UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

PROYECTO DE FIN DE CARRERA

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

Realizado por

MIGUEL LÓPEZ MOYA

Dirigido por

FRANCISCO JAVIER GONZÁLEZ CAÑETE

Dirigido académicamente por

NATALIA MORENO VERGARA

Departamento

LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

Málaga, Diciembre de 2015

UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

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

Presidente/a Dº/Dª. _____

Secretario/a Dº/Dª. _____

Vocal Dº/Dª. _____

para juzgar el proyecto Fin de Carrera titulado:

Desarrollo de un videojuego multijugador en perspectiva isométrica para dispositivos móviles Android con Bluetooth

realizado por Dº. Miguel López Moya,

dirigido por Dº. Francisco Javier González Cañete

y, en su caso, dirigido académicamente por Dª. Natalia Moreno Vergara

ACORDÓ POR _____OTORGAR LA CALIFICACIÓN

DE _____

Y PARA QUE CONSTE, SE EXTIENDE FIRMADA POR LOS COMPARECIENTES

DEL TRIBUNAL, LA PRESENTE DILIGENCIA.

Málaga, a ____ de _____ de 201_

El/La Presidente/a El/La Secretario/a El/La Vocal

Fdo: Fdo: Fdo:

Agradecimientos

A mi madre, por estar siempre a mi lado. A mi padre, por enseñarme que por muy duro que fuera el camino nunca hay que rendirse. A mis abuelos, Pepe y Rosario, y a mis tíos, Toño y Mari, porque sin su ayuda este proyecto no habría sido posible. A Marta, por ser un ejemplo, mi inspiración, mi apoyo más grande y por estar siempre a mi lado tanto en los buenos como en los no tan buenos momentos.

Contenido

Índice de figuras

Capítulo 1. Introducción

La industria del videojuego se ha mantenido en auge desde que Atari, en 1972, publicara el juego “Pong” [1]. Se trataba de un sencillo juego de tenis de mesa y en la figura 1-1 pude verse una pantalla de dicho videojuego. Exceptuando la crisis en el año 1983 debido al exceso de producción, el sector no ha parado de crecer año tras año. Según el estudio de EAE Business School [2], el mercado del videojuego en España movió 763 millones de euros en 2014, lo que supone un crecimiento del 31% respecto a los datos registrados en 2013. Se prevé que el mercado español de videojuegos crezca hasta los 890 millones de euros en 2018, un crecimiento anual superior al 4% anual. El tamaño del mercado mundial de videojuegos en el año 2014 es de 23.188 millones de euros, un crecimiento del 3% respecto a las cifras de 2013. Las previsiones que presenta el informe de EAE muestran que en el periodo 2014 – 2018 el mercado crecerá a unos ritmos muy interesantes. El tamaño del mercado de videojuegos en 2018 será de 26.022,78 billones de euros, lo que supone un crecimiento anual medio del 3%. En este estudio se tienen en cuenta solamente los productos para las plataformas PC y Mac y Consolas.

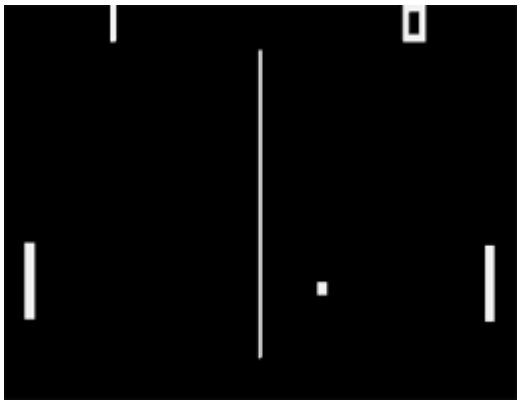

Figura 1 - 1. Pong de Atari

La industria de los videojuegos para dispositivos móviles está en pleno crecimiento gracias, en parte, a juegos free-to-play [3]. Los consumidores están invirtiendo gran cantidad de tiempo y dinero en juegos para smartphones y tablets. La empresa Newzoo [4], dedicada a la investigación dentro del mundo de los videojuegos, pronostica que para 2015, los ingresos globales producidos por juegos para dispositivos móviles eclipsarán a los ingresos de videojuegos para consolas por primera vez. Newzoo espera que el mercado del móvil genere más de 30 mil millones de dólares en todo el mundo, mientras que los juegos para consolas registrarán algo más de 26 mil millones de dólares. Según las predicciones de Newzoo, el mercado total de videojuegos en todo el mundo generará 91.950 millones de dólares en 2015.

En este proyecto se realizará una versión multijugador del videojuego clásico de los 80 para ordenadores de 8 bits, “Head over Heels”, para dispositivos móviles Android usando la tecnología Bluetooth.

En agosto de 1983 se presentó en el Reino Unido un videojuego que marcaría un punto de inflexión en la historia de los videojuegos: “Ant Attack” [5]. Lo que hacía especial a este juego desarrollado por Sandy White y distribuido por Quicksilva, era el uso de la perspectiva isométrica para representar un escenario, una ciudad, que podíamos recorrer con un chico o una chica. En la Figura 1-2 se muestra una pantalla de dicho videojuego.

Figura 1 - 2. Ant Attack

Si se es un neófito quizás puedas pensar, al ver una imagen del juego, que no tiene ningún atractivo. Sin embargo, debemos aprender a valorar la

originalidad y el empleo de nuevas técnicas en el desarrollo de software (y en cualquier otra disciplina), además de los miles de polígonos y colores utilizados. El objetivo de un juego debe ser divertir, entretener y, si es posible, enganchar al usuario. Si lo consigue sería un buen producto independientemente de que esté en blanco y negro y quepa en un disquete.

En las navidades de 1984, la compañía Ashby Computers & Graphics (A.C.G.), mas conocida como Ultimate, tras los tremendos éxitos cosechados en titulos como “Jet Pac” o “Sabre Wulf” deja boquiabierta a la industria informática con un producto que, para muchos programadores, todavía sigue siendo el cénit revolucionario: “Knight Lore”. La conocida técnica “Filmation”, empleada en su desarrollo, hace uso de la perspectiva isométrica a la vez que permite una libertad de manipulación de los objetos presentes en las estancias nunca vista hasta ese momento.

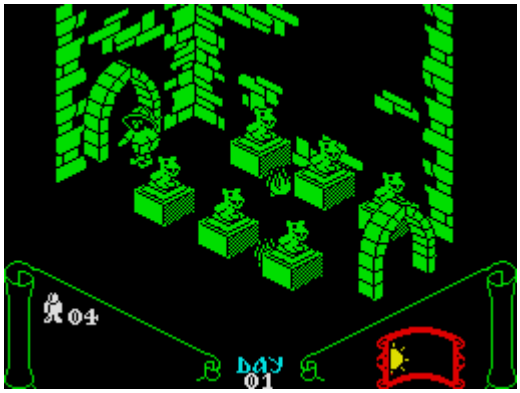

Figura 1 - 3. Knight Lore

Posteriormente, A.C.G. presentó el sistema “Filmation II” en su “Nightshade”. Podríamos considerarlo más una evolución de la técnica del “Ant Attack” que del mismo Filmation, pues se usaba la perspectiva isométrica con desplazamiento en un gran mapa. Sin embargo, no tuvo el impacto de su antecesor.

Multitud de juegos emplearon la técnica de Ultimate, entre ellos el protagonísta de este proyecto. “Head over Heels” apareció en 1987 de la mano de Ocean Software Ltd., la compañía mas poderosa en el desarrollo y distribución de videojuegos de Europa. Su programador, Jon Ritman, ya había

tenido un gran éxito con juegos como “Match Day” o “Batman”, pero este título lo conduciría, definitivamente, a la idolatría de muchos aficionados.

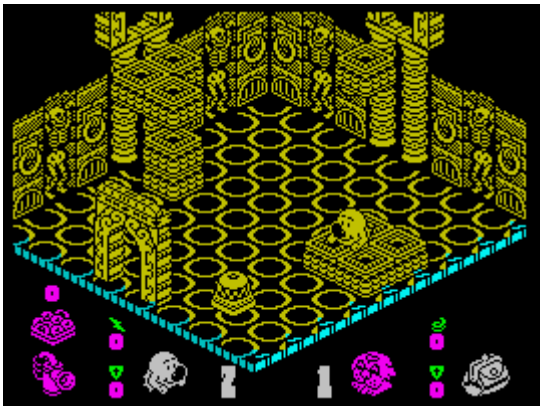

Figura 1 - 4. Head over Heels para ZX Spectrum

“Head over Heels” sigue la línea de “Batman” pero incluye una notable diferencia: el control de dos personajes simbióticos; en ocasiones por separado, en otras juntos. Esta característica realmente suma puntos de adicción para el juego. Los gráficos de Bernie Drummond superan notablemente a los de “Batman” y cada sala esta cuidada hasta el último detalle. El sonido y la música corrieron a cargo de Guy Stephens.

Estamos sin lugar a dudas ante una obra maestra y la culminación de la perspectiva isométrica de los videojuegos estructurados en salas. En el otro lado está esa maravilla de Francisco Menéndez y Juan Delcán titulada “La abadía del crímen”.

Ritman utilizó una técnica de programacion modular para desarrollar el juego, es decir, estructuró en partes distintas facetas del programa tales como el control del teclado, la representación en pantalla, etc. Esto le permitió su conversión a numerosos sitemas. Además, se hizo un aprovechamiento máximo de la memoria. Por ejemplo, el mapa con mas de 300 salas ocupa tan sólo 5 KB.

En “Head over Heels” se controlan dos personajes con habilidades bien diferentes. Head carece de pies y debe reptar para moverse, haciéndolo lentamente. Sin embargo, posee una especie de alas en sus brazos que le

permiten dar grandes saltos y planear en el aire. Heels, por su parte, tiene un salto muy disccreto y no planea al carecer de brazos, pero tiene unos enormes pies que le confieren gran velocidad de movimiento. Cuando se unen, el nuevo personaje adquiere las habilidades de ambos. Además Head puede usar una bocina para lanzar rosquillas a sus enemigos y dejarlos paralizados, mientras que Heels es capaz de usar un bolso para almacenar temporalmente un objeto.

Figura 1 - 5. Head over Heels para Atari ST

En el juego original se comienza con los protagonistas encarcelados en los calabozos del castillo de Blacktooth, separados y despojados de sus objetos. El primer objetivo debe ser escapar y encontrar tanto la bocina como el bolso, para después continuar avanzando hasta llegar al mercado que rodea al castillo. Allí, Head y Heels podrán volverse a unir y deberan tomar rumbo hasta la base que hay en la luna de Blacktooth. Dicha base alberga el centro de teletransporte del Imperio desde donde se pueden viajar a cualquiera de los cuatro planetas subyugados: Egyptus, Penitentiary, Safari y El Mundo del Libro.

El objetivo del juego consiste en encontrar y recuperar las cinco coronas perdidas. Hay una en cada planeta exterior y la quinta se encuentra en el mismo Blacktooth. Una vez conseguidas, Head y Heels deberán buscar el modo de volver a su planeta natal, volver a Freedom.

En este proyecto se desarrollara unprototipo de motor del juego. Con un par de estancias y las habilidades básicas de cada personaje sin contar Head con la bocina lanza rosquillas ni Heels con su bolso. Esta versión sera

exclusivamente multijugador (en concreto para dos jugadores). Cada personaje será controlado por un jugador con un dispositivo móvil. Cuando los dos personajes se unan accionando un botón, serán controlados por el jugador que inicialmente controlaba a Head, volviendo a ser controlado cada personaje por un jugador cuando Head y Heels se vuelvan a separar pulsando el mismo botón que se usa para unirlos. También se ha incluido un botón para el salto y una cruceta para controlar el movimiento de los personajes en cuatro direcciones.

Heels será controlado por el dispositivo en modo cliente, mientras que Head y Head over Heels (los dos protagonistas unidos) los manejará el dispositivo en modo servidor.

La música y sonidos incluidos se han obtenido del remake para PC del videojuego realizado por José Rodríguez Santos y Santiago Acha Jiménez [6]. Se han usado los gráficos realizados por Davit Masiá en 2003 [7] para los personajes, escenarios y elementos que aparecen en ellos.

Para los menús botones y marco del juego se han diseñado unos gráficos más modernos y acordes para usar en los dispositivos móviles. En el marco del juego se representa un inventario en el que aparecerán las vidas de cada personaje junto a un dibujo que representa a cada uno de los dos protagonistas. Este dibujo está a color o en tonos de gris según esté siendo controlado por el jugador o no respectivamente. Cuando los dos personajes protagonistas se unen apareceran los dos en color. El inventario está dividido en plataformas, en unas aparecen Head y Heels junto a sus vidas restantes y, las demás, están reservadas para situar los distintos artefactos que podrán ir consiguiendo durante el juego, que otorgarán habilidades especiales a los protagonistas del juego y que su implementación se ha dejado para expansiones futuras del juego.

En esta versión se han añadido enemigos que se moverán aleatoriamente en las cuatro direcciones por una estancia, no pudiendo pasar de una estancia a otra, y que sí contactan con alguno de nuestros avatares le quitarán una vida. Además, hay elementos estáticos como estacas que no

podremos tocar si no queremos que nos ocurra lo mismo. Los personajes controlados podrán pasar de una estancia a otra pasando por las puertas que comunican a éstas y el juego concluirá bien cuando las vidas de uno o dos de los personajes lleguen a cero (Juego Terminado), o consigamos la corona situada en una de las estancias (Mundo Liberado). Todo esto se analizará con mas detalle en el apartado *Diseño de la Aplicación*.

El documento presente esta estructurado en varios capítulos en los que se explica con detalle todas las partes del proyecto:

1. Introducción: es el presente capítulo, en el que se hace una descripción y justificación del proyecto, asi como de los objetivos a conseguir.
2. Tecnologías y herramientas empleadas: en este capítulo se detallan las herramientas software y hardware utilizadas.
3. Descripción de la aplicación: exposición en profundidad de la lógica de la aplicación, aclarando el funcionamiento de los menús y pantallas, las diferentes opciones y la lógica del juego.
4. Diseño de la aplicación: explicación de cómo se ha implementado la aplicación para lograr la funcionalidad deseada.
5. Plan de pruebas: exposición de la realización y los resultados de los testeos de la aplicación.
6. Conclusiones y líneas futuras: balance del proyecto y estudio de posibles mejoras o ampliaciones.

Capítulo 2. Tecnologías y herramientas empleadas

2.1. Introducción

En este capítulo se detallarán las tecnologías usadas para crear este proyecto, así como las herramientas software que se han utilizado y la labor para las que fueron empleadas.

Todas las herramientas que se han usado en este proyecto son de libre uso y distribución, siguiendo la filosofía general de búsqueda de la estandarización y mínimo coste global.

2.2. Entorno de desarrollo

Se deben aplicar varias plataformas que proporcionan distintas funciones para conseguir un entorno adecuado para la realización de la aplicación.

2.2.1. Java como lenguaje

Java es un lenguaje y plataforma que fue creada por la compañía Sun Microsystems y actualmente es propiedad de Oracle.

Java se creó como una herramienta de programación para ser usada en un proyecto de set-top-box (nombre con el que se conoce el dispositivo encargado de la recepción y decodificación de señal de televisión analógica o digital) en una pequeña operación denominada *The Green Project* en el año 1991. Java tuvo unos comienzos dubitativos, hasta 1994, cuando en Sun Microsystems reconocieron que la mejor forma de enfocar su desarrollo futuro

era orientado a Internet. A partir de entonces, Java floreció como uno de los lenguajes con más implantación, gozando actualmente de una excelente salud y de infinidad de desarrolladores, herramientas y una base de código muy extensa. Su sintaxis proviene parcialmente de C y C++, con el objetivo de acortar la curva de aprendizaje para los desarrolladores de estos lenguajes.

Las características que hacen de Java un lenguaje tan popular son:

- Es independiente de la plataforma. Es decir, una aplicación, una vez escrita, puede ser ejecutada en cualquier dispositivo, tal como reza el axioma de Java *"write once, run anywhere".* Esto es así porque cuando se compila una aplicación Java, se genera un código intermedio llamado *bytecode*, que ha de ser interpretado más tarde por la Máquina Virual Java (JVM), ésta sí dependiente del sistema operativo, presente en la gran mayoría de los sistemas.
- Seguridad: Esto se debe en parte a la generación de los *bytecodes*, ya que en Java, por su propio diseño, no se permite acceder a la máquina físicamente más que a través de librerías que sí dependen del sistema operativo y que no son estándares de Java.

Es software libre y los entornos de desarrollo integrados (IDE) más conocidos para programar en Java, como NetBeans y Eclipse, son gratuitos.

2.2.2. Java como plataforma

Java es una plataforma para la ejecución de programas. En contraste con otras plataformas. A diferencia de otras plataformas que están compuestas de procesadores físicos y de sistemas operativos, la plataforma Java consiste en una máquina virtual y un entorno asociado.

Existen distintas ediciones de la plataforma. La Java Standard Edition (Java SE) fue creada para el desarrollo de aplicaciones independientes que se ejecutan en ordenadores. Actualmente se utiliza además para el desarrollo de applets, que son unos programas que funcionan en contexto de navegador web.

En este proyecto se ha usado una edición especial de Java creada por Google para realizar aplicaciones que se ejecutan sobre sus dispositivos con el sistema operativo Android. Esta edición es conocida como la plataforma Android. Sobre ella, podríamos decir que está compuesta básicamente de bibliotecas del núcleo Java (en parte basadas en Java SE) y una máquina virtual llamada Dalvik. Este software colectivo se ejecuta sobre un núcleo de Linux modificado.

2.2.3. JDK (Java Development Kit)

Se necesita el Kit de desarrollo Java SE (JDK) para el desarrollo de programas Java. Éste contiene las herramientas precisas, incluido el compilador Java y un Entorno de ejecución Java (JRE – Java Runtime Environment) privado.

El instalador del JDK crea un directorio local con archivos que proporcionan información sobre el JDK y el código fuente de la biblioteca de clases estándar, así como varios subdirectorios:

- bin: Contiene herramientas del JDK organizadas, entre las que se incluye la de compilación de Java.
- jre: Posee en su interior la copia privada del JRE del JDK, la cual permite ejecutar programas Java sin tener que descargar e instalar el JRE público.
- lib: Alberga ficheros de biblioteca utilizados por las herramientas del JDK. Las herramientas del JDK se ejecutan en la línea de comandos mediante sus argumentos.

2.2.4. Kit de Android SDK

Es un conjunto de herramientas basadas en la línea de comandos. Es necesaria para crear, compilar y desarrollar proyectos Android. Cuenta además con un gestor de Dispositivos de Android Virtual (AVD – Android Virtual Device) y de Conjuntos de Herramientas de Desarrollo (SDK – Software Development

Kit), una herramienta que utilizará el emulador para instalar componentes SDK y originar dispositivos virtuales.

El gestor de SDK y AVD pueden descargar e instalar varios tipos de componentes tales como:

- Plataformas Android: Para cada versión oficial de Android existe una plataforma que debe ser incluida en las librerías de ejecución del SDK, una imagen del sistema que emplea el emulador y cualquier herramienta especialmente generada para dicha versión.
- Complementos para SDK: Complementos o librerías externas y herramientas que no pertenecen a ninguna plataforma específica.
- Controlador USB para Windows: Se necesita para depurar y ejecutar la aplicación en un dispositivo físico si se trabaja con el sistema operativo Windows.
- Ejemplos: Para cada plataforma existen una serie de ejemplos desarrollados para ella.
- Documentación: Copia local de la documentación de la última API del *framework* de Android.

2.2.5. Eclipse

Eclipse generalmente se utiliza en los desarrollos con Java y es un Entorno de Desarrollo Integrado (IDE – Integrated Development Enviroment) realizado en código abierto que se puede usar para generar aplicaciones con diferentes lenguajes de programación. Es recomendado por Google para el desarrollo de aplicaciones Android.

Esta plataforma fue desarrollada originalmente por IBM y conforma un entorno de desarrollo maduro, con una gran base de usuarios que ayudan a crear manuales, tutoriales y *plugins* para facilitar la programación. En la Figura 2-1 se observa el programa iniciado con parte de un código.

