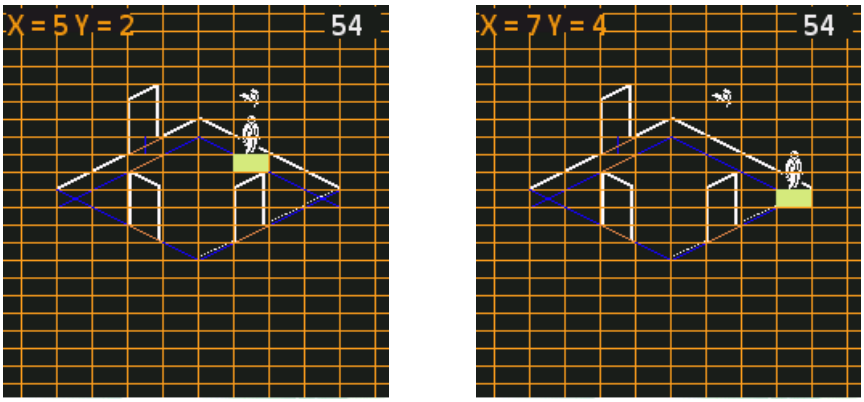


Figura 3.25 Colisión con celda con marca en aspa.

Cuando se analicen en profundidad las marcas para construir fronteras de colisión en los escenarios se verá que existen múltiples matices según la dirección y sentido del movimiento del personaje, matices que se diferencian a través de las etiquetas para construir las fronteras y no en base al movimiento que lleva el personaje principal.

Al respecto de la movilidad del personaje principal resulta de especial interés cómo aborda las entradas a los túneles o a las puertas que se pueden encontrar por los escenarios. Abordando primero la entrada a los túneles, hay que destacar, como ya se comentó, que cada entrada tiene un sentido determinado de entrada prefijado –este sentido coincide con uno de los sentidos de los ejes X e Y de la proyección. Pues bien, la marca de entrada al túnel ocupa una sola celda –cuando se vean las etiquetas que sirven para definir los túneles en el apartado dedicado a explicar los ficheros de configuración XML se profundizará a este respecto- mientras que la comprobación para ver si el personaje principal se encuentra sobre ella yendo en el sentido de entrada correcto consiste en una zona de comprobación en diagonal y perpendicular para la proyección con el sentido de movimiento del personaje principal, como la que se muestra en la **Figura 3.26** –celdas de color gris. Si la entrada al túnel es barrida por la zona de comprobación y el sentido del movimiento del personaje principal es el sentido de entrada al túnel, entonces éste entrará en él siempre que la entrada no esté cerrada por un derrumbe y lleve consigo el objeto linterna.

Al igual que ocurre con la marca de entrada al túnel, las marcas de las puertas suelen ocupar una sola celda también, aunque se rigen por el mismo principio de implantación sobre la *matriz de puertas* que las marcas de frontera y de dibujo por lo que pueden ocupar varias celdas en diagonal, como ocurre para las puertas uno y dos (véase mapa de la **Figura 2.22** en el **Capítulo 2**), y la entrada al patio de ejercicios donde se colocan dos consecutivas en diagonal. La **Figura 3.27** muestra las marcas de puertas para las entradas al patio de ejercicio.

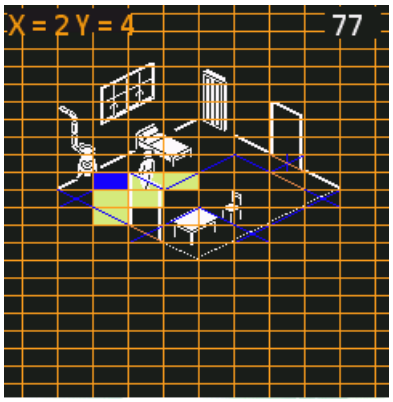


Figura 3.26 Imagen de la zona de barrido para detectar si el personaje se encuentra sobre túnel.

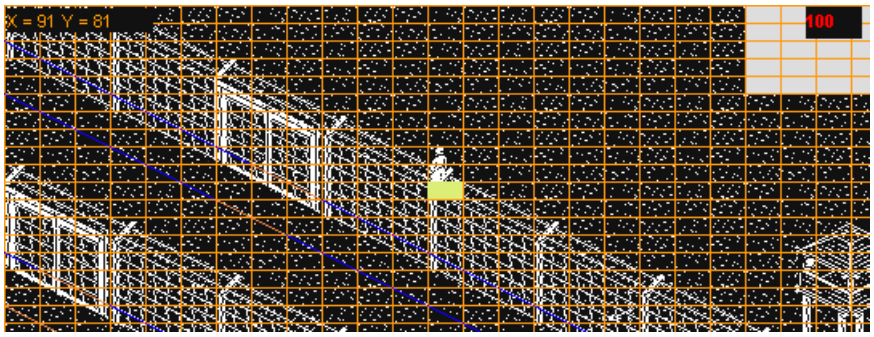


Figura 3.27 Imagen con las marcas para las puertas de entrada al patio de ejercicio.

A propósito de las entradas al patio de ejercicio, hay que destacar que las que se encuentran aledañas a cada lado de la valla enlazan entre sí, con lo que si, durante los eventos de tiempo que se encuentran abiertas, el personaje principal se posiciona sobre ellas, en el sentido de atravesar la valla, éste aparecerá al otro lado sobre la marca de puerta de salida con la que enlazaba la primera –el enlace de las puertas entre sí se observará con más detenimiento cuando se expliquen las etiquetas que definen las puertas en los ficheros XML de configuración.

La forma que tiene de atravesar una marca de puerta el personaje principal es sencilla, basta con que se coloque sobre ella y se mueva en el sentido que le permite cruzarla –cada marca de puerta indica un sentido válido para el que el personaje la cruza, cuando se vean las definiciones de los XML para las puertas se entenderá mejor este aspecto. No obstante, existen situaciones especiales donde la posición del personaje parece no coincidir con la marca. Estos casos ocurren cuando el personaje principal se posiciona sobre lo que se ha dado en llamar *intersticio* (en la imagen de la derecha de la **Figura 3.22** se puede ver un ejemplo de posicionamiento sobre *intersticio*).

Existe un comportamiento diferenciado con respecto a los intersticios según el personaje principal se encuentre en el escenario del patio exterior o en un escenario de

habitación interior, esto es debido al tamaño de las puertas que se dibujan en el patio exterior, puertas que pueden necesitar en algunos casos dos celdas en diagonal con marcas para llenar todo el umbral, como se observa en la **Figura 3.27**.

En el patio exterior cuando un personaje pasa a ocupar una posición en *intersticio*, de las dos celdas que une dicho punto *intersticial*, el personaje a efectos de la aplicación se hallará, realmente, en la celda superior de las dos antedichas, es decir, en la celda que tiene fila menor. Con este efecto el personaje si se coloca en el *intersticio* de una puerta con dos marcas como es el caso de las comentadas en el parágrafo anterior, resultará que a todos los efectos se encuentra en la marca superior de las dos señaladas. Igualmente, si se coloca en el *intersticio* de unión inferior que une la celda inferior (a la que le corresponde una fila mayor de las dos que tienen marcas de puerta) con otra que no tiene marca y que está en diagonal, entonces el personaje se encontrará, para la aplicación, como si estuviera en la marca de puerta inferior, pudiendo atravesar la puerta.

Este efecto es general para todo el patio exterior cuando se encuentra el personaje principal en un *intersticio*. Además, en las pruebas de este efecto se comprobó que resultaba natural para la perspectiva isométrica que, cuando se colocaba el personaje en un *intersticio* con marca de puerta por encima siguiendo la diagonal, el personaje pudiera atravesar la puerta.

En las habitaciones interiores, este comportamiento no existe. En ellas, todas las puertas tienen el ancho de una celda, por lo que no hay que considerar el efecto anterior. En cambio existe un efecto de desalineamiento de las puertas que obliga a introducir una variante. Este efecto de desalineamiento puede verse en la **Figura 3.28**. Para solucionarlo se introduce que en caso de caer el personaje principal en un *intersticio* en las paredes Norte y Oeste de la habitación, el personaje para la aplicación ocupara la celda previa en la que estaba antes de pasar al *intersticio*, haciendo la comprobación de si en la marca inferior en diagonal al *intersticio* hay marca de puerta y si el sentido del movimiento del personaje principal es el que lleva a atravesar la puerta.

Hay que destacar que este desalineamiento se produce por la necesidad desde el punto de vista gráfico de dejar media celda de separación mínima entre el contorno Norte y Oeste de la habitación (líneas blancas) y las marcas de frontera (en la **Figura 3.28** las líneas de color azul); este problema durante la fase de desarrollo –fase en la que se hicieron intensas pruebas para ver cuál era la mejor forma de fijar las fronteras y las líneas de contorno-, además, condujo a la introducción de las marcas de dintel especiales para las puertas, en la **Figura 3.28** la celda con el dibujo  . Esta marca indica la celda a la que se sale cuando se viene de la habitación

siguiente a la que se muestra en la figura, permitiéndose que se conserve el alineamiento entre las puertas en diagonal que se perdía por la separación de media celda que hay que introducir para conservar la perspectiva.

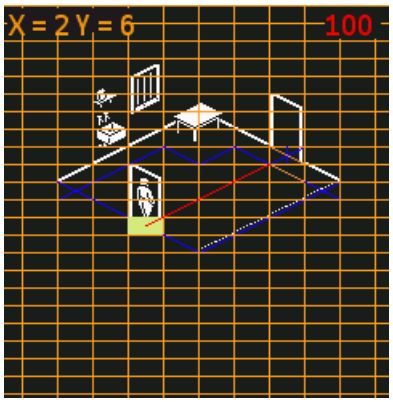


Figura 3.28 Imagen donde se muestra desalineación en diagonal de las puertas.

Con la introducción de dicha marca dintel, que a efectos del lenguaje de etiquetas consistirá en activar un atributo que haga referencia al dintel en una etiqueta de puerta como se verá más tarde, se conserva la alineación de las puertas. Es de destacar que cuando el personaje se encuentra en una marca dintel, éste no puede moverse en el sentido de las líneas de contorno, es como si estuviera debajo del marco de la puerta, sólo puede avanzar hasta el interior de la habitación o regresar hacia la habitación previa. Debe señalarse que las marcas de dintel sólo existen en las puertas de las habitaciones que se encuentran en una pared Norte u Oeste, las puertas que se encuentran en una pared Sur o Este no se ven afectadas por el problema de desalineamiento, ya que para estos casos las líneas de contorno de la habitación y de las fronteras coinciden. Cuando se aborden las etiquetas de definición de puertas para la matriz se verá cómo se convierte una marca de puerta en dintel.

3.9 Movilidad personajes secundarios

Los personajes secundarios tienen, en la aplicación, sus movimientos automatizados en función del evento temporal en curso y de la definición que se encuentre para ellos en los ficheros de *tiempo.xml* y *personajes.xml*. En función de este evento, se cargará para cada personaje una secuencia de movimientos particular. En el fichero *tiempo.xml*, en coordinación con el fichero *personajes.xml* cuando se requiera especificar aspectos particulares de algún personaje, se define para cada tipo de personaje: prisionero, soldado y general, cómo responden, los personajes, en cuanto a movimientos para cada uno de los eventos temporales posibles.

Existen dos tipos de respuestas que en el fichero de configuración se refiere por un atributo que toma el valor de "*predefinida-generica*" o "*predefinida-privada*". La definición de movimiento como "*predefinida-generica*" conduce a que todos los personajes de ese tipo, para ese evento concreto, se comporten con una secuencia de movimientos idéntica, con dicha secuencia de movimientos contenida en el fichero *tiempo.xml*. En el caso de "*predefinida-privada*", se tiene que la definición de la secuencia de movimiento para ese tipo de personaje y evento concreto se encuentra especificada para cada personaje, en particular, y no de forma genérica, en el fichero de configuración *personajes.xml*. Cuando se aborden detalladamente los ficheros de configuración se profundizará a este respecto. La **Figura 3.29** muestra un esquema del proceso para obtener la información acerca de la movilidad de los personajes secundarios.

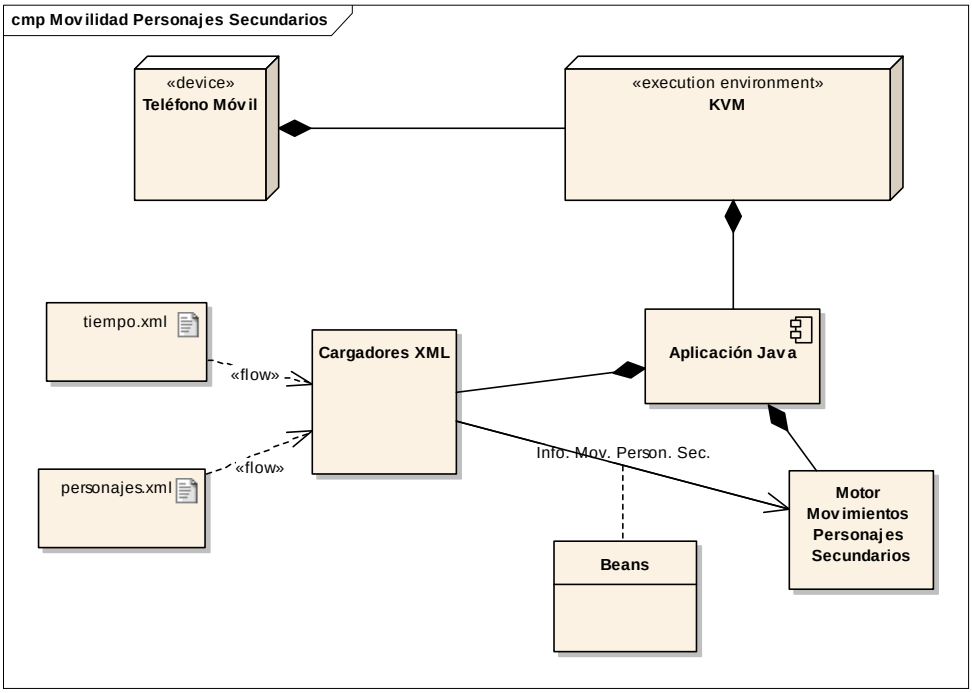


Figura 3.29 Flujo de la información en la movilidad de los personajes secundarios.

En la **Figura 3.29** aparecen dos entidades nuevas: una entidad llamada *Motor Movimientos Personajes Secundarios* y otra *Beans*. La entidad *Motor Movimientos Personajes Secundarios* es la encargada de generar en base a la información contenida en los ficheros XML y que le proporcionan los cargadores por medio de los *Beans*, la segunda entidad nueva que aparece, los movimientos de cada uno de los personajes.

Las clases *Beans* quieren significar todas las clases que posee la aplicación Java encargadas de albergar la información de los ficheros de configuración XML durante tiempo de

ejecución. Pueden verse como las clases que generan los objetos que mapean la información de los ficheros XML al entorno de orientación a objetos de Java.

Centrándonos en el motor de movimientos para los personajes secundarios, se va a profundizar en la explicación del algoritmo que permite a los personajes secundarios moverse de forma autónoma y con desplazamientos naturales en el entorno con obstáculos del patio exterior, al que se encuentran limitados.

Un movimiento en la aplicación se especifica por un destino en celda, indicándose la fila y la columna de la matriz donde debe trasladarse el personaje en cuestión –esta información se encuentra en los ficheros de configuración *tiempo.xml* y *personajes.xml*.

El algoritmo base de movimiento consiste en buscar la diagonal más cercana que conduce al personaje secundario al destino y seguirla una vez está sobre ella. Para ello, se divide primero la matriz de celdas en cuatro cuadrantes, basados en los ejes vertical y horizontal respecto de la celda destino. Dependiendo de la ubicación del personaje, es decir, de la celda en la que se encuentre el personaje; si ésta está en el primer, segundo, tercer o cuarto cuadrante, entonces se fija como destino primero alcanzar la diagonal del cuadrante en que se encuentre. En la **Figura 3.30** se puede ver un diagrama que facilita la comprensión del algoritmo descrito.

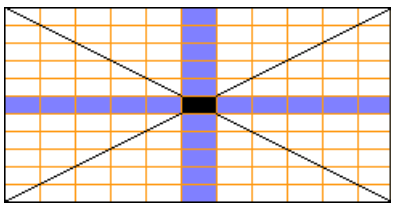
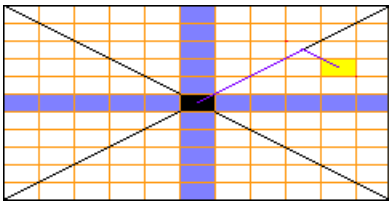
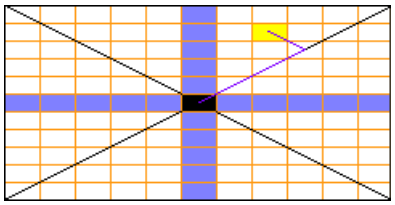


Figura 3.30 Ejes de división de cuadrantes y las diagonales para el camino *óptimo*.

Por cada cuadrante, la diagonal, que conduce a la celda de destino –diagonal negra de la **Figura 3.30 -**, divide el cuadrante en dos partes. El movimiento del personaje dependerá de la parte en la que se encuentre –en la aplicación se realizan las comprobaciones pertinentes para establecer en qué cuadrante se encuentra, así como, si está por encima de la diagonal o por debajo, para, así, fijar el sentido del movimiento. En la **Figura 3.31** se observa los dos ejemplos de desplazamiento para el primer cuadrante.



a) Partiendo debajo de diagonal.



b) Partiendo por encima diagonal.

Figura 3.31 Desplazamientos posibles para el personaje buscando la diagonal.

Resulta importante destacar que cuando el personaje se encuentra sobre el eje que divide la matriz en cuadrantes -celdas azules de la **Figura 3.30**-, la desambiguación del algoritmo funciona haciendo el movimiento hacia la diagonal más cercana en el sentido de giro de las agujas del reloj. Dentro de los desplazamientos anteriores, existe una variedad. Esta consiste en que el personaje no se encuentra en una celda, cuya diagonal de movimiento, hacia la diagonal que divide el cuadrante en dos, pasa por la mitad de una celda en la diagonal mencionada que divide al cuadrante, es decir, la diagonal de movimiento del personaje hacia la diagonal principal del cuadrante corta a ésta en un *intersticio* –véase el caso mostrado en la **Figura 3.32**.

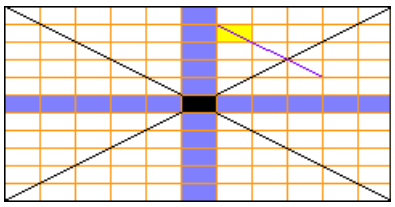


Figura 3.32 Diagonal que no divide en dos ninguna celda de la diagonal principal.

La solución que aporta el algoritmo de la aplicación consiste en conducir al personaje hasta la diagonal *sombra* sobre la que sí se posiciona y hacer que la siga después hasta el destino -la **Figura 3.33** muestra las diagonales *sombra* de las diagonales principales; para comprobar si éste llegó al destino –celda negra-, lo que se realiza es la comprobación de si en el perímetro de celdas inmediato a la posición del personaje se encuentra el destino, entonces, si es así, éste se considera que ha llegado –véase la **Figura 3.34** donde se muestra lo anterior.

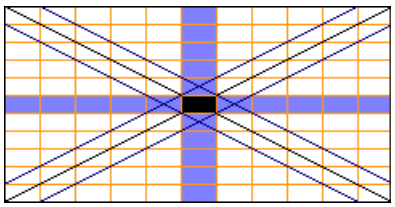


Figura 3.33 Imagen donde se aprecian las diagonales *sombra* en morado.

El perímetro a la celda donde se encuentra el personaje secundario y que se utiliza para saber si ha llegado al destino también aplica si el movimiento de los personajes se realiza por la diagonal principal como en la **Figura 3.31 –** en la **Figura 3.34** las celdas en gris son un ejemplo de perímetro de decisión para saber si el personaje ha llegado al destino. Este perímetro sirve también para decidir si el personaje secundario llegó a una puerta, por ejemplo para entrar a comer o dormir (esto se verá más adelante en este mismo apartado).

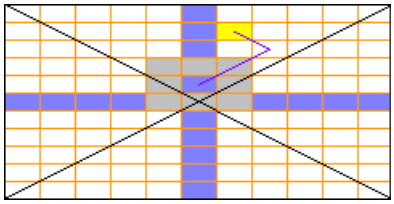


Figura 3.34 Imagen del movimiento cuando se sigue la diagonal *sombra* –línea morada.

En la movilidad mostrada hasta ahora, no se han tenido en cuenta los obstáculos estáticos que puede haber en el campo. Estos obstáculos los van a constituir las líneas de frontera para la limitación de los elementos del patio exterior como muros, vallas, paredes, barracones, torres de vigilancia, etc. Antes de seguir profundizando en el algoritmo que resuelve el movimiento de los personajes secundarios cuando se encuentran un obstáculo estático, se debe mostrar cómo se produce la colisión con éste por parte del personaje.

La colisión en el caso de los personajes secundarios comprende la mayoría de las situaciones ya vistas para el personaje principal, sólo que, en algunos casos concretos, se realiza una simplificación. Siguiendo con los diagramas de celdas mostrados hasta ahora para ver los desplazamientos, podemos ver los dos tipos de colisiones en los que existe diferencia respecto de los casos del personaje principal –ya mostrados en el apartado anterior:

- Colisión con *intersticio* de celdas que contienen dos marcas fronteras que impiden el paso: el personaje se mueve por una diagonal que se cruza con un *intersticio* de dos celdas en diagonal, ellas, que poseen marcas de frontera que impiden el paso en el sentido del movimiento. La **Figura 3.35** muestra esto último. En este caso el personaje secundario no se coloca sobre el intersticio como ocurría para el personaje principal.

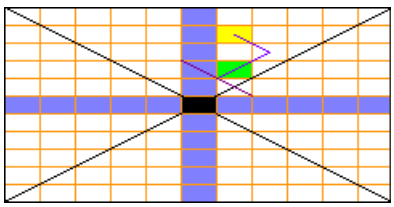


Figura 3.35 Colisión con el *intersticio* de dos celdas con marcas de frontera.

- Colisión con esquina *hacia dentro*: el personaje secundario en cuestión se desplaza por una diagonal que contiene marcas de frontera, pero la dirección y el sentido del movimiento están permitidos, llegando a una esquina *hacia dentro*. En este caso el personaje no se coloca justo en la esquina sino que se frena en la celda que forma la esquina de la diagonal que le estaba sirviendo de guía. La **Figura 3.36** muestra este caso en un diagrama de celdas.

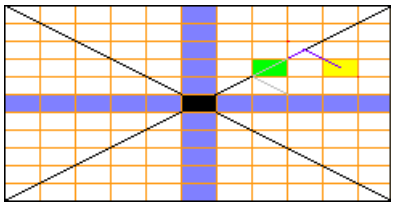


Figura 3.36 Imagen donde se muestra la colisión con una esquina *hacia dentro* –la celda verde es donde se frena el personaje al detectar la colisión con la esquina.

Una vez vistas y concretadas las peculiaridades que presentan los personajes secundarios en cuanto a las colisiones con las marcas fronteras se puede abordar el algoritmo que se sigue para resolver el camino ante un obstáculo –en la **Figura 3.37** se muestra el esquema genérico de un obstáculo para las diagonales que conducen a la celda de destino.

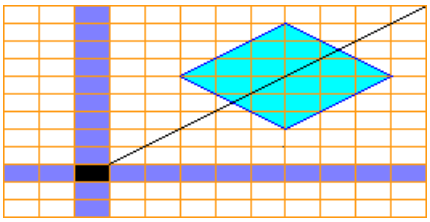
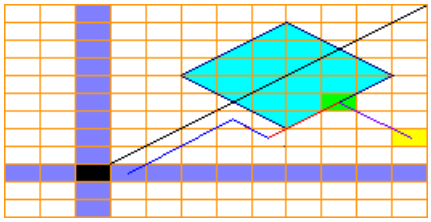


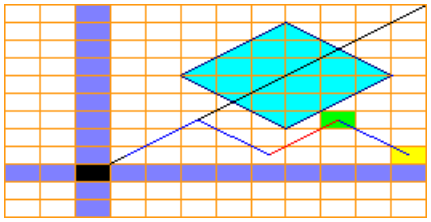
Figura 3.37 Esquema donde se muestra la diagonal principal y el obstáculo.

Existen dos situaciones de colisión que se pueden presentar para un personaje secundario en busca de destino:

- Colisión cuando todavía no se ha alcanzado la diagonal principal o diagonal *sombra*: en este caso el personaje secundario se desplaza buscando la diagonal principal o *sombra* que conduce a la celda destino, pero en su camino se encuentra con unas marcas fronteras que le impiden el paso, perpendiculares a su movimiento. Véase la **Figura 3.38** donde se muestra un esquema para este tipo de colisión –hay que tener en cuenta que este caso se generaliza para los cuatro cuadrantes y los dos sentidos de búsqueda de la diagonal principal, por cada uno, que se da.



a) Caso de colisión con celda.

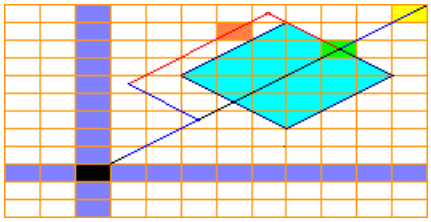


b) Caso de colisión con *intersticio*.

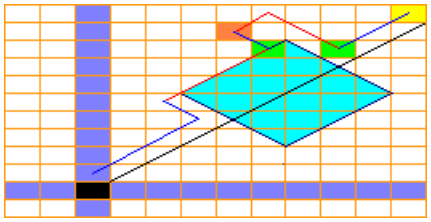
Figura 3.38 Colisión con un obstáculo sin alcanzar todavía no la diagonal principal o *sombra*.

El personaje bordeará el obstáculo siguiendo el sentido que le acerca a la celda de destino; una vez que el obstáculo se termine, se aplicará de nuevo el algoritmo normal de búsqueda de diagonal principal o sombra que le conduzca al destino –en las **Figuras 3.38 a)** y **b)** se muestran ejemplos de estos desplazamientos: la línea azul es el movimiento regido por el módulo normal del algoritmo, mientras que la línea roja representa los movimientos que realiza un módulo especial para resolver la colisión y buscar la celda de destino, la celda verde indica la celda donde se produce la colisión con el obstáculo.

- Colisión una vez que se sigue la diagonal principal o *sombra*: el personaje sigue la diagonal principal o sombra hasta que se encuentra con una marca frontera que lo frena, a diferencia del caso anterior, ahora existen dos posibles elecciones para bordear el obstáculo; la elección se realizará de forma aleatoria. La **Figura 3.39** muestra dos ejemplos de esta actuación –en ambos se elige la misma opción para bordear, pero en la aplicación es aleatorio, dándose unos esquemas parecidos a los que se representan en la figura. Una vez que se termina la frontera que le obliga a corregir el desplazamiento, profundiza una celda en el sentido de marcha primero -el de bajar por la diagonal principal o *sombra*- de modo que en esas condiciones (el personaje está en la celda de color bermellón) aplica el caso de resolución de búsqueda de diagonal principal o *sombra* con la posible variante de colisión anterior que puede conllevar.



a) Caso de colisión con celda.



b) Caso de colisión con *intersticio*.

Figura 3.39 Colisión con un obstáculo una vez que se sigue la diagonal principal o *sombra*.

Los esquemas representados para los dos tipos de colisiones anteriores con obstáculo se generalizan a los cuatro cuadrantes en unas representaciones análogas a las de las **Figuras 3.38** y **3.38**. Otro aspecto a destacar es que si en cualquiera de los casos anteriores el personaje colisionara, por ejemplo, con otro personaje, o bien, la forma del obstáculo en uno de los laterales que sigue fuera en esquina *hacia dentro* –como lo que se muestra en la **Figura 3.40-**, entonces el personaje elegiría un sentido aleatorio de los cuatro posibles para intentar salir del atasco, manteniéndose este estado hasta que dejara de colisionar y derivara a alguno de los casos de movimiento ya contemplados; puesto que, en todo momento, mantendría el objetivo de fila y columna de la celda fijada, una vez resuelta la situación de atasco, aplicaría la búsqueda de diagonal principal o *sombra* con todo el algoritmo que conlleva ya comentado, consiguiendo encontrar la forma de bordear el obstáculo.

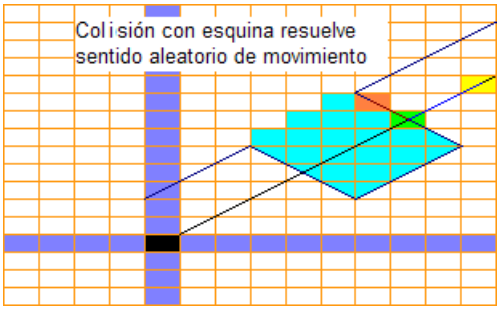
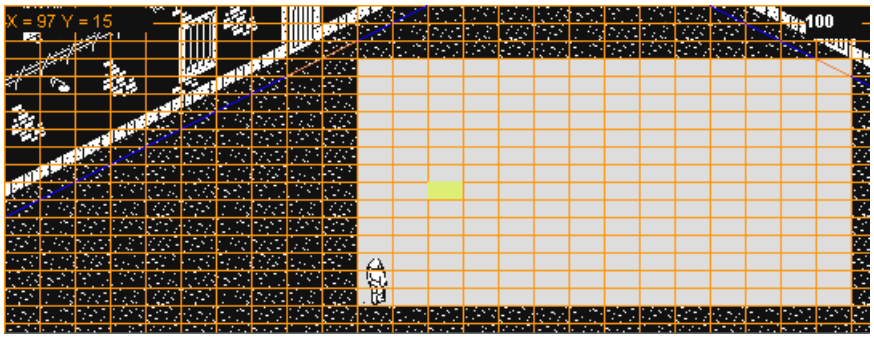


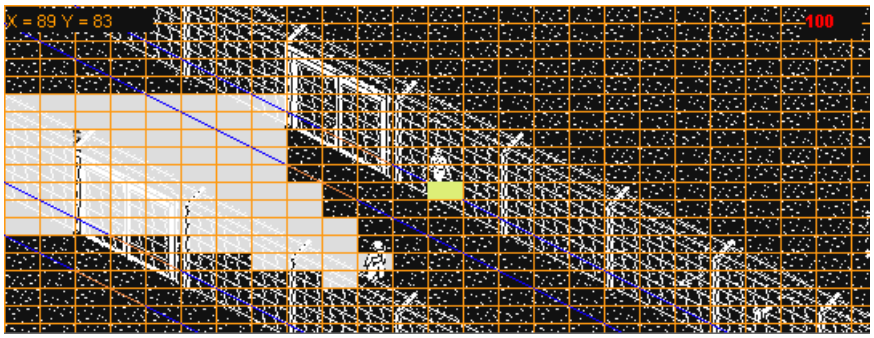
Figura 3.40 Colisión tras intentar bordear el obstáculo -resuelve mediante la fijación de un sentido aleatorio para el personaje de forma que busque salir del atasco; éste estado se mantiene mientras haya colisión.