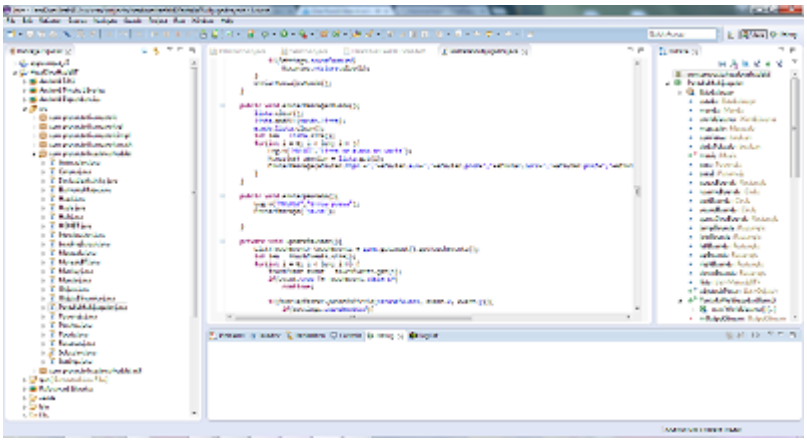

Figura 2 - 1. Eclipse en pleno funcionamiento

2.2.6. Complemento ADT para Eclipse

El complemento ADT recopila todas las herramientas que se encuentran en el kit Android SDK que ayuda a mejorar las capacidades de Eclipse. Se trata de una IDE basada en una arquitectura de complementos que se utiliza para ampliar sus capacidades gracias a desarrollos de terceros. Esta combinación integra en el entorno de trabajo de eclipse todas las herramientas de Android SDK de forma transparente.

2.2.7. Framework de Mario Zechner

En el capítulo 5 del libro Desarrollo de Juegos de Android de Mario Zechner [8] se encuentra la implementación de las clases y las interfaces de un *framework* realizado para juegos para codificar toda la mecánica de un juego en 2D de Android. Se ha decidido usar este *framework* para que ayudase al desarrollo del motor del juego. Esta elección se basó en que la aplicación no requiere una gran carga gráfica y que se busca simplicidad y trabajar con elementos bien documentados para poder hacer modificaciones futuras de forma más sencilla si se estimase oportuno.

Cabe destacar que Mario Zechner es además el creador de LibGDX, *framework* para el desarrollo de videojuegos multiplataforma, que soporta en la

actualidad Windows, Linux, Mac OS X, Android, iOS y HTML5 que es muy usado hoy en día.

2.2.8. Clase DeviceListActivity de AOSP

AOSP quiere decir Android Open Source Project y es un proyecto que trabaja para el manejo y desarrollo de Android desde el punto de vista de código abierto, libre de lo que Google encierra dentro del mismo código de Android y de todo lo referente a fabricantes y sus especificaciones.

En 2009 se dieron los procesos necesarios para poder incluir contribuciones externas a versiones oficiales de Android, siendo así la primera *release* de estas características la 2.0 (primer Eclair), que vio la luz a finales de ese mismo año.

Se ha extraído la clase DeviceListActivity.java de esa *release.* Esta ayudará al emparejamiento de dispositivos con Bluetooth.

2.3. Bluetooth

Bluetooth es la solución que se ha elegido en el presente proyecto para dar soporte multijugador a la aplicación. Se trata de una tecnología de radio de corto alcance que permite conectividad inalámbrica entre dispositivos remotos. Se diseñó pensando básicamente en tres objetivos: mínimo consumo, bajo precio y pequeño tamaño.

Opera en la banda de radio libre ISM (Industrial, Scientific and Medical) a 2’4 GHz. El rango de alcance de Bluetooth depende de la potencia empleada en la transmisión. Su máxima transmisión de datos es de 1 Mbps. La mayor parte de los dispositivos que usan Bluetooth transmiten con una potencia nominal de salida de 0dBm, lo que le permite un alcance de unos 10 metros en ambiente libre de obstáculos.

Para el acceso a Bluetooth, Android tiene su propia API en el paquete Android.bluetooth y requiere el emparejamiento previo de dos dispositivos antes de intercambiar datos.

2.4. Edición de imágenes

Ha sido necesario recurrir a un software específico para la edición de imágenes que aparecen en el videojuego.

GIMP (GNU Image Manipulation Program) es un programa de edición de imágenes digitales en forma de mapa de bits [9], tanto dibujos como fotografías. Es un programa libre y gratuito. Forma parte del proyecto GNU y está disponible bajo la *Licencia pública general de GNU* y GNU *Lesser General Public License.*

Es el programa de manipulación de gráficos disponible en más sistemas operativos (Unix, GNU/Linux, FreeBSD, Solaris, Microsoft Windows y Mac OS X, entre otros).

GIMP tiene herramientas que se utilizan para el retoque y edición de imágenes, dibujo de formas libres, cambiar el tamaño, recortar, hacer fotomontajes, convertir diferentes formatos de imagen, y otras tareas más especializadas. Se pueden también crear imágenes animadas en formato GIF e imágenes animadas en formato MPEG usando plugins de animación.

Los desarrolladores y encargados de mantener GIMP se esfuerzan en mantener y desarrollar una aplicación gráfica de software libre, de alta calidad para la edición y creación de imágenes originales, de fotografías, de iconos, de elementos gráficos de las páginas web y otros elementos artísticos de interfaz de usuario.

Los usos empleados en la elaboración de este proyecto incluyen la creación de gráficos, el cambio de tamaño, recorte y modificación de imágenes, la modificación de colores, combinación de imágenes usando un paradigma de capas, eliminación o alteración de elementos no deseados o la conversión

entre distintos formatos de imagen. Se puede observar una fotografía del programa en ejecución en la Figura 2-2.

Figura 2 - 2. GIMP en funcionamiento

2.5. Versión del juego de Jorge Rodríguez Santos

Para conocer a fondo el videojuego Head over Heels, además de las referencias vistas en el capítulo 1, se ha interactuado con una versión del juego original. Como no se disponía de ninguna videoconsola ni cartucho con el juego, la opción ha sido usar la versión del juego realizada por Jorge Rodríguez Santos y Santiago Acha Jiménez para PC [10] descargada desde su página web www.headoverheels2.com.

Esta versión aprovecha el motor isométrico *Isomot* en su versión 0.3 realizado por Ignacio Pérez Gil. Se caracteriza por el diseño gráfico *pixel-art* de Davit Masiá y está basada en la versión para Spectrum del videojuego original y su desarrollo es un calco de este. La resolución del juego es de 640 por 480 píxeles. El motor isométrico proyecta sombras y en las salas más grandes hay una cámara que sigue al personaje. Está disponible en veinte idiomas: afrikaáns, alemán, bable, catalán, croata, danés, eslovaco, español (España y América), finlandés, francés, gallego, húngaro, inglés, italiano, latín, polaco, portugués (Portugal y Brasil), ruso, serbio, valenciano y vascuence. Otra

característica a destacar es la posibilidad de grabar partida. En la Figura 2-3 se puede observar una imagen del videojuego.

Figura 2 - 3. Head over Heels de José Rodríguez Santos

2.6. Modelado UML

Resulta conveniente realizar una planificación previa que permita visualizar el esqueleto del sistema a la hora de afrontar el diseño de un sistema complejo. Con este fin se creó el Lenguaje Unificado de Modelado (UML – Unified Modeling Language) que permite modelar y representar la información con la que se trabaja en la fase de análisis y diseño. Es un lenguaje con notación expresiva que se usa para visualizar, construir y documentar un sistema.

UML es un lenguaje modelado, entendiendo como modelo una simplificación de la realidad. El objetivo del modelado de un sistema es capturar las partes esenciales del mismo. Para facilitarlo, se realiza una abstracción y se representa en una notación gráfica, lo que se conoce como modelado visual. El modelado visual permite manejar la complejidad de los

sistemas a diseñar o analizar y es independiente del lenguaje de implementación usado.

Proporciona como lenguaje unas reglas para permitir la comunicación y un vocabulario y como método formal de modelado permite un rigor mayor en la especificación y posibilita realizar una validación y verificación del modelo realizado.

Éstas son sus funciones:

- Visualizar. Permite representar de forma gráfica un sistema de manera que otra persona lo pueda entender.
- Especificar. Posibilita precisar cuáles son las características del sistema antes de ser implementado.
- Construir. A partir de modelos especificados se pueden construir los sistemas diseñados.
- Documentar. Los propios elementos gráficos se pueden utilizar como documentación del sistema desarrollado y pueden servir para su futura revisión.

Las tres clases de bloques de construcción que componen un modelo UML son:

- Elementos: son abstracciones de cosas reales o ficticias.
- Relaciones: se usan para ligar elementos entre sí.
- Diagramas: son colecciones de elementos con sus relaciones.

Por lo tanto, en un diagrama se tiene la representación gráfica de un conjunto de elementos con sus relaciones. UML ofrece una gran variedad de diagramas para visualizar el sistema desde distintas perspectivas con el fin de poder representar el sistema correctamente.

Los diagramas que se utilizan más adelante en el diseño de la aplicación son dos: de clases y de estados.

2.6.1. Diagrama de clases

Los diagramas de clases son los más utilizados en el modelado de sistemas orientados a objetos. Muestran un conjunto de clases, interfaces y colaboraciones, así como sus relaciones. Abarcan la vista de diseño estática de un sistema, como se puede ver en el ejemplo de la Figura 2-4.

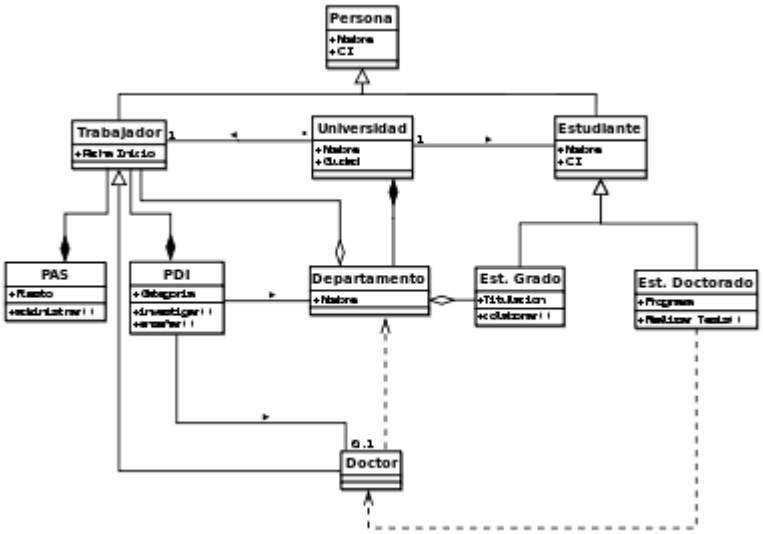

Figura 2 - 4. Diagrama de clases de personal universitario

2.6.2. Diagrama de estados

Son usados para modelar los aspectos dinámicos de un sistema. Se pueden usar para modelar una clase, un caso de uso o un sistema completo. Pueden servir para modelar la vida de un objeto, como se ve en la Figura 2-5.

La mayor parte de las veces se usan para modelar el comportamiento de objetos reactivos. Estos objetos son aquellos para los que la mejor forma de caracterizar su comportamiento es señalar cuáles son sus respuestas a los eventos lanzados desde fuera de sus contextos. Los objetos reactivos tienen un ciclo de vida bien definido, cuyo comportamiento se ve afectado por su pasado.

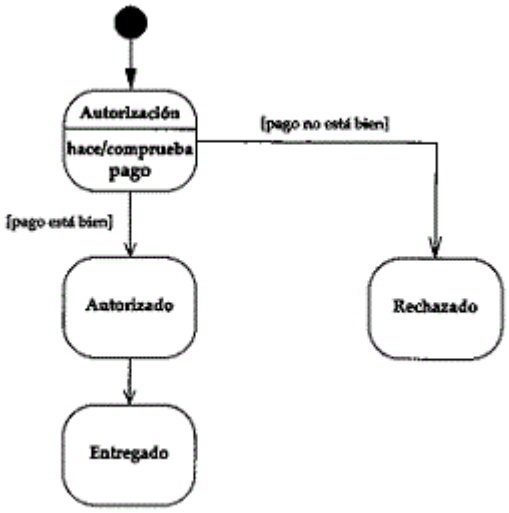

Figura 2 - 5. Diagrama de estados de autorización de pagos

Capítulo 3. Descripción de la aplicación

3.1. Introducción

Este capítulo, en cierta medida, es un manual de usuario detallado donde se explican todos los aspectos de la aplicación. Se explicará el desarrollo y las funciones del juego desde la perspectiva de usuario, sin entrar en detalles técnicos.

Se empezará con el resultado de la instalación del juego en un dispositivo. Posteriormente se realizará un recorrido por todas las pantallas de la aplicación, explicando las opciones, estados e imágenes que el usuario podrá encontrarse durante la ejecución del programa.

Después de presentar los menús y pantallas, se explicará en una sección con mayor detalle la descripción del mismo y la forma de proceder para jugar.

En el siguiente apartado del capítulo se analizaran los elementos que integran el juego tales como los elementos básicos que se muestran en cada pantalla. Por último, se dedicará una sección para el apartado sonoro en el que se indicarán los sonidos y cuando son utilizados.

3.2. Instalación

Para la instalación del videojuego será necesario un dispositivo con sistema operativo Android. Existen solamente dos formas de hacer la instalación, ya que no está subido a ninguna plataforma de distribución de aplicaciones como Aptoide o Google Play. Estas formas son:

- Mediante el programa Eclipse, con la conexión del dispositivo directamente a un ordenador que posea el código y ejecutándolo en dicho dispositivo.
- Obteniendo el archivo apk y ejecutándolo. Anteriormente el dispositivo debe haber sido configurado para aceptar la instalación de aplicaciones de fuentes desconocidas.

Después de realizar la instalación, si todo ha ocurrido de manera correcta, se mostrará en la pantalla del dispositivo un icono con el logotipo del juego junto al nombre HeadOverHeelsBT. Se puede ver el resultado en un dispositivo Samsung Galaxy Tab 2 7.0 en la Figura 3-1. Todas las capturas de pantalla de la aplicación ejecutándose que aparecen en este documento son obtenidas de ese mismo terminal.

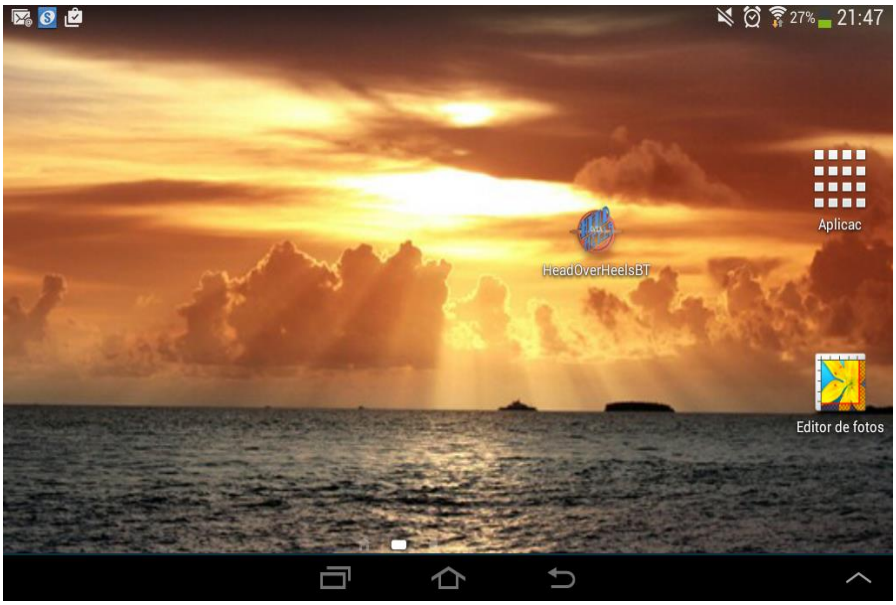

Figura 3 - 1. Icono del juego en el dispositivo.

3.3. Pantallas y menús

3.3.1. Inicio

Al iniciar el juego, la primera pantalla que aparece es una imagen del logotipo del juego sobre un fondo de cielo y nubes. Además se mostrará en la parte inferior central un botón con la palabra “Jugar”. Al pulsar dicho botón se

accederá a la siguiente pantalla del juego. En la Figura 3-2 se puede ver una captura de dicha pantalla.

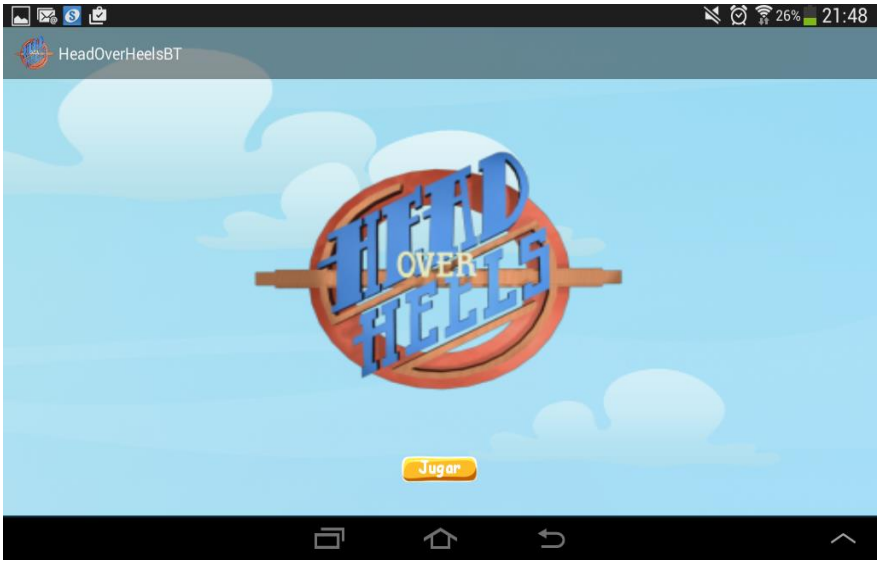

Figura 3 - 2. Pantalla de introducción de la aplicación.

3.3.2. Menú de selección de personaje

Al acceder a esta pantalla, lo primero que hace la aplicación es detectar si el dispositivo dispone de Bluetooth. Si no dispone, lo notifica con una venta emergente.

Figura 3 - 3. Ventana de activación del Bluetooth.

En caso de disponer de Bluetooth, pero estar éste desactivado, en la pantalla aparece una ventana emergente que da la opción de activarlo. Podemos observar este comportamiento en la Figura 3-3.

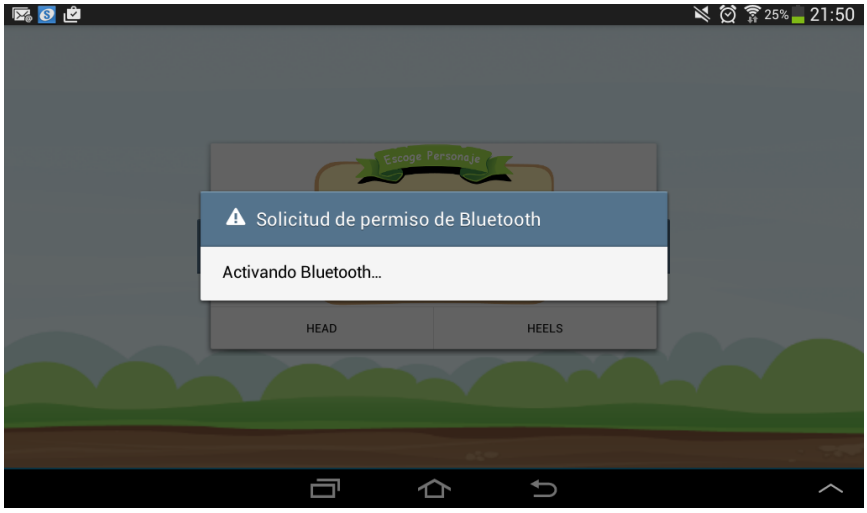

Figura 3 - 4. Mensaje mientras se activa el Bluetooth.

Si se permite la activación del Bluetooth, la aplicación notificará cuándo se esté realizando esta operación. Se puede apreciar en la Figura 3-4. Tras ser realizada, las consecuencias de esta acción se notificarán en la pantalla del dispositivo.

Con el Bluetooth activo, la aplicación dará dos opciones en una ventana emergente para seleccionar el personaje que se controlará durante el inicio del juego: HEAD (modo servidor de la partida) o HEELS (modo cliente). Se puede apreciar en la figura 3-5.

En caso de pulsar el botón HEAD, la pantalla no reflejará nada y el dispositivo se pondrá a la espera de ser enlazado con el dispositivo que controlará a HEELS. El dispositivo se mantendrá en esa espera hasta que notifique el intento de enlace de un dispositivo modo cliente (HEELS) con una solicitud de sincronización, de la que se hablará más adelante.

En caso de pulsar el botón HEELS, la aplicación dará la opción de buscar dispositivos o seleccionar algún dispositivo emparejado anteriormente.

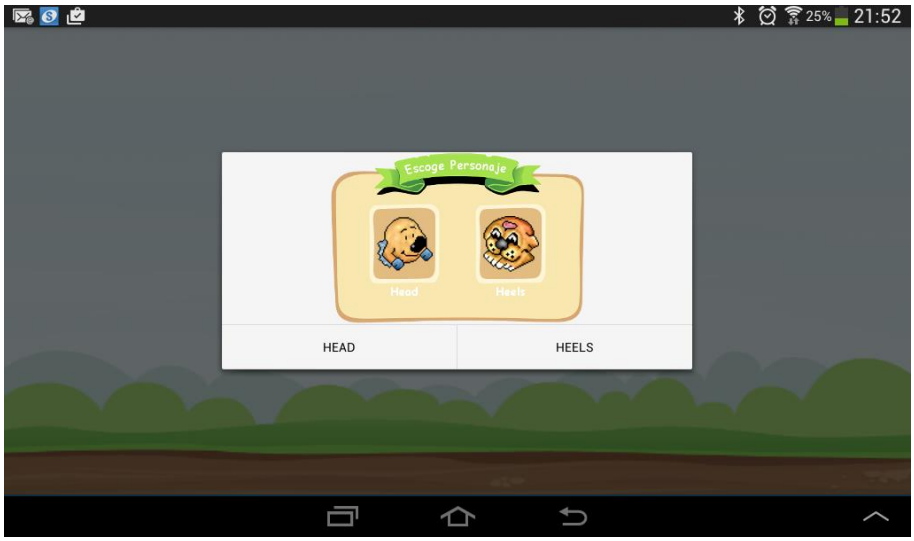

Figura 3 - 5. Pantalla de selección de personaje.

Si el dispositivo que actúa como servidor está dentro del área de Bluetooth, aparecerá en una lista junto a otros dispositivos con Bluetooth activo dentro de esa área. Si se selecciona el dispositivo que actúa como servidor de la partida dentro de esta lista y se aceptan las solicitudes de sincronización, se añadirá ese dispositivo a la lista de dispositivos emparejados.

Figura 3 - 6. Lista de dispositivos para conectarse.

Se puede ver en la figura 3-6 como se da la oportunidad de enlazar con distintos dispositivos previamente emparejados.

La solicitud de sincronización aparece en ambos dispositivos y, aunque lo normal es poder aceptarla o no mediante una ventana que aparecerá sin salir de la aplicación, en determinados dispositivos la solicitud aparece en otra área de éste, por lo que hay que aparcar momentáneamente la aplicación y, tras aceptarla, volver a ella. La Figura 3-7 muestra el caso normal, poder aceptar la sincronización sin necesidad de aparcar la aplicación.

Tras ser aceptada por ambos dispositivos, el dispositivo que controlará a HEELS y el controlará a HEAD irán a la siguiente pantalla.

Figura 3 - 7. Ventana para aceptar la solicitud de vinculación con otro dispositivo.

3.4. Pantallas durante la partida

Durante una partida pueden aparecer distintas pantallas. A continuación se detallaran las diferentes pantallas que se pueden encontrar.

3.4.1. Preparado

Una vez se inicia una partida, tanto en modo HEAD como HEELS, la primera pantalla que aparece contendrá el mensaje “¿Preparado? Pulse la pantalla para continuar”, y por detrás la estructura de la habitación en la que comenzará cada personaje el juego con todos los elementos que se situarán en

esta. El juego permanecerá parado hasta que uno de los jugadores pulse cualquier lugar de la pantalla y posteriormente dará comienzo la partida. Se puede ver esta pantalla en la Figura 3-8.

Figura 3 - 8. Pantalla Preparado.

3.4.2. Mundo Liberado

En el caso de que uno de los jugadores consiga la corona situada en una de las habitaciones en las que ocurre la partida aparecerá la pantalla Mundo Liberado. Contiene un letrero con el texto “Mundo Liberado” y un botón con el texto “Continuar” para salir y reiniciar la partida. Al reiniciar la partida los dos personajes comenzarán de nuevo con tres vidas y todo los demás elementos de la partida aparecerán como la primera vez que se inició. En la Figura 3-9 se muestra esta pantalla.

3.4.3. Juego Terminado

En caso de que las vidas de alguno o los dos personajes se agoten, aparecerá la pantalla de Juego Terminado, con un letrero indicativo con el texto de “Juego Terminado” y un botón con el texto “Salir” para reiniciar la partida al pulsarlo. Si se pulsa el botón para salir se reiniciará la partida. Ocurrirá lo

mismo al reiniciar la partida que en lo explicado respecto a la pantalla de Mundo Liberado. Esta pantalla se muestra en la Figura 3-10.

Figura 3 - 9. Pantalla de Mundo Liberado.

Figura 3 - 10. Pantalla de Juego Terminado.

3.4.4. Pantalla de pausa

Durante la partida, en la parte superior derecha de la pantalla, aparecerá un botón amarillo con el símbolo de pausa. Al pulsarlo aparecerá la pantalla de

pausa. En esta se mostrará un menú con distintas opciones y un letrero con el texto “Pausa”.

Cuando un jugador pulsa el botón de pausa en su dispositivo, los dos jugadores entrarán en el menú de pausa.

Las opciones de este menú se representan con tres botones de color amarillo. El primero para volver a la partida está representado por un botón con el típico símbolo de *play*. El segundo se utiliza para salir de la partida y es representado con un botón con un símbolo de una flecha circular. El tercero es para activar o desactivar el sonido según este éste desactivado o activado respectivamente. Si el sonido está activo este botón será representado por un símbolo de un altavoz tachado, mientras que si está inactivo, será presentado por el símbolo del altavoz sin tachar.

Todas estas opciones son accesibles con sus respectivos botones como aparecen en la Figura 3-11.

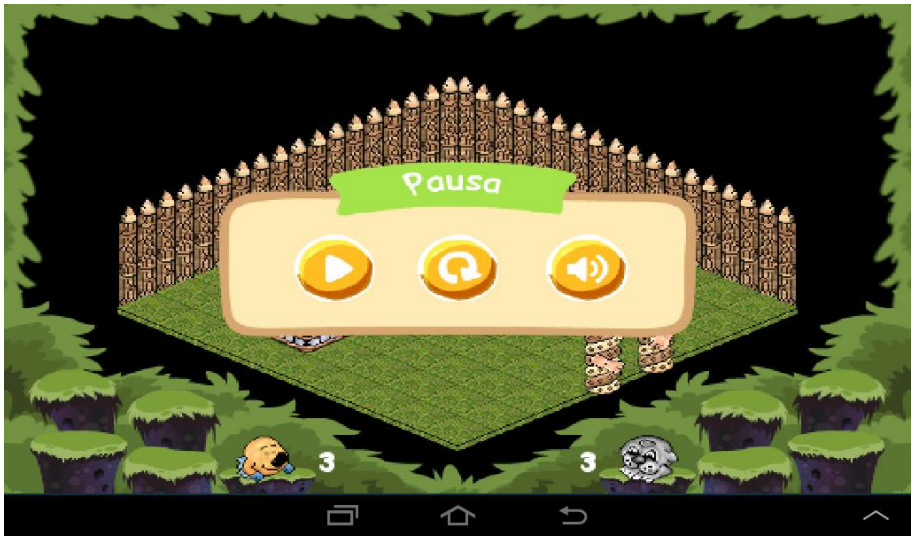

Figura 3 - 11. Menú de Pausa.

Si uno de los jugadores pulsa el botón para volver a la partida los dos jugadores volverán a la partida.

Si se pulsa el botón para salir de la partida aparecerá en la pantalla un mensaje para que el mensaje confirme si desea abandonar la partida. Contendrá el texto “¿Seguro qué quieres abandonar?”. También aparecerán un botón rojo para responder negativamente a esta pregunta, y uno verde para responder afirmativamente. En la Figura 3-12 se podrá observar dicho mensaje de confirmación.

Figura 3 - 12. Mensaje de confirmación para salir del juego.

3.5. Instrucciones de juego

El videojuego se ha implementado para juego multijugador exclusivamente. Podrán jugar dos jugadores simultáneamente. Un jugador controlara al personaje Heels desde un dispositivo, mientras que otro jugador controlará al personaje Head y Head over Heels cuando estén los dos personajes unidos.

Al comienzo de la partida un jugador controlará a Head y otro a Heels y cada uno de ellos aparecerá en una estancia distinta.

Para superar el juego se debe coger la corona para liberar el mundo. Cada personaje contará de inicio con tres vidas que pueden perder si colisionan con los pinchos situados en una habitación o con el mono situado en

otra. Al perder una vida un personaje aparecerá una animación que simula la desintegración del personaje. Los pinchos son elementos estáticos mientras que el mono se desplaza en cuatro direcciones libremente por la habitación en la que esté sin poder atravesar la puerta para acceder a la habitación colindante.

Si uno de los personajes se queda sin vidas el juego terminará y aparecerá la pantalla de juego terminado. Cuando los dos personajes estén unidos perderán una vida cada uno si colisionan con los pinchos o con el mono. Los pinchos pueden ser esquivados evitándolos por uno de sus lados o bien saltándolos.

Cada vez que un personaje pierde una vida y aún posee alguna, este volverá a aparecer en:

- Si es la primera vez que pierde una vida y no ha pasado de una habitación a otra, el personaje reaparecerá en la posición que tenía al inicio de la partida.
- Si los dos personajes están unidos y pierden una vida, volverán a ser dibujados unidos en la posición donde se unieron por última vez.
- Si el personaje ha pasado de una estancia a otra, volverá a ser dibujado en la estancia en la que murió, junto a la puerta. Ocurrirá de la misma forma si atravesaron la puerta los dos personajes unidos.

Tanto Head como Heels poseen habilidades bien diferenciadas como son:

- Head salta más alto pero anda más despacio que Heels.
- Heels es el doble de rápido que Head pero su salto es la mitad de alto.
- Al unirse los dos personajes, el personaje resultante, llamado Head over Heels, poseerá el salto de Head y la velocidad de Heels.

Tanto Head, como Heels, como los dos personajes juntos, podrán pasar de una estancia a otra atravesando las puertas situadas en cada una de estas. El personaje no podrá moverse fuera de los límites de las habitaciones a no ser que pasen de una a otra habitación por las puertas. Los límites son las paredes. Las paredes de la parte oeste y norte aparecen dibujadas en la pantalla de juego. Mientras que las paredes sur y este no están dibujadas. En la Figura 3-13 se puede observar una pantalla durante la ejecución de la partida.

Cuando los dos personajes estén en la misma estancia aparecerá la misma pantalla en los dispositivos de los dos jugadores. Cuando esto no ocurra, en la pantalla de cada dispositivo aparecerá la estancia en la que se encuentre el jugador controlado por el usuario de ese dispositivo.

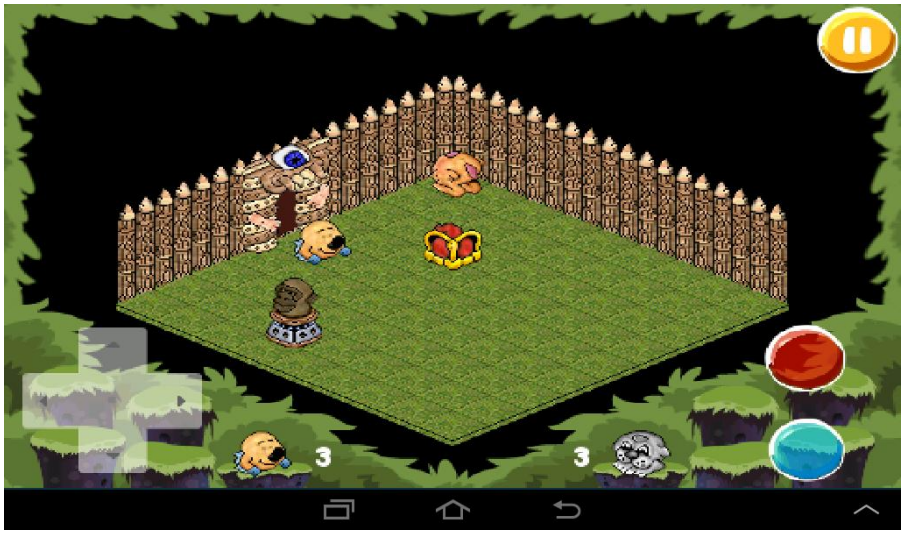

Figura 3 - 13. Imagen de la pantalla del juego durante la partida.

3.6. Controles del juego

Durante la partida, se dibujarán en la pantalla los controles del juego. Estos controles consisten en una cruceta direccional que representa las cuatro direcciones en las que pueden moverse los personajes, dos botones de acción (uno rojo y otro azul claro) y un botón de pausa. Los botones de acción y la cruceta direccional son semitransparentes para dejar ver lo que se dibuja en

pantalla tras ellos. Al pulsar el botón de pausa se pasará al menú de pausa, cuyo funcionamiento se ha explicado anteriormente.

3.6.1. Cruceta de dirección

Esta cruceta aparece en la parte inferior izquierda de la pantalla. Al pulsar el usuario la cruceta en una de las cuatros direcciones, el personaje controlado se moverá en dicha dirección, dejando de hacerlo cuando se levante el dedo de la pantalla. Si se está pulsando la cruceta mientras un personaje está saltando o cayendo esté se moverá en la dirección correspondiente hasta que deje de ser pulsada. En la Figura 3-13 se puede ver una imagen de dicha cruceta.

3.6.2. Botones de acción

En la parte inferior derecha aparecen dos botones. Por un lado, un botón rojo que al pulsarlo hará que el personaje controlado salte. Si se pulsa el botón rojo cuando el personaje está saltando no ocurrirá nada.

Por otro lado, un botón de color azul claro que servirá para juntar o separar a los dos personajes. El botón azul solo funcionará para el usuario que controle a Head o a los dos personajes juntos. Lo hará de la siguiente forma:

- Si se pulsa el botón cuando los dos personajes están separados y Head se encuentra sobre Heels, los dos personajes se unirán y el personaje Head over Heels pasará a ser controlado por el usuario que controlaba inicialmente a Head. El jugador dos permanecerá a la espera sin controlar a ningún personaje.
- Si el botón se pulsa cuando los dos personajes estén juntos y no estén saltando, los dos personajes se separarán. Esto hará que el jugador que controlaba a los dos pase a controlar a Head, mientras el otro controlará a Heels.
- Si se pulsa el botón cuando Head y Heels estén separados y Head no se encuentre sobre Heels no ocurrirá nada.

- Si se pulsa el botón cuando los dos personajes estén juntos y estén saltando no ocurrirá nada.

En la Figura 3-13 se pueden ver los botones de acción.

3.7. Marco de pantalla durante la partida

Mientras transcurre la partida aparece un marco que bordea toda la pantalla. En la parte de inferior de este marco aparecerán unas plataformas simétricas a un lado y otro de la pantalla. Estas plataformas se han implementado a modo de inventario en posibles versiones futuras para que aparezcan distintos ítems que potenciaran las habilidades o darán unas nuevas a los personajes.

En esta versión solamente aparecerán los iconos representativos de cada personaje junto al número de vidas que posee cada uno de ellos. Estos iconos se sitúan en las dos plataformas más céntricas de la pantalla. En la de la izquierda se representa a Head y sus vidas, mientras que en la de la derecha se representa a Heels y sus vidas. En la Figura 3-13 Se puede observar una imagen del marco de pantalla.

Los iconos de los personajes pueden aparecer en color o en tonos grises. El icono de un personaje aparecerá en color en la pantalla de un dispositivo cuando esté siendo controlado por el usuario de dicho dispositivo. De lo contrario, el icono del personaje aparecerá en tonos grises. Si los dos personajes están unidos los dos iconos aparecerán en color.

3.8. Elementos durante la partida

En esta sección se detallarán los distintos elementos podrán aparecer en una estancia durante la partida. Se comentarán tanto los posibles personajes controlados por el usuario, como los posibles obstáculos y enemigos que pueden restar vidas a los personajes al colisionar. También se explicarán los elementos que componen la estancia como las puertas, paredes y suelo; así como la corona para liberar el mundo. El juego se desarrolla en perspectiva

isométrica por lo que todos los elementos visuales dentro de las estancias en las que se desarrolla la partida son dibujados con esta perspectiva.

3.8.1. Head

Head es uno de los personajes que pueden ser controlados por el usuario. Es controlado por el usuario que juega en modo servidor. Sus características principales son que posee un gran salto puesto que tiene unas pequeñas alas que le permiten planear. Este salto le permite poder situarse encima de Heels entre otras cosas. Por otro lado su velocidad es pequeña debido a que carece de piernas y se desplaza reptando por el suelo. Se puede ver a Head en la Figura 3-14.

Figura 3 - 14. Head.

3.8.2. Heels

El otro personaje protagonista de la partida es Heels. Este personaje es controlado por el usuario que juega en modo cliente. Heels, al contrario que Head, posee una gran velocidad que le otorgan sus potentes piernas. El salto de Heels es menos potente que el de Head. Esto no le permite llegar a la misma altura cuando salta, por lo que no podrá situarse encima de Head. En la Figura 3-15 se ve una imagen de Heels.

Figura 3 - 15. Heels.

3.8.3. Head over Heels

Se trata en este caso de los dos personajes unidos. Será controlado por el usuario que juega en modo servidor. Este personaje es activado al situarse Head sobre Heels y pulsar el botón correspondiente el usuario que controla a Head. Posee el salto de Head y la velocidad para desplazarse en las cuatro direcciones de Heels. Se observa una imagen de Head over Heels en la Figura 3-16.

Figura 3 - 16. Head over Heels.

3.8.4. Mono

Es el enemigo dinámico que se ha implementado en esta aplicación. Se mueve aleatoriamente dentro de una estancia en las cuatro mismas direcciones en las que se mueven los personajes controlados por el usuario. No posee salto. Su velocidad es la misma que la de Head. No puede pasar de una estancia a otra a través de una puerta. En la Figura 3-17 se puede observar una imagen del mono.

Figura 3 - 17. Mono.

3.8.5. Pinchos

Los pinchos son un elemento estático que resta una vida a un personaje si contacta con ellos. En la Figura 3-18 se ve una imagen de los pinchos.

Figura 3 - 18. Pinchos.

3.8.6. Corona

La corona es un elemento estático del juego que se sitúa en una de las estancias del juego. Cuando un personaje contacta con ella se liberará el mundo y finalizará la partida apareciendo el mensaje “Mundo Liberado”. Tras esto se podrá reiniciar la partida y comenzar esta de nuevo. Se muestra una imagen de la corona en la Figura 3-19.

Figura 3 - 19. Corona.

3.8.7. Puerta

Las puertas son elementos que aparecen en las paredes de las habitaciones y sirven para que los personajes controlados por el usuario puedan pasar de una estancia a otra adyacente con esta. Se ve una imagen de la puerta en la Figura 3-20.

Figura 3 - 20. Puerta

3.8.8. Estancia

Las estancias son los lugares donde se desarrolla la partida. Están compuestas por un suelo delimitado por cuatro paredes. Las paredes norte y oeste son dibujadas en la pantalla mientras la sur y este no. En la pantalla de cada usuario aparece la estancia completa en la que está el personaje que está siendo controlado desde una vista isométrica. Se puede observar un ejemplo en la Figura 3-21.