Una situación análoga a las de las **Figuras 3.38 b) y 3.39 b)** es la colisión con un personaje –este hecho se produce a menudo cuando se llama a formar a los personajes, aunque también se da en el normal desarrollo de la acción del juego. En este caso, siguen siendo igual de válidos todos los aspectos del algoritmo que rige la movilidad de los personajes secundarios.

La movilidad de los personajes secundarios se vuelve más compleja en el caso de los guardias. Los guardias, además de tener todas las particularidades anteriores, poseen la de que pueden ver al personaje principal y lanzarse en su persecución si éste se encuentra en una zona prohibida o si ha hecho algo indebido. La **Figura 3.41** muestra el campo de visión de los dos tipos de guardias que existen en el campo: los soldados y el general. El campo de visión de ambos personajes es simétrico en cuanto a celdas, formando el del soldado un ángulo agudo y el del general un ángulo recto.



a) Campo de visión del general.



b) Campo de visión del soldado.

Figura 3.41 Campos de visión de los guardias.

Cuando un guardia ve al personaje principal en una zona prohibida o haciendo algo indebido se lanza en su persecución. Las acciones indebidas que automáticamente fijan el indicador de moral a rojo hasta que se detiene al personaje principal son: el que el personaje principal entre o salga de alguna puerta de intendencia (puertas uno, dos, tres y cuatro, véase el mapa de la **Figura 2.25** del **Capítulo 2**), que el personaje principal sea descubierto rompiendo la valla, que el personaje principal sea descubierto utilizando las herramientas para forzar una puerta o que, habiendo faltado el personaje principal a dos citas de recuento, éste sea visto por un guardia.

En cualquier caso, ya sea por haber hecho algo indebido o por estar en una zona prohibida el guardia va en busca del personaje principal. Para ello, al guardia se le va fijando en cada momento el destino de fila y columna de la celda que ocupa el personaje principal en su intento de huida, encargándose el motor de movimientos de los personajes secundarios de llevarlo hasta él.

Uno de los aspectos del motor de movimientos de los personajes secundarios y que resulta de gran interés para los guardias en sus persecuciones, es la capacidad que tienen los personajes secundarios de atravesar las puertas de la doble valla y los agujeros que el personaje principal haya podido hacer en la valla, si, cogiéndolos, va en camino a la celda que se fijara

como destino. Ésto se puede ver, claramente, cuando el personaje principal realiza un agujero en la valla, penetrando en el interior del perímetro de doble valla, dejándose descubrir por el soldado que patrulla por allí y volviendo a salir por el agujero primero que practicó en la valla. Si en su persecución el soldado descubre el agujero, que a efectos prácticos se traduce en que se coloque en la celda donde se encuentra la marca de la abertura, entonces éste cruzará el agujero en la valla y seguirá con la persecución bien hasta que lo alcance, o bien, hasta que el personaje principal abandone la zona prohibida. Una vez detenido el personaje principal, o bien que se haya salido de la zona prohibida, el soldado retoma su itinerario de patrullaje previo, para ello, regresa al interior de la doble valla, bien, porque descubre el agujero, o bien, porque el algoritmo le lleva a la entrada de la doble valla.

3.10 XML

Extensible Markup Language (XML) es un lenguaje para estructurar información en un documento que permite la creación de documentos claros y portables. Es una gran solución en el intercambio de datos entre aplicaciones y en la configuración de éstas. XML coincide en su fondo con las características que dieron origen a Java, por lo que juntos forman una pareja inmejorable y, en los últimos años, ha sido adoptado como un estándar a la hora de intercambiar datos o configurar aplicaciones en el mundo de la industria, pues proporciona un sistema con formato independiente de plataforma para indicar la información que se intercambia o que configura la aplicación.

El lenguaje XML es un lenguaje de marcas, como HTML, pero con una gran diferencia: permite la creación de marcas propias. Un elemento XML consta de ciertas partes fundamentales: elementos, atributos, entidades, DTD´s y XML *Schema*. Un *elemento* es el contenido delimitado por una etiqueta, desde la etiqueta de inicio hasta la etiqueta final, describiendo un dato. Un *atributo* es un valor que se incluye en la etiqueta de inicio de un elemento y se utiliza para proporcionar información adicional acerca del elemento. Una *entidad* es una representación virtual de un dato, ya sea texto o binario, a la que se puede hacer referencia en el documento XML; a la hora de analizar el documento las referencias a las entidades son sustituidas por sus valores.

Las etiquetas que pueden utilizarse para construir los elementos de un documento XML y los atributos que pueden tener se pueden definir, bien, en un fichero de tipo documento, *Document Type Definition* (DTD), o bien, en un fichero XML también expresado en lenguaje

específico para definición de XML: XSL (*Extensible Stylesheet Language*). La industria en la actualidad se decanta por el empleo de *XSL-schema* frente a DTD [22]. En ambos casos, si se dan estas definiciones de ficheros deben indicarse al comienzo del documento XML.

Un aspecto importante a la hora de tratar con ficheros XML es la herramienta parseadora utilizada. Los *parseadores* suelen ser componentes muy voluminosos, con una gran cantidad de código y que necesitan elevados recursos de memoria en tiempo de ejecución. En los dispositvos móviles MIDP los recursos de memoria suelen estar muy limitados. Se debe buscar un *parseador* diseñado para ser pequeño y ligero.

Los *parseadores* de código abierto suponen una alternativa francamente útil y atractiva. Proporcionan un elevado control, se pueden personalizar e, incluso, intentar reparar posibles errores.

Existen tres tipos fundamentales de *parseadores*:

- Tipo *model*: un *parseador model* lee un documento entero y crea una representación del documento en memoria. Estos *parseadores* usan muchos más recursos de memoria que cualquier otro tipo.

- Tipo *push*: este tipo de *parseador* lee a través de un documento entero, tan pronto como encuentra diferentes partes del documento, éste notifica a un *listener*.

- Tipo *pull*: un *parseador pull* lee en una vez sólo una pequeña cantidad de documento. La aplicación conduce al *parseador* a través del documento, mediante la petición de las siguientes partes a leer.

La tabla de la **Figura 3.42**, resume la oferta actual de pequeños *parseadores* de código abierto que son útiles para los dispositivos MIDP.

En la columna de tamaño se indica el tamaño del componente en un fichero *jar*. La columna MIDP indica si el componente *parseador* compilará sin tener que introducir modificaciones en un entorno MIDP. Y, por último, la columna tipo en el que se indica en qué tipo se engloba nuestro *parseador*.

Nombre	Tamaño	MIDP	Tipo
ASXMLP	6 kB	Sí	Push, model
kXML 2.0	9 kB	Sí	Pull
MinML 1.7	14 kB	No	Push
NanoXML 1.6.4	10 kB	Parche	Model
TinyXML 0.7	12 kB	No	Model
XParse-J 1.1	6 kB	Sí	Model

Figura 3.42 Tabla comparativa parseadores [23]

Por lo indicado previamente se debe buscar un *parseador* que no sea muy pesado en código, ni en memoria durante tiempo de ejecución –esto último recomienda evitar un *parseador* de tipo *model*, y decantarse por uno de tipo *push* o *pull*. Por otro lado, para simplificar la implantación del *parseador* y asegurar de esta forma la mayor portabilidad de la aplicación conviene evitar aquellos *parseadores* que tengan en la columna MIDP: *No*, ya que en éstos puede que haya que introducir modificaciones para poder implantar el *parseador* en el dispositivo móvil. Por todo esto, para manejar los ficheros XML en la aplicación desarrollada en este proyecto nos decantamos por la opción del *parseador* ASXMLP, que posee un peso en *jar* de 6 kB, uno de los más bajos, que no conlleva modificaciones para implantación en los dispositivos y que consiste en un término intermedio entre el tipo *push* y *model* que resulta aceptable. En este sentido, se habla de que se trata de un término intermedio entre *push* y *model* puesto que el *parseador* proporciona a un *listener* tres tipos de eventos pasándole estructuras de datos de pequeñas partes del documento:

- Encontrar comienzo de etiqueta: en este caso al método del *listener* correspondiente (*tagStarted*) le proporciona el nombre de la etiqueta y una tabla *Hash* (*Hashtable* Java) con los atributos de esa etiqueta si los tuviera.

- Encontrar fin de etiqueta: en este caso al método del *listener* correspondiente (*tagEnded*) le proporciona sólo el nombre de la etiqueta que se cierra.

- Encontrar texto plano: en este caso al método del *listener* correspondiente (*plaintextEncountered*) le proporciona un *string* con todo el texto encontrado.

El nombre de este parseador ASXMLP (*Al Sutton XML Parser*) proviene de su desarrollador: Al Sutton [24]. Este *parseador* constituirá la herramienta que procese los XML de configuración, entregando a las diferentes clases que se basan en ellos la información necesaria para que funcionen, el *parseador* con su *listener* correspondiente para el fichero XML

constituye lo que en el diagrama de la **Figura 3.2** se da en llamar bloque funcional de *Cargadores*.

3.11 Descripción de los ficheros de configuración XML

En este apartado se abordará la descripción de los diferentes ficheros de configuración XML que se desarrollaron en la aplicación. En ellos se encuentra encerrada gran parte de la lógica compleja del juego. En cierta medida pueden verse como un nivel de programación por encima del propio lenguaje Java, basado en las etiquetas XML que constituyen los ficheros de configuración.

Todo fichero XML posee una cabecera de presentación del documento. Cabecera que indica el tipo de fichero, concretamente en el caso de XML la cabecera en su formato más sencillo es <?xml version="1.0"?>. A continuación de esta cabecera aparece lo que se llama la marca o etiqueta raíz del documento. En esta marca raíz es norma habitual especificar el espacio de nombres al que hacen referencia las etiquetas del fichero XML para evitar cualquier ambigüedad en su interpretación estableciendo una naturaleza universal a las etiquetas al ligarse a una URI (*Uniform Resource Identifier*) –ésto se expresa mediante la inclusión de un atributo *xmlns*; en este proyecto el espacio de nombre se fijó a xmlns="http://www.etsit.uma.es/pfc/the-great-escape". Este espacio de nombres tiene un papel meramente virtual en el proyecto, puesto que no existe ningún recurso específico en la URI http://www.etsit.uma.es/pfc/the-great-escape. Otro hecho destacable es que este espacio de nombres será el espacio de nombres *por defecto* para todos los ficheros XML de la aplicación.

A continuación se da la descripción de todos los ficheros presentes en la aplicación, empezando por el fichero que hace referencia a las zonas prohibidas en el juego:

- *zonasprohibidas.xml*:

La marca raíz es *zonas-prohibidas-exterior* esta etiqueta se dispone en marca de inicio y cierra tal que:

```
<zonas-prohibidas-exterior xmlns="http://www.etsit.uma.es/pfc/the-great-escape">
    <!-- Resto de elementos xml -->
</zonas-prohibidas-exterior>
```

Dentro de esta marca se tendrá primero, en número arbitrario, la definición de las regiones consideradas prohibidas desde el inicio de la aplicación. Estas regiones estarán fijadas por rectas de diagonales en la matriz, fijándose uno de los dos semiplanos que se obtengan como zona prohibida. La **Figura 3.43** muestra un ejemplo de dicha recta diagonal y la zona prohibida.

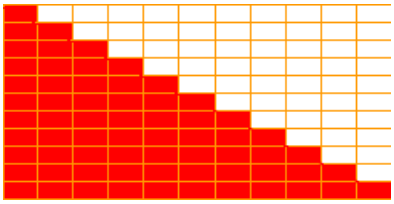


Figura 3.43 Imagen de una recta diagonal y zona prohibida en rojo asociada a ella.

La etiqueta que define estos semiplanos es en un ejemplo:

<linea-prohibida a="1" b="1" c="-114" zona="negativa" afecta-a="0"/>
<!--Al menos una -->

La línea prohibida se define en función de la ecuación general de la recta para el plano: $ay + bx + c = 0$. Para conservar la coherencia con la representación gráfica, eje X e Y cartesianos tienen como origen la esquina superior izquierda de la matriz. Este hecho conduce a que el movimiento por filas se haga siempre en el eje Y negativo, realizándose una corrección en el código para tener en cuenta este efecto. Teniendo en cuenta los ejes de la perspectiva isométrica, en definitiva, las diagonales de las celdas que definen los contornos en el juego, estas líneas en la aplicación tendrán siempre una pendiente de +1 o -1; hecho que coincide con fijar unos valores de *a =”1”*, *b =”1”* y *a =”1”*, *b =” -1”*, respectivamente.

Otro aspecto importante es la fijación de la celda por la que pasa la diagonal. Esto se controla mediante el parámetro *c*. Si deseamos que la recta pase por la celda (F0, C0) con pendiente +1, entonces basta con fijar c = F0 + C0, mientras que si buscamos una pendiente -1, entonces bastaría con fijar c = F0 – C0. Este aspecto no está embebido dentro del código para permitir al desarrollador, en el enfoque de motor genérico, poder trazar una línea de zona prohibida cualquiera en la matriz.

El atributo zona indica la zona que se considerará como región prohibida de los dos semiplanos que genera la recta. Con *zona=”negativa”*, en la aplicación, se realiza la comprobación de si siendo la ubicación del personaje principal en el patio exterior (F, C) con F

> 0, C > 0 –índices fila y columna de la matriz-, esta ubicación se encuentra dentro de la zona inferior a la línea prohibida, ya que sea cierto que $a(-F) + bC + c \leq 0$, de modo que el personaje principal estuviera realmente en zona prohibida; mientras que con *zona=”positiva”* la comprobación de si el personaje principal se encuentra en la zona prohibida se realiza comprobando la desigualdad $a(-F) + bC + c \geq 0$.

En el atributo *afecta-a*, se establece el tipo de personaje al que afecta la zona prohibida en concreto; si es 0 se trata de tipo prisionero, si es 1, de tipo soldado y si es 2, de tipo general. En esta aplicación siempre se establece que afecta al personaje principal de tipo prisionero. Pero se mantiene como concepto de generalidad para posibles líneas futuras del proyecto en las que las zonas prohibidas pudieran afectar a otros tipos de personajes.

Hay que destacar que se pueden poner tantas líneas prohibidas como se desee en la aplicación. Esto es gracias al empleo de *Colecciones* Java para albergar los datos de configuración. En concreto, gran parte de las estructuras de datos se basan en objetos *Hashtable* Java. Las zonas prohibidas se superponen unas sobre otras en base a la unión de conjuntos.

En nuestro caso, sólo se tiene la etiqueta de línea prohibida mostrada en el ejemplo y que barre todo el patio exterior de arriba a abajo, partiendo de la pared Norte del escenario. Para comprender cómo se *limpia* la zona central y el patio de ejercicios para que no sea zona prohibida, hay que recurrir a la siguiente etiqueta que nos encontramos en este fichero.

A las líneas prohibidas sigue la etiqueta *paralelogramos-habilitados* -sólo existe una- si es que tiene que haberla, ya que en la idea de generalidad de motor gráfico puede que no haga falta establecer paralelogramos habilitados que limpien zonas prohibidas. Entre el inicio y fin de esta etiqueta se definen los diferentes paralelogramos de regiones habilitadas.

```
<paralelogramos-habilitados>
    <!-- Resto de elementos xml -->
</paralelogramos-habilitados>
```

Los paralelogramos habilitados se definen mediante la etiqueta paralelogramo, a continuación se puede observar un ejemplo de definición de paralelogramo:

```
<paralelogramo afecta-a="0" lista-eventos="Libre,Ir A Formar,Ir A Comer,Libre 1,Ir A Ejercicio,Abandonar Ejercicio,Ir A Dormir">
    <!-- linea pend neg en fO = 14, cO = 112 lateral sup -->
```

```
<linea-paralelogramo a="1" b="1" c="-98" zona="negativa" />
<!-- linea pend pos en fO = 23 , cO = 63 lateral izq  -->
<linea-paralelogramo a="1" b="-1" c="86" zona="negativa" />
<!-- linea pend neg en fO = 80, cO = 88 lateral inf -->
<linea-paralelogramo a="1" b="1" c="-8" zona="positiva" />
<!-- linea pend pos en fO = 73, cO = 114 lateral der -->
<linea-paralelogramo a="1" b="-1" c="187" zona="positiva" />
</paralelogramo>
```

A la etiqueta *paralelogramo* le acompañan dos atributos: *afecta-a*, que ya se ha comentado, y *lista-eventos*. En el atributo *lista-eventos* lo que se expone es una lista con los eventos temporales para los que el paralelogramo se encuentra habilitado, es decir, los eventos en los que la zona definida por este paralelogramo deja sin efecto a cualquier zona prohibida que se extienda por ella. El paralelogramo mostrado en el ejemplo anterior comprende en su definición toda la zona central del patio exterior, por ello, el único evento temporal que no aparece en la lista es la Noche, ya que en ella todo el patio exterior se considera zona prohibida. En la **Figura 3.44** se puede observar un esquema de paralelogramo habilitado.

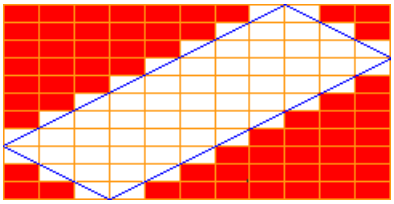


Figura 3.44 Imagen de un paralelogramo habilitado en la matriz de celdas –en blanco zona *limpiada* por el paralelogramo.

Dentro de la etiqueta se darán cuatro etiquetas *linea-paralelogramo* que, con el mismo concepto que *línea-prohibida*, definen los contornos del paralelogramo. A continuación, se muestra un ejemplo de esta etiqueta:

```
<!-- linea pend neg en fO = 14, cO = 112 lateral sup -->
<linea-paralelogramo a="1" b="1" c="-98" zona="negativa" />
```

Los atributos *a*, *b* y *c* tienen el mismo significado que para *línea-prohibida*, definen una de las cuatro rectas de un paralelogramo; mientras que *zona* indica la zona que es efectiva como permitida de igual forma que para *linea-prohibida*. El ejemplo nos indica la línea superior del paralelogramo para el centro del patio exterior. Esta forma de definir los contornos del paralelogramo entronca con la reutilización de los componentes de código empleados en la

definición de *línea-prohibida* y la idea de generalidad que se ha intentado dar al *pseudo-lenguaje de etiquetas* generado para esta aplicación.

De nuevo hay que decir que el número de paralelogramos que puede haber como elementos de la etiqueta *paralelogramos-habilitados* no está limitado, se pueden poner tantos como requiera el escenario del patio exterior para crear unas determinadas zonas prohibidas. Además, los paralelogramos se pueden superponer unos sobre otros.

Por último, en este fichero de configuración nos encontramos con la etiqueta que define qué habitación interior será considerada como celda de castigo a la que se trasladará al personaje principal una vez detenido. Esta etiqueta es *celda-castigo*. A continuación se muestra el ejemplo de esta etiqueta para la aplicación:

<celda-castigo nombre-habitacion="puerta-pared1-4-6-habitacion3"
url-xml="/habitaciones/habitacionesPuertaPared1_4_6/puertaPared1_4_6Habitacion3.xml"
en-puerta="0" tiempo-castigo="30"/>

Los atributos que acompañan a esta etiqueta son: *nombre-habitación*, que contiene el nombre de la habitación que se toma como celda de castigo; *url-xml*, que contiene la url donde se encuentra el fichero XML de configuración donde se define la habitación destinada a ser celda de castigo; *en-puerta*, que indica la puerta de dicha habitación donde será ubicado el personaje principal tras ser detenido y trasladado a la celda; y *tiempo-castigo* que comprende el número de iteraciones del *game loop* para la habitación que deberá permanecer en la celda de castigo el personaje principal hasta que se abra la puerta de la celda por la que entró, indicada en esta misma etiqueta en el atributo *en-puerta*, y pueda salir.

- *mapafronteraexterior.xml*:

La etiqueta raíz de este XML es:

<mapa-frontera-exterior xmlns="http://www.etsit.uma.es/pfc/the-great-escape">
<!-- Resto documento XML -->
</mapa-frontera-exterior>

El primer elemento de la marca raíz es la etiqueta que sirve para fijar la dimensión de la matriz en celdas. Se establece mediante la etiqueta autocontenida: *<dim-mapa num-filas="200" num-columnas="200"/>*, donde los atributos *num-filas* y *num-columnas* fijan el tamaño de la

matriz que contiene las marcas frontera -*matriz de fronteras*. Hay que destacar que todas las matrices para conservar coherencia con el desplazamiento del personaje deberán indicar un tamaño idéntico. Otro punto a destacar respecto de la dimensión de las matrices es el envoltorio de celdas de seguridad que se le aplica. A las matrices del juego se les añade un *envoltorio* de celdas vacías del tamaño de la pantalla del dispositivo; con estas celdas de seguridad, se evita que si se da que el personaje llegue a algún límite de la matriz tanto superior, inferior como lateral, por un fallo a la hora de construir los mapas, que no se quede sin celdas que recorrer la aplicación para representar esos bordes de la matriz propiamente dicha. Dentro de la aplicación se realizará el desplazamiento de los índices fila y columna de la matriz ampliada para que el personaje se encuentre en la ubicación correcta. En la **Figura 3.45** se muestra un esquema de esta idea de *envolver* la matriz.

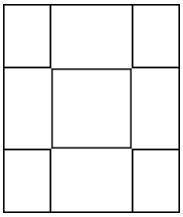


Figura 3.45 Esquema que representa el envoltorio de la matriz –el cuadrado central es la matriz, mientras que los rectángulos de las esquinas devienen del tamaño en celdas de la pantalla del dispositivo.

A esta etiqueta le sigue otra, ya mencionada, que especifica la ubicación de inicio del personaje principal. Esta etiqueta es *pos-ini-person*. Tiene dos grupos de atributos mutuamente excluyentes: *inicio-acostado* y el otro grupo formado por, *fila* y *columna*. A continuación se muestra ejemplo de uso de ambos:

<pos-ini-person inicio-acostado="si" />

En esta etiqueta se establece que el personaje principal debe aparecer de inicio acostado en la cama de su dormitorio, la información del dormitorio del personaje principal está embebida en el código de la aplicación. Mientras que con las etiquetas siguientes se establece que el personaje principal aparecerá de inicio en el patio exterior en la posición de la matriz indicada por los atributos *fila* y *columna*.

<pos-ini-person fila="123" columna="108" />

A continuación, le seguirá en el fichero XML una etiqueta que fija los límites desde donde se puede recoger los objetos abandonados en el campo. Esta etiqueta es:

<limite-recogida-objetos-patio-exterior fila="136" columna="88" />

Esta etiqueta con sus dos atributos *fila* y *columna*, establece la región en la que el control de objetos descubiertos cada amanecer es efectivo, quitando los escondites de piedra ya comentados en el **Capítulo 2**. Esta región será un paralelogramo de líneas en su lado Sur y Este paralelas a las líneas del vallado que va desde la celda de origen (0, 0), y que tiene la celda especificada en *fila* y *columna* como esquina Sureste del mismo.

Tras la etiqueta anterior sigue una etiqueta que da paso a las diagonales que delimitan los contornos de colisión en el patio exterior. Esta etiqueta es *diagonales*, entre su marca de inicio y fin se encuentran todas las diagonales de colisión del patio exterior. Se contemplan cuatro tipo de diagonales: *diag-pos-sup*, *diag-neg-sup*, *diag-pos-inf*, *diag-neg-inf*. A cada una de estas etiquetas le acompañan seis atributos: *fO* y *cO*, que indican la fila y columna de origen para introducir marcas en diagonal; *fD* y *cD*, que indican la fila y columna destino a partir de la cual debe dejarse de introducir marcas de la diagonal; *esp* que indica qué espaciado debe haber entre cada marca en la diagonal introducida en la matriz, partiendo de la marca de inicio; y, por último, *rompible*, la aparición de este atributo con el valor de *si* convierte la marca en rompible mediante el objeto especial de tenazas (estas marcas son las que se utilizan para hacer rompible la valla mediante las tenazas, aplicando un algoritmo de búsqueda de marca de frontera cercana en diagonal, para generar dos puertas que atraviesen la valla; si esa segunda marca de frontera rompible no se encontrara, entonces no se aplicaría la rotura de las diagonales de frontera).

Hay que destacar que las celdas de origen y destino deben estar en la misma diagonal de pendiente +1 o -1, en definitiva, siguiendo los ejes X e Y de la proyección isométrica, si no es así, la diagonal no será introducida en la matriz, dándose un aviso de error por consola. Otro aspecto que cabe destacarse es el hecho que la celda destino puede que no contenga marca en la matriz; esto es debido a que el espaciado puede provocar que en dicha celda no tenga que haber marca. A continuación, se muestra un ejemplo de uso de estas etiquetas:

<diag-pos-sup fO="1" cO="100" fD="24" cD="77" esp="0" rompible="si"/>

Los diferentes tipos diagonales de frontera responden a dos conceptos básicos a la hora de construir unos límites de colisión, que son la pendiente de la diagonal: que puede ser *pos* (+1) o *neg* (-1), y el sentido hacia el que se impide el paso con la marca: que puede ser *sup* (de

superior, por impedir el paso hacia la parte superior del escenario) e *inf* (de inferior, por impedir el paso hacia la parte inferior del escenario). En la **Figura 3.46** se puede ver un esquema de dichas diagonales con las zonas a las que no se puede acceder por las diagonales frontera en rojo.

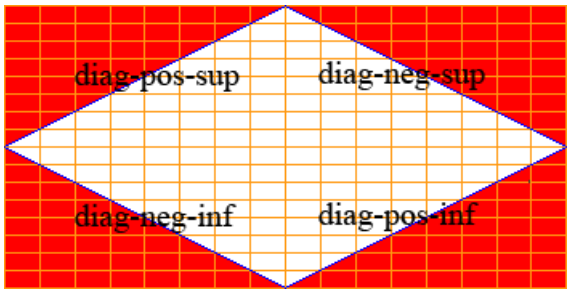


Figura 3.46 Diagonales frontera espaciado 0 y zonas no permitidas en rojo.

Además de las diagonales ya señaladas, en el interior de la marca *diagonales* también se puede dar las etiquetas de tipo aspa. Existen cuatro tipos de etiquetas aspa: *aspa-12*, *aspa-3*, *aspa-6* y *aspa-9*. Estas marcas tienen los mismos atributos que las marcas de diagonales vistas anteriormente. Un ejemplo de uso de esta etiqueta se muestra a continuación:

<aspa-3 fO="108" cO="45" fD="108" cD="45" esp="0" />

Las aspas son de utilidad para formar una esquina *hacia dentro* en una celda. La **Figura 3.47** muestra un ejemplo de aspa en la aplicación. Los cuatro tipos de aspas responden a los cuatro tipos de esquinas hacia dentro que pueden existir. El personaje principal en su movimiento por las fronteras puede colocarse sobre una de estas marcas. La clave de estas esquinas es que la marca aspa limitará el movimiento del personaje en dos sentidos, por ejemplo, la marca aspa-12, una vez que el personaje principal se ha colocado sobre ella impide los movimientos de éste asociados a las tecla arriba (sentido eje Y de la proyección negativo) y derecha (sentido eje X de la proyección positivo). La **Figura 3.48** muestra un esquema de matriz en el que se utilizan las cuatro aspas para generar un paralelogramo por donde moverse el personaje.

Con las marcas señaladas anteriormente de diagonales y aspas se construyen todas las fronteras que aparecen en el juego. La forma que se tuvo de construir los escenarios, tras pruebas intensivas para ajustar el método de representación y de colisiones, fue primero realizar todo el dibujo del escenario en cuestión y a continuación incorporar las marcas de frontera.

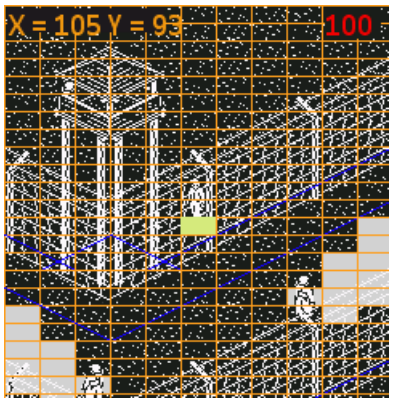


Figura 3.47 Imagen donde se aprecia el uso de las aspas –detalle a los pies de los postes de la torre de vigilancia.

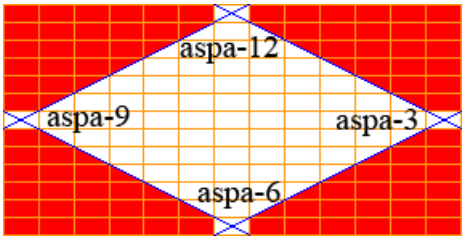


Figura 3.48 Esquema con los cuatro tipos de aspas.

Tras el fin de marca de *diagonales*, lo que nos encontramos en el fichero son las etiquetas de huecos. Un ejemplo de la marca *hueco*: *<hueco fO="184" cO="74" fD="184" cD="74" esp="0" />*; ésta posee los mismos atributos que las anteriores marcas. Su función es limpiar de marcas de diagonal o aspa introducida la matriz por la diagonal que define. Muchas veces su uso se limita a una sola celda como es el caso del ejemplo mostrado. Para que funcione correctamente debe colocarse después de la etiqueta que define marca de colisión introducida a limpiar.

- *mapadibujoexterior.xml*:

La marca raíz de este fichero es *mapa-dibujo-exterior*. Esta marca contendrá diferentes elementos. Al igual que en *mapafronteraexterior.xml* lo primero que se establece en este documento es la dimensión de la *matriz de dibujo*, coincidente con la ya indicada para la *matriz de fronteras*: *<dim-mapa num-filas="200" num-columnas="200"/>*. A esta marca sigue otra para indicar la secuencia de *sprites* que definen la apariencia gráfica del personaje principal:

<secuencia-sprite url="/imagenes/framesPrisionero.png" ancho-sprite="15" alto-sprite="30" />

En el atributo *url* se especifica el fichero de que contiene la secuencia del *sprite*, con *ancho-sprite* se indica el ancho en píxeles de cada *frame* del *sprite*, mientras que en *alto-sprite* lo que se indica es la altura en píxeles del *frame* para el personaje principal. Es de destacar que el *sprite* de todos los personajes consta de doce *frames* con tres *frames* por cada uno de los cuatro movimientos (uno con el pie izquierdo adelantado, otro con los pies juntos y el tercero con el pie derecho adelantado). En la **Figura 3.49** se muestra un ejemplo de *sprite* para el prisionero. Se debe destacar que los *sprites* de los personajes secundarios se definen en el fichero *personajes.xml*, por lo que se podría tener unos *frames* completamente diferentes para los personajes secundarios prisioneros y para el personaje principal.

Figura 3.49 Los doce *frames* personaje principal.

A continuación, se tendrá la marca *<imágenes><!-- elementos de imágenes --></imágenes>*; en su interior se señalarán todas las imágenes que se utilizarán para generar este escenario -el patio exterior. Para ello, se utiliza la siguiente etiqueta:

<imagen nombre="valla-der" url="/imagenes/vallaDer.png" />

Con el atributo *nombre* se está indicando el identificador que reconocerá a esta imagen en todo lo sucesivo a este XML, es decir, cuando posteriormente en el XML se quiera hacer la representación de esta imagen bastara con identificarla con su nombre. Por otro lado con *url* lo que se está indicando es el fichero, siempre en formato *png*, que contiene la imagen en cuestión.

Hay que destacar que no existe límite al número de imágenes, y, por tanto, de etiquetes *imagen*, que se pueden utilizar en los ficheros XML, siempre que los recursos de memoria del dispositivo lo permitan, ya que estas imágenes se albergarán en las estructuras de datos que proporciona Java: las colecciones -concretamente, se hace un amplio uso de la *Hashtable* de Java.

Tras esta marca de *imágenes* viene la de las diagonales de dibujo, de las que ya se ha hablado un poco en el **Apartado 3.7** de **Dibujo de escenarios** de este capítulo. La marca que las contendrá es *diagonales* (*<diagonales><!--diagonales de dibujo--></diagonales>*). Los elementos de esta marca tendrán la forma de etiqueta autocontenida como:

<diagonal nombre="pared-izq-sin-puerta" fO="23" cO="77" fD="0" cD="100" esp="5" anclaje="4"/>

El atributo *nombre* hace referencia al nombre de la imagen a utilizar, este nombre debe estar definido en el cuerpo de *imágenes*, si no fuera así, se daría un aviso de error durante la carga de este mapa –no obstante, la aplicación seguiría cargando las restantes imágenes y el resto del mapa. Le siguen a *nombre* los cinco atributos que definen la diagonal (*fO="23" cO="77" fD="0" cD="100" esp="5"*), ya comentados cuando se hizo referencia al fichero *mapafronteraexterior.xml*. El siguiente atributo es *anclaje*. Este atributo especifica el anclaje que llevará la imagen a la hora de insertarse en la matriz. Deteniéndonos un poco en el atributo *anclaje*, hay que decir que existen contemplados en la aplicación nueve anclajes. El anclaje de la imagen comprende la fijación del punto de referencia en la imagen y del punto de referencia en la celda. En la aplicación ambos puntos son coincidentes, es decir, si por ejemplo se fija como anclaje la esquina superior izquierda, *anclaje="1"*, entonces la imagen a dibujar tendrá como punto de referencia la esquina superior izquierda y se fijará ese punto de referencia en la esquina superior izquierda de la celda que contiene la marca de dibujo. La **Figura 3.50** muestra un esquema de los nueve tipos de anclaje.

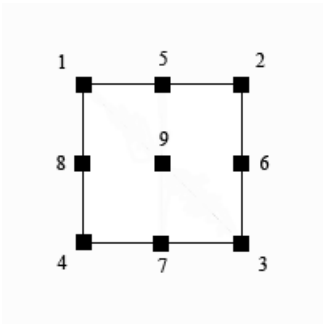


Figura 3.50 Esquema donde se muestran los anclajes de imagen y celda.

En la práctica los más utilizados para el dibujo en perspectiva isométrica son las esquinas, es decir, los correspondientes con los identificadores 1, 2, 3 y 4, ya que enlaza con las ideas de las diagonales de dibujo y de los ejes X e Y de la proyección.

En el ejemplo de diagonal mostrado en la página anterior no aparecen los tres siguientes atributos que puede llevar esta marca. Ahora se muestra un ejemplo de dicha marca completa, con todos los atributos que puede llevar:

<diagonal nombre="valla-der" fO="109" cO="102" fD="72" cD="65" esp="2" anclaje="3" plano="2" reflexion="1" anclaje-referencia="4" />

El atributo *plano* indica en plano de dibujo debe insertarse la imagen. Existen tres planos de dibujo, en el **Apartado 3.7 Dibujo de escenarios** se les nombró por plano *uno*, primer plano de dibujo, contiene el suelo y los objetos sobre los que proyectan sombra los personajes, plano *dos*, plano de dibujo de los personajes del juego, y plano *tres*, plano de dibujo de los objetos que proyectan sombra sobre el plano *dos*, es decir, sobre los personajes. En la aplicación los únicos planos accesibles a través de las marcas de dibujo son el plano *uno* y el *tres*. En concreto, al ser dos planos, se especifica que con *plano="1"* se accede a representar en el plano *uno*, mientras que con *plano="2"* se accede a representar en el plano *tres*. Cuando no se especifica el atributo *plano* la aplicación toma por defecto que dicha imagen está ubicada en el plano *uno*.

También se comprende que para dibujo se utilice la etiqueta *hueco* con igual formato a la ya comentada de *mapafronteraexterior.xml*. Con ella, se conseguirá el efecto de producir una limpieza en matriz de dibujo de marcas de dibujo previas que pudieran existir.

Los siguientes atributos *reflexion* y *anclaje-referencia* se encargan de establecer transformaciones en las imágenes a representar. Los indicadores de reflexión se traducen de la siguiente manera:

- *reflexion="1"*: Se realiza una transformación de *espejo* a lo largo del eje vertical central de la imagen. Resulta como en un volteo horizontal de la imagen.
- *reflexion="2"*: Se realiza una transformación de *espejado* y a continuación una rotación de 90º en el sentido de las agujas del reloj.
- *reflexion="3"*: Se realiza una transformación de *espejado* y a continuación una rotación de 180º en el sentido de las agujas del reloj.
- *reflexion="4"*: Se realiza una transformación de *espejado* y a continuación una rotación de 270º en el sentido de las agujas del reloj.
- *reflexion="5"*: Se realiza una rotación de 90º en el sentido de las agujas del reloj.
- *reflexion="6"*: Se realiza una rotación de 180º en el sentido de las agujas del reloj.
- *reflexion="7"*: Se realiza una rotación de 270º en el sentido de las agujas del reloj.

Por último referente al atributo *reflexion* la utilización de cualquier otro identificador diferente a los señalados no producirá ningún efecto sobre la imagen a representar; cuando no se coloca el atributo se entiende en la aplicación que no hay que realizar ninguna transformación.

El atributo *anclaje-referencia* se utiliza para fijar el píxel que va a servir de referencia en la imagen a la hora de representarla; se trata de establecer el píxel de la imagen que se deberá

fijar al píxel de la pantalla establecido según el atributo *anclaje* y la celda que ocupe la marca de dibujo en la matriz. A fin de simplificar un poco su manejo se establecen unos identificadores como los utilizados para el atributo *anclaje*. Véase la **Figura 3.50** para saber la traducción entre identificador y ubicación de píxel de referencia en la imagen.

A continuación en el fichero se tiene la marca *<marcas-especiales><!--etiquetas especiales--></marcas-especiales>*. En su interior nos encontramos con la etiqueta especial autocontenida *suelo*:

<suelo nombre="suelo" esp-ver-suelo="5" esp-hor-suelo="5"/>

Esta etiqueta se encarga de generar el suelo del escenario principal –el *césped* de puntitos blancos que constituye el fondo del patio exterior, dibujo contenido en el plano de dibujo uno, con la particularidad de que siempre es lo primero que se dibuja de dicho, haya o no más marcas.

Al ser una representación periódica a lo largo de todo el plano, no tiene marcas en la *matriz de dibujo*, está desligada de ella, evitándose posibles problemas de superposición de marcas de dibujo. Indicándose el espaciado en vertical y horizontal en celdas para la matriz de cada cuanto tiene que dibujarse con los atributos *esp-ver-suelo* y *esp-hor-suelo*, respectivamente. El atributo *nombre* hace referencia a la imagen contenida en el cuerpo de *imagenes* donde está el suelo.

Por último en el interior de *marcas-especiales* se tiene la etiqueta *objetos-especiales* etiqueta que podrá contener en su cuerpo las etiquetas que definen un objeto especial –de los que puede recoger el personaje principal- para el escenario del patio exterior. Las etiquetas de objetos contempladas para esta aplicación son:

- *llave*:

<llave nombre="Llave Verde" fO="94" cO="102" url-imagen="/imagenes/llave.png" abre-puerta="8" en="patio-exterior" />

Esta etiqueta posee los siguiente atributos que se pueden apreciar en el ejemplo anterior: *nombre*, indica el nombre del objeto que aparecerá por pantalla, por ejemplo cuando se recoja, *fO* y *cO*, indica la fila y la columna en la que se ubicará el objeto en la matriz, *url-imagen*, indica el fichero de imagen *png* que contiene la imagen del objeto, *abre-puerta*, indica el

identificador de puerta que abre la llave –véase comentarios al fichero *mapapuertasexterior.xml*-, y *en*, atributo que hace referencia al nombre del escenario donde la llave abre puerta con identificar igual al señalado.

El resto de etiquetas para establecer los objetos tienen, todas, los mismos cuatro atributos primeros de la etiqueta *llave* (*nombre*, *fO*, *cO* y *url-imagen*). El nombre de la etiqueta coincide con el nombre del objeto por lo que se tendrán las etiquetas: *pala*, *linterna*, *herramientas*, *documentos*, *radio*, *tenazas*, *bolsa* y *brújula*.

- *mapapuertasexterior.xml*:

La etiqueta raíz de este documento es *mapa-puertas-exterior*. Al igual que en los casos anteriores lo primero que se realiza en este documento es establecer la dimensión de la *matriz de puertas* para el patio exterior. Esta dimensión se establece mediante la etiqueta *dim-mapa*, ya comentada en el caso de *mapafronteraexterior.xml* y *mapadibujoexterior.xml*.

Tras la etiqueta que establece la dimensión del mapa para las puertas, se tendrá la etiqueta *puertas* (*<puertas><!--puertas en el patio exterior --></puertas>*). Los elementos interiores de esta marca serán las puertas definidas en el escenario patio exterior. Para definir las puertas se tiene la etiqueta *puerta*. A continuación se muestra un ejemplo de esta etiqueta, que irá contenida entre la marca de inicio y fin de *puertas*.

<!-- puerta pared nº 5 -->

<puerta id="8" enlace-con="puerta-pared5-habitacion1" url-xml="/habitaciones/habitacionesPuertaPared5/puertaPared5Habitacion1.xml" entrada="0" tipo="diag-neg-sup" control-entrada-salida="si" inicio-activo="no" bidireccional="si" fO="49" cO="143" fD="49" cD="143" />

El primer atributo que tiene esta marca es el atributo *id*. Cada puerta de cada escenario tendrá un identificador, *id*, único y entero mayor o igual que cero, que se utilizará para hacer referencia a ella en cualquier salto posible junto con el nombre de la habitación en la que se encuentra la puerta con dicho identificador.

El siguiente atributo, *enlace-con*, indica el nombre de la habitación interior o, en el caso de que aparezca de la forma *enlace-con=”patio-exterior”*, el patio exterior del castillo, al que conduce la puerta con el *id* señalado en este escenario patio exterior. Relacionado con este atributo está el siguiente *url-xml*, en él se define la ubicación del fichero XML de la habitación

con la que enlaza esta puerta del patio exterior. En el caso de que el enlace sea entre puertas del mismo patio exterior, caso contemplado cuando en el atributo *enlace-con* se coloca el valor *patio-exterior*, este atributo deberá estar vacío, es decir, se presentará de la forma *url-xml=""*. El atributo *entrada* indica el identificador de la puerta destino en la habitación indicada en *enlace-con*, leyendo la marca del ejemplo del párrafo anterior sería: en el escenario patio exterior (puesto que la etiqueta se encuentra en el fichero *mapapuertasexterior.xml*) define la puerta con identificador *8* que enlaza con la puerta *0* de la habitación *puerta-pared5-habitacion1* y que se encuentra definida en el fichero XML */habitaciones/habitacionesPuertaPared5/puertaPared5Habitacion1.xml*.

El atributo *tipo* indica el tipo de diagonal de entrada que posee la puerta. Estas diagonales de entrada coinciden con las diagonales frontera descritas en el comentario al fichero *mapafronteraexterior.xml*, y pueden observarse en la **Figura 3.46**.

El atributo *control-entrada-salida* es un atributo opcional, si no se coloca la aplicación le dará por defecto el valor de *control-entrada-salida="no"*. Cuando aparece, para que tenga efecto debe tener el valor de *control-entrada-salida="si"*. Con él, se indica que la puerta del patio exterior está bajo una vigilancia especial por parte de los guardias del campo; de modo que si un guardia observa al personaje principal entrando o saliendo de ella, automáticamente activará la alarma de búsqueda del prisionero, por considerarse que el personaje principal ha hecho un acto indebido. Este atributo sirve para generar el control de entradas y salidas que se efectúa sobre las puertas de intendencia (puertas uno, dos, tres y cuatro del mapa del **Capítulo 2** de la **Figura 2.21**).

Con el atributo *inicio-activo* se controla si la puerta definida para el escenario patio exterior se encuentra abierta de inicio (caso en que *inicio-activo* toma el valor de *si*), o si la puerta se encuentra cerrada de inicio (caso en que inicio-activo toma el valor de *no*). En el último caso, cada vez que se produzca un reset del campo de prisioneros porque amanezca un nuevo día en el campo todas las puertas que estuvieran puestas a *inicio-activo="no"*, se comprobarán y si estuvieran abiertas se volverían a cerrar como era su estado inicial.

El atributo *bidireccional* se utiliza para crear una puerta con cerradura por los dos lados de la puerta. Para un correcto funcionamiento de este atributo la puerta de destino debe indicarse como puerta con *inicio-activo="no"*, es decir, puerta cerrada de inicio y también debe aparecer el atributo *bidireccional="si"* para que estén enlazadas. Así cuando una de las dos puertas se abra, por ejemplo por el uso de las herramientas entonces la otra puerta también se abrirá.

Todo lo anterior referente al atributo *bidireccional* sirve para distinguir respecto de la construcción de una puerta batiente con una sola de las puertas cerrada con cerradura y la otra abierta. Esto se produce cuando la puerta de partida se establece como de *inicio-activo="no"*, es decir, cerrada y la puerta destino se establece como puerta de *inicio-activo="si"*, de esta forma se pueden construir dos tipos de puertas, las puertas batiente con cerradura en una sola de las puertas y las puertas con cerradura a operar por los dos lados de las puertas. Debe destacarse que en cualquiera de los casos una vez abierta la puerta, ya se por utilizar una llave o las herramientas, la puerta permanecerá abierta hasta la revisión que se realiza en el campo todos los amaneceres, momento en que se volverán a cerrar todas las puertas que estuvieran abiertas.

Por último, se tienen los atributos que definen la diagonal de marcas a introducir sobre la *matriz de puertas*: *fO* y *cO* definen la celda de origen en la matriz de la diagonal que se seguirá para introducir las marcas de puertas, mientras que *fD* y *cD* define la celda destino en la matriz para dicha diagonal. Estas diagonales se pueden trazar en cualquier sentido, siguiendo siempre direcciones paralelas a los ejes de la proyección isométrica.

No existe limitación en cuanto al número de puertas que se pueden introducir, siempre que los recursos de memoria del dispositivo lo permitan, las estructuras de datos donde se alberga la información de los *Beans* para las puertas están basadas en las colecciones Java que presenta J2ME.

- *tuneles.xml*:

El documento comienza con la etiqueta raíz *tuneles*. En su interior se encuentran las definiciones de los túneles que se pueden dar en la aplicación de videojuego. La definición de un túnel se realiza a través de la etiqueta *tunel*, a continuación se puede ver un ejemplo de uso de esta etiqueta:

<tunel id="0" nombre-tunel="tunel1-patio-exterior-a-barracon2-habitacion1-TUNEL-A" tiempo-tunel="10" url-img-tunel-bloq="/imagenes/salidaTunelBloqueada.png" url-img-tunel-desbloq-ida="/imagenes/salidaTunelDesbloqueadaIda.png" url-img-tunel-desbloq-vuelta="/imagenes/salidaTunelDesbloqueadaVuelta.png">

La etiqueta de inicio consta de una serie de atributos: *id*, es el identificador del túnel, utilizado para un indexado rápido en tabla Hash de éste en la aplicación –entero mayor o igual que cero, único-; *nombre-tunel*, contiene el nombre del túnel con el que se le conocerá a efectos del interior de la aplicación, parecido al nombre de la habitación; *tiempo-tunel*, contiene el número de bucles del *game loop* que comprenderá atravesarlo de inicio a fin, siempre que la

salida por el otro extremo no esté bloqueada; *url-img-tunel-bloq*, contiene la ubicación del fichero con la imagen de la salida del túnel bloqueada por un derrumbe –hay que tener en cuenta que sólo se considera que el personaje se encontrará la salida bloqueada cuando vaya en sentido de dentro del campo hacia fuera del perímetro, siendo esta imagen de salida bloqueada la que se contiene en la url; *url-img-tunel-desbloq-ida*, contiene la url de la imagen del movimiento de ida, considerándose este sentido el ya comentado anteriormente; y *url-img-tunel-desbloq-vuelta*, contiene la url de la imagen del movimiento de vuelta en el túnel. Estas imágenes de ida y vuelta están enlazadas con las etiquetas que definen las entradas y que se comentarán a continuación.

La etiqueta *entrada-tunel* define las dos entradas que puede tener un túnel; hay que destacar que se establece para la aplicación que la entrada con identificador 0 es la que se considera como entrada natural de sentido ida hacia el perímetro de vallas, de esta forma las imágenes anteriores quedan enlazadas correctamente con su sentido correspondiente. Un ejemplo de estas etiquetas que se contendrán en el cuerpo de *tunel* es:

<entrada-tunel id="0" ubicacion="barracon2-habitacion1" url-xml="/barracones/barracon2/barracon2Habitacion1.xml" tipo="diag-pos-sup" inicio-bloqueado="no" fO="3" cO="1"/>

<entrada-tunel id="1" ubicacion="patio-exterior" url-xml="" tipo="diag-neg-inf" inicio-bloqueado="si" fO="85" cO="70"/>

Los atributos que posee son: *id*, identificador de la entrada, ida 0, vuelta 1, *ubicacion*, indica el nombre de la habitación interior donde se encuentra ubicada la entrada que se está definiendo –si la ubicación fuera el patio exterior, ésta se codifica con la cadena *patio-exterior*, *url-xml*, da el fichero XML que define la habitación interior en la que se encuentra la entrada –en el caso de estar ubicada la entrada del túnel en el patio exterior deberá tener la forma de cadena vacía (*url-xml=""*), *tipo*, al igual que el atributo con el mismo nombre para las etiquetas de *puerta* define un sentido de entrada para el túnel, sentido que se conservará dentro del túnel para indicar si se avanza o en caso de pulsar la tecla de movimiento de sentido opuesto realizar el camino de vuelta por el túnel, *inicio-bloqueado*, atributo que bloquea dicha entrada del túnel, haciendo necesario el uso de la pala para desbloquearla desde la superficie, si es el caso de que se vaya a introducir por ella el personaje, o desde el interior del túnel, si es el caso de que dicha entrada al túnel sea la salida en el camino que recorre el personaje a través del interior del túnel –puede tomar dos valores *si* o *no*, y, por último, *fO* y *cO*, definen la celda de la matriz donde se encuentra el túnel en el escenario especificado por *ubicación*.

- *tiempo.xml*:

La etiqueta raíz es *tiempo*, contendrá diferentes elementos. El primero es el comprendido entre la marca de inicio y fin para una etiqueta llamada *eventos-horario*. A esta etiqueta le acompañará un atributo: *periodo*. El valor de *periodo* es un número entero. Indicará la duración en milisegundos de un día en el juego. Un ejemplo de esto comentado es: *<eventos-horario periodo="960000">*

En el cuerpo de *eventos-horario* se tendrán los diferentes eventos temporales que se dan el juego. Estos eventos se definen a través de las etiquetas *evento*. A continuación se muestran tres de estas etiquetas autocontenidas:

<evento nombre="Libre" delay-inicial="0" texto-pantalla="LEVANTARSE" />
<evento nombre="Ir A Formar" delay-inicial="60000" texto-pantalla="FORMAR" />
<evento nombre="Libre" delay-inicial="900000" texto-pantalla="AMANECER" reset-campo-prisioneros="si" />

El atributo *nombre* se utilizará más tarde para hacer referencia a las acciones a realizar cada vez que se dé el evento en cuestión que lo posee. En la aplicación desarrollada, se utilizan las definiciones de las siguientes acciones: *Libre*, *Ir A Formar*, *Formar*, *Ir A Comer*, *Comer*, *Libre*, *Ir A Ejercicio*, *Ejercicio*, *Abandonar Ejercicio*, *Libre 1*, *Ir A Dormir*, *Dormir* y *Noche*. Algunas de estas acciones se repiten en los dieciséis eventos que se emplean para construir la rutina temporal del juego.

El atributo *delay-inicial* indica el momento de aparición del evento en el período indica en el atributo de *eventos-horario*. El otro atributo de *evento texto-pantalla* indica la cadena de texto que se mostrará por pantalla cuando el evento salte.

El atributo *reset-campo-prisioneros* obliga en el evento en el que aparece a que se produzca un reset del campo, es decir, se cierran las puertas que hubieran sido abiertas, se reparan las aberturas en el vallado y se rastrean los posibles objetos que hubiera sido soltados por el campo, siendo devueltos a su ubicación original.

A la marca de fin para *eventos-horario* le siguen tres bloques de etiquetas encapsuladas en la marca madre *acciones-evento* cada uno. A la etiqueta de inicio de *acciones-evento* le acompaña un atributo: *tipo-personaje*. Este atributo establece que el bloque de acciones que se

comprende en el interior de la etiqueta *acciones-evento* quedará circunscrito al tipo de personaje secundario que establezca. Si el valor de *tipo-personaje* es 0, para los prisioneros, si es 1, para los soldados, y si es 2, para el general.

Dentro de *acciones-evento* se tendrán tantos cuerpos de etiquetas de *accion* como eventos con nombre diferentes hayan sido definidos en *eventos-horario*. La etiqueta *accion* tiene la forma siguiente: *<accion nombre="Libre"><!-- definición de la acción a realizar --></accion>*.

En la definición de *accion* habrá dos etiquetas que se pueden presentar tanto en forma autocontenida como en forma de bloque con cuerpo. Estas etiquetas son: *secuencia-saltos* y *fin-accion*. Combinándolas adecuadamente junto con las definiciones contenidas en el fichero *personajes.xml*, se consiguen los movimientos de los personajes secundarios en el juego.

La etiqueta *secuencia-saltos* posee un atributo que es *tipo*. Este atributo puede tomar los siguientes valores: *predefinida-generica* y *predefinida-privada*. En *predefinida-generica* todos los personajes a la que afecte esta acción se moverán por igual según la definición contenida en *secuencia-saltos* y *fin-accion*. En *predefinida-privada* cada personaje tendrá una definición de saltos privada y particular que estará comprendida en el fichero *personajes.xml*. En el caso de *tipo="predefinida-privada"* la etiqueta será autocontenida (*<secuencia-saltos tipo="predefinida-privada" />*), mientras que para el caso de *tipo="predefinida-generica"*, la etiqueta tendrá un cuerpo donde se definan los saltos a realizar. Estos saltos se indicarán a través de la etiqueta *salto*. Un ejemplo de cuerpo de la etiqueta *secuencia-saltos* se muestra a continuación:

```
<secuencia-saltos tipo="predefinida-generica">
        <salto id="0" fD="33" cD="78"/>
        <salto id="1" fD="20" cD="72"/>
</secuencia-saltos>
```

A la etiqueta salto le acompañan tres atributos: *id*¸ el identificador, entero mayor o igual que cero, que indica el orden que ocupa el salto en la secuencia, *fD* y *cD*, fila y columna de la celda de destino del salto en la matriz –será el motor de movimientos para los personajes secundarios el que se encargue de llevar a los personajes hasta su destino, una vez cargado éste. Cabe destacarse que el número de saltos que se pueden introducir en una secuencia no está limitado.

La otra etiqueta que sirve para definir la acción es *fin-acción*. Le acompaña un solo atributo, *tipo*, en función del cual será autocontenida o con cuerpo. Los distintos tipos de fin de acción posibles son: *estatico-izq*, *estatico-der*, *estatico-arr* y *estatico-aba*. Estos valores indican que el personaje secundario en cuestión llegará a la celda de destino o, en su defecto, al perímetro de celdas inmediatas y permanecerá estático con la orientación izquierda, derecha, arriba o abajo.

Un aspecto importante es cuando el personaje secundario es trasladado a la puerta del comedor o a la puerta de su dormitorio en un evento en el que este personaje debe entrar dentro de esas habitaciones. En este caso, unos controladores especiales de la aplicación: el *ControladorDormir* y el *ControladorComer*, se encargan de que una vez que el personaje llegue al perímetro de la puerta de entrada a dicha habitación introducirlo en ella de forma controlada. De igual manera estos controladores se encargan de extraer a los personajes de forma controlada cuando se cumpla el tiempo que deben estar dentro de la habitación. Pues bien, en este tipo de traslados a entradas a habitaciones en las que serán introducidos los personajes secundarios el fin de acción que se emplea es el estático anterior señalado. La forma de este fin de acción autocontenido es: *<fin-accion tipo="estatico-der" />*.

Por otro lado, está la forma de fin de acción con cuerpo. Esta se da cuando se establece el *tipo="aleatorio"*. Cuando se usa este tipo de fin de acción en el cuerpo de *fin-accion*, se tiene otra etiqueta, *region*, que especifica la región en la que se moverá de forma aleatoria el personaje secundario al que afecte. A continuación se muestra un ejemplo del uso de *fin-accion* en esta circunstancia:

```
<fin-accion tipo="aleatorio">
    <region fO="0" cO="100" fD="84" cD="98" />
</fin-accion>
```

La etiqueta *region* tiene cuatro atributos que definen un rectángulo de celdas en la matriz por el cual los personajes secundarios, afectados por la forma de resolver esta acción, se moverán aleatoriamente. Para ello, el motor de movimientos para los personajes secundarios cargará aleatoriamente un destino dentro de dicha celda y lo llevará hasta ella, con la diferencia ahora de que si se produce una colisión el motor de movimientos no persiste en trasladar al personaje secundario hasta su destino aleatorio fijado previamente, sino que simplemente cambia a otro destino aleatorio dentro de la región. Este hecho constituye una diferencia respecto a como actúa el motor de movimientos para cumplir las secuencias de saltos. En este último caso, el motor sí persiste en el intento de trasladar al personaje principal a su destino,

haciendo todas las correcciones de movimientos necesarias para resolver posibles colisiones. En este sentido, el simplificar el algoritmo para las colisiones en un final de acción aleatorio supone la ventaja de que como las regiones suelen ser pequeñas, como es el caso del patio de ejercicio cuando los prisioneros y los guardias deambulan por él, conduce a una agilización de los movimientos, frente a la situación de repetida colisión y su correspondiente resolución.

Hay que destacar que el ajuste de todos estos parámetros requirió un período de pruebas muy intenso hasta fijar unos términos de configuración que permitieran abordar de manera satisfactoria la compleja movilidad de los personajes secundarios en el patio exterior.

- *personajes.xml*:

Su marca raíz es *personajes*. Como elementos tendrá unas marcas con cuerpo llamadas *personaje*, que definirán cada uno de los personajes secundarios que existirán en el juego. No existe límite de partida al número de personajes secundarios que se pueden introducir, salvo las capacidades de procesado y memoria de que disponga el dispositivo móvil.

La etiqueta *personaje* define cada uno de los personajes y además termina por complementar la movilidad de éstos que no estuviera contemplada en *tiempo.xml*. Posee una serie de atributos que son: *nombre*, cadena de texto del nombre por el que se conocerá al personaje en la aplicación, *tipo*, *0*, si se trata de prisionero; 1, si se trata de soldado; y 2, si se trata de general, *url-sprite*¸ contiene la url del fichero donde se encuentra el *sprite* de doce *frames*, tres por cada uno de los cuatro movimientos, que definen la apariencia gráfica de este personaje, *ancho-sprite* y *alto-sprite*, cada una indicadora del tamaño de un *frame* del personaje en píxeles para el ancho y el alto, de forma que pueda ser extraído por la aplicación del fichero imagen, *fO* y *cO*, indicadores de la fila y columna de la matriz de celdas del patio exterior en la que aparecerá el personaje de arranque de la aplicación, y, por último, *lista-eventos-temporales*, contiene la lista de los eventos temporales, se refiere al atributo *nombre* que designa la acción, ante los que responde el personaje (la omisión de uno de estos eventos produce que cuando se dé éste, el personaje simplemente se quede parado). A continuación se muestra ejemplo de estos atributos:

<personaje nombre="prisionero1" tipo="0" url-sprite="/imagenes/framesPrisionero.png" ancho-sprite="15" alto-sprite="30" fO="44" cO="109" lista-eventos-temporales="Libre,Ir A Formar,Formar,Ir A Comer,Comer,Ir A Ejercicio,Ejercicio,Abandonar Ejercicio,Libre 1,Ir A Dormir,Dormir,Noche" >

Dependiendo del tipo de personaje que se esté definiendo y de cómo se definiera su movilidad para las diferentes acciones en *tiempo.xml* existirán diversas etiquetas en el cuerpo de *personaje*.