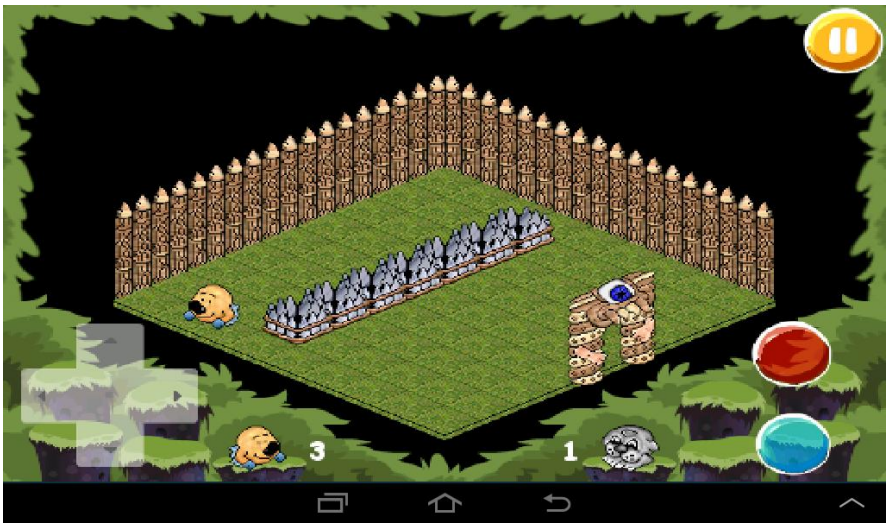

Figura 3 - 21. Estancia durante la partida.

3.9. Audios

La aplicación dispone de efectos de sonido y de música que pueden ser deshabilitados durante la partida del juego.

La aplicación cuenta con una melodía que comenzará a sonar durante la partida al cargarse por primera vez la pantalla Preparado.

Todos los sonidos y melodía del juego suenan únicamente si la opción de sonido está activa. El usuario puede activar o desactivar el sonido desde el menú de pausa durante la partida y esta opción será guardada. Incluso si se interrumpe la partida, la próxima vez que se entra la última opción de sonido guardada será cargada.

Durante el desarrollo de la partida existen 5 sonidos diferentes implementados. Estos sonidos son:

- Un sonido que se escuchará cuando se pulse el botón de pausa o alguno de los botones incluidos en el menú de pausa.
- Un sonido que se reproducirá cuando un personaje salte.
- Un sonido que se escuchará cuando un personaje pierda una vida.
- Un sonido para cuando uno o los dos personajes pierdan todas sus vidas y se termine la partida.
- Un sonido para cuando se consigue la corona.

Capítulo 4. Diseño de la aplicación

4.1. Introducción

Este capítulo trata sobre la forma en la que se ha implementado la aplicación para conseguir la funcionalidad descrita en el anterior capítulo. Se explicará de qué forma se han organizado las diferentes clases que componen el programa y cómo se han implementado para contribuir a la funcionalidad global de este. Para empezar se describirá el marco software donde se sitúa la aplicación desarrollada.

4.1.1. Marco software de la aplicación

La aplicación Android de videojuego se ejecutará en el entorno proporcionado por el dispositivo. La interacción con el usuario en esta aplicación software se hará a través del teclado, la pantalla y el altavoz del dispositivo. El terminal responderá a las acciones realizadas por el usuario mediante teclado o pulsando la pantalla táctil, mostrando los resultados de esta interacción y del desarrollo del videojuego a través de la pantalla y el altavoz en caso de que el sonido esté habilitado.

La tecnología Android se constituye con un modelo de capas para lograr un alto grado de portabilidad de las aplicaciones. Dicha distribución permite acceder a las capas más bajas a través del uso de librerías para que así el desarrollador no tenga que programar a bajo nivel las funcionalidades que se necesitan para que una aplicación utilice los componentes hardware de los dispositivos móviles. Cada una de las capas usa elementos de la capa inferior para realizar sus funciones. Se puede ver un resumen de estas capas en la Figura 4-1.

Figura 4 - 1.Resumen de la estructura de capas de Android

El núcleo del sistema operativo Android está basado en el kernel de Linux. Actúa como una capa de abstracción entre el hardware y las demás capas de la arquitectura. Esta capa no es accedida directamente por el desarrollador. Se deben utilizar las librerías situadas en capas superiores.

Las librerías nativas de Android están desarrolladas en C o C++ y compiladas para la arquitectura hardware propia del teléfono. Dichas librerías están implementadas normalmente por el fabricante, que también se encarga de instalarlas en el dispositivo antes de su puesta en venta. El objetivo de éstas es proporcionar funcionalidad a las aplicaciones para tareas que repiten frecuentemente, evitando tener que codificarlas cada vez y garantizando que se llevan a cabo de la manera “más eficiente”.

El entorno de ejecución de Android no se considera una capa en sí mismo, puesto que también está formado por librerías. En éste encontramos las librerías con las funcionalidades habituales de Java y otras específicas de Android.

Dentro del entorno de ejecución de Android, el componente principal es la máquina virtual Dalvik. Las aplicaciones se desarrollan en Java y son compiladas en un formato específico para que esta máquina virtual las ejecute.

Con esto se consigue la ventaja de que las aplicaciones se compilan una sola vez y de esta manera estarán listas para distribuirse con la garantía total de que podrán ejecutarse en cualquier terminal Android que disponga de la versión mínima del sistema operativo para la que fue desarrollada la aplicación.

El framework de aplicaciones es la siguiente capa y está formada por todos los servicios y clases que usan directamente las aplicaciones para realizar sus funciones. La mayoría de los componentes del framework son librerías Java que acceden a los recursos situados en las capas anteriores mediante la máquina virtual Dalvik.

La capa superior de esta pila software la forman las aplicaciones. Esta capa incluye todas las aplicaciones del dispositivo, tanto las que usan interfaz de usuario como las que no, tanto las instaladas por el usuario como las que vienen de serie con el dispositivo, tanto las nativas (desarrolladas en C o C++) como las administradas (programadas en Java). Se puede observar con detalle toda esta estructura en la Figura 4-2.

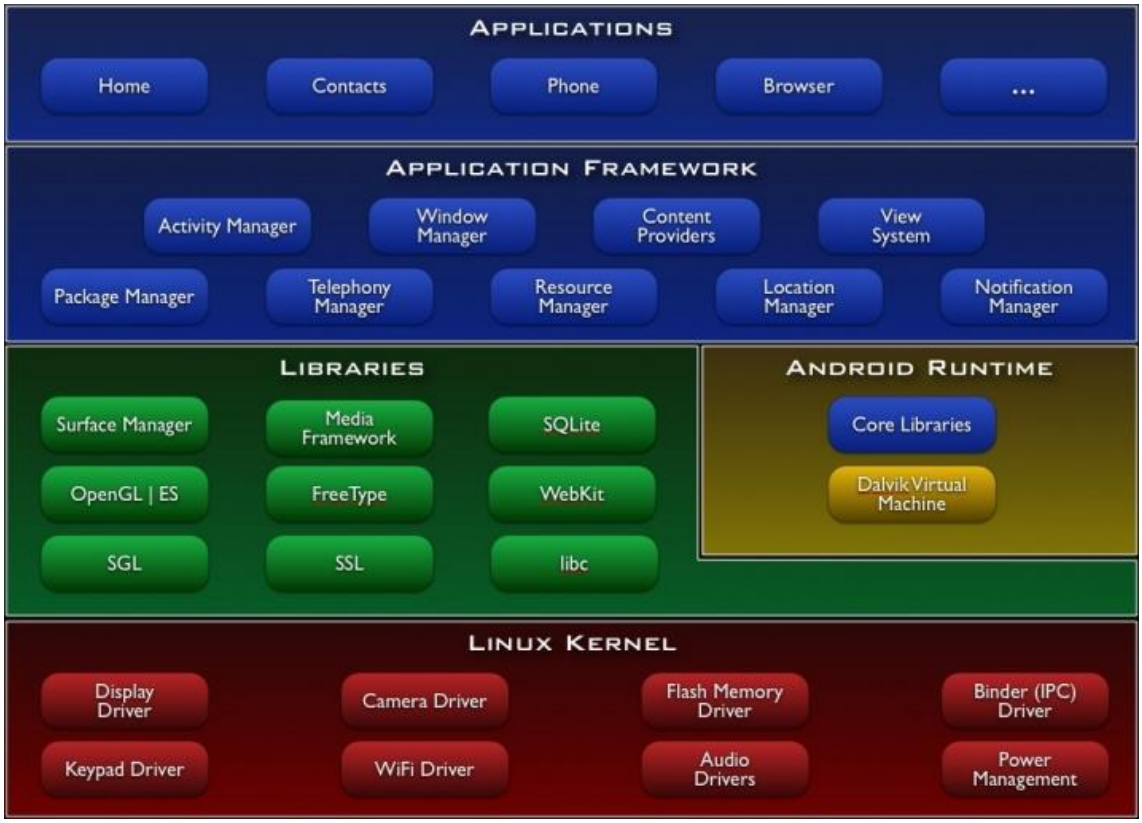

Figura 4 - 2. Estructura de capas oficial de Android.

Una vez que se ha introducido el entorno software sobre el que se implementa la aplicación desarrollada, ahora se explicarán los aspectos particulares del diseño y la implementación de ésta.

4.2. Visión general de la aplicación

Conviene conocer algunas características sobre las clases dentro de Android para establecer un punto de referencia desde el que tener una visión general de la aplicación, es decir, por un lado, la distribución e identificación de las clases implementadas, así como las relaciones entre estas, y por otro lado, un diagrama que represente de forma esquemática el funcionamiento global del sistema.

4.2.1. Herencia

La facilidad que nos ofrece para la reutilización de código es uno de los grandes atractivos de la programación orientada a objetos. Esto evita que la programación llegue a ser muy repetitiva. La reutilización de código consiste en no volver a escribir repetidas veces los mismos algoritmos para resolver situaciones similares. La clave para conseguir esto en Java, y por lo tanto, en Android, es la herencia.

Esto consiste en definir una nueva clase, que herede de una clase padre y añada el nuevo atributo que la diferencie. De esta manera podremos añadir nuevas funcionalidades sin tener que modificar la clase de partida. Además, es posible no disponer del código de una clase para poder heredar de ella.

La herencia consiste en crear una clase a partir de otra clase padre. La clase hija heredará los atributos y métodos de la clase padre y podrá crear nuevos atributos y métodos para completar su comportamiento. Esta clase hija también tendrá la posibilidad de redefinir los métodos de la clase padre.

4.2.2. Layouts

Los denominados *layouts* son elementos no visuales utilizados para controlar la distribución, posición y dimensiones de los controles que se insertan dentro de ellos.

Cuando se programa para Android se anima a declarar las actividades, con sus correspondientes elementos, en archivos XML que son independientes del código. De esta forma se gana flexibilidad y claridad.

En estos ficheros de describirá la manera en la que se agrupan los elementos de la actividad utilizando etiquetas XML de una forma muy similar a la utilizada en el desarrollo web. Las herramientas de desarrollo interpretan este fichero y generan el código necesario.

Las ventajas de este esquema son:

- Facilidad y rapidez para cambiar la disposición gráfica.
- Simplicidad y claridad en la descripción de las interfaces gráficas.
- Separación entre el código de la aplicación y la representación gráfica de la interfaz.

No obstante, es posible generar la interfaz escribiendo en el código de la clase y esto es una necesidad cuando la interfaz es dinámica y los elementos se deben generar durante la ejecución del programa de manera inmediata.

4.2.3. Hilos

El *software* que se ejecuta un dispositivo Android está basado en eventos. Esto quiere decir que el flujo de ejecución es dirigido por eventos de entrada. Dichos eventos pueden realizarse en cualquier orden y en cualquier instante de tiempo.

La ejecución del *software* basado en eventos generalmente se basa en una denominada “cola de eventos”. En dicha cola se guardan los eventos de entrada según el orden de llegada. Para cada cola, el hilo lee los eventos de la

cola en un bule infinito, uno tras otro, y llama a las rutinas que manejan los eventos correspondientes.

Un mismo hilo puede ejecutar todas las rutinas. En este caso, los eventos son ejecutados uno tras otro, de forma secuencial. Otra alternativa sería ejecutar cada rutina por separado en distintos hilos. En dicho caso, las rutinas se ejecutarán simultáneamente, en paralelo. Esta manera de trabajar tiene la ventaja de que el hilo encargado de la cola de eventos puede atender de forma rápida nuevos eventos, al no tener que mantenerse a la espera de que finalice la ejecución de cada tarea.

4.2.4. División de la aplicación

Para la realización del videojuego se ha dividido la estructura en dos bloques. El primero es el bloque de introducción y selección de personaje. Las clases que lo componen tienen como padre la clase Activity, la cual es proporcionada por Android. La parte lógica permanece escrita en su clase y la parte gráfica en su *layout* asociado.

Por otro lado, el segundo bloque está constituido por el motor del juego. Contiene las clases que generan la parte “jugable” de la aplicación. Hay gran cantidad de las clases que lo constituyen que heredan de alguna clase del *framework* descrito en el apartado 2.2.7. Todo el apartado gráfico ha sido diseñado en una de las clases que lo forman.

En la clase Settings se guarda y cargan los ajustes de sonido del juego.

4.2.5. Clases implementadas en el bloque de introducción y selección de personaje

Implementan la pantalla de introducción y la de selección de personaje, y por tanto la selección entre servidor o cliente. También tienen la función del establecimiento de la conexión Bluetooth entre los dispositivos.

Las clases son:

- Introduccion: inicia la aplicación y muestra una pantalla de inicio
- Seleccion: desarrolla la pantalla de selección de personaje. Se encarga también del establecimiento de la conexión Bluetooth. Dentro de ella se implementan dos hilos:
    - HiloServidor: establece y mantiene la conexión del dispositivo que está en modo servidor.
    - HiloCliente: establece y mantiene la conexión del dispositivo que está en modo cliente.
- DeviceListActivity: registra y guarda los dispositivos con los que se enlaza además de otras ayudas para la conexión Bluetooth.

La forma en la que estas clases están desarrolladas se muestra en el diagrama de clases de la Figura 4-3.

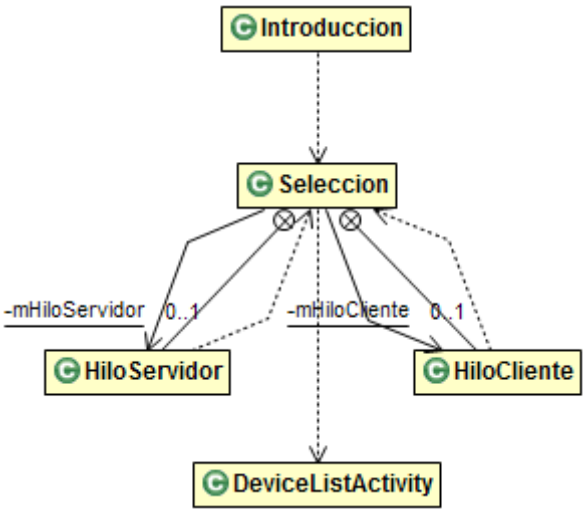

Figura 4 - 3. Diagrama de clases del bloque de introducción y selección de personaje.

4.2.6. Clases implementadas en el bloque del motor del juego

En este bloque, las clases implementadas controlan tanto el desarrollo de la partida y los elementos que lo componen como el apartado visual.

Las clases que lo constituyen son:

- HOHBT: arranca el juego y crea el hilo que se mantendrá escuchando la interacción del usuario con la pantalla.
- LoadingScreen: carga las imágenes y los archivos sonoros en Recursos. Carga también los ajustes.
- Recursos: lista de recursos que se van a usar.
- Settings: se cargan y guardan las opciones de audio.
- PantallaMultijugador: crea toda la interfaz gráfica y controla la interacción del usuario con la aplicación. También se encarga del envío de mensajes e interpretación de mensajes recibidos vía Bluetooth.
- Mundo: actualiza lo que sucede en la partida con cada uno de sus elementos.
- MensajeBT: sirve de puente para que los eventos de Mundo puedan ser enviados por Bluetooth en PantallaMultijugador.
- Objeto: clase padre de la que heredan los elementos de la partida.
- Animacion: implementa una animación de un objeto dinámico.
- ObjetoDinamico: hereda de objeto. Implementa un Objeto con movimiento.
- ElementoMapa: hereda de Objeto e implementa un elemento que compone la estancia.
- Mapeado: compone una estancia con paredes y suelo.
- Puerta: hereda de Objeto e implementa las puertas de las estancias.
- Personaje: hereda de Objeto e implementa los personajes controlados por el usuario.
- Head: hereda de Personaje. Es el personaje Head.
- Heels: hereda de Personaje. Es el personaje Heels.
- Hoh: hereda de Personaje e implementa al personaje Head over Heels.
- Monkey: es el mono enemigo durante la partida. Hereda de ObjetoDinamico.
- Pinchos: heredera de Objeto e implementa los pinchos que se encuentran como obstáculo en la partida.
- Corona: corona para liberar mundo. Hereda de Objeto.

La forma en que estas clases están relacionadas se representa en el diagrama de clases de la Figura 4-4. Se aprecia cómo, partiendo de HOHBT, se llega a la clase PantallaMultijugador, y cómo esta se relaciona con Mundo, clase encargada de toda la lógica automática.

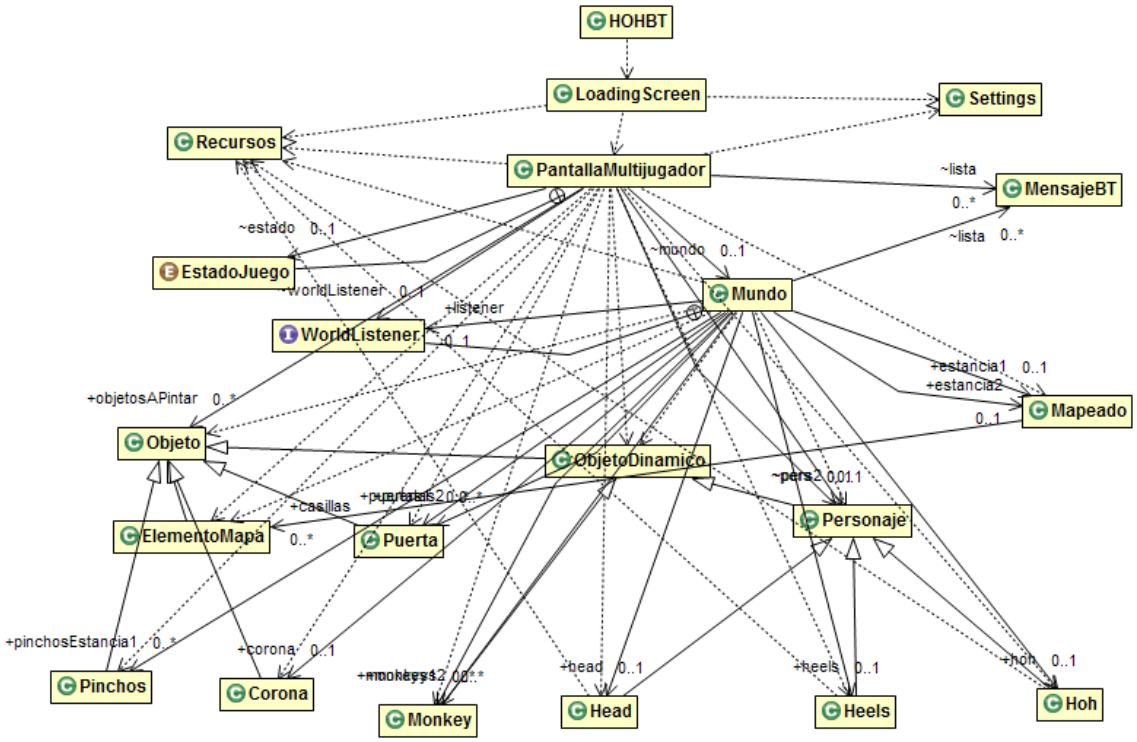

Figura 4 - 4. Diagrama de clases del bloque del motor del juego

La relación entre los dos bloques viene con el arranque de la clase HOHBT en Seleccion.