Si el tipo se estableció a prisionero (*tipo="0"*), entonces habrá dos bloques de etiquetas al inicio del cuerpo: el que responde a la configuración de acceso al dormitorio cuando sea tiempo de dormir para el prisionero (etiqueta, *dormitorio*) y el que se refiere a la configuración de acceso al comedor para cuando sea tiempo de comer (etiqueta, *comedor*). Cada prisionero que se defina deberá llevarlas, para el correcto funcionamiento: en los dormitorios, porque prisioneros diferentes para la aplicación van a dormitorios diferentes, y en el comedor, por similitud con el caso anterior, aunque sólo exista un único comedor para el juego, pero pudiendo contemplar para una línea futura de expansión la inclusión de más de un comedor. A continuación, se muestra un ejemplo de estas marcas señaladas para los prisioneros.

```
<dormitorio nombre="barracon2-habitacion2" lista-eventos-tiempo-llevar="Ir A Dormir,Dormir,Noche" evento-tiempo-sacar="Libre">
    <puerta-dormitorio fD="45" cD="108" />
</dormitorio>

<comedor nombre="puerta-pared1-4-6-habitacion5" lista-eventos-tiempo-llevar="Ir A Comer,Comer" evento-tiempo-sacar="Libre">
    <puerta-comedor fD="20" cD="72" />
</comedor>
```

Ambos bloques son parecidos en cuanto a atributos y marcas internas. El atributo *nombre* indica la habitación interior que a tal efecto actuará como dormitorio o comedor. El siguiente atributo *lista-eventos-tiempo-llevar* indica los eventos temporales en los que un personaje secundario que se encuentre por el patio exterior y para el cual su definición de secuencia de saltos le permita alcanzar la puerta de entrada de la habitación designada como dormitorio o comedor, será introducido dentro de la habitación.

Existe una lista de eventos para comprender todo el intervalo de tiempo que el personaje secundario puede estar dentro de la habitación, de modo que si en su deambular por el patio exterior se retrasara y no pudiera alcanzar la entrada en el evento *Ir A …*, entonces tendría otra oportunidad con los siguientes eventos comprendidos en la lista para entrar. Esto es así ya que se establecen dos controladores en la aplicación que el *ControladorDormir* y el *ControladorComer*, que introducirán al prisionero en el interior de la habitación si éste se acerca

al perímetro de la puerta, por eso se establecen los eventos en los que un prisionero puede ser introducido dentro de la habitación, para evitar que en el deambular aleatorio de algún caso éste fuera introducido por error dentro de la habitación.

El atributo *evento-tiempo-sacar* especifica el evento concreto en que los controladores de los que se ha estado hablando en el párrafo anterior, extraen a los prisioneros de las habitaciones de dormitorio y comedor. Las etiquetas *puerta-dormitorio* y *puerta-comedor* junto con sus atributos definen concretamente la ubicación de la puerta de entrada de la habitación dormitorio y comedor. Estas etiquetas pueden resultar redundantes pues ya se conoce la habitación a la que se tiene que trasladar, pero se trata de una forma directa de pasar la información de la ubicación de la puerta a los controladores, sin introducir mayor complejidad a las definiciones de puertas en el fichero *mapapuertasexterior.xml*.

Las siguientes marcas estarán presentes en todos los personajes secundarios para los cuales en *tiempo.xml* se definiera una acción como secuencia de saltos predefinida privada. La marca es:

```
<secuencia-saltos-privada accion="Ir A Formar">
        <salto id="0" fD="10" cD="99"/>
        <fin-accion tipo="estatico-izq" />
</secuencia-saltos-privada>
```

Existe una marca de este tipo por cada acción predefinida privada que se definiera en el fichero *tiempo.xml* correspondiente para el tipo de personaje que esté definiendo la etiqueta *personaje*. Posee un atributo *accion* que indica el nombre de la acción correspondiente en la que toma el control de la movilidad del personaje secundario. El cuerpo es idéntico al ya explicado en el fichero *tiempo.xml* para la etiqueta *secuencia-saltos*: unas etiquetas secuenciadas de *salto* y una de *fin-accion*.

- *cruzroja.xml*:

La marca raíz es *cruz-roja*. Posee como elementos las etiquetas: *horario-llegadas*, *envoltorio-objetos-cruz-roja* y *objetos-cruz-roja*.

La primera etiqueta: *<horario-llegadas periodo="960000" instante-de-llegada-de-objeto="15000" />*, configura el *Timer* de la Cruz Roja para la entrega de los objetos de la Cruz Roja. El atributo *periodo* define el período de tiempo en milisegundos en el que se irán

entregando los objetos, mientras que *instante-de-llegada-de-objeto* indica el instante en milisegundos dentro del período en el que se entregarán los sucesivos objetos, uno cada vez que se cumpla dentro de cada período. Como requisito externo se debe fijar que el período de la *cruzroja.xml* será el mismo que el período de *tiempo.xml*. Además, el instante de llegada del objeto habrá que fijarlo para que coincida dentro del *slot* de tiempo para el evento *Levantarse.*

La siguiente etiqueta que nos encontramos es *envoltorio-objetos-cruz-roja*, a continuación se muestra un ejemplo de su uso:

<envoltorio-objetos-cruz-roja nombre="Paquete" url-imagen="/imagenes/cajaCruzRoja.png" />

El atributo *nombre* indica el nombre que se le dará en la aplicación al envoltorio. Los objetos de la Cruz Roja se entregan envueltos en un envoltorio. La imagen de este envoltorio se obtiene del atributo *url-imagen*. En la **Figura 3.51** se muestra la habitación del Botiquín con la caja de la Cruz Roja en el centro de la misma.

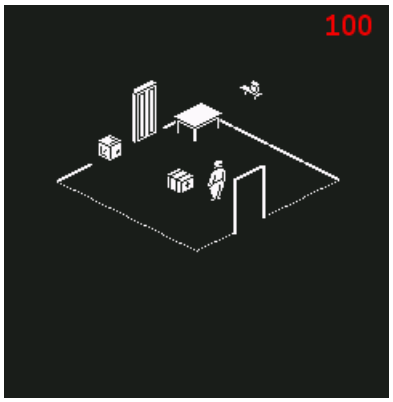


Figura 3.51 Imagen del envoltorio a los objetos de la Cruz Roja en la habitación del Botiquín.

En el cuerpo de la etiqueta *objetos-cruz-roja* se introducen las etiquetas de los objetos que se irán entregando en la aplicación. Nótese que se pueden entregar todos los objetos que han sido definidos en la aplicación, incorporando la etiqueta del objeto en los términos que se vieron para los objetos especiales en la explicación del fichero *mapadibujoexterior.xml*. Véase el ejemplo propio de la aplicación que se muestra a continuación:

<objetos-cruz-roja>
<bolsa nombre="Bolsa" fO="3" cO="3" url-imagen="/imagenes/bolsa.png" />
<tenazas nombre="Tenazas" fO="3" cO="3" url-imagen="/imagenes/tenazas.png" />
<brujula nombre="Brujula" fO="3" cO="3" url-imagen="/imagenes/brujula.png" />

</objetos-cruz-roja>

El cargador de este fichero pasará al controlador de la Cruz Roja la información de los objetos a entregar, que junto con el *Timer* de la Cruz Roja que se configure también a través del fichero, se realizará la entrega correctamente de éstos, ajustándose a lo indicado al respecto en el **Capítulo 2** de descripción de la aplicación.

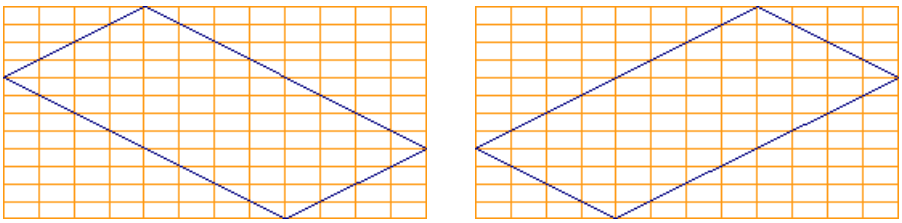
- *ficheroshabitacionesinteriores.xml*:

En la aplicación existen veinticinco ficheros de configuración de este tipo, uno por cada habitación existente. La marca raíz para estos ficheros XML es *habitacion*. Aparte del atributo ya comentado referente al espacio de nombres que acompaña a toda marca raíz de la aplicación, se tienen en este caso otros atributos adicionales: *nombre*, indica el nombre de la habitación por el que se le conocerá dentro de la aplicación, *scroll*, sirve para indicar si la habitación tendrá activada la opción de scroll con arrastre de pantalla, caso de poner *scroll="on"* o de no aparecer este atributo, o para indicar si la habitación tendrá la opción de scroll desactivada, caso de poner *scroll="off"*, útil en las habitaciones pequeñas que caben completamente por pantalla; de forma que se reducen las condiciones a comprobar de antemano en el motor de movimientos para personaje principal en las habitaciones, y *habitacion-permitida*, indica, si aparece, con el valor a *si* que se trata de una habitación permitida para el personaje principal, en el juego sólo existen dos de estas habitaciones que son las que conforman el barracón del personaje principal, si no aparece o se le da el valor de *no* o cualquier valor diferente de *si*, se toma por habitación no permitida, poniéndose a rojo el indicador de moral cuando el personaje principal entre dentro.

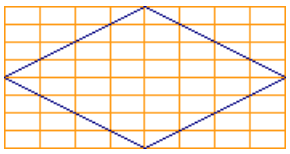
A continuación viene la etiqueta de *dim-mapa*. Esta etiqueta tiene el mismo comportamiento que las ya comentadas con idéntico nombre para los ficheros *mapafronteraexterior.xml*, *mapadibujoexterior.xml* y *mapapuertasexterior.xml*, sólo que en este caso con una sola *dim-mapa* le definimos las dimensiones de las tres matrices para la aplicación.

En la aplicación existen dos tipos de habitaciones utilizadas: uno es el que constituye las habitaciones pequeñas y el otro es el que da lugar a las habitaciones grandes. Pues bien, para las habitaciones pequeñas se tienen unas dimensiones: *<dim-mapa num-filas="8" num-columnas="8" />*, es decir, una matriz de ocho filas por ocho columnas para construir la habitación. Por el otro lado, para las habitaciones grandes se tiene: *<dim-mapa num-filas="12" num-columnas="12" />*, es decir, una planta de matriz sobre la que trabajar de doce filas por doce columnas. Abstrayendo un poco un esquema básico para los dos tipos de habitaciones, en la **Figura 3.52** se muestran los *esqueletos* en cuanto a contornos de frontera para las

habitaciones que se dieron en construir para la aplicación en base a este fichero de configuración.



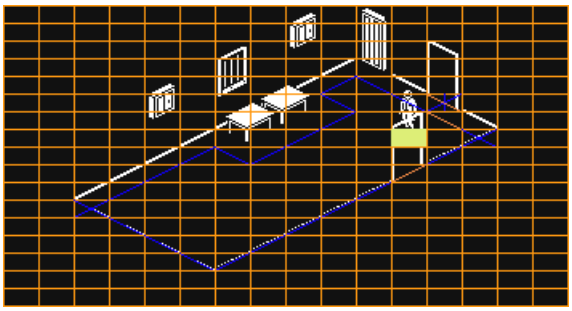
a) Imágenes de los *esqueletos* para las habitaciones grandes del juego.



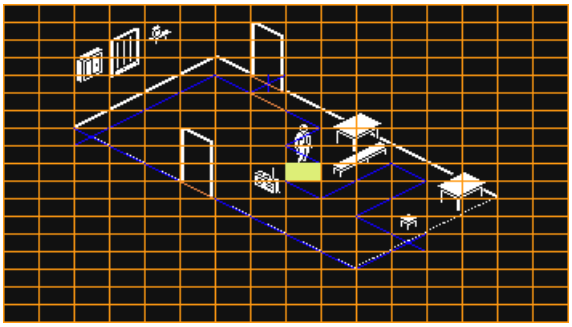
b) Esqueleto de las habitaciones pequeñas.

Figura 3.52 Estructuras de las habitaciones creadas para el juego.

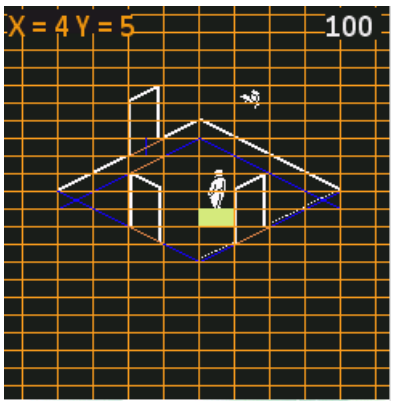
En la siguiente secuencia de imágenes (**Figura 3.53**) se pueden observar algunos ejemplos de estas estructuras de las habitaciones en la propia aplicación de videojuego a través de la visión de diseño. Las matrices de doce por doce y ocho por ocho se insertan en el centro de una matriz ampliada que es sobre la que desplaza la pantalla, idéntico principio al explicado en la **Figura 3.45**.



a) Imagen de habitación grande, con el largo en el sentido del eje X de la proyección isométrica.



b) Imagen de habitación grande, con el largo en el sentido del eje Y de la proyección isométrica.



c) Imagen de habitación pequeña.

Figura 3.53 Imágenes de ejemplo de las habitaciones del juego.

Estas habitaciones se construyen siguiendo el principio de etiquetas ya comentado para el patio exterior en sus ficheros de configuración, sólo que en el caso de las habitaciones interiores todo se comprende en un único fichero. Para ello, el documento XML se divide en varios bloques por las etiquetas: <*mapa-frontera*>, <*puertas*>, <*mapa-dibujo*> y <*objetos-especiales*>.

Abordando la primera a través de un ejemplo se tiene la estructura que se muestra a continuación:

```
<mapa-frontera>
    <diagonales-front>
        <diag-pos-inf fO="11" cO="8" fD="9" cD="10" esp="0"/>
        <diag-pos-sup fO="3" cO="1" fD="1" cD="3" esp="0"/>
        <!-- …-->
    </diagonales-front>
</mapa-frontera>
```

El bloque *mapa-frontera* define las fronteras de colisión de para el escenario. Para ello, en el bloque de la etiqueta *diagonales-front* se usan las mismas etiquetas de diagonales que en el fichero *mapafronteraexterior.xml*.

El siguiente bloque en el fichero es el que corresponde a *puertas*. Un ejemplo de su estructura se muestra a continuación:

```
<puertas>
```

<puerta id="0" enlace-con="puerta-pared1-4-6-habitacion12" url-xml="/habitaciones/habitacionesPuertaPared1_4_6/puertaPared1_4_6Habitacion12.xml" entrada="1" tipo="diag-pos-sup" inicio-activo="no" bidireccional="si" dintel-activo="si" fO="2" cO="1" fD="2" cD="1" />

<puerta id="1" enlace-con="puerta-pared1-4-6-habitacion12" url-xml="/habitaciones/habitacionesPuertaPared1_4_6/puertaPared1_4_6Habitacion12.xml" entrada="1" tipo="diag-pos-sup" inicio-activo="si" fO="3" cO="1" fD="3" cD="1" />

</puertas>

En su interior se definen las puertas de las habitaciones interiores a través de la etiqueta *puerta*, que resulta idéntica a la ya explicada en el fichero de *mapapuertasexterior.xml*. Nótese la única diferencia con respecto a las etiquetas de dicho fichero que es el atributo *dintel-activo*. Si se presenta con valor igual a *si*, la marca de puerta se convierte en la marca de dintel de puerta. Esta marca, como ya se comentó en el **Apartado 3.7 Dibujo de escenarios**, se introduce desde el fichero de configuración XML para corregir el efecto de desalineamiento que se produce cuando saliendo de una puerta que está en diagonal con la siguiente el personaje queda desalineado por la separación de media celda que se da en las paredes Norte y Oeste de las habitaciones entre las marcas de dibujo de contorno de la habitación y las marcas de colisión o frontera. (Véase la **Figura 3.28**)

Capítulo 4.
Diseño e implementación de la aplicación

4.1 Introducción

La aplicación J2ME de videojuego se tendrá que ejecutar en el entorno proporcionado por el dispositivo móvil. Como en la mayoría de las aplicaciones software, la interacción con el usuario se realizará a través del teclado, la pantalla y el altavoz del dispositivo.

El dispositivo responderá a las acciones realizadas por el jugador mediante el teclado, mostrando los resultados de dicha interacción y del desarrollo del juego a través de la pantalla y el altavoz. La **Figura 4.1** muestra el escenario de interacción del jugador con el dispositivo.

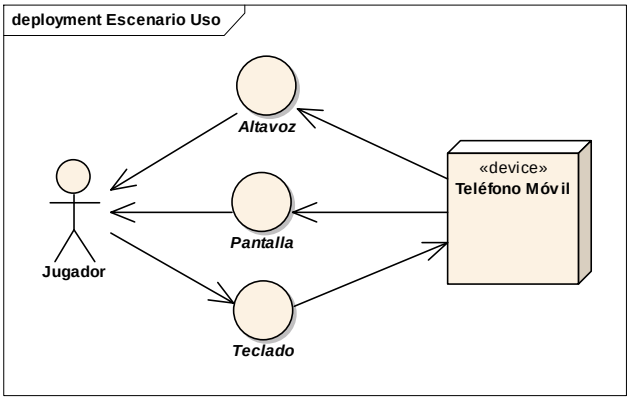


Figura 4.1 Escenario de interacción del jugado con el dispositivo.

La arquitectura esquemática del entorno de ejecución software que posee el dispositivo móvil para ejecutar la aplicación e interactuar con el jugador puede apreciarse en la **Figura 4.2**. En ella, se tiene la existencia de tres interfaces físicas de comunicación con el usuario, ya señaladas: teclado, pantalla y altavoz; el SO (Symbian, en nuestro caso) y la Máquina Virtual de Java (KVM), con diferentes hilos de aplicaciones que se ejecutan en ésta.

El SO controla el acceso a las interfaces físicas de comunicación con el usuario. Éste proporciona las rutinas y servicios necesarios a la Máquina Virtual de Java para que las aplicaciones Java puedan hacer uso, a su vez, del teclado, la pantalla y el altavoz.

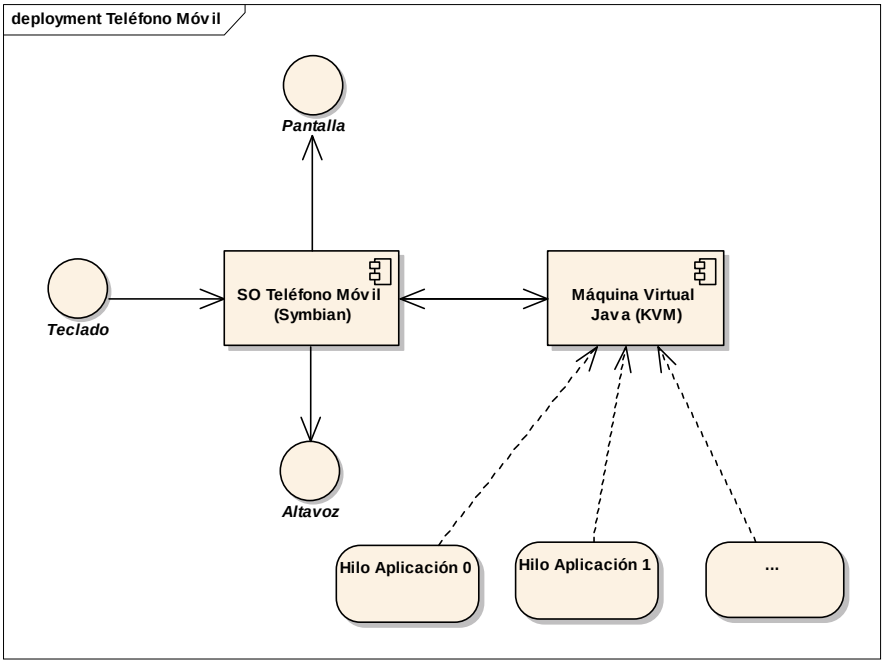


Figura 4.2 Esquema de la arquitectura software sobre el que se ejecuta la aplicación.

La máquina virtual de Java instancia un hilo de ejecución concurrente por cada aplicación J2ME lanzada. De esta forma, se pueden tener varias aplicaciones Java ejecutándose a la vez en el dispositivo. Además, dentro de cada hilo de ejecución de aplicación pueden existir a su vez otros sub-hilos propios de la aplicación que habrán sido contemplados por el programador. La máquina virtual irá alternando su ejecución en función de prioridades y eventos fijados por el programador, pero siempre estarán confinados al entorno propio de su hilo de aplicación.

Un aspecto a destacar de este proyecto es el empleo de las capacidades *multihilo* de Java, como ya se señaló en el capítulo anterior. La aplicación hace uso de los *threads* Java de manera importante para separar diferentes ámbitos de ejecución del videojuego, así como, para controlar el desarrollo temporal de la aventura gráfica. En la **Figura 4.3** se puede observar esta estructura.

El empleo de múltiples *threads* obliga a un mayor refinamiento de la arquitectura de la aplicación, incorporándose conceptos de la programación concurrente desde el enfoque Java. Java simplifica el modelo de programación concurrente para dotar al lenguaje de sencillez y robustez.

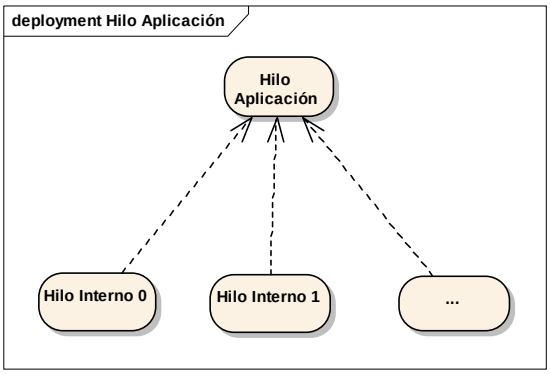


Figura 4.3 Hilos internos al Hilo de aplicación.

Un concepto importante de la programación concurrente para el manejo en Java de los *threads* son los monitores. La sincronización de diferentes bloques de código y métodos de las clases Java se consigue en base a la idea de monitores, básicamente. En este proyecto se ha intentado hacer el uso más claro y sencillo de esta idea. Más adelante, en el desarrollo de este capítulo, se profundizará más en el tema.

Como ya se ha mencionado en la introducción al proyecto, la tecnología J2ME sigue un modelo de capas para conseguir un alto grado de portabilidad de las aplicaciones. En la base está el SO del dispositivo móvil (Symbian, en este proyecto), por encima se ubica la máquina virtual de Java (KVM). Y por último, antes de hallarnos la aplicación, un nivel de API´s (*Aplication Programming Interface´s*) que proporcionan un conjunto de componentes y utilidades para la programación -estos dos niveles últimos señalados aíslan a la aplicación del SO que pueda estar debajo, con la consiguiente ventaja ya señalada de la portabilidad. Esta estructura puede verse en la **Figura 4.4**.

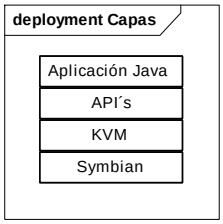


Figura 4.4 Estructura en capa de la tecnología J2ME.

En esta introducción al capítulo se ha mostrado, a grandes rasgos, el marco software en el que la aplicación a desarrollar se moverá. A continuación, se abordarán los aspectos ya particulares del diseño y la implementación de la aplicación.

4.2 Aproximación desde las interfaces de usuario

La visualización por pantalla se hace en J2ME principalmente mediante los objetos *Canvas*. En concreto, para los videojuegos existe una subclase de *Canvas*: *GameCanvas* (en el paquete j*avax.microedition.lcdui.game*), que aporta ventajas adicionales respecto a la clase padre como pueden ser el doble búfer, el refresco de la pantalla, controles adicionales sobre los botones del teclado, etc. Esta misma clase también se encarga del control de las teclas pulsadas por teclado.

Un segundo objeto de utilidad para visualizar por pantalla y controlar el teclado es la instancia de la clase *List* (del paquete *javax.microedition.lcdui*). Esta clase despliega por pantalla una lista de opciones que se pueden seleccionar desde teclado, permitiendo fácilmente la creación de árboles de menús.

También se utiliza en este proyecto una clase hermana de la anterior: la clase *Form* (del mismo paquete que *List*). Esta clase se empleará en el modo de lectura de texto sólo; si bien, permite mayores posibilidades de uso como formulario. Ambas clases, *List* y *Form*, son herederas de la superclase abstracta *Screen*, clase padre de todas las clases de interfaz de usuario de alto nivel para J2ME: *Alert*, *Form*, *List* y *TextBox*.

Para el control del sonido está la interfaz *Player* (del paquete *javax.microedition.media*). A través de la creación de objetos *Player,* mediante alguno de los métodos estáticos *createPlayer* de la clase *Manager* (del mismo paquete), tenemos acceso al control de los sonidos que se van a reproducir en el altavoz del dispositivo.

En la **Figura 4.5** se muestra un esquema básico de una aplicación en base a las interfaces de comunicación con el jugador.

Existe también un tercer elemento que ha sido utilizado en este proyecto, y que permite el control del teclado: los objetos *Command*. Estos objetos permiten implementar funciones especiales desde teclado. Todas las clases que escriban por pantalla tendrán la posibilidad de incorporar estos objetos, así como otro objeto que implemente la interfaz *CommandListener* para detectar cuándo son pulsados los comandos y desarrollar las acciones correspondientes.

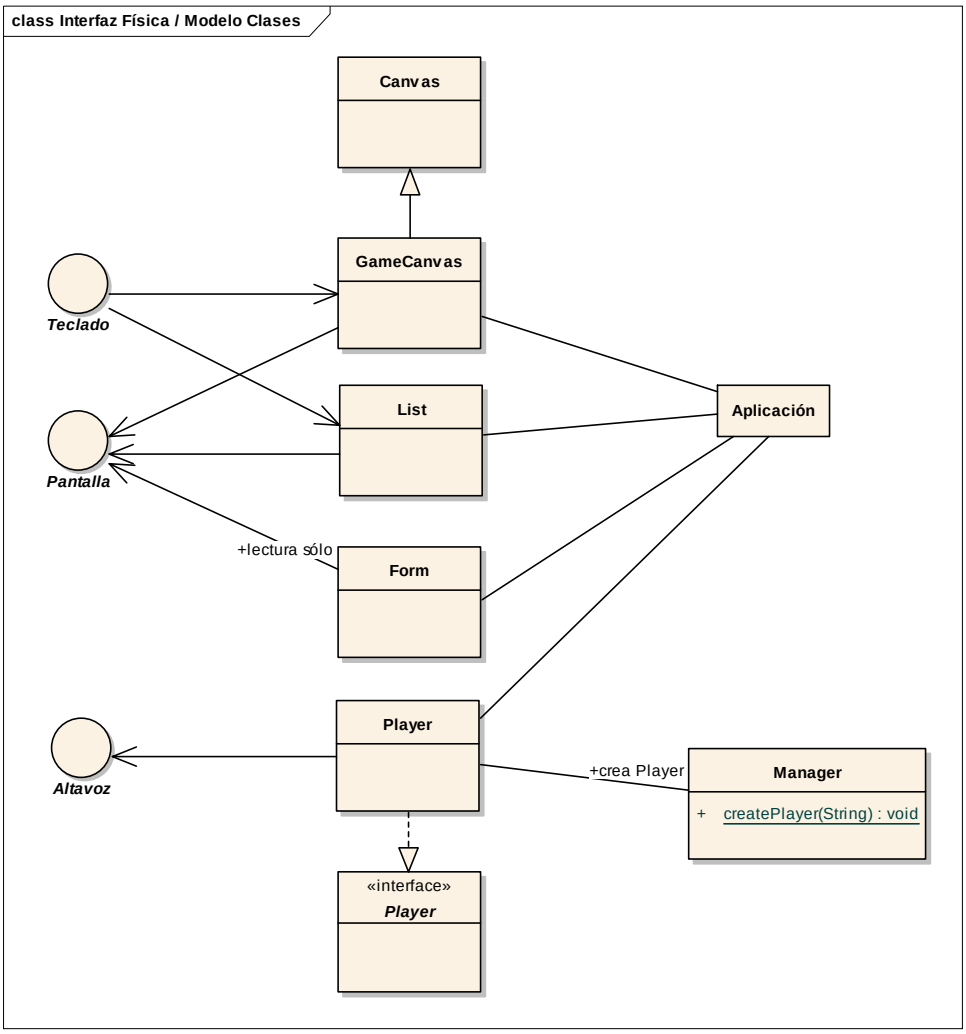


Figura 4.5 Esquema de clases e interfaces J2ME para la comunicación con usuario.

Todas las clases que trabajan con la pantalla y, por ende, con el teclado son herederas de la superclase abstracta *Displayable*. En esta clase se tienen los métodos públicos *addCommand* y *setCommandListener*. Con *addCommand* se indica los comandos que van a aparecer por pantalla cuando ese *Displayable* se muestre, mientras que *setCommandListener* fija para esa pantalla concreta el *listener* que interpretará los botones de comandos pulsados.

En la **Figura 4.6** se puede ver un esquema resumen de la arquitectura de herencia de clases para control de pantalla en J2ME.

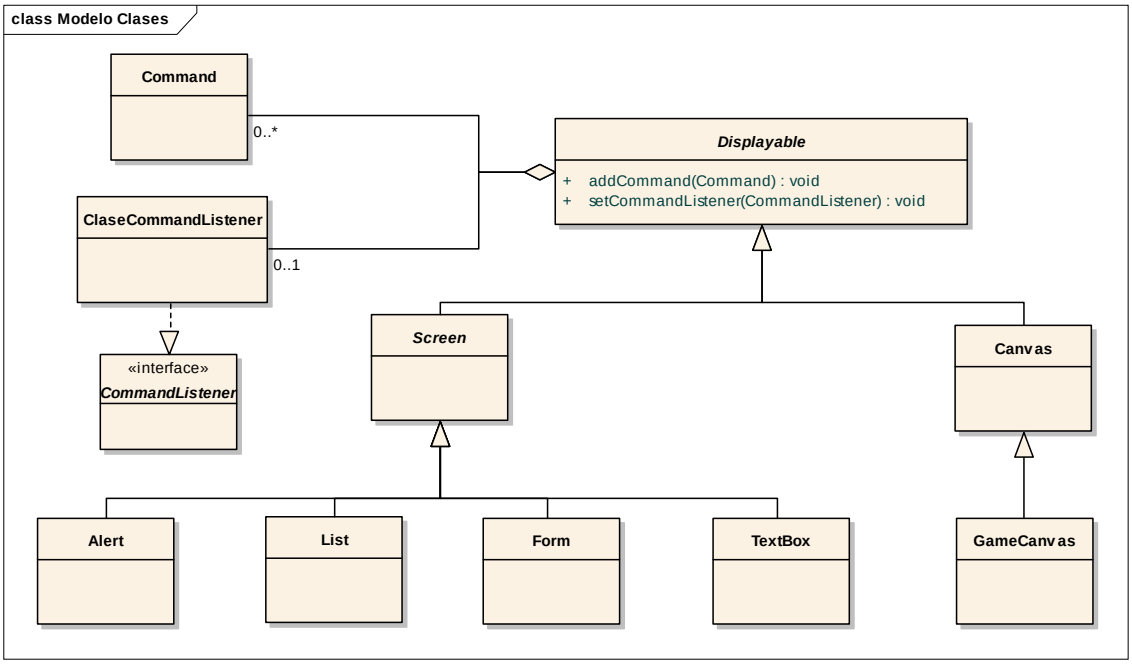


Figura 4.6 Diagrama de relación clases para control pantalla y el teclado.