4.2.7. Diagrama de estados de la aplicación

El diagrama de estados es una representación esquemática del funcionamiento global de la aplicación. Se muestra en la Figura 4-5. Muestra de forma simplificada la descripción del funcionamiento que se realizó en el capítulo 3 de este documento.

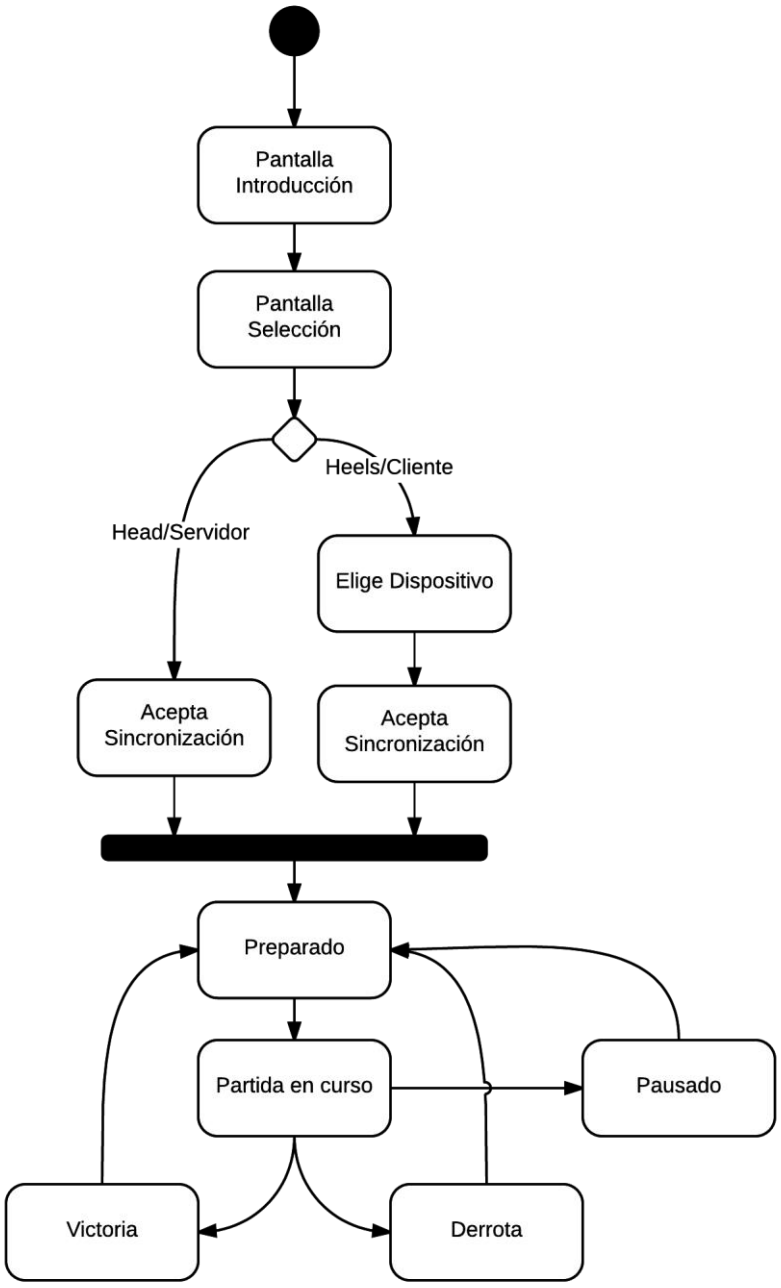

Figura 4 - 5. Diagrama de estados de la aplicación

4.3. Utilidades generales para la aplicación

En este apartado se incluyen una serie de clases e interfaces que han sido utilizadas para todo el código, tales como las clases heredadas por las clases implementadas, los mecanismos de lectura y escritura de datos y las herramientas usadas para el apartado sonoro y gráfico de la aplicación.

4.3.1. Clases heredadas

Se pasará a describir los aspectos básicos de las clases e interfaces padre de algunas de las clases implementadas.

Activity

La clase Activity es la base de cualquier aplicación Android que posea interfaz de usuario. Si una aplicación tiene interfaz de usuario tendrá al menos una clase Activity o una clase que herede de esta.

Una Activity está conformada por dos partes: una parte lógica y otra gráfica. Por un lado, la parte lógica es un fichero .java. Es la clase que se implementa para poder manipular, interactuar y colocar el código de esa actividad.

Por otro, la parte gráfica está constituida por los *layouts* a los que se ha hecho referencia en la sección 4.2.2. Contiene todos los elementos que se ven en una pantalla, declarados con etiquetas similares a las del HTML, es decir, que el diseño de una aplicación Android es parecido al de una página web. Por decirlo de alguna forma, XML es un primo cercano de HTML.

En una Activity se debe implementar al menos un método. Éste es el método onCreate, que es en donde se genera la actividad. Se puede decir que es donde se le da vida. En una parte de este método se enlaza la parte lógica con la parte gráfica de la Activity.

Con respecto del ciclo de vida de una Activity, se debe conocer que cuando es iniciada una nueva Activity, se pone al inicio de la pila de ejecución y se convierte en la Activity en ejecución. La Activity que estaba ejecutándose antes siempre estará por debajo en la pila y no volverá a primer plano hasta que la nueva Activity no exista. En la Figura 4-6 se puede observar el ciclo de vida de este tipo de clases.

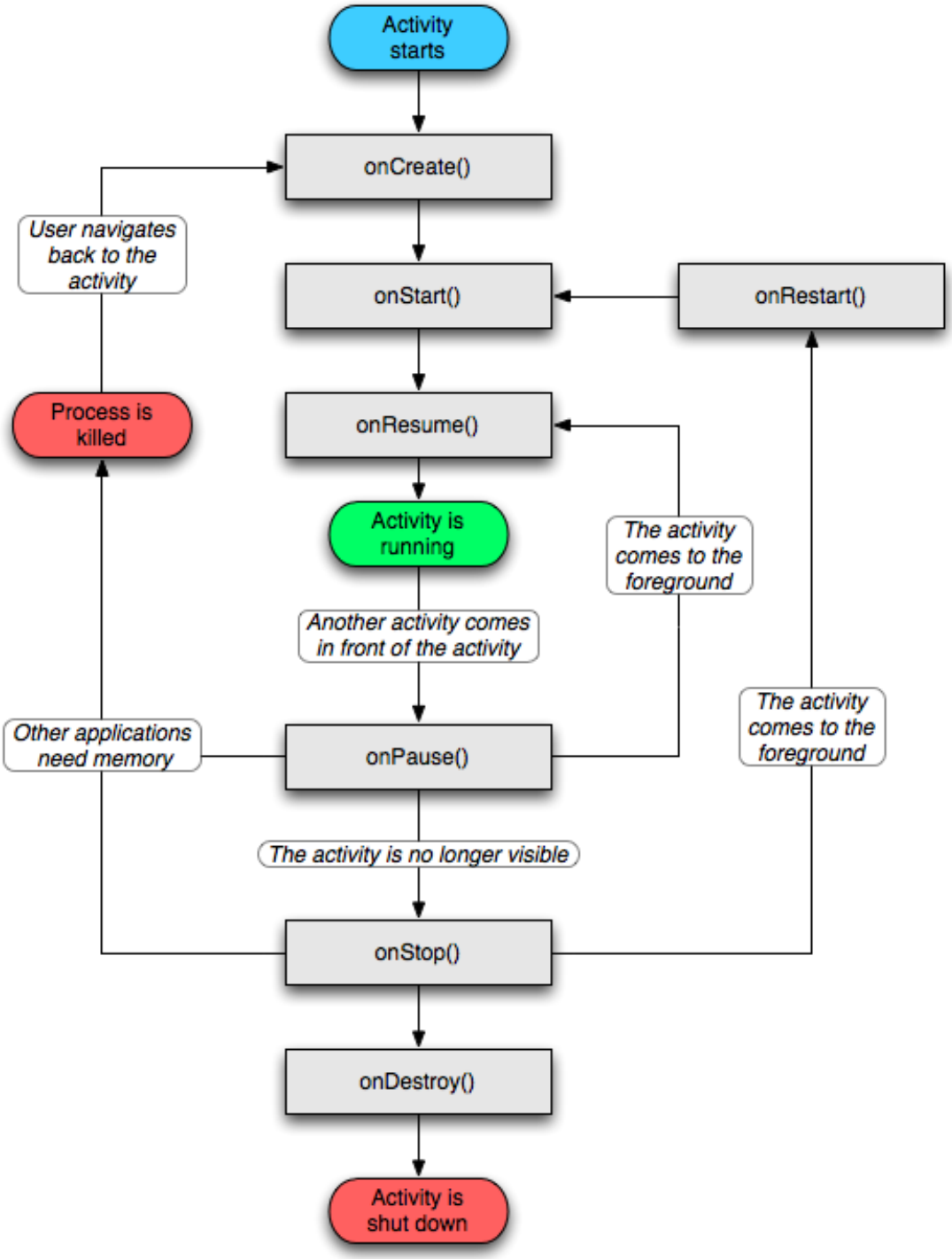

Figura 4 - 6. Ciclo de vida de la clase Activity

Se usan los métodos *startActivity* y *startActivityForResult* para iniciar una Activity desde otras clases. El primero lanza una nueva Activity sin esperar resultado al finalizarse. El segundo lanza una Activity de la que se espera un resultado una vez finalice. El método *onActivityForResult* será llamado dentro

de la primera Activity con los datos proporcionados cuando la segunda Activity finalice.

Game

Esta interfaz está incluida en el framework de Mario Zechner, comentado en el capítulo 2.2.7 y es usada en el bloque del motor del juego. Su implementación tiene los fines de:

- Configurar la pantalla y el componente de la interfaz de usuario e integrarse en todas las llamadas que se generen para que pueda recibir los eventos procedentes de las entradas de usuario y de la ventana.
- Iniciar el hilo de la ejecución principal.
- Guarda un registro de la pantalla principal y mandarle que se actualice y muestre su contenido cada vez que se repita el bucle principal.
- Transferir los eventos de la ventana que se envíen desde el hilo de ejecución de la interfaz de usuario hasta el del bucle principal.
- Garantizar el acceso a otros módulos desarrollados en el framework.

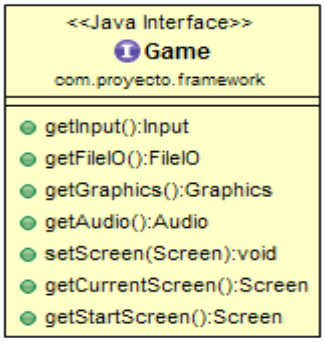

Figura 4 - 7. Interfaz Game

Su implementación será responsable de generar una nueva versión y de registrar la actividad de algunos módulos de bajo nivel. La Figura 4-7 muestra la interfaz Game con los métodos que incluye.

Con el método *setScreen* se define la pantalla Screen (que será explicada en este documento más adelante) de Game. Este método y los

métodos que devuelven ejemplos de los módulos de bajo nivel solo se implementará una vez, junto con la creación del hilo de ejecución interno, el gestor de la ventana y la lógica que será empleada con el bucle principal. Ésta pedirá constantemente a la pantalla que se actualice y muestre su contenido en el dispositivo. El método *getCurrentScreen* devuelve la pantalla que está activa.

Más adelante se describe la clase abstracta AndroidGame que implementa la interfaz Game. AndroidGame implementará todos los métodos exceptuando *StartScreen*. Con este método se obtiene una instancia de la primera pantalla del juego.

Screen

Es una clase abstracta que también se incluye en el framework de Mario Zechner. Se creó abstracta en vez de una interfaz para permitir implementar algo de contabilidad y así escribir menos código cuando se fuese a implementar. Puede acceder a módulos de bajo nivel de la clase Game para reproducir sonidos, dibujar el contenido de la pantalla y leer y escribir datos en archivos.

Configura una nueva pantalla Screen cada vez que se necesite. Con esto se pueden implementar las transiciones de pantalla dentro de sus instancias. Cada una de éstas decidirá cuándo tendrá lugar y si habrá otros casos de Screen que se basarán en su estado.

Los métodos *update* y *present* son los encargados de actualizar el estado de la pantalla y presentarlo en el dispositivo. Serán llamados una única vez en cada una de las repeticiones del bloque principal.

Por otro lado, los métodos *pause* y *resume* serán llamados cuando se detenga y retorne el juego.

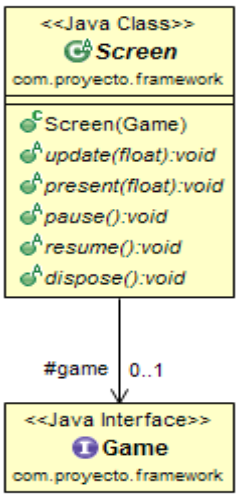

Figura 4 - 8. Clase Screen

El encargado de eliminar el objeto Screen actual y liberar todos los recursos del sistema que tuviese reservados es el método *dispose*. Se puede observar con detenimiento la clase en la Figura 4-8.

AndroidGame

Esta clase también pertenece al framework de Mario Zechner. Hereda de la clase Activity e implementa la interfaz Game. Se ha creado con el fin de ser llamada sólo una vez y que de ésta se puedan obtener todos los elementos que se necesiten.

Como puede verse en la Figura 4-9 es una clase algo compleja. Lo más destacado para este proyecto de esta clase, está en el método *onCreate*. Hará que se trabaje en modo pantalla completa con la orientación que se requiera y con un tamaño original de 480x320. La clase es capaz de convertir las coordenadas del punto de pantalla que ha tocado el usuario al sistema fijo de coordenadas que se está usando.

Otra de las ideas destacadas implementadas en la clase AndroidGame se encuentra en el método *setScreen*, que es heredada del interfaz Game. Se encarga de comunicar a la Screen activa que se detenga temporalmente (que se pause) y se libere para dejar sitio a una nueva instancia de Screen. Cuando el intervalo de tiempo es igual a cero, a esta nueva instancia se le pide que

retome la actividad y que se actualice, asignando el miembro Screen a la nueva pantalla Screen.

Un método que se utiliza muy a menudo es *getGraphics.* Este devuelve la instancia al elemento que ha generado la llamada. Este elemento siempre es una de las implementaciones Screen del juego.

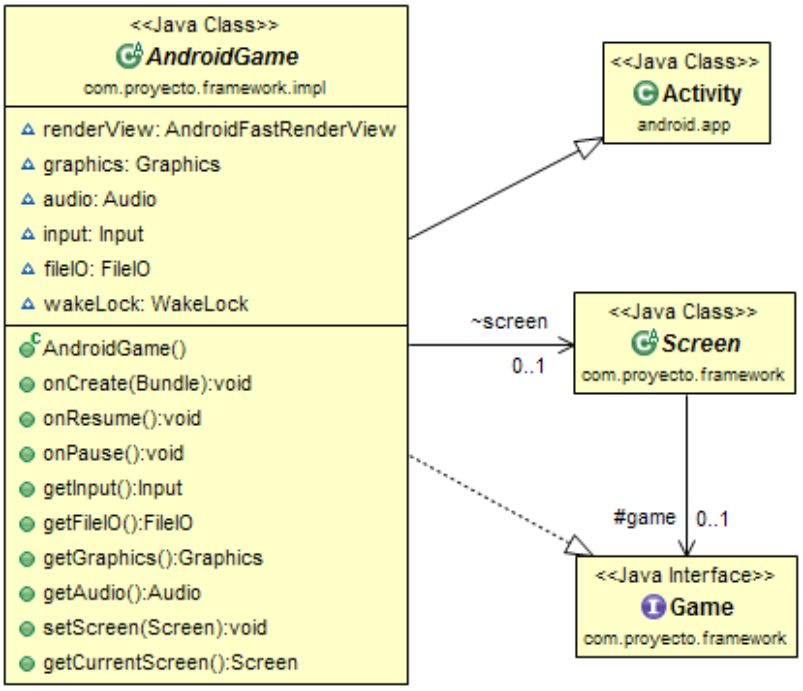

Figura 4 - 9. Clase AndroidGame

4.3.2. Interfaz gráfica en el bloque de introducción y selección

Widget

El paquete Widget es un paquete oficial de Android. Contiene elementos del interfaz de usuario (UI), en la mayoría de los casos visuales, para usar en la pantalla de la aplicación.

1) ImageButton

Define un botón con una imagen asignada. Se le puede asignar una posición en la pantalla así como su tamaño entre otras propiedades.

2) ImageView

Permite mostrar imágenes en la aplicación. Su propiedad más interesante es *android:src*, que permite indicar la imagen a mostrar. También se puede definir el tamaño de esta imagen.

3) Toast

Un toast es una ventana emergente que tiene un pequeño mensaje temporal para el usuario. La clase Toast ayuda a crearlos y mostrarlos.

Cuando esta ventana es mostrada al usuario, aparece como un elemento flotante por encima de la aplicación. Nunca se enfoca a ella. La idea es que sea lo menos intrusivo posible, mientras se muestra al usuario la información que se desee.

El principal método de la clase que se ha usado es *makeText*, que genera el toast estándar con el mensaje a mostrar.

App

Este paquete contiene clases de alto nivel que encapsulan el modelo de la aplicación Android. Se trata de un paquete importante ya que en él se encuentra la clase Activity que es el núcleo de las aplicaciones, que ya se comentó con anterioridad.

1) AlertDialog

Muestra un texto emergente con uno, dos o tres botones. Cada botó puede relacionarse con una acción. Se utiliza para crear una pregunta o elección al usuario cuyas posibles respuestas se reflejen en los botones. El pulsar uno de los botones hará que el programa realice una opción que indiquemos.

4.3.3. Interfaz gráfica del motor del juego

Graphics

El *framework* de Mario Zechner utiliza dos interfaces para usarlos con los módulos, pero en este proyecto sólo se utiliza el interfaz Graphics. Las imágenes que se quieran utilizar se cargan en un buffer virtual del componente de la UI donde se dibuja la imagen. Con esto, el interfaz puede:

- Cargar imágenes almacenadas en disco y guardarlas en la memoria para ser dibujadas más tarde.
- Dibujar las imágenes que se hayan cargado al inicio del buffer, pudiendo dibujar la imagen completa o bien parte de ella en una posición concreta.

El método *newPixmap* cargará una imagen con el formato JPEG o PNG y el nombre del fichero se especifica como un recurso en el archivo APK de la aplicación.

El método *drawPixmap* se encarga de dibujar partes de una imagen cargada en unas coordenadas específicas. Incluso se puede recortar la imagen ya que se ha de especificar cuál es el lugar de la imagen que se tiene en cuenta como el inicio de ésta y cuál será el tamaño a partir de este inicio.

Cabe destacar la enumeración en una lista llamada PixmapFormat de los diferentes formatos que se pueden usar: ARGB8888, ARGB4444 o RGB565.

4.3.4. Lectura y escritura de datos

IO

1) BufferedReader

Esta clase se utiliza para realizar clases Reader de bajo nivel de una forma eficiente y sencilla de usar. Los BufferedReaders leen grandes cantidades de un fichero a la vez, y mantienen esta información en el buffer. En

el momento en el que se pregunta por el siguiente carácter o la siguiente línea de información, la respuesta se recupera del buffer, con lo que se minimiza el número de ocasiones que se tiene que leer desde dicho fichero. Esta clase provee métodos como *readLine,* que nos permite leer la siguiente línea de caracteres de un archivo.

2) BufferedWriter

Se utiliza esta clase para crear clases de bajo nivel. Los BufferedWriters escriben grandes cantidades de información en un archivo, lo cual minimiza el número de veces que las operaciones de escritura de archivos se llevan a cabo. La clase BufferedWriter tiene un método que se llama *newLine*, el cual genera separadores de línea propios de la plataforma de manera automática.