Un enfoque muy utilizado es declarar que la propia clase *Displayable*, que controla lo mostrado por pantalla, implemente la interfaz de *CommandListener* para los comandos que se le hayan añadido. En la **Figura 4.7** se muestra un ejemplo de esto último sobre la clase *GameCanvas*.

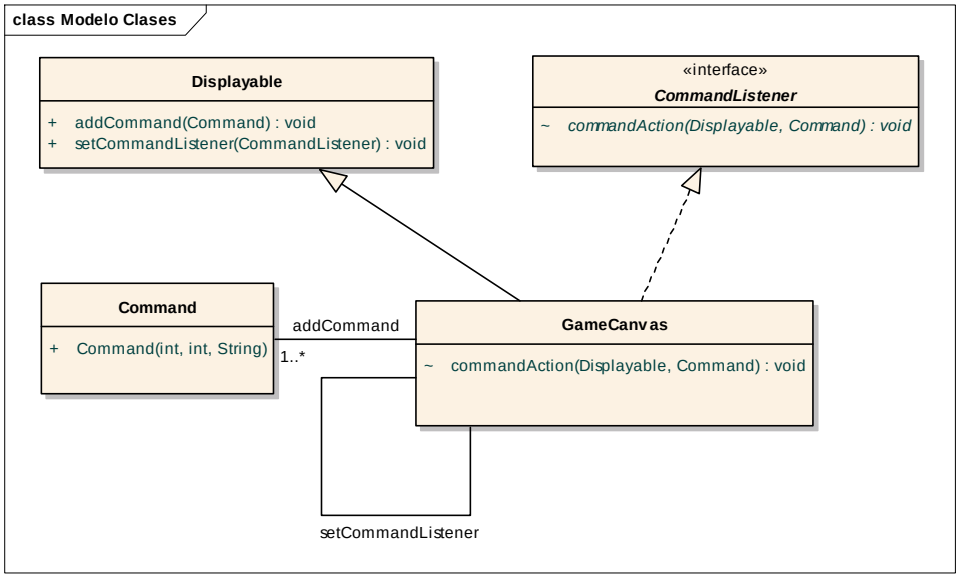


Figura 4.7 Clase *Diplayable* con la implementación propia de la interfaz CommandListener.

En la aplicación se empleó este tipo de arquitectura de control para manejar los comandos que se despliegan cuando se pulsa la tecla de Opciones para el caso del motor gráfico que controla el dibujo del escenario principal.

Estos objetos de control sobre las interfaces físicas de comunicación deberán ser manejados por el resto de la aplicación para mostrar el desarrollo normal de una aplicación de videojuego.

En el proyecto los motores gráficos: *MotorGraficoExteriorCanvasThread* (encargado de representar el escenario principal del patio exterior), *MotorGraficoInteriorCanvasThread* (encargado de representar los escenarios de las habitaciones interiores) y *TunelCanvasThread* (encargado de representar los túneles) son subclases de *GameCanvas*, gracias a ello se consigue capturar los datos de movimientos introducidos desde teclado y se representan por pantalla los dibujos de los escenarios actualizados.

Por otro lado se tiene las instancias de clases *List* que se utilizan para generar los menús de la aplicación, conjuntamente con las clases *Form*. El primer tipo de clases, *List*, se integra en el patrón de menús de *Ben Hui* [25] que se comentará más adelante en este capítulo; mientras que la clase *Form* se empleará para construir un formulario de sólo lectura que genere la pantalla de las instrucciones, como nodo final del árbol de menús generado con el patrón de *Ben Hui*.

La instanciación de objetos *Player's* en la aplicación se hace corresponder con cada sonido que cabe la posibilidad de que sea reproducido. En concreto, se tendrán tres *players*: el que reproduce la música -en formato *midi*- para cuando se consigue una fuga exitosa, el que reproduce el sonido de la alarma cuando el personaje principal es descubierto realizando algo indebido y el que reproduce el sonido de la campana cuando se hace llamamiento a algún evento temporal –estos dos últimos en formato *mp3*.

Los sonidos estarán contenidos en el *jar* de la aplicación, existiendo un fichero en el formato correspondiente por cada uno de ellos. Para controlar la reproducción de los sonidos se envuelven las instancias de las clases *Player's* con una clase de control: *ControladorSonido*, más adelante en este capítulo se aborda dicho controlador.

4.3 Estructura de parseadores

Existe un *parseador* XML por cada tipo diferente de fichero de configuración presente en la aplicación. La aplicación, como ya se mencionó en el apartado dedicado a XML del capítulo anterior, emplea la estructura de *parseador* ASXMLP (parseador de Al Sutton). Para su empleo basta con incorporar el *jar* con los fuentes binarios del *parseador* a la librería de la aplicación y crear una clase que implemente la interfaz *XMLEventListener* del paquete *com.alsutton.xmlparser*. En esta aplicación, además, se creó una superclase (*XMLListener*, en el paquete *parseadorXML*) del que deben heredar todas las clases que implementen la interfaz anterior, a fin de incorporar métodos generales que pueden ser de utilidad a la mayoría de los *parseadores* que se creen.

Las clases que implementan la interfaz anterior aparecen nombradas en el proyecto con el nombre *XMLListener<entidad parseada>* (con *<entidad parseada>* se indica el ámbito de actuación para el *parseador*, a fin y al cabo, el tipo de fichero XML que "*parsea"*). La **Figura 4.8** muestra un ejemplo completo de la estructura señalada para el *parseador* encargado de cargar las habitaciones interiores cada vez que éstas se lanzan.

La lista de *parseadores* es la siguiente:

- En el paquete *motorGraficoExterior.parseadoresXML* se tienen los pars*eadores* para los ficheros *mapafronteraexterior.xml*, *mapadibujoexterior.xml* y *mapapuertasexterior.xml*, que respectivamente son *XMLListenerFrontera*, *XMLListenerDibujo* y *XMLListenerPuertas*.
- En el paquete *motorGraficoInterior.parseadoresXML* está el *parseador* encargado de cargar los ficheros de las habitaciones interiores: *XMLListenerHabitacion*.
- En el paquete *cruzRoja.parseadorXML* se encuentra el correspondiente *parseador XMLListenerCruzRoja* para el fichero *cruzroja.xml*. Recuérdese que este *parseador* no sólo se lanza al comienzo del juego, sino también cada vez que el personaje principal es arrestado, y, tras ser llevado a la celda de castigo, se le indica al personaje que salga al patio exterior tras el amanecer de un nuevo día.
- En el paquete *tiempo.parseadorXML* se tiene al *parseador XMLListenerTiempo* encargado de parsear el fichero *tiempo.xml*. A este *parseador* le ocurre igual que al de la Cruz Roja, tras la estancia del personaje principal en la celda de castigo se vuelve a lanzar para resincronizar el tiempo.

- En *tuneles.parseadorXML* nos encontramos el correspondiente *parseador* para *tueneles.xml*: *XMLListenerTuneles*.
- En *zonasProhibidas.parseadorXML* el *parseador XMLListenerZonaProhibida* se encarga del fichero *zonasprohibidasexterior.xml*.
- En *personajes.ParseadorXML*: *XMLListenerPersonajes* para *personajes.xml*.

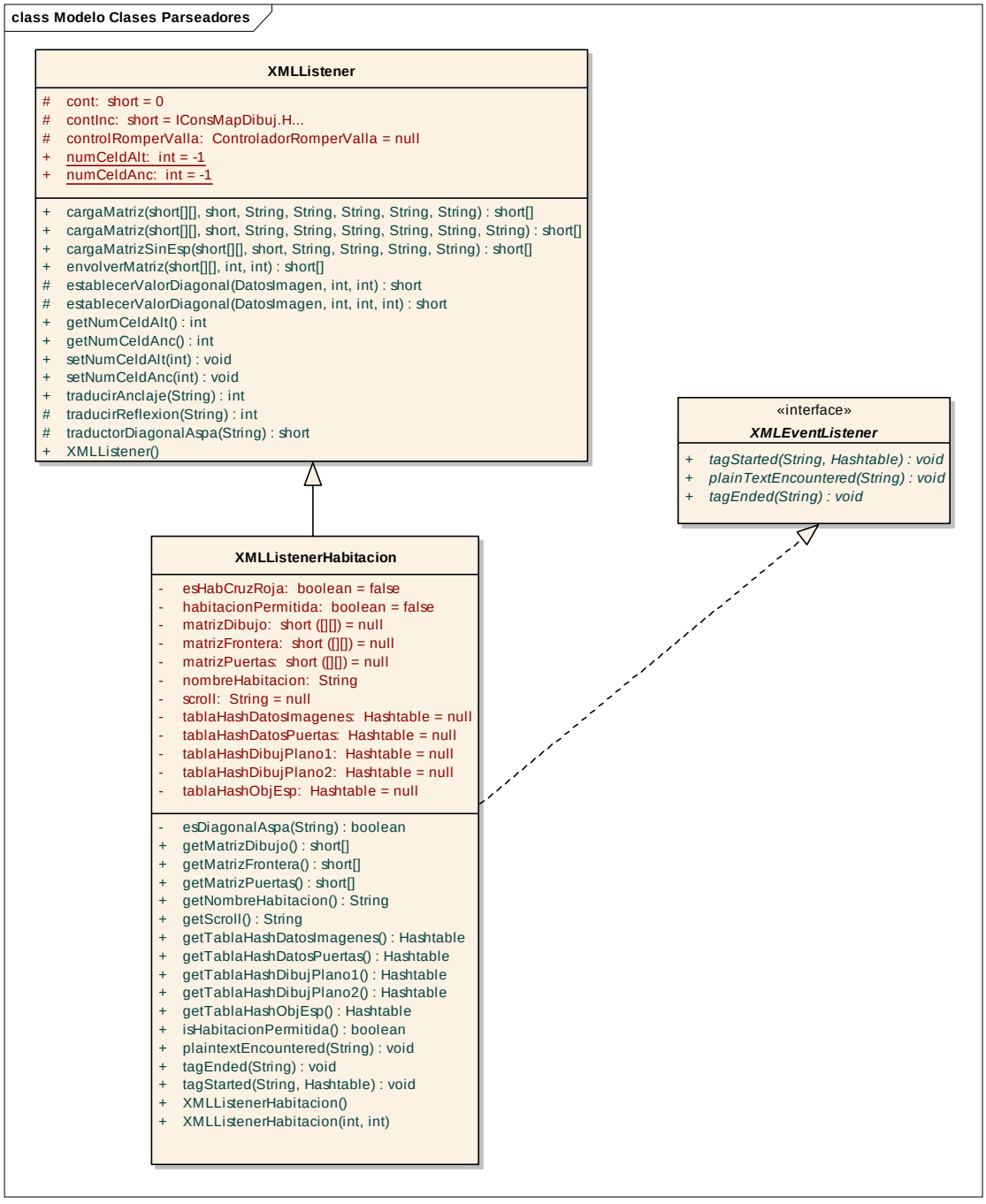
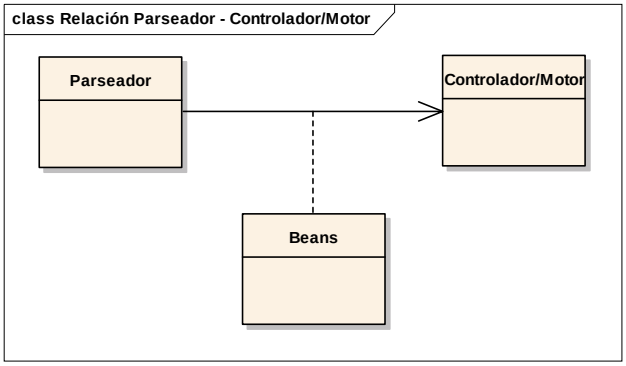


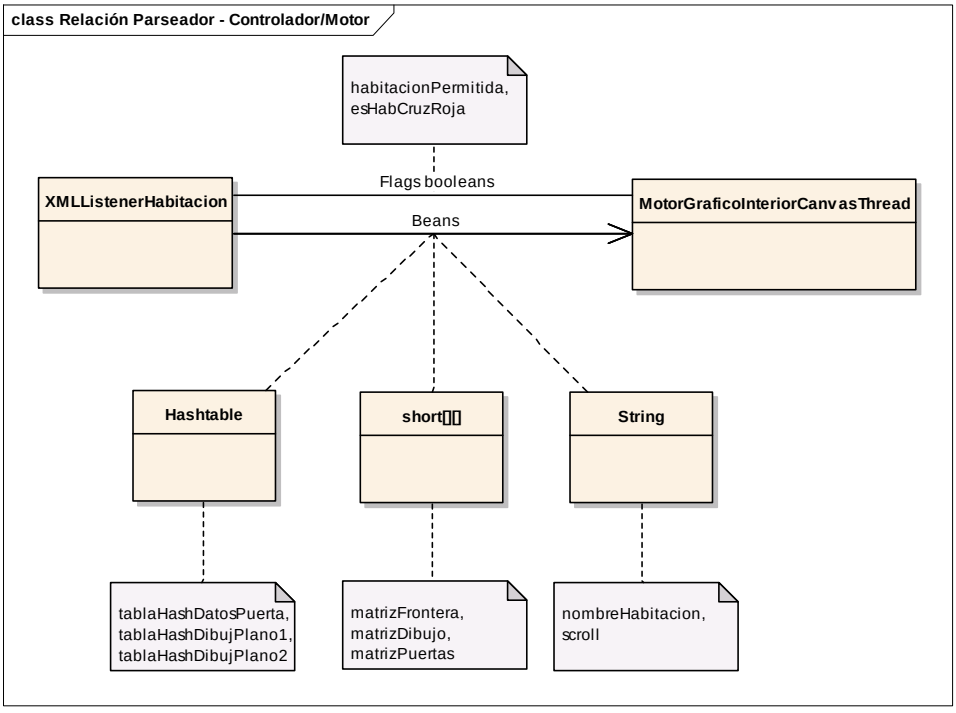
Figura 4.8 Ejemplo de estructura del *parseador* para las habitaciones interiores.

Los *parseadores* se encargarán de cargar en los diferentes motores y controladores de la aplicación las estructuras de datos en las que se mapea la información contenida en los diferentes ficheros XML. Estas estructuras de datos ya fueron referenciadas durante el capítulo

anterior como *Beans*. En la **Figura 4.9 a)** se puede observar la relación entre el *parseador* y el *controlador/motor* en términos generales, mientras que para la **Figura 4.9 b)** se muestra la misma relación sólo que concretando para el caso de *XMLListenerHabitacion* y *MotorGraficoInteriorCanvasThread* al que proporciona las estructuras de datos acerca de la habitación interior para operar.



a) Relación general entre *parseador* y Controlador/Motor.



b) Detalle de la relación entre *parseador* de habitaciones interiores y el motor gráfico para éstas

Figura 4.9 Esquemas de relación *parseador* y *Controlador/Motor*.

La estructura anterior de relación es general para todos los *parseadores*. Otro aspecto importante de la aplicación y relacionado con los *parseadores* lo constituyen los controladores y los motores de la aplicación.

4.4 Motores y controladores

La aplicación cuenta con dos motores gráficos *pesados* para los escenarios del patio exterior y de las habitaciones interiores (*MotorGraficoExteriorCanvasThread* y *MotorGraficoInteriorCanvasThread*), más una tercera clase Java, más liviana, encargada del control y la visualización de los túneles (*TunelCanvasThread*). Los motores gráficos propiamente dichos heredan a su vez de una tercera clase donde se *generalizan* las características comunes de ambos. Esta superclase es la de MotorGrafico. A continuación, en la **Figura 4.10** podemos observar un esquema de clases para los motores gráficos.

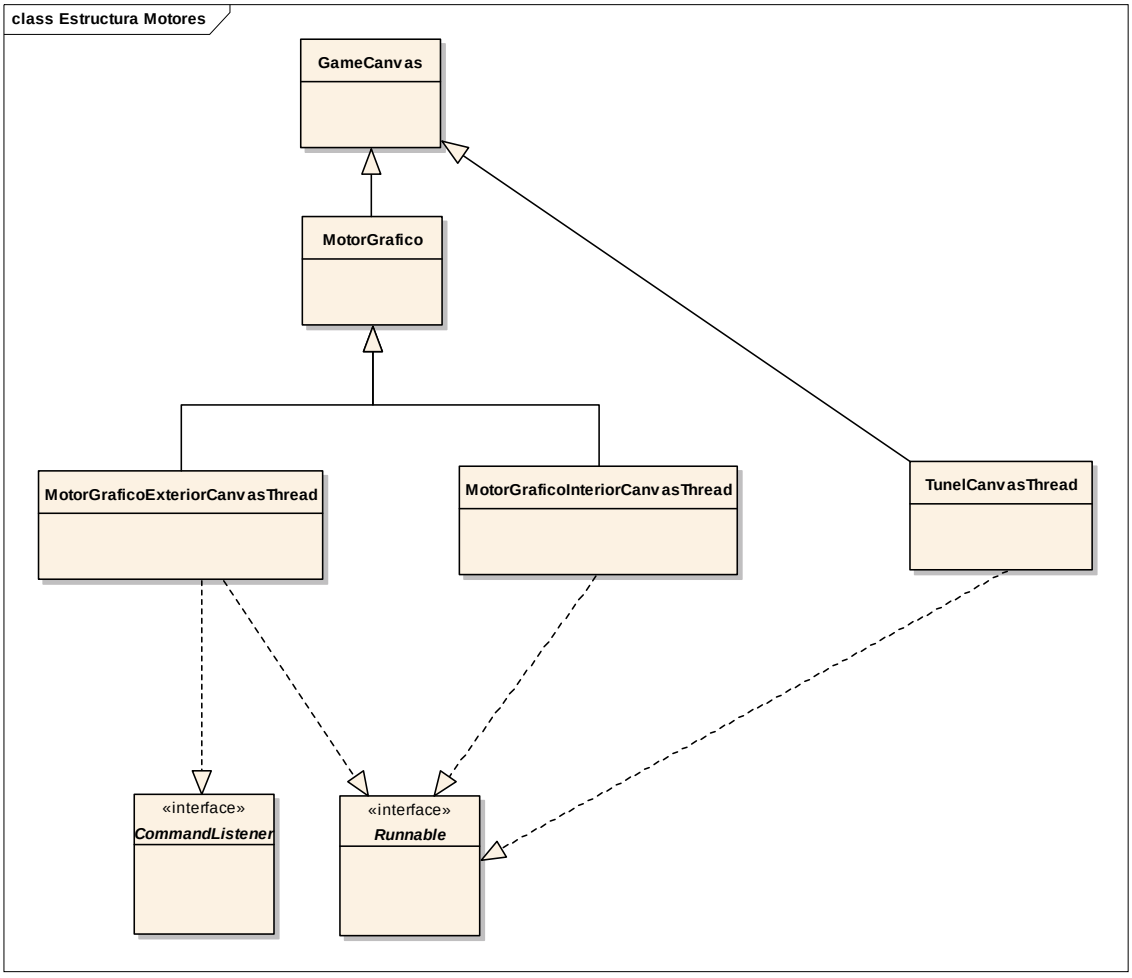


Figura 4.10 Esquema que muestra la estructura de los motores gráficos de la aplicación.

Todo el control de comandos en la aplicación asociado a los tres *GameCanvas* está centralizado en el método *CommandAction* que obliga a implementar la interfaz *CommandListener* en *MotorGraficoExteriorCanvasThread* -los dos *canvas* restantes lo único que hacen es añadir los comandos del motor gráfico exterior y establecer como *listener* a dicho motor gráfico exterior.

La clase *GameCanvas* proporciona a las clases que las hereden el control de lo representado en pantalla y de las teclas pulsadas a través, principalmente, de los métodos: *paint(Graphics g),* donde se codifica toda la representación gráfica para el canvas; *repaint()*, método para poder reinvocar a *paint* en sucesivas iteraciones del *game loop*; *keyPressed(int key)*, que indica la tecla pulsada; y *keyRelased(int key)*, que indica la tecla liberada.

En los métodos *paint* de cada uno de los tres motores gráficos se establece el control final de lo que se representará por pantalla. A este método le complementa en los motores interior y exterior un método *abstracto* heredado de *MotorGrafico*: *hacerDibujo*(Graphics g), donde se concentra la mayor parte del código de control para dibujar -la matriz de dibujo que genera el escenario en los planos uno y tres, invocación al método del controlador de los personajes para hacer su dibujo en el plano dos y la generación de la visión de diseño.

Además, complementando a los métedos *keyPressed* y *keyRelased*, en los motores gráficos exterior e interior, se tiene los métodos de control de la movilidad del personaje principal que derivan en la ubicación que éste ocupará en la matriz del juego para representar. Estos métodos provienen del método abstracto de MotorGrafico *movimientoPersonajePrincipal()*. En él, se establece el control de las colisiones para el personaje principal en los términos correspondientes de cada escenario, patio exterior y habitaciones interiores y el control de la toma de puertas y entradas a túneles con sus consiguientes efectos de carga de nuevos escenarios.

El mecanismo para lanzar los *threads* es a través de la interfaz *Runnable*. Una clase que implemente esta interfaz poseerá un método *run()* que se constituirá el cuerpo de ejecución del *thread*. Este método *run()* para los *CanvasThread* de la aplicación contendrá un bucle o *game loop*, que comprenderá todo el control en ejecución de los escenarios. La finalización de cada *thread* se realiza saliendo del bucle y dejando que termine el método *run.*

En cuanto a los *threads* de la aplicación se va a detallar el mecanismo de sincronización mediante monitores Java que se sigue. Cada uno de los tres motores posee una referencia a objeto de la clase *Monitor* (clase vacía definida en el proyecto para facilitar la sincronización, ya que los objetos que se instancien de ella serán los que se utilicen para suspender los hilos y despertarlos).

El esquema *multithreading* seguido es sencillo. El hilo principal es el *MotorGraficoExteriorCanvasThread* que está cargado en memoria todo el tiempo que se está

jugando. Este hilo se suspende para, bien lanzar un *MotorGraficoInteriorCanvasThread*, o bien, un *TunelCanvasThread* (estos dos hilos últimos se pueden conocer como secundarios). El paso de habitación en habitación, o de habitación a túnel y viceversa, se hace lanzando el hilo secundario correspondiente al nuevo escenario y dejando que el hilo del que se provenía termine. La vuelta desde un hilo secundario al principal conlleva despertar dicho hilo con la actualización de la posición del personaje principal y dejar que el hilo secundario termine.

El hilo principal instancia un objeto *Monitor* que se irá pasando de hilo secundario en hilo secundario mientras éstos se estén ejecutando, una vez que se vuelva al patio exterior el hilo secundario antes de morir despierta al hilo principal gracias al objeto *Monitor* que le pasó el hilo principal al primer hilo secundario.

A continuación se muestra el código para suspender el hilo principal en la aplicación, donde *mon* constituye la instancia de la clase vacía *Monitor*:

```
synchronized (mon) {
      // Código previo a la suspensión del hilo
      ...
      mon.wait();
      // Código que sigue a sacar de la suspensión el hilo
      ...
}
```

Este bloque se encuentra dentro del bucle *game loop* de run, alcanzándose una vez que el personaje toma una puerta o entrada a túnel, a través de la activación de un *flag*.

Por otro lado, la estructura de control de monitores que poseen los hilos secundarios consiste en la contrapartida de despertar el hilo principal:

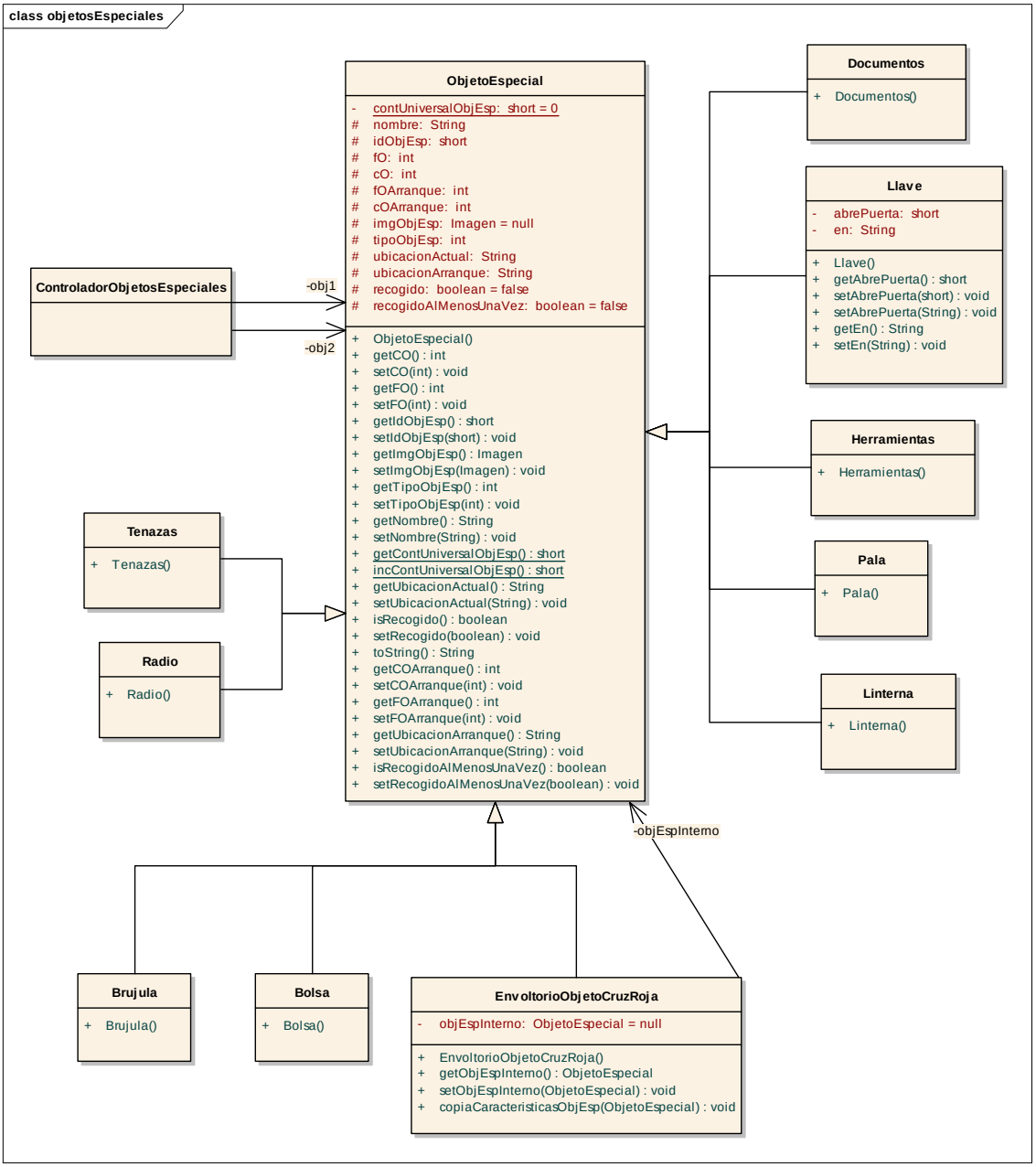
```
synchronized (mon) {

      mon.notifyAll();

}
```

Este bloque de código se encuentra al final del método *run*. El utilizar bloques de código sincronizados –estos bloques se ejecutan de forma atómica- es una cuestión estilística de programación, ya que los hilos secundarios despiertan al principal al finalizar *run*; si bien, en las restantes veces que se utiliza la sincronización entre hilos no es una mera cuestión estilística – este aspecto se verá más adelante tras abordar los controladores de la aplicación.

Los controladores se encargan de los aspectos especiales del juego. La lista de controladores existentes en la aplicación comprende:

- *ControladorCitas*: encargado de comprobar que el personaje principal acude a las citas (*Formar*, *Comer* y *Ejercicio*) donde se hace recuento de personajes, de modo que si faltase a dos o más se activa un *flag* para que los guardias del campo lo detengan si le ven.
- *ControladorComer*: encargado de introducir, sentar a la mesa y extraer a los personajes secundarios del comedor. Además controla el sitio en que se sienta y levanta el personaje principal.
- *ControladorCruzRoja*: encargado de gestionar la entrega de los objetos de la Cruz Roja en el Botiquín. Colabora con la clase *TimerCruzRoja* para sincronizar las entregas en el tiempo al comienzo de cada día nuevo.
- *ControladorDormir*: encargado de introducir a los personajes secundarios en las camas de sus dormitorios en los barracones y de sacarlos al patio exterior cuando se dé el evento *Levantarse*.
- *ControladorGanarPartida*: controla si el personaje principal toma por una de las tres únicas salidas entre el cerco de piedras, de modo que si lleva los objetos necesarios deriva la acción a una pantalla de enhorabuena por haber conseguido la evasión, y si no, lo devuelve al campo encerrándolo en la celda hasta que se dé el amanecer de un nuevo día.
- *ControladorMoral*: lleva todo la comprobación de la moral, realizando los descuentos o incrementos en la moral según el personaje sea detenido con o sin objetos, o vaya recogiendo objetos y usándolos.
- *ControladorObjetosEspeciales*: se encarga del control referente a la recogida, suelta, uso y descubrimiento por parte de los guardias de los objetos que se hayan desperdigados por el campo. La **Figura 4.11 a)** muestra el diagrama de esta importante clase de la aplicación y la **Figura 4.11 b)** sus relaciones con otras clases importantes de su mismo paquete.

a) Diagrama de clase del ControladorObjetosEspeciales.



b) Relaciones del *ControladorObjetosEspeciales* y demás clases del paquete *objetosEspeciales*.

Figura 4.11 Diagramas de clases del *ControladorObjetosEspeciales* y clases colaboradoras.