3) InputStreamReader

Es una clase para cambiar un *stream* de bytes a un *stream* de caracteres. Los datos que se leen de la fuente son transformados a caracteres por un convertidor que puede ser provisto o se puede usar el convertidor por defecto. InputStreamReader contiene un buffer de bytes leídos del *stream* de la fuente, los cuales son transformados a caracteres conforme se vayan necesitando. Dicho buffer tiene un tamaño de 8KB.

4) OutputStreamWriter

Esta clase es usada para cambiar un *stream* de caracteres a un *stream* de bytes. Se convierten los datos escritos en el *stream* de salida a bytes por un conversor que puede ser provisto o usar el conversor por defecto. El conversor por defecto lo obtiene el “conversor de archivos” proporcionado por el sistema. OutputStreamWriter tiene un buffer de bytes para ser escritos en el *stream* de salida y convertir los caracteres conforme se necesiten. El tamaño de ese buffer es de 8KB.

5) OutputStream

Clase que representa un *stream* de bytes de salida.

6) PrintWriter

Es un objeto que permite imprimir representaciones formateadas de una salida de *stream* de texto. Su método *println* presenta la cadena que se carga y hace que lo siguiente a escribir se haga en una nueva línea. Tiene un método llamado *flush* que se asegura de que todos los datos pendientes sean enviados al objetivo.

Content

1) Context

Esta clase proporciona un interfaz que posee toda la información global del entorno de una aplicación. Es una clase abstracta cuya aplicación está prevista por el sistema Android. Permite el acceso a los recursos y las clases específicas de la aplicación, además de las convocatorias de las operaciones a nivel de aplicación, como pueden ser el lanzamiento de las actividades, la difusión y la recepción de los *Intents*, etc.

Integer

1) parseInt

Analiza la línea especificada como un valor decimal de tipo *Integer*.

String

1) trim

Copia la cadena eliminando cualquier espacio en blanco del principio de final de la cadena.

2) split

Divide una cadena de caracteres usando la expresión que se le especifique. En este proyecto se ha utilizado el símbolo coma para que marque el final de una sección de la cadena.

4.3.5. Gestor de sonidos

Media

1) MediaPlayer

Es una clase que se usa para el control de acciones visuales o sonoras de archivos de video y audio y de *streams*.

Audio

Con esta interfaz se encuentra la manera de cargar los sonidos en memoria y controlar la reproducción de archivos de audio precargados y la reproducción del *stream* del sonido que no se desea cargar.

La interfaz audio también permite crear nuevos objetos Music y Sound. Un objeto Sound representa un efecto sonoro que se ha guardado en memoria, mientras que un objeto Music representa un archivo de audio que se reproduce sobre la marcha.

1) newSound

Toma el nombre del archivo como argumento y lo relaciona con la interfaz Sound. Posteriormente, se puede ejecutar sobre éste el método *play* indicando el volumen como argumento.

2) newMusic

Relaciona el nombre del archivo que se le pasa como argumento con la interfaz Music. En esta ocasión, sus métodos permiten la reproducción del *stream* de música. Además se puede pausar o detener completamente a éste y definir un bucle de reproducción. En dicho bucle, cuando se llega al final del archivo de audio, se volverá a reproducir desde el principio. También se le puede definir el volumen.

4.3.6. Interacción con el usuario

View

1) View

Esta clase representa el bloque básico para la construcción de componentes de la interfaz de usuario. Ocupa un área rectangular en la pantalla y es la responsable tanto del dibujado en ella como de registrar el evento de ser pulsado. Se trata de la clase base para los widgets, de los que se ha hablado anteriormente en este documento, que son usados para crear componentes UI interactivos, como por ejemplo los botones.

2) OnclickListener

Interfaz definida para avisar siendo invocada cuando una View sea cliqueada.

Content

1) DialogInterface

Es un interfaz utilizado para la creación de un dialogo que ejecute algún código cuando un ítem de dicho dialogo sea cliqueado.

4.3.7. Interacción entre clases

IO

1) IOException

Señala un error general de entrada/salida. Se pueden especificar los detalles de dicho error si se llama al constructor de éste.

OS

1) Bundle

Es una clase que sirve para contener tipos primitivos y objetos de otras clases. Con ella se pueden pasar datos de una Activity a otra.

Util

1) List

Una List es una lista de elementos que mantienen un orden. Cada elemento de la lista tiene un índice a través del cual se puede acceder a él, siendo el índice del primer elemento el cero. Normalmente, las List permiten elementos duplicados, a diferencia de los Sets.

2) ArrayList

Es una implementación de List, creada como una matriz.

Content

1) Intent

Un Intent es una descripción abstracta de una operación a realizar. En la aplicación se ha usado junto a los métodos descritos anteriormente *startActivity* o *startActiviyForResult* para comenzar una nueva Activity.

2) AcitvityInfo

Se encarga de proporcionar la información que puede saberse de una Activity. Dicha información es recogida del AndroidManifest.xml.

4.3.8. Bluetooth

util

1) UUID

UUID (Universally Unique Identifier) es una representación única de 128 bits que se usa para identificar la aplicación en una conexión Bluetooth.

bluetooth

1) BluetoothAdapter

Esta clase es el punto de entrada para toda la infraestructura Bluetooth de la aplicación. BluetoothAdapter representa el adaptador Bluetooth, es decir, el dispositivo físico. Con esta clase se pueden descubrir otros dispositivos Bluetooth activos cerca del dispositivo, acceder a la lista de dispositivos emparejados, instanciar un objeto de la clase BluetoothDevice usando una MAC conocida o crear sockets para la comunicación con otros dispositivos.

Su método estático *getDefaultAdapter* retorna una referencia del adaptador local Bluetooth del dispositivo que ejecuta la aplicación.

El método *listenUsingRfcommSocketToServiceRecord* crea un socket de servidor con un nombre y un UUID determinados. El nombre y el UUID son registrados en el SDP (Service Discovery Protocol). Este proporciona a los clientes la información que necesitan para descubrir un servicio.

2) BluetoothDevice

Se usa para solicitar una conexión con un dispositivo Bluetooth remoto a través de un BluetoothSocket. Además se utiliza para solicitar información acerca del dispositivo, como puede ser el nombre, la MAC, clase, etc.

Su método *createRfcommSocketToServiceRecord* es el encargado de generar un socket de cliente listo para conectarse al socket de servidor con el UUID que se proporcione.

3) BluetoothServerSocket

Para que se pueda establecer una conexión, uno de los dispositivos debe tomar el rol de Servidor (conexión pasiva) y el otro el rol de Cliente (conexión activa). Esta clase representa a un *socket* escuchando en el lado de la aplicación servidora.

En el instante en el que el cliente solicita una conexión, un objeto de esta clase se encarga de retomar un objeto de la clase BluetoothSocket que representará al servidor, siempre y cuando la conexión haya sido aceptada.

4) BluetoothSocket

Es la clase que permite una comunicación punto a punto con otro dispositivo enviando y recibiendo información en forma de *streams* de datos. Representa una interfaz con otro dispositivo Bluetooth en forma de socket.

Sus métodos *getInputStream* y *getOutputStream* devuelven los flujos de información que se pueden usar para la transmisión de datos.

Net

1) UnknowHostException

Esta clase se ejecuta cuando no puede realizarse la tarea dada con la dirección de un host proporcionado. Con esto se puede crear alguna rutina para que, en caso de ocurrir un error, la aplicación realice una actividad distinta a si hubiera tenido éxito.

4.3.9. DeviceListActivity.java

Es una Activity que aparece como un diálogo. Es la encargada de obtener y mostrar los dispositivos a los que la aplicación puede conectarse y conseguir la dirección de aquel dispositivo al que el usuario quiera conectarse. También diferencia entre los dispositivos ya enlazados anteriormente (los muestra inicialmente) con los nuevos dispositivos descubiertos, para los cuales

se tiene que realizar una búsqueda. Muestra una lista de los dispositivos vinculados y los descubiertos.

4.4. Permisos

Una aplicación Android recoge los permisos que se le otorgan a la aplicación y algunas características generales en el AndroidManifest.xml. Durante el proceso de instalación de la aplicación, se informa al usuario de dichos permisos. Se puede cancelar la instalación en el caso de no estar de acuerdo con otorgar dichos permisos.

En esta aplicación se utilizan los siguientes permisos:

- WRITE_EXTERNAL_STORAGE: Para poder escribir en memoria externa.
- WAKE_LOCK: Permite el acceso a los bloqueos de energía para mantener el procesador durmiendo o mantener la pantalla apagada. En esta aplicación se utiliza para que el dispositivo permanezca siempre activo durante la partida.
- BLUETOOTH: Permite la conexión entre aplicaciones y dispositivos pareados por Bluetooth.
- BLUETOOTH_ADMIN: Para que una aplicación pueda buscar y parear dispositivos Bluetooth.

4.5. Bluetooth

Recordando lo ya comentado en el capítulo 2 de este documento, Bluetooth es una tecnología de radio de corto alcance (de unos diez metros aproximadamente en un ambiente libre de obstáculos), que opera en la banda libre de radio ISM a 2,4 GHz y que permite conectividad inalámbrica entre dispositivos remotos. Android permite el desarrollo de aplicaciones que usen tecnología Bluetooth utilizando su propia API en el paquete *Android.bluetooth.*

A la hora de diseñar la funcionalidad multijugador, el primer aspecto importante es fijar un protocolo de comunicación sobre el que se va a sustentar

la estructura. En esta aplicación el enfoque que se ha seguido es utilizar una estructura cliente-servidor sobre Bluetooth limitando el número de clientes a solo uno. Se usa el protocolo de RFCOMM (Radio Frequency Communication) que establece las sesiones de comunicación usando las direcciones Bluetooth de los dos terminales. Dichas sesiones están ligadas a cada par único de direcciones Bluetooth de los dispositivos pareados.

4.5.1. Pasos previos al emparejamiento

Para empezar, se crea una variable general para el tipo BluetoothAdapter. Una vez que se entra a la pantalla de selección de personaje, a través del método *BluetoothAdapter.getDefaultAdapter.* Esta variable se asocia con el adaptador Bluetooth.

Posteriormente se realiza la comprobación de si el dispositivo soporta tecnología Bluetooth comprobando que la variable a la que se ha asociado el adaptador no es nula. Si la variable resulta nula, se indica a través de un *toast* que el dispositivo no soporta Bluetooth.

Seguidamente se comprobará que el Bluetooth del dispositivo esté activado con la respuesta booleana la característica *isEnabled* de la variable. En el caso de que no esté activo, se lanzará un *Intent* para permitir la posibilidad de activarlo a través de la función ACTION_REQUEST_ENABLE. También se hará visible el dispositivo en su entorno para que pueda ser detectado por otros dispositivos para que así le envíen una solicitud de vinculación.

Una vez hecho esto, se creará un diálogo mediante *AlertDialog* preguntando con qué personaje se desea jugar, y por lo tanto, si se desea jugar en modo servidor o modo cliente.

4.5.2. Funcionamiento

Emparejar dispositivos

Con anterioridad a que los datos puedan ser enviados entre dos dispositivos a través de Bluetooth, los dispositivos tienen que estar emparejados. Una vez que los dispositivos tengan el Bluetooth activado y visible, el emparejamiento entre ellos requiere los siguientes pasos:

- Uno de los dispositivos, en el caso de esta aplicación el dispositivo Cliente (con el que se controlará a Heels), debe explorar su entorno para ver si hay otros dispositivos Bluetooth. A través de esto, encontrará todos los dispositivos en su rango de alcance con el Bluetooth activado, entre ellos el dispositivo Servidor (con el que se jugará con Head).
- El dispositivo Cliente debe enviar una solicitud de vinculación con el dispositivo Servidor. La solicitud entrante se muestra en los dos dispositivos junto a una clave de enlace generada por el sistema que se puede aceptar o denegar. Si los dos dispositivos aceptan la solicitud, estos quedan emparejados.

Transmisión de datos

La transmisión de datos en Bluetooth está basada en un modelo cliente-servidor con *sockets*. Un socket es una interfaz a una red de datos. Esto quiere decir que un programa puede leer datos entrantes de un *socket* y escribir los de salida del mismo sin necesidad de saber el funcionamiento interno de la red. Los datos siempre son transmitidos a través de un par de *sockets*: Los datos escritos por el socket del servidor se pueden leer desde el socket del cliente y viceversa. Se pueden considerar por tanto, que los dos *sockets* se pueden considerar como los extremos de un canal de datos entre el servidor y el cliente.

Para el envío de datos a través de un par de *sockets* Bluetooth en Android se requieren los siguientes pasos:

- En el dispositivo que actúa como servidor, la aplicación debe generar un "*socket* servidor" con un UUID y esperar una solicitud de conexión de un dispositivo cliente.
- En el dispositivo que actúa como cliente, la aplicación tiene que crear un "*socket* de cliente" y mandar una solicitud de conexión al *socket* de servidor, identificándole por su UUID.
- El dispositivo servidor debe aceptar la solicitud. Posteriormente a esta aceptación, el cliente y el servidor estarán conectados. Tras producirse esto compartirán un canal RFCOMM para la siguiente transmisión de datos. El servidor crea un nuevo *socket* para la conexión.
- El cliente y el servidor deben obtener los flujos de datos de entrada y salida de sus *sockets* respectivos. Todos los datos que el cliente escriba en su flujo de salida aparecerán en el flujo de entrada del servidor y viceversa. La transferencia de datos se puede realizar en ambas direcciones.

4.5.3. Modo servidor

Cuando se selecciona como personaje a controlar a Head (modo servidor), la rutina del diálogo que nos pregunta el personaje a elegir, hará que se cree un hilo llamado HiloServidor que escuchará las conexiones entrantes y pasará el *socket* a la clase PantallaMultijugador donde la conexión *socket* será establecida.

Este hilo comienza creando un objeto temporal que posteriormente será asignado al *socket* servidor. El objeto temporal usará el método del adaptador *listenUsingRfcommWithServiceRecord* con el UUID y el nombre de la aplicación que también es usada en el código cliente. Una vez se crea el *socket* en el objeto temporal, se pasa al *socket* servidor. Posteriormente el hilo se dará por creado y volverá a la rutina del diálogo.

Esta rutina será la que hará arrancar el hilo generado. El hilo permanecerá a la escucha hasta que ocurra una excepción y el *socket* es devuelto. Cuando se devuelve la conexión *socket*, se establece dicho *socket* en

la clase PantallaMultijugador y se informa a una variable booleana dentro de esa clase, que es el servidor. Después de esto, se pasa a la clase HOHBT para iniciar la partida.

La clase HOHBT es hija de AndroidGame. Carga los recursos del juego hasta finalizar en la clase PantallaMultijugador en la cual ya se ha trasladado el socket correspondiente.

4.5.4. Modo cliente

Es activado cuando se pulsa en la opción de elegir a Heels en el diálogo de selección de personaje. Esta opción lanza la actividad definida en la clase DeviceListActivity (explicado anteriormente). Para lanzarla utilizará el método *startActivityForResult* que esperará la recepción de una respuesta al terminar dicha actividad, marcada por la etiqueta REQUEST_CONNECT_DEVICE.

El resultado se verá reflejado en el método onActivityResult. En el caso de que los resultados dados indiquen que se terminó bien la Activity y que el paquete de datos dados no esté vacío, cogerá de esos datos la dirección MAC del dispositivo remoto y utilizará dicha dirección para crear una variable de la clase BluetoothDevice.

Posteriormente se generará el llamado HiloCliente que intentará conectarse con el otro dispositivo y pasar el *socket* a la clase PantallaMultijugador cuando la conexión *socket* esté establecida de manera similar al modo servidor.

Se utiliza un método de la variable BluetoothDevice para conectar el *socket* de Bluetooth con el dispositivo cuya dirección fue recibida. Dicho método es llamado *createRfcommSocketToServiceRecord* con el UUID de la aplicación.

Cuando se ha creado el hilo, el método *onActivityResult* activará dicho hilo que lo primero que hará será cancelar la búsqueda de nuevos dispositivos con el fin de que la conexión no se ralentice mucho. Después intentará conectar el dispositivo con el *socket*. Si logra la conexión, establecerá dicho *socket* en la clase PantallaMultijugador e informará a una variable booleana

dentro de esa clase, que es el cliente. Posteriormente se arranca la clase para cargar los recursos del juego para llegar al final a la clase PantallaMultijugador y empezar así la partida.

4.5.5. Implementación de la funcionalidad Bluetooth dentro del juego

Ya se ha cargado previamente tanto el socket como el modo en el que entra el dispositivo en la clase PantallaMultijugador mediante el método llamado *setBluetoothSocket*. Dicho método se encarga de crear una variable estática de la clase *BluetoothSocket*, llamada mSocket, con el socket importado y en una variable estática booleana se guardará si se actúa como servidor o cliente.

La clase PantallaMultijugador, una de las primeras cosas que tiene que hacer es activar el método *inicializarBT* que se crea dentro de esta clase. Este método creará dos objetos. Uno de tipo BufferedReader al que se le hará la correspondencia con el flujo de datos a leer del *socket*. El otro, del tipo PrintWriter se le hará correspondencia con el flujo de datos a escribir en el *socket*.

Posteriormente la clase PantallaMultijugador creará un nuevo hilo llamado empezarARecibirMensajes que ejecuta la rutina necesaria para leer los mensajes recogidos del flujo de datos a leer del *socket* mientras esté funcionando.

En la siguiente sección se describirá cómo es el modelo de los mensajes Bluetooth para que se comprendan los pormenores de cómo se realizan el envío y recepción de mensajes.

Cuando se llama al método *RecibirMensajes,* se crea una variable de tipo *string* a la que se le proporcionará el valor de la línea leída por el objeto BufferedReader. A esa variable se le aplica el método *trim* y el resultado de este método se establece como el nuevo valor de la variable.

El hilo empezarARecibirMensajes hará, mientras esté funcionando, una rutina que consiste en crear una matriz de String, insertar en esa matriz el

resultado del método *RecibirMensajes* donde se meterá cada dato en una de las casillas mediante el método *split* y arrancar el método *RutinaMensajeRecibido* trasladándole dicha matriz.

En la rutina del método *RutinaMensajeRecibido* se compararán los datos recibidos en la primera casilla de la matriz con unos caracteres. Cuando se encuentra una coincidencia, se ejecuta la rutina asignada para dicha coincidencia.

Para mandar mensajes se utiliza el método *EnviarMensaje* con un *String* asociado. La función de este método es introducir el *String* en el flujo de salida y ejecutar el método *flush* a dicho flujo.

4.5.6. Intercambio de mensajes Bluetooth

Mientras que el envío de mensajes se puede hacer en cualquier momento de la rutina general llamando al método *EnviarMensajes*, la recepción es consecuencia de un hilo que se genera e inicia al principio y comprueba continuamente si el flujo de entrada contiene datos.

Modelo del mensaje Bluetooth

El mensaje enviado a través del socket establecido contiene solamente una cadena *String*. Dicha cadena está dividida mediante comas con máximo de ocho secciones.

En su primera sección se introducirán los caracteres que indicarán la rutina que se quiere ejecutar. En el resto de secciones se introducirán los datos necesarios para ejecutar la rutina. Estos datos son introducidos en su equivalente de tipo *String* y así serán recibidos, por lo que si se trata de datos de un tipo distinto a *String*, habrá que usar un método para poder pasarlos a tipo *String*.

Mensajes de sincronización y cambios de estado

Es necesario transmitir y recibir distintos datos o momentos para poder sincronizar el inicio, interrupción o salida de la partida del juego. Dichos datos son incluidos dentro de las secciones cuya cabecera carca la acción.

1) Se pausa la partida

Este mensaje será enviado cuando uno de los dispositivos cambie su estado a Pausado sin añadir más datos. Solamente actúa si el dispositivo que recibe el mensaje está en estado Funcionando dentro de la partida. La Figura 4-10 muestra el aspecto que presenta el mensaje de pausa.

pausa							

Figura 4 - 10. Formato de los mensajes pausa

2) Se reinicia una partida tras mundo liberado o juego terminado

Una vez finalizada una partida tras haber conseguido la corona para liberar el mundo o bien haber terminado el juego con la muerte de uno o los dos protagonistas, en ambos dispositivos se da la posibilidad de reiniciar la partida. El primer dispositivo que pulse alguno de los botones que lleva a esas acciones enviará el mensaje de reiniciar. En la Figura 4-11 se puede observar el modelo de este mensaje.

reiniciar							

Figura 4 - 11. Formato de los mensajes reiniciar

3) Juego terminado

Cuando el juego pasa al estado GameOver, el dispositivo envía este mensaje para comunicar el cambio de estado. Este mensaje no contiene más datos, y tras recibirlo, la rutina comprueba si el dispositivo que lo recibe tiene el estado Funcionando, y si lo tiene, realiza los pasos necesarios para que su estado cambie a GameOver. Se puede ver la estructura de este mensaje en la Figura 4-12.

gameover							

Figura 4 - 12. Formato de mensajes gameover

4) Mundo liberado

Tras liberar el mundo el estado del juego pasa a Victoria. Tras ello el dispositivo enviará un mensaje para comunicarlo. Este mensaje solamente contiene la cadena victoria. Tras recibir este mensaje el dispositivo que lo recibe pasará su estado a victoria. Se observa el formato de este mensaje en la Figura 4-13.

victoria							

Figura 4 - 13. Formato de los mensajes victoria

5) Se inicia o retoma la partida

Si al iniciar una partida se pulsa la pantalla en uno de los dispositivos cuando éste se encuentra en estado Preparado, o se encuentra en estado Pausado y se pulsa el botón para volver a la partida, este dispositivo enviará el mensaje con la palabra adelante en la primera sección para comunicar al otro dispositivo el momento de iniciar o volver a la partida. Tras recibir este mensaje el dispositivo que lo recibe pone el estado del juego a Funcionando. La estructura de este mensaje se puede ver en la Figura 4-14.

adelante							

Figura 4 - 14. Formato de mensajes adelante

Mensajes de información durante la partida

1) Se actualiza un personaje

Cuando se mueve un personaje durante el juego, o dicho personaje salta o el personaje está explotando (cuando pierde una vida), el dispositivo enviará un mensaje para comunicarlo. En la primera sección del mensaje se encontrará

la cadena con la palabra personaje. En la segunda sección estará la información correspondiente a la estancia de dicho personaje, en la tercera la posición en el eje X de éste, en la cuarta la posición en el eje Y, en la quinta la posición en el eje Z, en la sexta la dirección del personaje, en la séptima su estado y en la octava el número de vidas restantes del personaje. El dispositivo que recibe el mensaje actualizará la posición del personaje correspondiente según los datos recibidos. La estructura del mensaje se ve en la Figura 4-15.

personaje	Estancia	posición X	posición Y	posición Z	dirección	estado	Vidas

Figura 4 - 15. Formato de mensajes personaje

2) Los dos personajes se unen

Cuando Head se encuentra sobre Heels y se pulsa el botón correspondiente para unirlos, los personajes Head y Heels se unen y el personaje controlado por el dispositivo que controlaba inicialmente a Head y que, por lo tanto, actúa como dispositivo servidor, será Head Over Heels. El dispositivo servidor enviará un mensaje para informar al dispositivo cliente de esto para que una a los dos personajes también. El mensaje contendrá la cadena unir en su primera sección. El dispositivo cliente no volverá a controlar al personaje Heels mientras que no reciba un mensaje desde el dispositivo servidor para separar a los dos personajes y permanecerá a la espera sin poder controlar ningún personaje. El formato de este mensaje se observa en la Figura 4-16.

unir							

Figura 4 - 16. Formato de los mensajes unir

3) El personaje Head Over Heels pierde una vida

Cuando este personaje pierde una vida, tanto Head como Heels perderán una vida y el dispositivo servidor enviará un mensaje con la palabra muerte en su primera sección sin ninguna información más. Al recibir este

mensaje, el dispositivo cliente actualizará las vidas de los dos personajes. Se observa el formato de este mensaje en la Figura 4-17.

muerte							

Figura 4 - 17. Formato de mensajes muerte

4) Los dos personajes se separan

Si los dos personajes están unidos y se pulsa el botón para separarlos en el dispositivo servidor, los personajes Head y Heels se separarán y se enviará un mensaje para informarlo, además, el dispositivo servidor pasará a controlar a Head. El dispositivo cliente recibirá un mensaje con la palabra separar en la primera sección sin ninguna otra información. Tras esto, el dispositivo cliente separa a los dos personajes y Heels volverá a ser controlado por el dispositivo en modo cliente. La estructura del mensaje es la que se ve en el Figura 4-18.

separar							

Figura 4 - 18. Formato de mensajes separar

5) Para un personaje

Si el usuario pulsa una posición en la pantalla que no sea ningún botón de control del personaje, al levantar el dedo se enviará el mensaje con la palabra parar en su primera sección para parar el movimiento del personaje controlado. Además, el estado del personaje pasa a ser PERSONAJE_STATE_STOP. El dispositivo que recibe este mensaje cambiará el estado del personaje que no está siendo controlado por dicho dispositivo a PERSONAJE_STATE_STOP. Se observa el modelo de este mensaje en la Figura 4-19.

parar							

Figura 4 - 19. Formato de mensajes parar

6) Actualiza la posición de un enemigo

En el momento que un enemigo (en este caso un mono) actualiza su posición en la pantalla, el dispositivo en modo servidor enviará un mensaje con la palabra mono en la primera sección para que, al recibir el mensaje, el dispositivo en modo cliente actualice la posición del mono correspondiente según la información recibida en el mensaje. En la segunda sección del mensaje se incluirá el índice del mono dentro de una lista de monos que se usa para distinguirlo en el caso de que haya varios monos en la misma estancia. En la tercera sección se encontrará la posición en el eje X del mono, en la cuarta la posición en el eje Y, en la quinta la posición en el eje Z y en la sexta la dirección del mono. Se puede ver la estructura del mensaje en la Figura 4-20.

mono	Índice	posición X	posición Y	posición Z	dirección		

Figura 4 - 20. Formato de mensajes mono

4.6. Introducción y selección de personaje

Estas dos secciones de la aplicación han sido realizadas usando los layouts correspondientes a sus clases. Las clases que definen estas secciones son la clase Introduccion y la clase Seleccion con sus layouts con el mismo nombre con la primera letra en minúscula.

4.6.1. Layouts

El layout de cada clase marca la disposición de cada texto, botón o imagen que se muestre. En esta aplicación solo se han usado para la disposición de botones e imágenes de fondo. Cada objeto debe poseer una identificación única. Si se necesita algún elemento externo, como puede ser el fondo de una imagen, debe indicarse la ruta de dicho objeto. Para cada objeto se puede definir la altura, anchura, orientación, márgenes y posición dentro de unos límites.

Con los botones, se define su tipo dentro del propio layout. Se ha seleccionado un botón representado por una imagen del tipo denominado ImageButton.

4.6.2. Pantalla de introducción

Se muestra al iniciar la aplicación y es definida por la clase Introduccion que lo primero que se hace es ajustar su orientación a modo apaisado.

Posteriormente se carga la disposición de la pantalla del layout asociado. Se asignan su imagen de fondo con el logo del juego y un botón compuesto por una imagen con el texto “Jugar” que al ser pulsado hará que se pase a la pantalla de selección de personaje. Al ser una clase que hereda de Activity, la presentación de esta clase se forjará en el método *onCreate*. Se hace que al pulsar el botón de “Jugar”, mediante el método *setOnclickListener,* se entre en el método *onClick.*

El método *onClick* indica qué hacer en caso de que el botón sea pulsado. En este caso se pasará a la pantalla de selección de personaje recurriendo a *Intent* y *startActivity* para iniciar la Activity Selección y finalizando la Activity Introduccion tras ello. Posteriormente comenzará toda la rutina explicada en la sección 4.5 dedicada al Bluetooth.

4.6.3. Pantalla de selección de personaje

Definida por la clase Seleccion. También hereda de la clase Activity y su presentación se establecerá en el método *onCreate*, en el que se establecerá su imagen de fondo según su layout asociado. También se iniciará el diálogo para seleccionar el personaje y, por lo tanto, si se juega en modo servidor o cliente mediante el método *crearDialogoModo*. En este método se asignará la imagen de fondo para el diálogo emergente para la selección del personaje.

En la clase Seleccion se inicia la rutina de Bluetooth ya explicada anteriormente en la sección 4.5 de este documento.

4.7. Guardar/Cargar datos

La carga y guardado de datos de la aplicación ha sido desarrollada en la clase Settings. Se ha utilizado el interfaz FileIO del framework de Mario Zechner para el guardado y carga de datos en ficheros de texto. Este interfaz es implementado en la clase AndroidFileIO también incluida en el framework antes mencionado. En esta clase se definen los métodos necesarios para la lectura y guardado en ficheros.

En la clase Settings se han implementado el método *load* para la carga de datos desde un fichero de texto, y el método *save* para guardar los datos en un fichero de texto. En este proyecto se ha decidido guardar los ficheros en memoria externa siempre que sea posible. Para ello se ha incluido el permiso WRITE_EXTERNAL_STORAGE en el fichero AndroidManifest.xml que permitirá a la aplicación guardar en memoria externa.

Para esta aplicación, en los ficheros solo se guarda la información referente a si el sonido del juego está activo o no. Para ello, en la clase Settings se crea una variable booleana llamada soundEnabled que estará inicializada a true la primera vez que se entre en el juego. La clase Settings también cuenta con una constante de tipo String llamada file cuyo valor es ".headoverheelsbt" que será el nombre que tendrán los ficheros para guardado y carga de datos en esta aplicación.

Al método *load* se le pasará como parámetro una instancia de FileIO. Éste intentará leer una línea del fichero de entrada y cargará la información correspondiente al sonido en la variable soundEnabled.

Al método *save* también se le pasa como parámetro una instancia de FileIO. Este método intentara escribir en una línea del fichero la información referente al sonido del juego obtenida de la variable soundEnabled.

4.8. Elementos y recursos del videojuego

Entre los elementos que componen la partida del juego se tiene, por un lado, el mapeado que formara la habitación o estancia y, por otro, los

elementos incluidos dentro de una estancia como, por ejemplo, los personajes controlados por el usuario, los enemigos o los obstáculos.

El mapeado estará compuesto por el suelo y las paredes de la estancia. Tanto estos elementos que componen la estancia como todos los demás elementos de la partida, extienden una clase base llamada Objeto.

Cabe destacar otra clase llamada ObjetoDinámico, que heredará de la clase Objeto y que servirá para definir los elementos que pueden moverse por las estancias durante la partida.

4.8.1. La clase Objeto

Es la clase básica de la que heredan todos los elementos que componen la partida. En esta clase se definen cuatro constantes que representan la dirección del objeto. Estas constantes son llamadas LEFT, RIGHT, UP y DOWN.

Por otro lado, se han definido ocho constantes que representan la clase de objeto. Esas constantes son: CORONA, HEAD, HEELS, HOH, E_MAPA, MONKEY, PINCHOS y PUERTA.

Además, a la clase se le añaden distintas características en forma de variables: posicionX, posicionY, posicionZ, esfera, clase, estancia y dir.

Las tres primeras características son de tipo *float,* esfera es del tipo Sphere, y las tres últimas son de tipo *int.*

El tipo Sphere es implementado en el *framework* y define una esfera a partir de un punto con coordenadas en tres dimensiones y el valor de su radio.

Las características posicionX, posicionY y posicionZ definirán su posición en la pantalla, la característica esfera una esfera que lo envuelve, clase la clase, estancia la estancia en la que se encuentra y dir su dirección.

Se ha implementado esta clase como Comparable. Con la redefinición del método *compareTo* se establece un orden según la suma de las variables posicionY y posicionZ. Esto se ha implementado con el fin de ser utilizado a la

hora de dibujar los elementos en la pantalla por el orden establecido. El objeto que tenga un valor mayor en la suma de posicionY y posicionZ será dibujado posteriormente.

Se han definido dos constructores para esta clase. En el primero de ellos se le pasan como parámetros el radio de su característica esfera y su clase. Construirá un objeto con la información relativa a la característica esfera y clase de este. En el segundo constructor, se le pasarán como parámetros los valores de sus características posicionX, posicionY, posicionZ, el radio de la característica esfera y el valor de su característica clase. Este constructor generará un objeto con esas características establecidas según los valores de los parámetros.

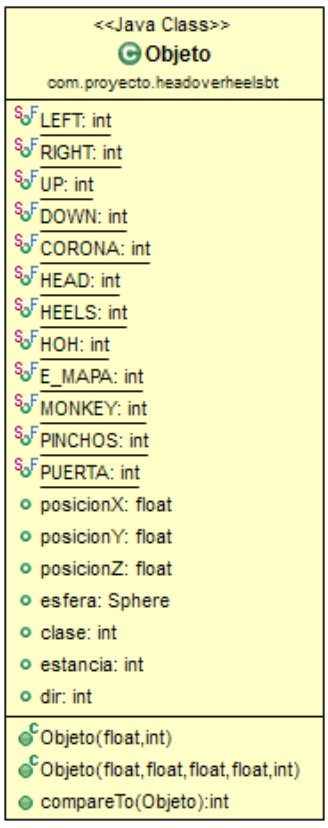

Figura 4 - 21. Clase Objeto

4.8.2. La clase ObjetoDinamico

Como se ha comentado anteriormente, esta clase hereda de la clase Objeto. Llamaremos a los objetos de esta clase objetos dinámicos a partir de

ahora. Se le han añadido las características animacion de tipo Animacion y avanceX y avanceY de tipo *float*. Las características avanceX y avanceY establecen el incremento de su posición en los ejes X e Y cuando esté moviéndose. Se puede decir que definen la longitud de un paso del objeto dinámico. La clase Animacion se explica a continuación.

Animacion

La clase Animacion define una animación para un objeto dinámico. Sirve para establecer qué imagen se dibujará por pantalla cuando un objeto dinámico se esté moviendo por la pantalla. Posee cuatro atributos: una lista de elementos de tipo Cuadro (este tipo se define dentro de la clase Animacion) llamada cuadros, un entero llamado indice*,* un *float* llamado tiempo, y otro *float* llamado duracion.

El tipo *Cuadro* cuenta con dos características: tiempo de tipo *float* e imagen de tipo Pixmap (implementado en el *framework* de Mario Zechner). Pixmap es, resumiendo, una imagen. También posee dos constructores. Uno sin parámetros de entrada que asigna el valor *null* a imagen y cero a tiempo. El otro asigna los valores que se le pasan como parámetro a las características del tipo *Cuadro*.

Por otro lado, la clase Animacion tiene un constructor en el que se inicializa la lista cuadros y se establece el valor de duracion a cero. También dentro del constructor se llama al método *iniciar.* El método *iniciar* establece los valores de las características tiempo e índice a cero.

La clase Animacion posee un método llamado *sumaCuadro* que añade a la lista cuadros un nuevo elemento y actualizará el valor de la característica duracion de la clase. Se le pasan como parámetros un Pixmap y un *float.*

Se ha definido un método llamado *anima* al que se le pasa un parámetro de entrada de tipo *float*. Este método actualizará el índice del cuadro dentro de la lista cuadros cuya imagen será dibujada en pantalla según el parámetro de entrada que será el tiempo que lleva moviéndose el objeto dinámico.

Por último, el otro método de la clase Animacion es *getCuadro*. Este método devolverá el Pixmap a pintar.

La clase ObjetoDinamico cuenta además con tres métodos constructores:

- Uno posee como parámetros de entrada el radio de la esfera del objeto dinámico, su clase, su avanceX y su avanceY. Llama a un constructor de la clase padre al que le pasa como parámetros los valores del radio de la esfera y la clase.
- Otro tiene como parámetros de entrada los valores de su posicionX, posicionY, posicionZ, el radio de la esfera, su avanceX y avanceY. Llama al constructor del padre pasándole como parámetros de entrada los valores para establecer la posicionX, posicionY, posicionZ, radio de la esfera y la clase.
- Otro idéntico al anterior exceptuando que, además se le pasa como parámetro de entrada la animacion del objeto dinámico.

Los dos últimos métodos que tiene la clase ObjetoDinamico son *avanza* y *siguientePosicion*.

El método *avanza* sirve para que el objeto dinámico de un paso. No posee ningún parámetro de entrada. Actualiza la posicionX y la posicionY según el valor de dir (dirección), avanceX y avanceY (longitud del paso). Además actualiza las coordenadas de la característica esfera.

Por último, el método *siguientePosicion* devolverá un Vector2. La clase Vector2 es tomada del *framework* de Mario Zechner y representa un punto en dos dimensiones (eje X y eje Y). El valor que devuelve este método dependerá de la dirección del objeto dinámico y la longitud de su paso. Este valor sería el de la siguiente posición del objeto dinámico tras dar un paso en la dirección actual.

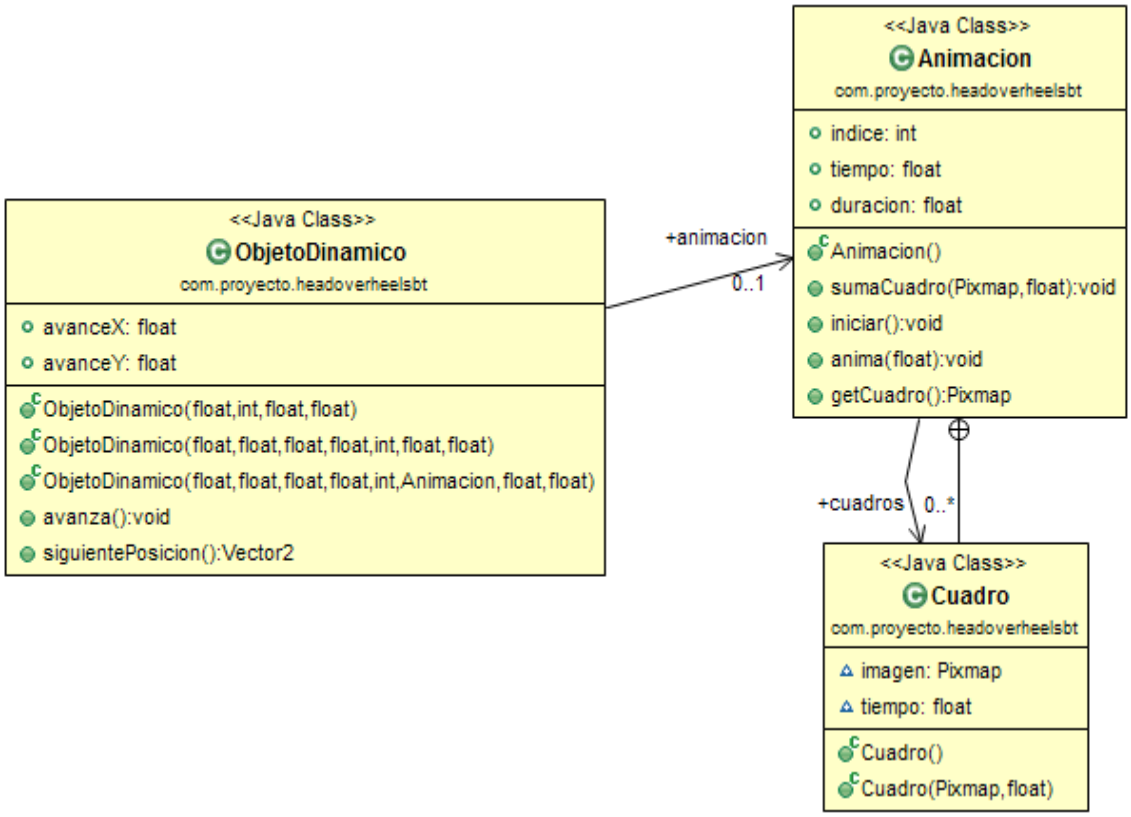

Figura 4 - 22. Clase ObjetoDinamico

4.8.3. La clase ElementoMapa

La clase ElementoMapa implementa el suelo y las paredes de la estancia. Hereda de la clase Objeto. En ella se definen tres constantes que son el tipo de elemento. Estas constantes son: SUELO, PARED_NORTE y PARED_OESTE. Se define además otra constante llamada ELEMENTO_RADIO. Es el radio del elemento.