- *ControladorPersonajes*: superclase que engloba todas las características de los controladores para los personajes del juego. Se constituye en superclase de los controladores específicos para los personajes autómatas de la aplicación. En la **Figura 4.12** se puede observar el diagrama de clases para éstos.

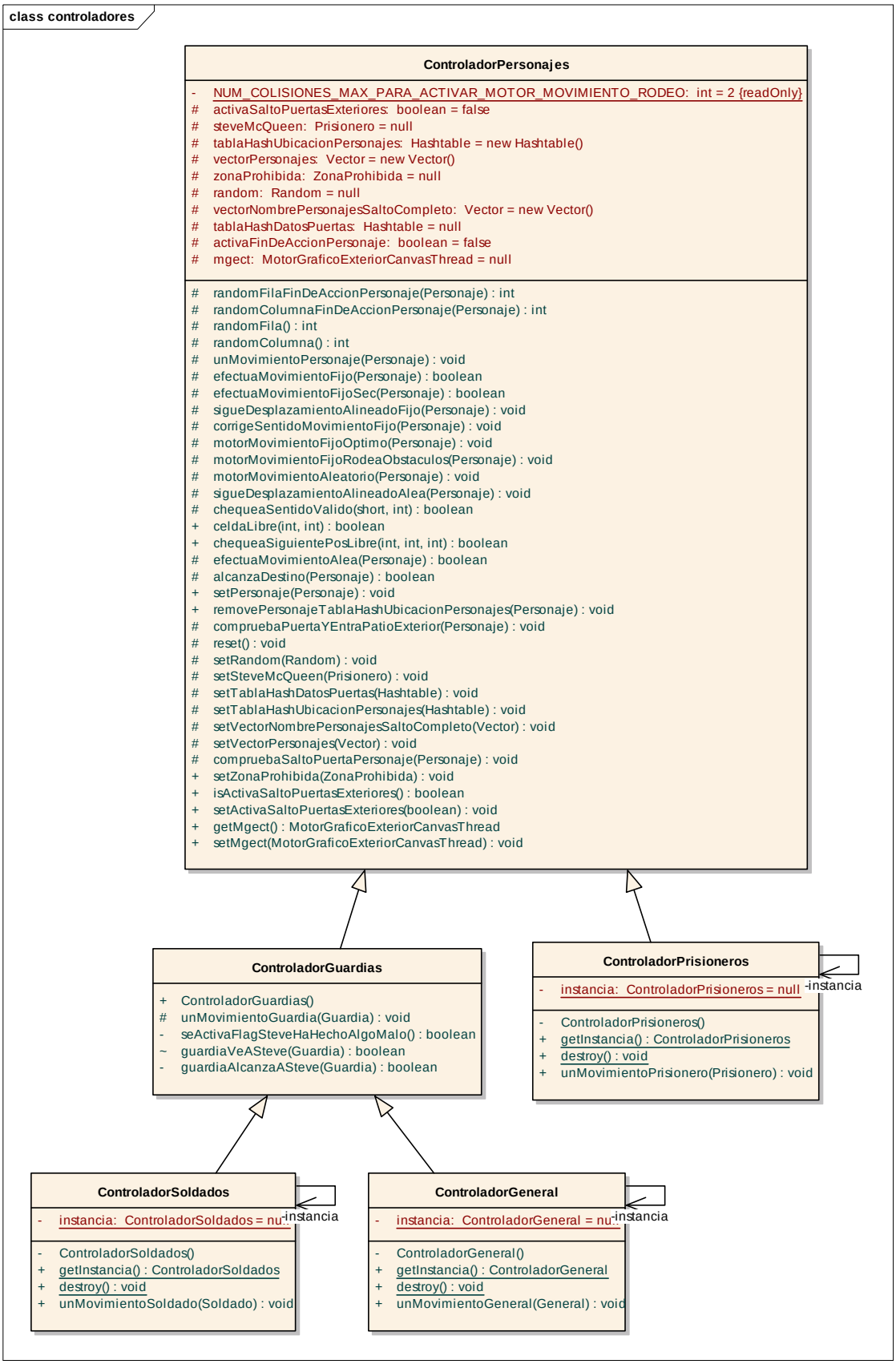


Figura 4.12 Esquema de clases para los controladores de los personajes.

En la **Figura 4.12** se aprecian la jerarquía de clases para los controladores, siendo los controladores finales *ControladorPrisioneros*, *ControladorSoldados* y *ControladorGeneral*. Cada uno se encarga de los aspectos de la movilidad específicos de cada tipo de personaje.

- *MetaControlPerson*: se trata de un meta-controlador. En él, se aglutinan los tres controladores para personajes, dándose un portal de acceso común mediante esta clase.
- *ControladorRomperValla*: se encarga de generar las puertas de rotura de la valla (diagonales frontera marcadas como rompibles en el XML mapafronteraexterior.xml), de introducirlas en la *matriz de puertas* para su posible uso por todos los personajes del juego y de eliminarlas de dicha matriz una vez se haga efectivo el reset del campo cada día nuevo que amanece.
- *ControladorSonido*: se encarga de controlar la activación y desactivación de los *players* del juego. Éstos son tres el sonido de campana, alarma y la música de *La gran evasión* para cuando consigues la fuga.
- *Tiempo*: se encarga de ir entregando los sucesivos eventos que acontecen en el campo y que rigen el comportamiento de los personajes.
- *ZonaProhibida*: se encarga del chequeo de si el personaje principal está en una zona prohibida o no, teniendo en cuenta el evento tiempo en curso.
- *ControladorTuneles*: se encarga de controlar el acceso a los túneles, llevando el control de si cada entrada está abierta o cerrada para permitir la entrada o no al túnel desde habitación interior o desde el patio exterior.
- *ControladorVibracion*: se encarga de activar y desactivar la vibración para el juego. Existen sólo dos tipos de vibración: una cuando se da aviso de un evento nuevo y la segunda cuando un guardia va a detener al personaje principal.

Todos los controladores de la aplicación implementan un patrón software *singleton*. Este patrón está especialmente indicado para clases Java como los controladores, de las cuales sólo existirá una instanciación (por cada una) en la aplicación. Por otro lado, se trata un patrón que simplifica enormemente el código, ya que no hay que estar introduciendo referencias a los cargadores en todas las clases donde sean utilizados, bastando con obtener la instancia del controlador sobre la que invocar el método que se desee. En la **Figura 4.13** se puede ver un diagrama de clase donde se resume el patrón *singleton*.

En este patrón, la instancia de la clase se establece como un campo privado estático de la misma. Proporcionándose dos métodos por defecto estáticos que tienen acceso a éste por defecto: *getInstancia()*, que crea la instancia única de la clase en la aplicación, si no existe, y

devuelve su referencia; y *destroy()*, que establece a *null* la instancia privada estática para que el recogedor de basura de Java puede liberarla.

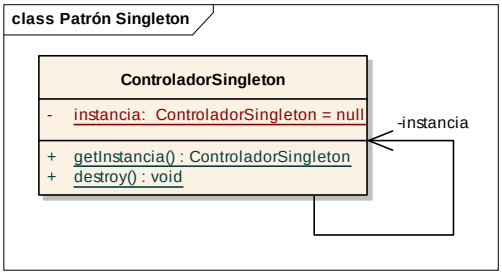


Figura 4.13 Esquema de ejemplo del patrón *singleton*.

La mayoría de los controladores poseerán un método de reset del controlador. Este método entronca con el reset del campo cada día nuevo que amanece, es decir, dedicado a las funciones correspondientes de reparar vallas, cerrar puertas abiertas, reposicionar objetos abandonados y visibles para la aplicación en sus ubicaciones originales. Se trata de un reset ligero, el controlador en ningún momento es destruido y vuelto a lanzar, salvo para los casos del *ControladorCruzRoja* y *Tiempo*,por razones de sincronía y no accesibilidad a los hilos de la clase *Timer* en segundo plano, cuando el personaje trasladado a la celda es liberado en el amanecer de un nuevo día.

Volviendo a retomar la sincronización de los tres motores debe destacarse el especial cuidado a tener a la hora de relanzar el juego por completo desde el menú de juego nuevo. Al emplear el patrón *singleton* en los controladores hay que asegurarse de que el anterior hilo principal de *MotorGraficoExteriorCanvasThread* ha terminado, destruyendo los antiguos controladores antes de cargar los controladores de inicio para el nuevo hilo principal. Esto se consigue forzando que tras empezar el método *run* se haga el chequeo de un *flag* estático antes de la carga de los controladores, que sólo dará paso, sin suspender el nuevo hilo, a menos que se haya ejecutado un bloque de código donde se destruyen los controladores al final del método *run* del antiguo hilo principal. En términos esquemáticos de código fuente Java sería:

```
run() {
      if (!MotorGraficoExteriorCanvasThread.controladoresReseteados) {

      // Suspendo el hilo principal nuevo hasta que el hilo principal
      // antiguo haya reseteado los controladores

      synchronized(MotorGraficoExteriorCanvasThread.monMGECTControlado
resReseteados) {
```

```
            MotorGraficoExteriorCanvasThread.monMGECTControladoresReseteados
                                                .wait();

                  }

            }
            // Es en el propio hilo principal donde se carga el juego

            this.cargaJuegoInterno();

            // Cuerpo central de run

            // game loop ...

            // Final de run

            // Destruyendo las instancias de los controladores
            ZonaProhibida.destroy();

            Tiempo.destroy();


            ControladorGeneral.destroy();
            ControladorSoldados.destroy();
            ControladorPrisioneros.destroy();
            ControladorMoral.destroy();
            ControladorObjetosEspeciales.destroy();
            ControladorTuneles.destroy();
            ControladorRomperValla.destroy();
            ControladorCruzRoja.destroy();
            ControladorDormir.destroy();
            ControladorComer.destroy();
            ControladorCitas.destroy();
            ControladorGanarPartida.destroy();
            ControladorSonido.destroy();


            TimerObjCruzRoja.destroy();

            MetaControlPerson.destroy();

            this.tablaHashDatosPuertas = null;

// Despertamos de la suspensión un posible hilo principal nuevo que
// tuvo que esperarse a que este hilo reseteara los controladores.

            MotorGraficoExteriorCanvasThread.controladoresReseteados = true;

            synchronized(MotorGraficoExteriorCanvasThread.monMGECTControlado
resReseteados) {

            MotorGraficoExteriorCanvasThread.monMGECTControladoresReseteados
                                            .notifyAll();
                        }

} // Fin de run
```

Hay que destacar que el *flag* (`controladoresReseteados`) se pone a *false* en el momento de seleccionar la opción en el menú de juego nuevo. En este esquema se sincronización de hilos se hace indispensable el uso de los monitores en el cuerpo de un bloque *synchronized*, para asegurar el correcto funcionamiento y sincronía entre hilos.

Relacionado con los dos motores gráficos externos que puede haber en la aplicación cabe destacarse la variante del patrón *singleton* empleada para obtener la instancia del objeto que propicia el hilo principal. Para ello, en la clase de arranque MIDlet de la aplicación: *PrisioneroMovil* poseerá la referencia del hilo principal en curso, pudiendo ser accesible para el resto de clases sin ambigüedad, ya que la posibilidad de tener dos instancias de *MotorGraficoExteriorCanvasThread* impide el empleo propiamente en ella del patrón. En la **Figura 4.14** se aprecia un esquema simplificado de clases donde se muestra esto último.

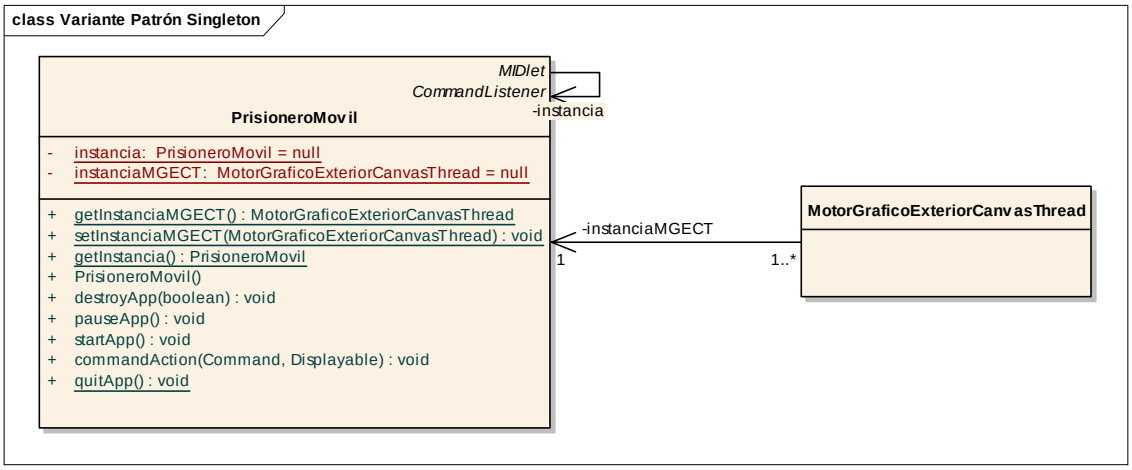


Figura 4.14 Estructura para proporcionar la referencia al objeto que propicia el hilo principal.

Al respecto del *game loop* hay que destacar que su estructura es bastante parecida para los tres motores gráficos. A continuación se muestra un esquema de los métodos principales que se ejecutan en él, extraído de *MotorGraficoExteriorCanvasThread* para los restantes resulta similar:

```
// Chequeo si hay que sacar personaje secundario del dormitorio.
     ControladorDormir.getInstancia().sacarPrisionerosDeLaCama();

// Chequeo si hay que sacar personaje secundario del comedor.

ControladorComer.getInstancia().sacarPrisionerosDelComedor();

// Realiza movimiento de todos los personajes secundarios,
// resolviendo colisiones, entradas en dormitorios y comedores,
// modos de rastreo de los guardias, en definitiva el motor de
// movimientos para los personajes secundarios.
```

```
this.metaControlPerson.unMovimiento();

// Chequeo del movimiento del personaje principal, resolviendo
// colisiones, toma de puertas y entradas a los túneles.

this.movimientoPersonajePrincipal();

// Refresco del dibujo de la pantalla del juego actualizado.

repaint();
```

4.5 Gestión de menús

Para la gestión de los menús se emplea el patrón de *Ben Hui* [25] con algunas modificaciones. Este patrón permite crear de forma controlada menús de opciones seleccionables que se despliegan en árbol, como los de la aplicación. En la **Figura 4.15** se muestra el esquema original de clases de este patrón.

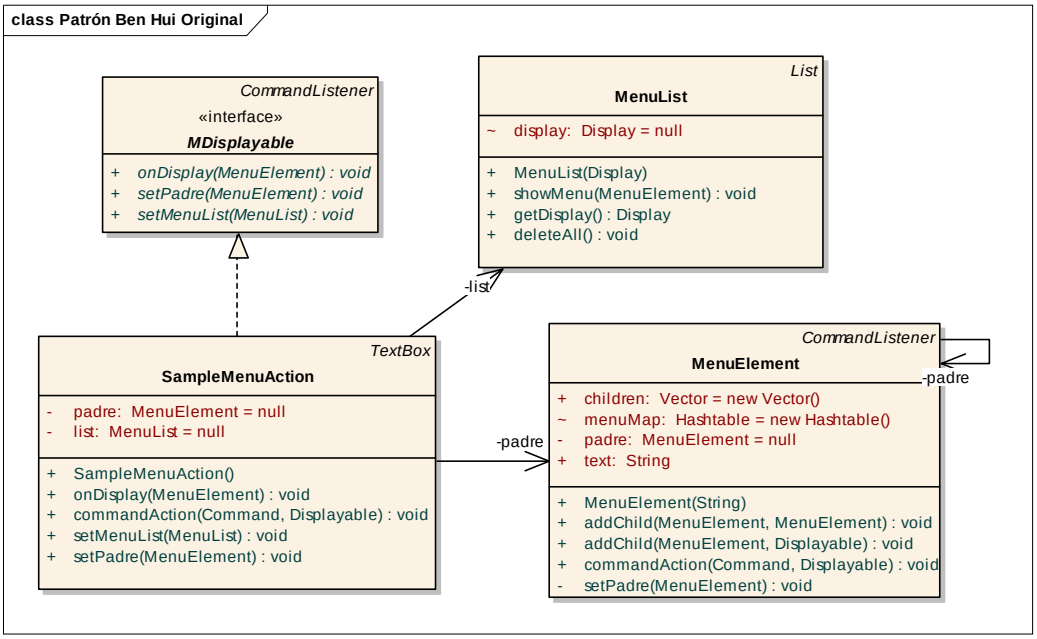


Figura 4.15 Esquema original del patrón de *Ben Hui* para menús.

En este patrón se tiene la clase MenuList encargada de mostrar los menús a través del método *showMenu*. Los menús se definen en la clase *MenuElement*, donde cada hijo que se le añada al nivel de menú puede ser bien otro menú desplegable con opciones igual que el que nos atañe, o directamente un objeto *Displayable* final. Las acciones finales u objetos *Displayable* implementan la interfaz MDisplayable para un correcto funcionamiento en la navegación. Este patrón fue modificado en la aplicación para permitir las funcionalidades múltiples adicionales

de navegación por los menús como el botón de atrás, los botones On/Off y el lanzamiento del juego. En la **Figura 4.16** se muestra la modificación introducida al patrón.

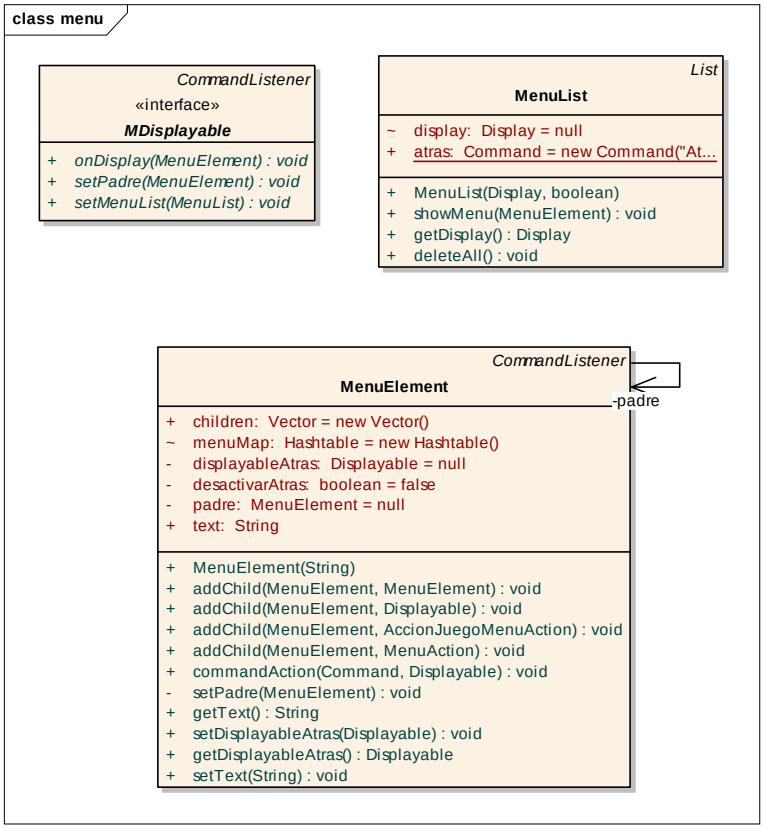


Figura 4.16 Patrón de *Ben Hui* modificado para la aplicación.

También hay que destacar las acciones (*actions*) finales que complementan al esquema de clases anterior. La **Figura 4.17** muestra estas acciones. En la figura se observan las clases *Action*:

- *ContinuarJuegoMenuAction* y *JuegoNuevoMenuAction* que implementan la interfaz *AccionJuegoMenuAction*. Las clases que implementen esta interfaz se añadirán como elementos finales a un *MenuElement*. En el método acción estará contenida la acción del juego programada que abandona los menús.
- *SonidoOnOffMenuAction* y *VibracionOnOffMenuAction* implementan la interfaz *MenuAction*. Esta interfaz está enfocada a permitir los efectos de actualización del menú como botón de encendido y apagado, continuándose en el ámbito del menú desplegado.
- *InstruccionesMenuAction* que implementa *MDisplayable* del patrón original y extiende a la clase *Form*. Se utiliza como formulario de solo lectura donde están las instrucciones del juego, con la vuelta al menú del que se desplegó a través del botón *Atrás*.

- *SalirMenuAction* formulario de salida (extiende la clase *Form*), el paso hacia atrás de vuelta al menú se hace mediante el botón de *No*, de ahí que no implemente la interfaz *MDisplayable*, ya que ésta introduciría el botón de *Atrás*.

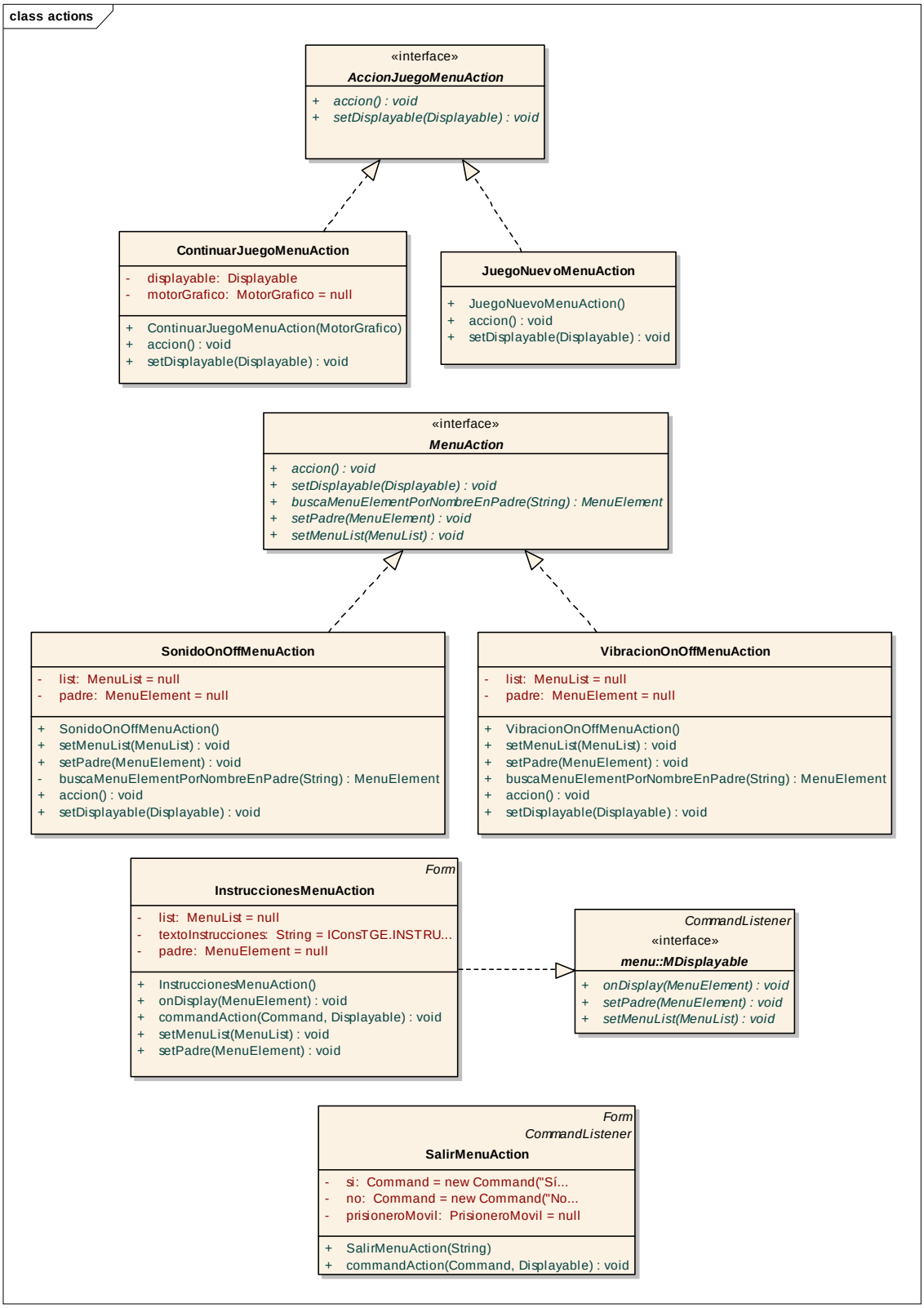


Figura 4.17 Paquete *pattern.menu.actions* las acciones finales para el patrón modificado.

4.6 Bluetooth

Bluetooth es una tecnología de radio de corto alcance, que permite conectividad inalámbrica entre dispositivos remotos. Se diseñó pensando básicamente en tres objetivos: pequeño tamaño, mínimo consumo y bajo precio.

Opera en la banda libre de radio ISM (banda internacional científico-médica) a 2.4 Ghz. Su máxima velocidad de transmisión de datos es de 1 Mbps. El rango de alcance *Bluetooth* depende de la potencia empleada en la transmisión. La mayor parte de los dispositivos que usan *Bluetooth* transmiten con una potencia nominal de salida de 0 *dBm*, lo que permite un alcance de unos 10 metros en un ambiente libre de obstáculos.

Mientras que el hardware *Bluetooth* avanzó muy rápidamente, hasta hace relativamente poco no había manera de desarrollar aplicaciones Java *Bluetooth* – hasta que apareció *JSR 82* [26], que estandarizó la forma de desarrollar aplicaciones *Bluetooth* usando Java. Ésta esconde la complejidad del protocolo *Bluetooth* detrás de unos *API's* que permiten centrarse en el desarrollo, en vez de los detalles de bajo nivel de *Bluetooth*.

El API intenta ofrecer las siguientes capacidades:

- Registro de servicios.
- Descubrimiento de dispositivos y servicios.
- Establecer conexiones *RFCOMM*, *L2CAP* y *OBEX* entre dispositivos.
- Usar dichas conexiones para mandar y recibir datos (las comunicaciones de voz no están soportadas).
- Manejar y controlar las conexiones de comunicación.
- Ofrecer seguridad a dichas actividades.

Los *API's* Java para *Bluetooth* definen dos paquetes que dependen del paquete *CLDC javax.microedition.io*:

- *javax.bluetooth*
- *javax.obex*

4.7 Diseño funcionalidad multijugador por bluetooth

Antes de afrontar este diseño, hay que destacar que la funcionalidad no llegó a implementarse en la aplicación debido al excesivo sobrecoste en tiempo que habría conllevado. No obstante, conforme al título del proyecto se hace el diseño de esta funcionalidad, dejando como posible línea futura su traslado a la aplicación J2ME.

El primer aspecto importante a la hora de diseñar la funcionalidad multijugador es fijar el protocolo de comunicación sobre el que se va a sustentar la estructura. El enfoque seguido será de una estructura de *cliente-servidor* sobre *Bluetooth* -limitando el número de clientes a uno sólo, es decir, el modo multijugador sólo tendría dos jugadores. Las tres opciones de protocolo de comunicación son: *RFCOMM*, *L2CAP* y *OBEX*.

RFCOMM es también conocido como perfil de puerto serie (SPP). Este protocolo emula múltiples conexiones entre puertos serie *RS-232*. El protocolo establece las sesiones de comunicación utilizando las direcciones *Bluetooth* de los dos puntos terminales. Una sesión puede tener más de una conexión -el número de conexiones depende de la implementación. Las sesiones, por otro lado, están ligadas a cada par único de direcciones *Bluetooth* de los dispositivos que se conectan. Además, un dispositivo podrá tener más de una sesión, entendiéndose por esto que podrá estar conectado a más de un dispositivo *Bluetooth*, es decir, que *RFCOMM* no queda limitado a una única sesión.

L2CAP (Logical Link Control and Adaptation Protocol) se trata de un protocolo que posee dos tipos de comunicaciones: una orientada a conexión (bidireccional), y otra no orientada a conexión (unidireccional). El API de JSR 82 no soporta para este protocolo el tipo de comunicación no orientada a conexión. *L2CAP* requiere de la configuración de una serie de parámetros fundamentales del canal de comunicación que se habrán de negociar entre los dispositivos *Bluetooth*. Estos parámetros son:

- La Unidad Máxima de Transferencia (*MTU*): valor del *payload* máximo que el dispositivo que envía la petición puede atender. Por defecto se tiene 672 *bytes*.
- Tiempo de descarte: cantidad de tiempo durante el cual el administrador del canal intenta transmitir satisfactoriamente el paquete antes de descartarlo. Puede establecerse que se retransmita el paquete continuamente hasta que se reciba confirmación de su llegada o el enlace caiga.

- Calidad de Servicio (*QoS*): opción que describe el flujo de tráfico. No está soportada por la API.

OBEX es también conocido como protocolo de intercambio de objetos. *OBEX* no se encuentra definido en la API JSR 82 de *Bluetooth*, sino que posee su propia API. *OBEX* es un protocolo diseñado por *IrDa* (*Infrared Data Association*) para intercambiar objetos entre clientes y servidores mediante el establecimiento de sesiones *OBEX*. Para J2ME se optó por extender la API de OBEX para dar cobertura a *Bluetooth.* OBEX implementa la transferencia de objetos estableciendo una sesión, mediante una petición CONNECT. Ésta termina mediante una petición DISCONECT. Entre estas dos peticiones, el cliente puede traer objetos del servidor mediante GET, o enviarlos mediante PUT. Los objetos pueden ser archivos, vCards, arrays de bytes, etc.

En un principio los tres protocolos de comunicación podrían utilizarse para realizar la comunicación *Bluetooth*. No obstante, hay que destacar que en el caso de OBEX su uso supone la creación virtual en los dos dispositivos móviles con intención de comunicarse de un cliente y un servidor OBEX, puesto que la comunicación ha de ser bidireccional y en OBEX es el cliente el que lleva (PUT) o trae (GET) los objetos desde el servidor. Por otro lado, para *L2CAP* el hecho de que se tengan que configurar parámetros de Unidad Máxima de Transferencia de datos tanto en recepción como transmisión y tiempos de descarte introduce aspectos de diseño a tener en cuenta a la hora de definir los mensajes a transmitirse por la aplicación y que puede considerarse en cierta forma excesivo para la naturaleza de los mensajes que se espera enviar entre los dos dispositivos.

De los anteriores motivos se desprende que una solución satisfactoria al problema de la comunicación *Bluetooth* se puede conseguir mediante el empleo del protocolo de *RFCOMM*. Así, de los tres protocolos de comunicación se elige la opción de *RFCOMM* debido a la versatilidad y mayor sencillez.

Una aplicación que ofrezca un servicio basado en el perfil de puerto serie (*SPP*) –protocolo *RFCOMM*- es un servidor *SPP*. Una aplicación que inicie una conexión a un servicio *SPP* es un cliente *SPP*. Cliente y servidor residen en los extremos de una sesión *RFCOMM*. El servidor *SPP* registra su servicio en el *SDDB* (*Service Discovery DataBase*), y como parte del proceso de registro, se añade un identificador de canal (*channel identifier*) al *ServiceRecord* por la implementación.

Establecer una conexión satisfactoria consta de una serie de pasos: inicialización de la pila *Bluetooth* (muy dependiente del fabricante del dispositivo), descubrimiento de dispositivos y servicios, manejo del dispositivo y comunicación. Al final del proceso lo que queda es un flujo o *stream* de datos. Para el caso de nuestra aplicación debería tenerse dos flujos, uno por cada sentido *in*, *out* en el servidor y el cliente. Sobre estos flujos irá la información que deben intercambiar el cliente y el servidor para que el modo multijugador funcione.

Para la inicialización de la aplicación Bluetooth hay que tener en cuenta el *BCC* (*Bluetooth Control Center*). Los dispositivos *Bluetooth* que implementen la JSR 82 pueden permitir que múltiples aplicaciones se estén ejecutando concurrentemente. El *BCC* previene que una aplicación pueda perjudicar a otra. El *BCC* puede ser una aplicación nativa, una aplicación en un API separado, o sencillamente un grupo de parámetros fijados por el proveedor que no pueden ser cambiados por el usuario.

En la fase de descubrimiento los dispositivos *Bluetooth* descubren otros dispositivos con los que poder realizar la comunicación. Los dispositivos inalámbricos son móviles, motivo por el cual necesitan un mecanismo que permita encontrar, conectar, y obtener información sobre las características de otros dispositivos con los que conectarse.

Cualquier aplicación puede obtener una lista de dispositivos en su radio de cobertura. Estos dispositivos los encuentra, usando las funciones, o bien *startInquiry()* (no bloqueante) o *retrieveDevices()* (bloqueante). *startInquiry()* requiere que la aplicación tenga especificado un *listener,* el cual es notificado cuando un nuevo dispositivo es encontrado después de haber lanzado un proceso de búsqueda. Por otra parte, si la aplicación no quiere esperar a descubrir dispositivos (o a ser descubierta por otro dispositivo) para comenzar, puede utilizar *retrieveDevices()*, que devuelve una lista de dispositivos encontrados en una búsqueda previa o bien unos que ya conozca por defecto.

A continuación se muestra código esquemático para el proceso de descubrimiento de dispositivos, basado en el método *startInquiry()* no bloqueante, y servicios mediante el protocolo *RFCOMM* para *Bluetooth*, este código formará parte del cliente SPP:

```
//Lista de dispositivos y servicios encontrados
public static Vector dispositivos_encontrados = new Vector();
public static Vector servicios_encontrados = new Vector();
public static int dispositivo_seleccionado = -1;

//Objetos Bluetooth necesarios
public LocalDevice dispositivoLocal;
public DiscoveryAgent da;
```

```
//Código de ejemplo para búsqueda de dispositivos.

…
      try {
            dispositivoLocal = LocalDevice.getLocalDevice();
            dispositivoLocal.setDiscoverable(DiscoveryAgent.GIAC);
            da = dispositivoLocal.getDiscoveryAgent();
            da.startInquiry(DiscoveryAgent.GIAC,new Listener());
      } catch(BluetoothStateException be){
            be.printStackTrace();
      }

…
//Código de ejemplo para búsqueda de servicios.

…
//Buscamos el servicio de puerto serie en el dispositivo seleccionado
RemoteDevice dispositivo_remoto = (RemoteDevice)
dispositivos_encontrados.elementAt(dispositivo_seleccionado);

            try{
            // Buscamos en el puerto serie 0x1101 en el servidor SPP
                  da.searchServices(null,new UUID[]{new
UUID(0x1101)},dispositivo_remoto,new Listener());

            }
            catch(BluetoothStateException be){
                  mostrarAlarma(be, this.c, 0);
            }

…
```

Al emplear el método *startInquiry* y *searchServices* se hace necesario el empleo adicional de un *listener*, que implementa la interfaz *DiscoveryListener* del paquete *javax.bluetooth*. Este *listener* recibe la notificación del descubrimiento de nuevos dispositivos y de los servicios asociados. A continuación se muestra un ejemplo de código para el *listener*:

```
//Se implementa el DiscoveryListener
public class Listener implements DiscoveryListener{
//Se implementan los métodos de la interfaz DiscoveryListener

      public void deviceDiscovered(RemoteDevice dispositivoRemoto,
DeviceClass clase){

            System.out.println("Se ha encontrado un dispositivo
      Bluetooth");

            dispositivos_encontrados.addElement(dispositivoRemoto);

      }

      public void inquiryCompleted(int completado){

            System.out.println("Se ha completado la búsqueda de
      dispositivos");

            if(dispositivos_encontrados.size()==0){
```

```
            //Notificar que debe realizar una búsqueda nueva de
            //dispositivos.
            …
            }
            else{
            //Mostrar los dispositivos encontrados, para que
            //puedan seleccionarse.
            …
            }

      }

      public void servicesDiscovered(int transID, ServiceRecord[]
servRecord){

            for(int i=0;i<servRecord.length;i++){

                  ServiceRecord servRecordAux = servRecord[i];

                  servicios_encontrados.addElement(servRecordAux);
            }
      }

      public void serviceSearchCompleted(int transID, int respCode){

            System.out.println("Terminada la búsqueda de servicios");

            if(servicios_encontrados.size()>0){
            //Mostrar los servicios encontrados en el puerto serie.
            …
            }
            else{
            //Notificar que no se encontró ningún servicio de puerto
            //serie.
            …
            }
      }
}
```

Al otro lado de la comunicación debe encontrarse el dispositivo *Bluetooth* junto con el servicio correspondiente dispuesto a ser descubierto. Un esquema de código que configura esta parte de servidor SPP es:

```
…
public boolean fin = false;
//Objetos Bluetooth necesarios
public LocalDevice dispositivoLocal;
public DiscoveryAgent da;
public StreamConnectionNotifier servidor;
…
//Se le da un nombre a la aplicación
String nombre = "Ejemplo SPP";
//Se define un UUID único para este servicio. Se elige uno a nuestro
//gusto.
UUID uuid = new UUID(0xABCD);
servidor = null;//Similar a un socket servidor
```

```
//Se intenta crear una conexión, usando SPP
      try{
            servidor =
(StreamConnectionNotifier)Connector.open("btspp://localhost:"+uuid.toS
tring()+";name="+nombre);
      //Se obtiene el service record
            ServiceRecord rec = dispositivoLocal.getRecord(servidor);
      //Se rellena el BluetoothProfileDescriptionList usando el
      //SerialPort version 1
            DataElement e1 = new DataElement(DataElement.DATSEQ);
            DataElement e2 = new DataElement(DataElement.DATSEQ);
      //Se agrega el puertoserie fijado de antemano 0x1101, coincide
      //con el que buscará el cliente SPP
            e2.addElement(new DataElement(DataElement.UUID,new
      UUID(0x1101)));
      //Versión 1
            e2.addElement(new DataElement(DataElement.INT_8,1));
            e1.addElement(e2);
      //Se agrega al service record el BluetoothProfileDescriptionList
      //con un índice fijado por el diseñador libremente.
            rec.setAttributeValue(0x0009,e1);

      }
      catch(Exception e){
            e.printStackTrace();

      }
```

Tras el código previo se puede abordar el proceso de conexión y recepción de datos, un ejemplo de código que cumpliría con esto último es:

```
StreamConnection sc = null;//Similar a un socket cliente.
RemoteDevice rd = null;

while(!fin){

      try{

            //Se aceptan conexiones del cliente y se obtiene el objeto
            //remoto.
            sc = servidor.acceptAndOpen();
            rd = RemoteDevice.getRemoteDevice(sc);
            //Se obtiene el input stream del objeto remoto.
            DataInputStream in = sc.openDataInputStream();
            //Se lee el mensaje.
            String msj = in.readUTF();//Por ejemplo lectura de cadena.
            sc.close();
            …

      } catch(Exception e){
            e.printStackTrace();
      }

}
```

En cuanto a la transmisión de los datos, se puede tener un esquema de código como el que sigue:

```
ServiceRecord sr = (ServiceRecord)servicios_encontrados.elementAt(0);

//Se obtiene la URL asociada a este servicio en el dispositivo remoto
String URL =
sr.getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT,false);

try{

      //Obtiene la conexión y el stream de este servicio
      StreamConnection con = (StreamConnection)Connector.open(URL);
      DataOutputStream out = con.openDataOutputStream();

      //Escribimos datos en el stream, por ejemplo una cadena.
      out.writeUTF(msg);
      out.flush();

      //Cerramos la conexión
      out.close();
      con.close();
      mostrarAlarma(null, c, 2);

} catch(Exception e){
      e.printStackTrace();
}
```

La arquitectura *cliente-servidor* de la aplicación se servirá de los bloques de código anteriores para construir el cliente y el servidor, tratándose de un cliente SPP y un servidor SPP. En cada una de los partes se lanzará un hilo encargado de la comunicación con un bucle de recepción de datos y un método para transmitir los mensajes (*enviarMensaje(byte[])*). En la **Figura 4.18** se muestra un diagrama de clases posible para esta arquitectura.

Además, cada una de las clases dispondrá de un método adicional *procesaMensajeCliente()* en el cliente y *procesaMensajeServidor()* en el servidor, que se encargarán de tomar el mensaje recibido en formato array de bytes, interpretarlo y notificar al controlador correspondiente las modificaciones que deba realizar.

La necesidad de una arquitectura de *cliente-servidor* proviene de la sincronía temporal que debe existir en el juego. Esta sincronía afecta al evento tiempo actual, la llegada de objetos en la Cruz Roja, las posiciones de cada personaje en los escenarios y los reset de campo (con las funciones de arreglar vallas, cerrar puertas y recuperar objetos sueltos). De aquí surge la asimetría en el proceso que obliga a fijar a una de las entidades para el modo multijugador como servidor y a la otra como cliente.

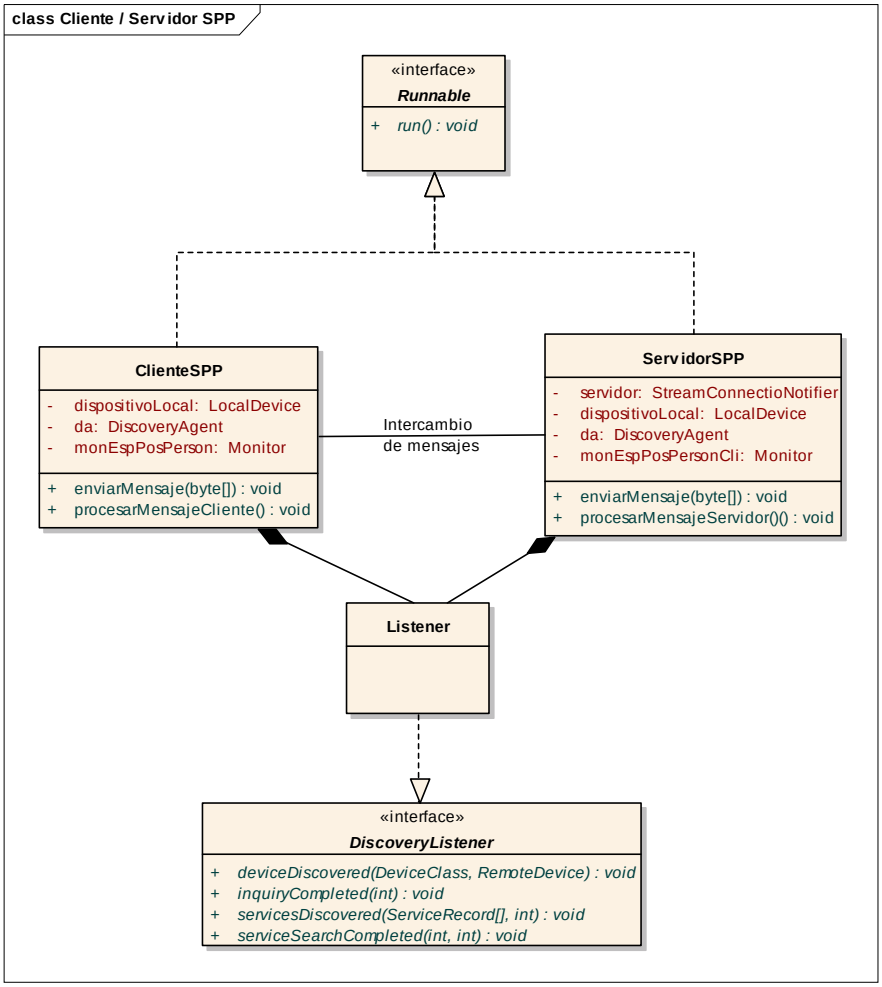


Figura 4.18 Diagrama de clases posible para estructura cliente / servidor SPP.

El servidor tendrá como funciones exclusivas el llevar el hilo temporal de eventos, la llegada de los objetos de la Cruz Roja y el reset del campo, comunicándoselos al cliente a través de mensajes para que éste actualice su hilo temporal, los objetos de la Cruz Roja y proceda al reset del campo en cada día nuevo que amanece. Estos mensajes no requieren de espera de sincronía por parte del cliente.

Por otro lado, se considera que el servidor realizará el cálculo de las posiciones de todos los personajes secundarios teniendo en cuenta la posición de los dos personajes principales; para ello, deberá recibir un mensaje en el que se actualice la posición de éste en el cliente; de modo que se pueda tener en cuenta en los cálculos de posición del resto de personajes. El servidor se sincronizará con el cliente para recibir el mensaje de posición del personaje principal del cliente –este mensaje se enviará de forma periódica en cada ciclo de ejecución del *game loop* del cliente. Los mensajes de posición de los personajes secundarios se enviarán de forma periódica en cada ciclo de ejecución del *game loop* del servidor. Este envío

de los mensajes de posición de los personajes producirá una espera de sincronía en el cliente para recibirlos antes de representar por pantalla, parecida a la espera que acontecerá en el servidor para recibir el mensaje de posición del personaje principal del cliente. La sincronía adicional que se introducirá en los *game loop* del cliente y el servidor, estarán bajo control de las clases *clienteSPP* y *servidorSPP* respectivamente. En la **Figura 4.18** se observan las dos referencias a monitores, uno para el *game loop* del cliente y otro para el del servidor, en los que se basará el mecanismo de sincronía y los cuales se controlarán por el *clienteSPP* y *servidorSPP*. La **Figura 4.19** muestra un esquema de sincronía temporal para los *game loops* del cliente y el servidor con los mensajes enviados en cada momento.

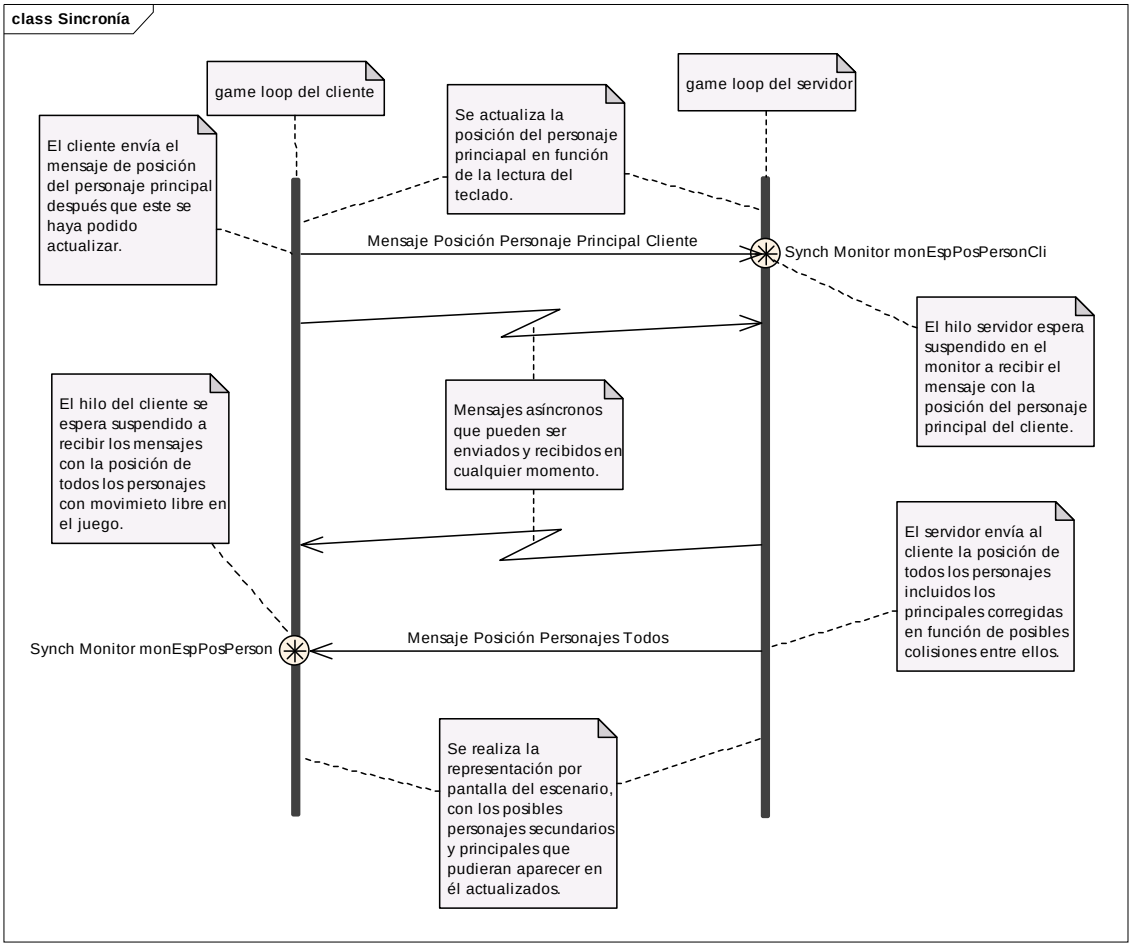


Figura 4.19 Esquema de sincronización entre los dos *game loops* del cliente y el servidor.

Otro mecanismo importante para el diseño de la comunicación mediante *Bluetooth* es la detección de interrupción de la comunicación. Para ello se propone el lanzamiento de *Timers* en el cliente y en el servidor antes de suspender el hilo en los monitores de sincronización respectivos; cumplido el tiempo de dichos *Timers* sin haber recibido el mensaje que saca el hilo de la suspensión se entenderá que la comunicación se ha interrumpido por algún fallo y se

comunicará mediante algún mensaje por pantalla. Cada *timer* tendrá un *flag* que indique si es efectivo o no cuando se cumpla. A este *flag,* que indica si el *timer* ha sido desactivado porque llegó el mensaje esperado o no, sólo se accederá mediante métodos sincronizados a través de la colocación de la etiqueta *synchronized* en el *getter* y *setter* para asegurar un correcto funcionamiento multihilo.

También en ambos sentidos se podrá enviar mensajes de notificación asíncronos como los de notificación de evento temporal nuevo, llegada de objeto en la Cruz Roja o el reset del campo en el amanecer de un nuevo día. Otros mensajes de este tipo serán: el mensaje donde se notifique que el cliente solicita permiso para recoger objeto del suelo (sentido cliente-servidor) y que se le conceda este permiso por parte del servidor (sentido servidor-cliente), que se ha soltado un objeto al suelo, que se rompe la valla, que se haya abierto una puerta, que se haya abierto la entrada a un túnel.

Otro aspecto importante será la sincronización de arranque de las dos aplicaciones en el modo multijugador. Una vez realizada la conexión Bluetootoh, se iniciará un intercambio de mensajes que aseguren un arranque parejo del juego en cada dispositivo móvil. Para ello se propone que el servidor envíe un mensaje especial de arranque y se suspenda en el hilo principal sin mostrar el juego hasta que el cliente le conteste con el mismo mensaje. Tras lo cual ambas partes empezarán el *game loop* mostrando en la pantalla la acción del juego de cada una. En la **Figura 4.20** se muestra un esquema de esta comunicación.

Hay que destacar que para introducir la colaboración entre los jugadores en modo multijugador se incorpora la funcionalidad del objeto especial radio. De modo que uno de los dos personajes puede recoger dicho objeto y usarlo para atraer la atención de los guardias que estén en el radio de alcance del sonido del aparato, haciendo que éstos abandonen su puesto de patrullaje y acudan a la radio, escuchándola durante un tiempo determinado, lo que puede permitir que el otro personaje principal pueda realizar acciones prohibidas sin el control del guardia entretenido con la radio. Por ejemplo, en el modo multijugador puede fijarse un guardia del campo a la entrada del Botiquín, de manera que para poder recoger un objeto entregado por la Cruz Roja hiciera falta utilizar la radio, mediante el compañero de fuga, para distraerlo.

Otra forma de colaboración es la manera de escaparse de los personajes principales. Los personajes principales deben llevar entre ambos los tres objetos especiales: brújula, documentos, y, además, la comida –objeto nuevo para el modo multijugador-, para que la fuga tenga éxito. Existirá un tiempo máximo entre la fuga del primero de ellos y del segundo, de modo que si no se cumpliera en ese tiempo la fuga del segundo, el primero sería arrestado y trasladado a la

celda de castigo. El control del tiempo máximo podrá realizarse mediante un *timer* como los que ya se han comentado en el proyecto anteriormente.

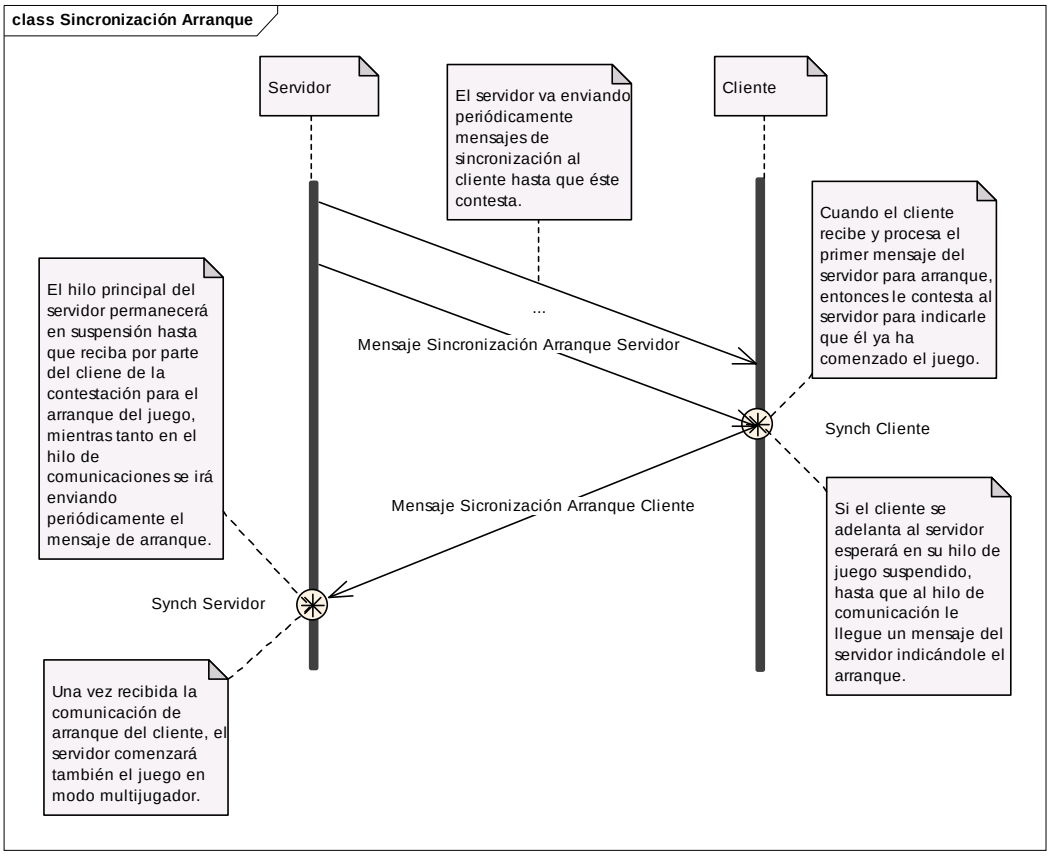


Figura 4.20 Esquema de comunicación para sincronizar el arranque del juego -una vez conectados el clienteSPP y el servidorSPP a nivel del protocolo RFCOMM.

A continuación se va a hacer una descripción de los mensajes que deberían enviarse para una funcionalidad con dos jugadores, así como, del formato de paquete que se propone.

Primero, se explicarán los diferentes campos que constituyen el formato de los paquetes a enviar:

- El primer campo es el identificador de mensaje: un byte que estará en todo mensaje a enviar y que constituirá el inicio del mismo. En el byte, el *nibble* de mayor peso indica el sentido del mensaje, siendo 0 sentido posible tanto cliente-servidor como servidor-cliente, 1 sentido servidor-cliente, y 2 cliente-servidor. Por otro lado, el *nibble* de menor peso indica dentro del sentido del mensaje correspondiente el tipo de mensaje concreto. La distinción de sentido es una cuestión meramente clasificatoria, ya que sólo se tendrán dos dispositivos en conexión Bluetooth para la modalidad multijugador.

- El campo Id. Objeto se corresponde en los mensajes con el identificador único de objeto que cada objeto posee en la tabla del *ControladorObjetosEspeciales*. Consistirá en el mapeo del tipo short al paquete a enviar, ocupando 2 bytes.

- El siguiente campo que se puede presentar es el de Long. (Longitud). Este campo está presente en todos los mensajes de longitud variable, es decir, en todos aquellos que presenten un campo de Nombre Ubicación, Nombre Túnel o Evento Actual que consistirá en el mapeo a array de bytes de un String. A este campo se le reserva un byte. Long. indica el tamaño de la parte variable que se envía, es decir, la longitud en bytes del Nombre Ubicación, Nombre Túnel o Evento Actual presente en el mensaje. Ésta será de un valor máximo de 255 bytes.

- Relacionado con el anterior, se tiene el campo de Nombre Ubicación. Éste campo llevará los bytes -siempre en longitud menor que 256- del nombre del escenario (nombre de habitación o la cadena "*patio-exterior*") que resulte de interés en el mensaje. Es un campo de longitud variable, de ahí la necesidad de ir acompañado de un campo previo de Long. para poder extraerlo.

- Pos. X y Pos. Y son campos que representan la posición de algún personaje u objeto en la matriz del juego, ya sea correspondiente al escenario principal patio-exterior o a alguna habitación interior. Son campos que se conseguirán del mapeo a bytes de variables de tipo int por lo que ocuparán cuatro bytes en cada mensaje.

- Id. Puerta es un campo que se refiere al identificador de la puerta en la tabla de puertas abiertas presente en el *ControladorPuertasAbiertas* –nótese que cada puerta, al igual que los objetos especiales poseen un identificador único que se les asigna tras la lectura de los ficheros de configuración XML. Se trata de un short por lo que ocupará dos bytes en el mensaje.

- El campo Partida Ganada / Perdida indica si la partida se ha ganado por los jugadores o no. Se indicará a través de un solo byte, que tomará el valor 0x01 si es ganada o 0x00 si es perdida. Es importante destacar que para este diseño no se propone un campo que mapee una variable boolean, ya que este tipo de variables no tienen definido tamaño en el estándar java con precisión.

- El campo Permiso Sí / No se utiliza para indicar si uno de los prisioneros tiene permiso o no para recoger un objeto del suelo. Al igual que el campo anterior se emplea un byte para indicar si se tiene permiso, en cuyo caso se envía el 0x01, si no se tiene permiso para recoger el objeto del suelo el valor 0x00.

- Evento Actual se trata de un campo originado por el mapeo a bytes de la cadena de evento temporal en curso que se produce en el servidor. Se trata de un campo de longitud variable teniendo como máximo los 255 bytes que permite el campo Long. que lo precederá.

- Nombre Túnel será el mapeo a bytes del nombre del túnel sobre el que se realiza la apertura de una entrada al mismo bloqueada. Se trata de un campo variable procedente de un String al que precede un campo Long. que indica su longitud.

- Id. Entrada se trata del campo que indica la entrada del túnel que es abierta. El campo tendrá el tamaño de un byte de modo que con el valor 0x00 se indica la entrada al túnel con identificador 0, mientras que con 0x01 se indica la entrada al túnel con identificador 1.

A continuación, se abordan los mensajes a enviarse por parte del cliente y el servidor.

En los dos sentidos, cliente → servidor y servidor→cliente; los mensajes que se muestran a continuación tendrán una naturaleza no periódica y asíncrona. Constituyen la parte simétrica del diseño para la funcionalidad multijugador puesto que estará presente tanto en el cliente como en el servidor. Los mensajes serían:

- *Soltar objeto*: cuando uno de los personajes principales, ya sea el del servidor o del cliente, suelte un objeto, entonces se envía un mensaje de indicación al antagonista para que actualice las tablas de objetos en su correspondiente *ControladorObjetosEspeciales* y aparezca el objeto en la posición donde ha sido soltado.

00	Id.Objeto (short)	Long. (byte)	Nombre Ubicación (String)	Pos. X (int)	Pos. Y (int)

- *Romper valla*: cuando uno de los personajes principales rompa el perímetro de la valla, entonces envía este mensaje al compañero para que actualice la rotura de valla en la tabla de puertas correspondiente del *ControladorPuertasAbiertas*.

01	Pos. X (int)	Pos. Y (int)

- *Abrir puerta*: cuando uno de los personajes principales abra una puerta, se envía este mensaje al compañero para que actualice la tabla de puertas abiertas en el *ControladorPuertasAbiertas*.

02	Long. (byte)	Nombre Ubicación (String)	Id. Puerta (short)

- *Fin aplicación*: mensaje que ha de enviarse cuando la partida se gana porque los dos prisioneros consiguen escaparse en un corto período de tiempo con los objetos necesarios para la fuga, se pierde porque uno de los dos prisioneros queda sin moral.

03	Partida Ganada/Perdida (byte)

- *Abandono de la aplicación*: mensaje que se envía cuando uno de los dos jugadores abandona la partida al otro, de forma que también se deberá cerrar la el juego para el otro jugador.

04

- *Quitar piedra túneles*: mensaje que se envía cuando uno de los dos jugadores abre la entrada a un túnel en el juego, para que se actualice en la aplicación del compañero. Este jugador puede estar en el lado del cliente o en el lado del servidor.

05	Long. (byte)	Nombre Túnel (String)	Id. Entrada (byte)

Estos mensajes actualizarían las estructuras de datos y el dibujo por pantalla en el receptor correspondiente.

En sentido solo servidor→cliente; en este sentido exclusivo se envían mensajes no periódicos y periódicos que requieren de la sincronización por parte del cliente (como es el mensaje para *Posición de cada personaje en el juego*):

- *Evento tiempo actual*: se envía al cliente el evento temporal actual que rige en la aplicación, para que lo notifique por pantalla, con esto se garantiza la sincronización en los eventos mostrados por pantalla.

10	Long. (byte)	Evento Actual (String)

- *Llegada objeto Cruz Roja*: mensaje que notifica a la aplicación cliente el objeto de la Cruz Roja que ha llegado.

11	Id. Objeto (short)

- *Posición de cada personaje en el juego*: la aplicación cliente en cada ciclo de ejecución del *game loop* esperará suspendida hasta su llegada, para representar por pantalla a los personajes.

12	Id. Person. (short)	Long. (byte)	Nombre Ubicación (String)	Pos.X (int)	Pos.Y (int)

Se enviarían tantos mensajes de este tipo como personajes. Si la ubicación para un personaje secundario se expresa como la del comedor o de un dormitorio, se entenderá que éste puede encontrarse, o bien, sentado a la mesa del comedor, o bien, acostado en el dormitorio.

- *Reset del campo prisioneros: arreglar valla, cerrar puertas y recuperar objetos*. Mensaje para indicar al cliente que haga un reset, arreglando las vallas rotas, cerrando puertas y devolviendo a su ubicación original los objetos sueltos por el campo y que no estén en una ubicación segura.

13

- *Permiso para coger objeto del suelo*: mensaje de contestación del servidor hacia el cliente a la solicitud de petición de permiso para recoger objeto del suelo, con esto se evita cualquier posible conflicto entre los dos personajes principales que pudieran intentar recoger a la vez el mismo objeto.

14	Permiso Sí/No (byte)

- *Cliente detenido fuera del campo*: cuando el personaje principal del cliente consigue escaparse, pero transcurre el tiempo de para que el segundo personaje (el del servidor) se escape y no lo consigue, o éste no lleva los objetos necesarios para la fuga entonces el servidor cumplido el tiempo o detectado que no llevan los objetos para la fuga, le envía al cliente este mensaje para que

introduzca al personaje principal del cliente el la celda de castigo. Nótese que para el modo multijugador habrá que añadir un calabozo adicional.

En sentido solo cliente → servidor, mensajes asimétricos que el cliente debe enviar al servidor para notificar su posición y el deseo de coger un objeto, son peticiones que pueden dar lugar a conflictos por lo que es el servidor el que tiene que resolverlas y contestar al cliente con lo que debe hacer éste:

- *Posición del prisionero principal (cliente)*: este mensaje es esperado por el servidor suspendido para poder calcular la posición de los demás personajes en función del movimiento que haga el prisionero principal del cliente. Haya o no haya movimiento del cliente debe enviarse siempre, con la posición del personaje, puesto que el servidor esperará a que llegue para salir de la suspensión.

20	Longitud (byte)	Nombre Ubicación (String)	Pos. X (int)	Pos. Y (int)

- *Petición del cliente Coger objeto*: este mensaje se tiene que enviar cuando el personaje principal del cliente recoja un objeto suelto en suelo. Se trata de una solicitud al servidor para que le dé permiso al cliente a tomar el objeto. El servidor resolverá cualquier posible conflicto de colisión y responderá con un mensaje de *Permiso para coger objeto del suelo*.

21	Id. Objeto (short)	Long. (byte)	Nombre Ubicación (String)	Pos. X (int)	Pos. Y (int)

- *Activar radio*: si es el cliente el que activa la radio debe enviar un mensaje al servidor indicándole dónde se ha activado ésta para que el servidor lo tenga en cuenta para hacer que los posibles guardias que la escucharan se acerquen hasta ella. Si es el personaje principal del servidor el que activa la radio puesto que está en el lado que controla los movimientos de los personajes, entonces no hace falta que envíe ningún mensaje para el cliente.

22	Nombre Ubicación (String)	Pos. X (int)	Pos. Y (int)

- *Fuga del cliente*: si el cliente consigue escaparse lo notifica al servidor, para que éste activa un contador con el tiempo que tiene el segundo prisionero para escaparse o compruebe si se ha producido el final del juego por conseguir escapar.

23

El juego tiene que modificar algunos aspectos para en base a estos mensajes poder desarrollarse. Por la necesidad de sincronía temporal entre los dos jugadores, cuando uno de los personajes sea detenido, en la celda de castigo no transcurre un día completo, sino sólo un determinado tiempo, para así no romper la posible acción del segundo jugador. Además, la cama vacía del tercer barracón será la que se le asigne en el modo multijugador al segundo personaje principal. El número de objetos permanecerá inalterado, por lo que deberán colaborar para intentar la fuga. Se desactivaría la opción de relanzar la aplicación. Una vez uno de los dos jugadores perdiera o ganase la partida, la aplicación se terminaría.

Resulta evidente que para tener esta estructura de *cliente-servidor* en base a los paquetes de información anteriores habrá que introducir modificaciones en los controladores correspondientes tanto en la versión de cliente como de servidor, para que puedan transmitir los paquetes en el momento de producirse. Estas modificaciones pueden ser:

En los dos sentidos cliente → servidor y servidor→cliente:

- *Soltar objeto* entraría en el campo de *ControladorObjetosEspeciales*.
- *Romper valla* afectaría a *ControladorRomperValla*.
- *Abrir puerta* afectaría *ControladorPuertasAbiertas.*
- *Fin aplicación* produce la aparición de un *canvas* donde se informa del motivo de fin: partida ganada, partida perdida y error en la recepción de paquetes.

En sentido solo servidor→cliente:

- *Evento tiempo actual* afectaría al controlador Tiempo
- *Llegada objeto Cruz Roja* afectaría al controlador *Cruz Roja*
- *Posición de cada personaje secundario en el juego* afectaría a *MetaControlPerson* y los controladores de personajes que se encuentran tras él.
- *Reset del campo prisioneros: arreglar valla, cerrar puertas y recuperar objetos* tendría el mismo efecto que el reset normal ya existente. (Puede que este mensaje no hiciera falta el evento Amanecer enviado en un mensaje Evento tiempo actual puede desencadenar el mismo efecto)
- *Permiso Coger objeto*: este mensaje afectaría a *ControladorObjetosEspeciales* ya que da el visto bueno para recoger el mensaje o no, por posible conflicto con el personaje del servidor que intenta recoger a la vez el mismo objeto.

En sentido solo cliente → servidor:

- *Petición del cliente Coger objeto* entraría en el campo de *ControladorObjetosEspeciales*.
- *Posición del segundo prisionero (cliente)* afectaría al controlador *MetaControlPerson*, en concreto a *ControladorPrisioneros*.
- *Activar radio* que afectaría a los controladores *ControladorObjetosEspeciales* y *ControladorGuardias*.
- *Fuga del cliente* este mensaje dejaría en *standby* al cliente mientras el segundo personaje intenta la fuga. Podría aparecer por pantalla del cliente un mensaje del tipo: *Esperando al compañero de fuga …*

Capítulo 5.
Conclusiones y líneas futuras

5.1 Conclusiones finales.

Este proyecto aborda el diseño y la implementación de una aplicación de videojuego en perspectiva isométrica con capacidades multijugador vía Bluetooth para los dispositivos móviles en base a la plataforma Java J2ME. Se tomó como modelo de inspiración para el juego un juego en perspectiva isométrica de mediados de la década de los 80: *The Great Escape*.

La aventura gráfica original en la que se basa el proyecto se inspira a su vez en la famosa película homónima (en español, *La gran evasión*) donde un grupo de prisioneros de guerra aliados, recluidos en un campo de concentración nazi, tratará de evadirse, a través de la construcción de dos túneles que les conduzcan lejos del doble perímetro de alambre de espino que circunda todo el campo.

En el juego original se tiene un personaje principal, recluido en un campo de concentración. La acción transcurre en una especie de castillo fortaleza nazi, con dos laterales rodeados por un doble vallado de alambre de espino, los laterales este y sur; y los dos restantes, los laterales oeste y norte, por las paredes del castillo con puertas de entrada a habitaciones interiores. Además, entre las paredes del castillo y el doble alambrado, se tiene un espacio abierto -el patio exterior del castillo- donde se ubican tres barracones de prisioneros. El juego en el momento de su aparición aportó la novedad de un hilo temporal que guiaba la actividad en el campo. Existen tres tipos de personajes: prisionero, soldado y general, cada uno con una movilidad libre de autómata que se ajusta en cada momento al evento temporal en que se encuentra la aventura. Los distintos eventos temporales se van sucediendo dando lugar a la rutina habitual de un campo de concentración con eventos, por ejemplo, de ir a formar, ir a comer o ir a dormir; de esta forma los días se suceden en el campo de concentración teniendo el personaje principal que cumplir con la rutina diaria para no despertar sospechas.

El juego contaba, además, con una serie de habitaciones interiores del castillo que el personaje debía investigar, algunas de ellas cerradas, teniendo que utilizar los objetos especiales llaves o herramientas para abrirlas, y donde podía recoger otros objetos especiales que le

servirían para la fuga. Para alcanzar la meta de la fuga, el personaje principal sólo dispone de dos formas, o bien, rompiendo la doble valla mediante el empleo de las tenazas con el riesgo añadido de que puede ser descubierto por los soldados que patrullan por el interior del doble vallado, o bien, encontrando los túneles y sirviéndose de ellos para escaparse. Además, para que la fuga sea exitosa, el personaje principal debe llevar consigo dos objetos especiales de los que puede encontrar por el campo. En todo momento, se encuentran por el campo de prisioneros múltiples guardias –los soldados y el general- que patrullan y observan que el personaje principal no se acerque a las zonas prohibidas, lanzándose en su persecución si en algún momento éste es descubierto en una de ellas o haciendo algo indebido. En definitiva, el juego trata de transmitir la atmósfera opresiva de un campo de concentración en la Alemania nazi, y en esa atmósfera el jugador debe conducir a su personaje para llevarlo a una fuga exitosa.

Durante todo el desarrollo de la aplicación se ha pretendido ser fiel al guión original de la aventura, así como a la apariencia gráfica de la misma. Esto se aprecia en la reproducción fidedigna que se ha conseguido de los escenarios tanto exteriores como interiores del juego original y en el conjunto de rutinas temporales que se suceden en el juego, muy semejantes a las que acontecen en el juego original.

En las primeras fases del desarrollo de la aplicación una idea fue tomando fuerza: gran parte de la configuración de la aplicación debería residir fuera del propio código fuente. El discurrir natural de este principio en J2ME conduce a la consideración de ficheros externos de configuración. En este punto surge XML como base tecnológica ineludible para abordar desde un punto de vista estándar y ordenado la construcción de los ficheros de configuración, aparte de todo el soporte existente en J2ME para una solución XML.

Antes de llegar a la solución mediante XML que convertiría la aplicación en una aproximación bastante exhaustiva a un motor gráfico configurable mediante ficheros XML, se hicieron diversas pruebas que principalmente lidiaban con las matrices que habían de construir los escenarios.

En un principio la codificación de las matrices de los escenarios se incluía directamente en el propio código. No obstante, se vio que resultaba un enfoque inviable para el tamaño del escenario principal del juego: el patio exterior del castillo fortaleza. Estas matrices, entonces, se trasladaron a ficheros de texto externos de formato privado. Con la progresiva inclusión de mayor número de tópicos en los ficheros de configuración, el uso de ficheros en formato XML fue una necesidad para poder avanzar de manera controlada en el desarrollo. Sin embargo, debe destacarse que el empleo de los ficheros XML de configuración llevó a una complejidad mayor

en la aplicación, características como la movilidad de los personajes en el juego, que residen en los ficheros de configuración XML, debían abordarse desde un punto de vista de generalización para un conjunto de posibilidades bastante amplio y condujeron a una mayor complejidad de los motores de movimiento de los personajes secundarios y de interpretación de las instrucciones contenidas en los ficheros XML.

Una ventaja que se derivó del uso de ficheros de configuración XML fue que durante la fase intensiva de construcción de los escenarios tanto para el patio exterior como las habitaciones interiores se tuvo que hacer un número elevado de pruebas en los emuladores de SUN y en el de Nokia para la serie S60 de modo que el resultado final fuera satisfactorio, teniendo que hacer cambios en etiquetas y atributos XML de los ficheros. Las facilidades de *parseo* que J2ME presenta para los ficheros XML proporcionaron un campo de actuación ordenado en el código fuente que facilitó la construcción de los escenarios.

Otra idea importante y de gran relevancia para la realización del juego consistió en la matriz de celdas isométricas en que se divide el escenario y la correspondencia de éstas con las celdas de las matrices internas de la aplicación de dibujo, fronteras y puertas y los correspondientes ficheros XML donde se configuran. Gracias a esta separación se tiene un mayor grado de libertad en la construcción de los escenarios. Debe destacarse el concepto de las diagonales para introducir marcas en las respectivas matrices que entronca con los ejes de coordenadas X e Y de la proyección isométrica, y de las cuales se sirve la aplicación para insertar los bloques de imágenes de los escenarios, las fronteras de colisión para los personajes con movimiento libre y las marcas de puertas hacia otros escenarios.

En lo referente a la generación de los escenarios, debe destacarse que sin las capturas tomadas del emulador *CCS64* del juego original y el posterior procesado de las mismas en la aplicación *Adobe Photoshop CS*, sobre todo, a la hora de manipular las imágenes para conseguir las zonas de transparencia adecuadas que permitieran en base a los tres planos del *z-buffer* de dibujo una correcta representación de la profundidad de los escenarios isométricos, no habría sido posible llevar a una correcta finalización el proyecto.

También se ha tenido un especial cuidado en el desarrollo de la interfaz de menús de la aplicación. Partiendo del patrón de Ben Hui para los menús, pero introduciendo modificaciones allí donde éste se quedaba limitado para lo deseado en la aplicación, se han conseguido unos menús navegables de uso muy intuitivo. Se debe destacar en estos menús la navegabilidad, así como los botones de activación y desactivación tanto del sonido como de la vibración. En este sentido también destacan los botones de comandos del menú en tiempo de juego.

En lo referente a la estructura interna de la aplicación las diferentes funcionalidades internas más importantes se han intentado encapsular en un conjunto de controladores. Estos controladores siguen un patrón software *singleton* para facilitar su manejo en las diferentes clases que hacen uso de ellos. En cada controlador se engloba una funcionalidad específica del juego como pueden ser los objetos especiales cuyo control se encuentra en el *ControladorObjetosEspeciales*, el discurrir del complejo hilo temporal en el campo de prisioneros que se encuentra en la clase *Tiempo*, entre otros.

La movilidad de los personajes en la aplicación constituye otro aspecto destacable de la misma. Durante el desarrollo se hizo un especial esfuerzo en conseguir una movilidad natural y fluida de los personajes a través de los escenarios. La movilidad del personaje principal a través de las celdas isométricas y siempre limitada por las fronteras de colisión puede decirse que ha resultado muy satisfactoria. Se puede decir que presenta un alto grado de fluidez tanto en el escenario exterior como interior, siendo las características de estos diferentes (en el patio exterior el personaje principal queda centrado en la posición media de la pantalla, fijo en ella, y al desplazarse se produce un efecto como de desplazamiento de la "ventana" a través de la cual se ve el campo con él siempre centrado en medio; mientras que en las habitaciones interiores el desplazamiento del personaje es libre, produciendo un efecto de *scroll* o arrastre de la pantalla cuando éste llega a los límites de la pantalla). También está, aunque con mayor sencillez, el desplazamiento del personaje principal por el interior de los túneles con la capacidad de cambiar el sentido del movimiento y de uso de objetos.

En cuanto a la movilidad de los personajes, sin duda, las capacidades de movimientos de los personajes secundarios supusieron otro gran reto. El control de la movilidad reside en el conjunto de controladores encargados de los personajes del juego y concretamente en unos métodos de estos que reciben el nombre de motores de movimiento. Existen tres tipos de personajes en la aplicación con movilidad automática: los prisioneros, los guardias y el general. Cada uno de ellos tiene unas características diferentes, por ejemplo los guardias y el general deben poder descubrir al personaje principal y perseguirlo hasta atraparlo por el patio exterior si éste estaba realizando alguna acción indebida. El algoritmo que conduce a los personajes secundarios por la matriz de celdas –el escenario patio exterior concretamente- y resuelve las posibles colisiones tanto con fronteras como con otros personajes en su movimiento libre constituye otro de los elementos clave de la aplicación, habiéndose conseguido un resultado de movilidad natural y satisfactoria

También debe destacarse la estructura multihilo de motores gráficos encargados de representar los escenarios del juego como son el patio exterior, las habitaciones interiores y los túneles. A este respecto destaca la sincronización entre los diferentes hilos conseguida en base a los monitores Java. Como ya se comenta en el proyecto, la división de los distintos escenarios en diferentes hilos conlleva un encapsulamiento de las características de los mismos que favorece el desarrollo de la aplicación de modo elegante, pero teniendo la contrapartida del especial cuidado que se debe tener a la hora de sincronizar los diferentes hilos en los saltos entre escenarios.

Un aspecto importante para la construcción de los escenarios fueron las pruebas en el teléfono móvil Nokia E65 que el departamento puso a disposición del proyecto. Un hecho que produjo gran incertidumbre en el desarrollo del proyecto fue el constatar que versiones de la aplicación que en los emuladores tanto de Sun como de Nokia no presentaban fallos, al trasladarlos al dispositivo móvil físico presentaban fallos. Para depurar estos fallos se tuvo que hacer uso de las herramientas de visionado de trazas de J2ME mediante Bluetooth para los teléfonos móviles de Nokia de la serie S60 presentes en el SDK FP1 3rd Edition. Este aspecto de la depuración en el dispositivo físico móvil de fallos no presentes en los emuladores nos revela un hecho importante del desarrollo de aplicaciones para los dispositivos móviles: la fuerte dependencia de la ejecución de la aplicación del dispositivo móvil. Aún utilizando una plataforma como J2ME que destaca por la búsqueda de la portabilidad de las aplicaciones, debe tenerse muy en cuenta los dispositivos móviles físicos sobre los que pueda correr la aplicación, realizándose pruebas en ellos que aseguren un correcto funcionamiento. A este respecto la aplicación desarrollada en este proyecto se ha comprobado que funcionaba con un rendimiento satisfactorio en los siguientes teléfonos móviles:

- Nokia E65.
- Nokia N95.
- Nokia N81.
- Sony Ericsson K800i.
- Nokia 6110.

Como ya se ha comentado en el proyecto, el diseño de la aplicación se centra en los dispositivos móviles de la serie S60 de Nokia. No obstante, como se desprende de la inclusión en la lista anterior de un dispositivo Sony Ericsson dada la portabilidad de las aplicaciones de J2ME la aplicación posee un rendimiento satisfactorio en teléfonos de otros fabricantes siempre y cuando éstos sean de unas prestaciones medio-altas.

Es de destacar que, aunque en un primer momento el objetivo del proyecto tomó como base la aventura y apariencia gráfica del juego: *The Great Escape*, ya citado, el enfoque de utilización de ficheros de configuración XML dota a la aplicación de unos principios generales de construcción de juegos en perspectiva isométrica –en cierta medida las etiquetas XML contenidas en los ficheros constituyen un pequeño lenguaje de etiquetas para la realización de juegos en perspectiva isométrica con características parecidas a las del juego tomado como referencia. En base a estos principios se podría construir un juego con una apariencia gráfica y estructura completamente distinta a la del actual.

La búsqueda de unas prestaciones altas en el juego supuso la realización de un arduo proceso de depuración de la aplicación. En este sentido, una vez alcanzada la fase de desarrollo de Bluetooth se vio que la implementación de ésta supondría un excesivo sobrecoste en tiempo. No obstante, conforme al título del proyecto se hizo un especial hincapié en el diseño del modo multijugador vía Bluetooth, quedando reflejado éste en la memoria del proyecto.

Finalmente, el desarrollo de todo proyecto de ingeniería supone un reto intelectual de gran envergadura. Un esfuerzo continuado a lo largo de un período de tiempo en el que se van sucediendo avances con el descubrimiento de nuevos problemas a resolver. Durante todo el proceso, la idea de partida se va concretando a unas bases de ingeniería, se va formalizando paulatinamente, y esa idea da pie a otras, a ramificaciones, que también formarán parte del proyecto. Nunca se puede considerar que un proyecto sea fruto de una sola persona, siempre existen muchas personas que contribuyen a la realización, de manera directa o indirecta, personas que con su esfuerzo ayudan al avance de la ingeniería y a la mejora de la calidad de vida de toda la sociedad.

5.2 Líneas futuras.

La principal línea futura que queda abierta en este proyecto es la implementación de la parte Bluetooth según las directrices de diseño contenidas en esta memoria. Como ya se ha comentado en el apartado anterior, esta funcionalidad no se trasladó a código fuente debido al excesivo coste en tiempo que habría supuesto; sin embargo, se realizó el diseño en detalle para facilitar una posible realización futura. Sin duda, la clave de esta línea futura reside en la sincronización entre el cliente y el servidor, en cada ciclo de los *game loops*, puesto que cada uno necesita de datos que debe proveerle su contrapartida.

Otra de las posibles líneas de ampliación de la aplicación de videojuego consiste en la extensión de la movilidad de los personajes secundarios al interior de las habitaciones. En el proyecto desarrollado la movilidad de los personajes secundarios se circunscribe al escenario principal –el patio exterior. Extendiendo los criterios de diseño de la movilidad de los personajes a los escenarios interiores, se considera que podría resultar una línea futura de realización factible, si bien el salto entre habitaciones interiores puede llevar a consideraciones adicionales.

La posibilidad de guardar el estado de la partida en base a RMS (*Record Management System*) que proporciona J2ME sería otra línea futura a desarrollar. En este sentido, el abordar esta línea de ampliación posiblemente conduciría a tener que reelaborar el control temporal que permite ir desarrollando los eventos temporales del juego, puesto que el guardar la partida en una aventura gráfica con un hilo temporal que es pieza clave para su desarrollo obliga necesariamente a respetar ese hilo temporal a la hora de restablecer la partida en el punto donde fue guardada.

También y como una posible línea futura de mayor complejidad cabría la posibilidad de realizar un encriptado de los ficheros de configuración XML, de modo que su contenido no fuera accesible a terceros y evitar que se pudieran utilizar las capacidades de motor gráfico configurable para desarrollar variantes del juego. Esta línea de desarrollo comprendería el empleo de alguna herramienta de encriptado de ficheros, a través de la cual se pasarían los ficheros de configuración XML a residir en el jar de despliegue de la aplicación, además de la incorporación de un módulo de desencriptado en la aplicación que cargaría estos ficheros encriptados para desencriptar posteriormente para uso normal de la aplicación. Sin duda, habría que probar si este proceso de desencriptación no supusiera un coste en tiempo elevado que ralentizara la aplicación. Algunos ejemplos de librerías ligeras para encriptación en J2ME en la actualidad son: Masabi EncryptME [27] o The Legion Bouncy Castle [28]. El primero consiste en una librería propietaria extremadamente ligera; mientras que la segunda es más pesada, pero de código abierto.

Otras líneas de ampliación de la aplicación podrían contemplar el movimiento libre del personaje principal en los túneles con un desarrollo de éstos en mapa laberíntico, contemplando la posibilidad de que se pudieran esconder objetos en ellos. La inclusión de las luces de vigilancia por la noche, simulando los focos; luces que seguirían al personaje principal si fuera éste descubierto por alguna de ellas. Además, podrían incluirse los perros que patrullan por el interior del doble vallado. Todas estas líneas en un principio se consideran ampliaciones naturales y que no deberían acarrear excesivas complicaciones.

Bibliografía y Referencias

Bibliografía

Agustín Froufe Quintas y Patricia Jorge Cárdenas, “Java 2 Micro Edition (J2ME). Manual de usuario y tutorial”, editorial Ra-Ma, edición 2004. Manual de consulta para J2ME.

Alberto Gutiérrez Martínez y Francisco José García Peñalvo, “Juegos de calidad comercial en J2ME”, Revistas Profesionales S.L., Sólo programadores, Nº 122 - 126, 2005.

Cay S. Horstmann y Gary Cornell, “Java 2. Volumen 1. Fundamentos. Volumen 2. Características avanzadas”, editorial Pearson Educación, edición del 2003.

Fco. Javier Ceballos Sierra, “Java 2: Curso de programación”, editorial RA-MA, edición del 2000.

Forum Nokia, Overview of Multiplayer Mobile Game Design, Versión 1.1 del 2003.

Jorge Rubira, “Desarrollo de juegos para móviles con J2ME”, Revistas Profesionales S.L., Sólo Programadores nº 120 y 121.

Manuel Jesús Prieto Martín, “Desarrollo de juegos con J2ME”, Ra-Ma, 2005.

Manuel J. Prieto, “Curso de J2ME. JAVA 2 MICROEDITION”: http://www.javahispano.org/contenidos/es/curso_de_j2me/?menuId=jH&onlypath=true

Pedro Daniel Borches Juzgado, “Java 2 Microedition: Soporte Bluetooth”, Universidad Carlos III de Madrid, Versión 1: http://www.it.uc3m.es/celeste/docencia/j2me/tutoriales/bluetooth/EstudioTecnologico1_0.pdf

Sergio Gálvez Rojas, Lucas Ortega Díaz, “Java a tope: J2ME”, Universidad de Málaga, edición electrónica. Tutorial sobre la plataforma J2ME: http://www.lcc.uma.es/~galvez/J2ME.html

S60 Platform, “Designing Isometric Adventure Games”, Versión 1, 2006.

Referencias

[1] Estadísticas y estudios acerca del mercado de los juegos para dispositivos móviles, presente en la página Web de Qualcomm.
http://brew.qualcomm.com/brew/en/developer/brew_gaming/market_research/

[2] Página Web de los videojuegos de la empresa THQ para dispositivos móviles.
http://us.thqwireless.com/

[3] Página Web de los videojuegos de la empresa EA para dispositivos móviles.
http://www.eamobile.com/Web/

[4] Página Web de los videojuegos de la empresa Glu para dispositivos móviles.
http://www.glu.com/emea/Pages/home.aspx

[5] Página Web de los videojuegos de la empresa Capcom para dispositivos móviles.
http://www.capcommobile.com/

[6] Página Web de los videojuegos de la empresa Gameloft para dispositivos móviles.
http://www.gameloft.es/

[7] Página Web de la compañía Sun Microsystems, referente a la tecnología de la plataforma J2ME.
http://java.sun.com/javame/index.jsp

[8] Página Web de la plataforma para videojuegos Brew de Qualcomm.
http://brew.qualcomm.com/brew/en/developer/brew_gaming/

[9] Página Web de la plataforma para videojuegos Ex-En de Infusio.
http://developer.in-fusio.com/exen/information_technology.php

[10] Página Web del Sistema Operativo para móviles Symbian.
http://www.symbian.com/

[11] Página Web del Sistema Operativo para móviles de Microsoft.
http://www.microsoft.com/windowsmobile/en-us/default.mspx

[12] Página Web de la compañía que desarrolló el Sistema Operativo Palm (actualmente propiedad de la compañía Access)
http://www.palmsource.com

[13] Página Web de la compañía Apple donde se pueden encontrar las características del Sistema Operativo del iPhone.
http://developer.apple.com/iphone/

[14] Página Web con información acerca del Sistema Operativo Openmoko.
http://www.openmoko.com/

[15] Página Web del Sistema Operativo Access Linux Platform.
http://alp.access-company.com/

[16] Página Web del Sistema Operativo QTopia.
http://www.qtopia.net/

[17] Página de la empresa ARM Limited, dueña de la arquitectura ARM, sobre la que corren en exclusiva el Sistema Opertivo Symbian.
http://www.arm.com/

[18] Página con información general acerca del video juego *The Great Escape*.
http://en.wikipedia.org/wiki/The_Great_Escape_(video_game)

[19] Publicación *on line* donde se recoge al grupo de creadores del videojuego original de The Great Escape.
http://www.crashonline.org.uk/36/denton.htm

[20] Página Web del emulador CCs 64
http://www.computerbrains.com/ccs64/

[21] Página Web donde se recoge la publicación: Micromanía, Año 2, Nº 18 Diciembre de 1.986. http://blogs.vandal.net/52116/vm/195941642008

[22] Recomendaciones del organismo W3C para XML.
http://www.w3.org/TR/REC-xml/

[23] Página Web con el artículo on line de Sun Microsystems donde se realiza comparativa entre diferentes *parseadores* XML para J2ME.
http://developers.sun.com/mobility/midp/articles/parsingxml/

[24] Página Web donde encontrar el Al Sutton XML Parser.
http://asxmlp.sourceforge.net/

[25] Página Web con el artículo de Ben Hui:
Big designs for small devices.
Four J2ME design patterns simplify interactive content development.
http://www.javaworld.com/javaworld/jw-12-2002/jw-1213-j2medesign.html

[26] Página Web con la API de J2ME: JSR82.
http://java.sun.com/javame/reference/apis/jsr082/

[27] Página Web de la librería propietaria de encriptación Masabi Encryptme.
http://www.masabi.com/tech_encryptME.html

[28] Página Web con la API de encriptación libre de The Bouncy Castle Legion.
http://www.bouncycastle.org/

Otras reseñas bibliográficas:

http://java.sun.com/javame/reference/apis.jsp
Enlace donde se encuentran las API de J2ME.

http://java.sun.com/products/sjwtoolkit/
Página Web de Sun donde se encuentran el Wíreless Toolkit, emulador de Sun para J2ME, y documentación para desarrolladores.

http://www.forum.nokia.com/
Página Web de Nokia donde se encuentran recursos como SDK para la serie S60 y documentación para desarrolladores.

http://webdelprofesor.ula.ve/nucleotrujillo/alperez/teoria/cap_02-sistemas_de_proyeccion/02-proyeccion_cilindrica.htm

Página Web con información acerca de la perspectiva isométrica y dimétrica.

http://es.wikipedia.org/wiki/Perspectiva_isom%C3%A9trica
Página Web con información acerca del uso de la perspectiva isométrica en los juegos.