Esta clase posee, además de las características heredadas de Objeto, una nueva característica de tipo *int* llamada tipo que será el tipo de elemento del mapa. Los objetos de esta clase que tengan como tipo SUELO serán una baldosa del suelo, las que sean del tipo PARED_NORTE una porción de la pared norte de una estancia y las que sean del tipo PARED_OESTE una porción de la pared oeste.

Tiene un constructor al que se le pasan como parámetros de entrada las coordenadas del elemento. Este constructor inicializa por defecto su

característica tipo a SUELO y además un método constructor de la clase Objeto a la que le pasa como parámetros las coordenadas, ELEMENTO_RADIO y E_MAPA.

Se implementan también en esta clase los métodos *setTipo,* que asigna un nuevo valor a la característica tipo, y *setPosicion* que le asigna unas nuevas coordenadas. Al método *setTipo* se le pasa como parámetro el nuevo tipo y a *setPosicion* los valores de las nuevas coordenadas en el eje X y en el eje Y.

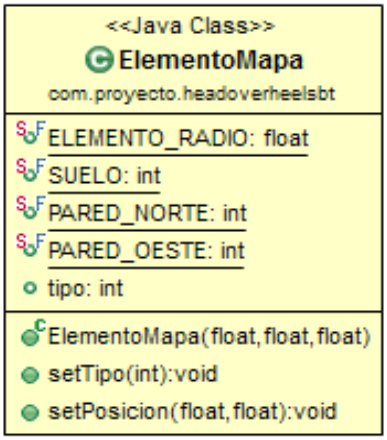

Figura 4 - 23. Clase ElementoMapa

4.8.4. La clase Mapeado

Esta clase implementa una estancia, que está compuesta por el suelo y las paredes. Posee una característica llamada ancho que será el ancho de la estancia. Ese ancho vendrá definido por el número de baldosas que tenga el ancho de la estancia más uno. Además, posee una matriz de elementos del tipo Elemento mapa llamada casillas y cuatro atributos de tipo Vector2 que serán las cuatro esquinas de la estancia.

La clase Mapeado tiene un constructor que inicializa las posiciones de todos los elementos que componen el mapeado y sus esquinas. Al constructor se le pasa como parámetro el ancho del mapeado.

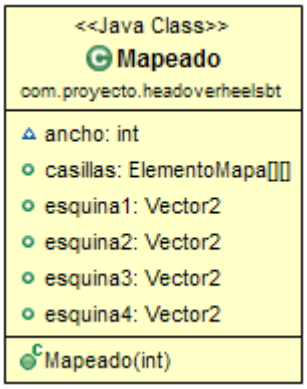

Figura 4 - 24. Clase Mapeado

4.8.5. La clase Personaje

La clase personaje implementa las características y comportamiento de los personajes controlables por el usuario. En este proyecto hay tres clases que heredan de esta. Estas clases son Head, Heels y Hoh. La clase Personaje es también heredera de la clase ObjetoDinámico.

En esta clase se definen tres constantes de tipo *int* que son las siguientes: PERSONAJE_STATE_STOP, PERSONAJE_STATE_MOVE y por último PERSONAJE_STATE_EXPLODING. Estas constantes representan el estado en el que está el personaje. Además, se ha definido la constante de tipo float PERSONAJE_EXPLODING_TIME que será el tiempo que durará la animación del personaje cuando pierde una vida.

A la clase Personaje se le han añadido las siguientes características:

- state: estado del personaje.
- lives: vidas del personaje.
- estanciaAnt: estancia anterior en la que se encontraba el personaje. Se utiliza para el cambio de estancia de un personaje cuando atraviesa una puerta.
- posIniX, posIniY y posIniZ: se utilizan para volver a dibujar un personaje cuando pierda una vida en la posición correcta.
- dirIni: para situar al personaje orientado hacia la dirección que indica cuando pierde una vida.

- enPuertaV y enPuertaH: de tipo booleano se usan para saber si un personaje está en una puerta.
- jumpin y fallin: indican si un personaje está saltando o cayendo respectivamente.
- aDerecha, aIzquierda, aArriba, aAbajo, aInicial: son de tipo Animacion y representan las animaciones de un personaje según la dirección que tenga el personaje.

Esta clase posee tres métodos constructores:

- El primero posee como parámetros de entrada el radio del personaje, la clase, y los valores de avanceX y avanceY para la longitud de un paso del personaje. Llama a un constructor del padre y además inicializa los valores de las características state, jumpin, fallin, enPuertaV y enPuertaH.
- El segundo constructor posee como parámetros de entrada los mismos que el primero y además se añaden los valores de sus características posicionX, posicionY, posicionZ y estanciaAnt. También llama a un constructor de la clase padre e inicializa los valores de los atributos dir, lives, state, estancia, estanciaAnt, posIniX, posIniY, posIniZ, jumpin, fallin, enPuertaV y enPuertaH.
- Por último, el tercero es idéntico al segundo, exceptuando que, además se le pasa como parámetro de entrada una animación que será la animación inicial que tendrá el personaje, es decir, será el valor del atributo aInicial.

Se ha definido para la clase Personaje un método llamado *setAnimacion,* que tiene un parámetro de entrada de tipo Animacion. Ese parámetro se le asignará al atributo animación del personaje.

También se han añadido los métodos *jump* y *fall*, que serán implementados en las clases herederas de la clase Personaje, y el método *hit* que pondrá el valor PERSONAJE_STATE_EXPLODING al atributo state del personaje.

Figura 4 - 25. Clase Personaje

A continuación se explicarán las tres clases herederas de la clase.

4.8.6. La clase Head

Es la que implementa al personaje Head. Heredera de la clase Personaje como se ha comentado anteriormente, la clase Head contiene tres constantes definidas de tipo *float*: HEAD_RADIO, PASO_X y PASO_Y. El radio de Head sería la primera constante, mientras que PASO_X y PASO_Y establecen la longitud del paso de Head.

Esta clase posee dos métodos constructores:

- El primero posee como parámetros de entrada las coordenadas del personaje y la estancia en la que se encuentra. Llama a un constructor del padre y carga de la clase Recursos los valores de las animaciones del personaje.
- El segundo se comporta de forma similar. Posee un parámetro más que es la animación inicial del personaje. Esta animación es un parámetro en la llamada al constructor del padre que se hace en este método.

Uno de los parámetros que se les pasa a los constructores del padre en los dos casos será el de la constante HEAD definida en la clase Objeto y será la clase de objeto.

Además, en la clase Head se sobrescriben los métodos *jump* y *fall* del padre que implementan el comportamiento del personaje cuando salta y cae.

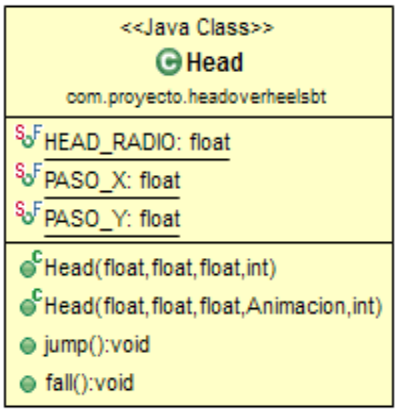

Figura 4 - 26. Clase Head

4.8.7. La clase Heels

Implementa el comportamiento del personaje Heels. Hereda de la clase Personaje y posee los mismos métodos que la clase Head con una implementación similar. Se definen también tres constantes de tipo *float* que son: HEELS_RADIO, PASO_X y PASO_Y. Como en la clase Head son el ancho del personaje y la longitud de su paso.

Uno de los parámetros que se les pasa a los constructores del padre en los dos casos será el de la constante HEELS definida en la clase Objeto y será la clase de objeto.

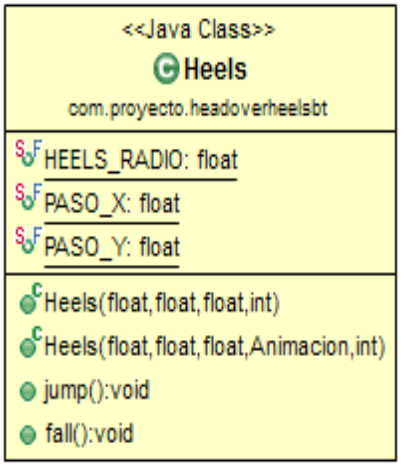

Figura 4 - 27. Clase Heels

4.8.8. La clase Hoh

Implementa el comportamiento del personaje Head over Heels. Es casi similar a las dos clases anteriores y también hereda de la clase Personaje. También tiene tres constantes que definen lo mismo que la clase Head y la clase Heels. Esas constantes son: HOH_RADIO, PASO_X y PASO_Y.

Existen dos diferencias en esta clase con respecto a las clases Head y Heels. La primera es que posee un constructor más que no tiene ningún parámetro de entrada. Ese constructor llama a un constructor del padre al que le pasa como parámetros los valores de las constantes HOH_RADIO, PASO_X y PASO_Y. Además este método constructor establece los valores de las animaciones del personaje.

Uno de los parámetros que se le pasará a los constructores del padre será el valor de la constante HOH que será la clase de objeto.

La otra diferencia respecto a las clases anteriores es que sobrescribe el método *avanza* de la clase ObjetoDinamico.

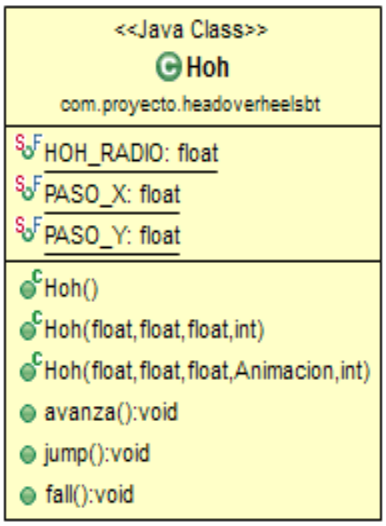

Figura 4 - 28. Clase Hoh

4.8.9. Clase Monkey

Implementa el comportamiento del enemigo Mono. Hereda de la clase ObjetoDinamico. Tiene tres constantes que son MONKEY_RADIO, PASO_X y PASO_Y. Tiene un método constructor con las coordenadas del mono y la estancia en la que está como parámetros de entrada. Este método llama a un constructor del padre e inicializa los valores de la dirección del personaje y la estancia en la que se encuentra.

Se le pasa como parámetro al constructor del padre el valor de la constante definida en la clase Objeto HOH que será la clase de objeto.

Figura 4 - 29. Clase Monkey

4.8.10. Clase Puerta

La clase puerta implementa las puertas para pasar de una estancia a otra durante la partida. Hereda de la clase Objeto. Tiene tres constantes que son: PUERTA_RADIO, PUERTA_LR y PUERTA_UD. Estas dos últimas definen el tipo de puerta, que será puerta del lado izquierdo o derecho de la estancia en el primer caso, y puerta del lado superior e inferior de la estancia en el segundo caso.

Se le añade un nuevo atributo llamado estanciaSig que será de tipo *int* y significará la estancia a la que pasará el personaje al atravesar la puerta. La clase puerta poseerá un solo método que es el constructor que llama a un constructor del padre al que le pasa entre otros parámetros el valor de la constante PUERTA de la clase Objeto como tipo de objeto.

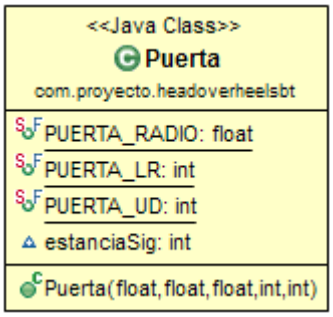

Figura 4 - 30. Clase Puerta

4.8.11. Clase Corona

La clase corona implementa la corona que deben conseguir los personajes para liberar el mundo. Tiene una constante que será el radio de su esfera y es llamada CORONA_RADIO.

Tiene un atributo llamado estancia que será la estancia en la que se encuentra.

Posee un método constructor con las coordenadas y la estancia como parámetros de entrada. Llama al padre pasándole entre los parámetros el valor

de la constante CORONA de la clase Objeto. Además, este método establece el valor del atributo estancia.

Figura 4 - 31. Clase Corona

4.8.12. Clase Pinchos

Esta clase implementa los pinchos, que son obstáculos que encuentra el personaje. Hereda de la clase Objeto y posee una constante que ha sido llamada PINCHOS_RADIO. Posee un constructor al que se le pasan las coordenadas de los pinchos. Ese constructor llama a su vez a un constructor del padre al que le pasa como parámetros las coordenadas, PINCHOS_RADIO y el valor de la constante PINCHOS de la clase Objeto.

Figura 4 - 32. Clase Pinchos

4.8.13. Recursos

En este juego se utilizan dos tipos de recursos: sonoros y gráficos. Los sonoros han sido tomados de la versión del juego Head over Heels de Jorge Rodríguez y Santiago Acha, exceptuando el sonido push.wav. Son guardados en la carpeta assets. Estos archivos se podrían dividir en música y sonidos. Los sonidos son:

- dead.wav: se utiliza en el momento en el que un personaje pierde una vida.
- gameOver.wav: se utiliza cuando uno de los personajes o los dos pierden todas sus vidas y se llega al estado de GameOver.
- jump.wav: se usa cuando el personaje controlado salta.
- push.wav: se usa cuando se pulsa un botón que no sea un botón de acción.
- victory.wav: se utiliza cuando se consigue la corona y se llega al estado de Victoria.

Se ha utilizado solo un archivo de música llamado music.ogg y es la música que suena durante la partida.

Los archivos gráficos también se guardan en la carpeta assets. Son de tipo png. La mayoría son también extraídos del juego de Jorge Rodríguez y Santiago Acha. Algunos han sido modificados con Gimp y otros han sido creados con la misma herramienta. Estos archivos gráficos son:

- blueButtom.png: botón azul de acción.
- corona.png: dibujo de la corona.
- GameOver.png: se usa cuando se llega al estado GameOver.
- gamepad.png: cruceta de dirección.
- head.png: dibujo de Head. Aparece junto al número de vidas de Head cuando este personaje esté siendo controlado.
- headNA.png: igual que el anterior pero en tonos grises. Se usa cuando Head no está siendo controlado.
- headD0.png, headD1.png, headD2.png, headL0.png, headL1.png, headL2.png, headR0.png, headR1.png, headR2.png, headU0.png, headU1.png y headU2.png: se usan para las animaciones de Head.
- heels.png: dibujo de Heels. Aparece junto al número de vidas de Heels cuando este personaje esté siendo controlado.
- heelsNA.png: igual que el anterior pero en tonos grises. Se usa cuando Heels no está siendo controlado.

- heelsD0.png, heelsD1.png, heelsD2.png, heelsL0.png, heelsL1.png, heelsL2.png, heelsR0.png, heelsR1.png, heelsR2.png, heelsU0.png, heelsU1.png y heelsU2.png: se utilizan para las animaciones de Heels.
- hohD0.png, hohD1.png, hohD2.png, hohD3.png, hohL0.png, hohL1.png, hohL2.png, hohL3.png, hohR0.png, hohR1.png, hohR2.png, hohR3.png, hohU0.png, hohU1.png, hohU2.png y hohU3.png: se utilizan para las animaciones de Head over Heels.
- inGameBackground.png: fondo negro durante la partida.
- items.png: compuesto por algunos botones, números de las vidas, menú de pausa y mensaje de confirmación para salir de la partida.
- losa.png: una baldosa del suelo durante la partida.
- monkey.png: imágenes del mono.
- MundoLiberado.png: aparece cuando se consigue la corona.
- paredx1.png: una porción de pared norte.
- paredy1.png: porción de pared oeste.
- personajeExplota0.png, personajeExplota1.png, personajeExplota2.png, personajeExplota3.png, personajeExplota4.png, personajeExplota5.png: se utilizan para la animación de un personaje cuando pierde una vida.
- pinchos.png: dibujo de pinchos.
- Preparado.png: se usa para el mensaje de estado Preparado al iniciar la partida.
- redButtom.png: Botón de acción rojo.
- safariDoorL.png: puerta de la pared oeste.
- safariDoorR.png: puerta de la pared este.
- safariFrame.png: marco de pantalla durante la partida.

Todos estos recursos se almacenan en la clase Recursos para poder facilitar el acceso a ellos. Esta clase contiene objetos públicos y estáticos que se encargan de contener los *Pixmap, Animacion, Sound* y *Music* de los recursos. Hay un objeto estático para cada imagen, sonido y animacion que se

ha cargado en la carpeta assets. Una animación consiste en un conjunto de imágenes.

4.8.14. Entre la introducción y el juego. Carga de recursos.

Tras la pantalla de selección de personaje, la aplicación entrará en la pantalla de la partida. En este apartado se verán los pasos que se preparan antes de entrar en las clases que llevarán el peso del juego.

Lo primero que hace la aplicación, una vez se haya seleccionado el personaje a controlar, será arrancar la clase HOHBT, que será la responsable de generar y reproducir la primera pantalla del juego. Se ha derivado de la clase AndroidGame e implementado el método *getStartScreen* con el que obtiene una instancia de la clase LoadingScreen que se comentará más adelante. Esta acción permitirá iniciar el juego junto con todos los elementos necesarios para que funcione, desde cargar todos los recursos gráficos y sonoros, entrada del usuario, cargar las opciones de un fichero, hasta el inicio del hilo de ejecución del bucle principal.

La clase LoadingScreen se obtiene a partir de la clase Screen. Se implementa un constructor que obtiene una instancia de Game que recibe del constructor de la clase padre gracias a que fue llamada usando el método *getStartScreen*.

Posteriormente, se implementa el método *update* que se utiliza para cargar los recursos y configuración del juego. Para los recursos se utiliza el método *Graphics.newPixmap* con el que se genera un nuevo Pixmap en los objetos definidos en la clase Recursos. Se especifica el formato del color que tienen los Pixmap. El formato del fondo negro de la partida será RGB565, mientras que el resto de imágenes será ARGB444. De esta forma se conserva memoria y se incrementa la velocidad del juego. También se crean las animaciones de los personajes usando el método *sumaCuadro* de la clase Animacion comentada anteriormente. Además, se crean los efectos de sonido y la música y se guardan en los objetos correspondientes de la clase Recursos.

Se cargarán los ajustes de configuración del juego recuperándolos desde un fichero con el método *load* de la clase Settings.

Por último, se llama al método *setScreen* para pasar a una Screen llamada PantallaMultijugador que es la clase donde se desarrolla todo el peso del juego.

4.9. Lógica del videojuego

El desarrollo del motor del juego se desarrolla todo en unas pocas clases, que estarán muy atareadas durante el transcurso de la partida. La primera clase llamada se ayudará de una clase que contendrá la mayoría de la lógica automática. Será utilizada para fijar los elementos de la partida al inicio y la irá llamando cada cierto tiempo (deltaTime) para que haga que todos los elementos realicen su acción automática correspondiente.

El modelado y la presentación gráfica se realizaran en la primera clase cargando el estado de la partida en ese momento y dibujándolo en pantalla cada cierto periodo de tiempo.

4.9.1. Lógica automática

La lógica automática se desarrolla en la clase Mundo. Lo primero que se hace en esta clase es definir una larga lista de constantes y variables.

Las constantes definen el estado del juego y las dimensiones de las instancias y son cinco:

- WORLD_STATE_RUNING: la partida está en transcurso.
- WORLD_STATE_GAME_OVER: la partida ha terminado muriendo algún o los dos personajes
- WORLD_STATE_VICTORY: la partida se ha terminado consiguiendo la corona.
- ANCHO_ESTANCIA_1: ancho de la estancia uno.
- ANCHO_ESTANCI_2: ancho de la estancia dos.

Las variables

Las variables se dividirán según su función para poder ser explicadas mejor conociendo el marco en el que actuarán.

Variables de personajes:

- head: de tipo Head y es el personaje Head.
- heels: de tipo Heels y es el personaje Heels.
- hoh: de tipo hoh y es el personaje Head over Heels.

Variables de estado de personajes:

- headOverHeels: de tipo boolean y será true si el personaje Head se encuentra sobre Heels y false en caso contrario.
- joinHOH: de tipo boolean, será true si los dos personajes están unidos y false en caso contrario.

Variables de elementos de la partida:

- corona: de tipo Corona representa a la corona para liberar el mundo.
- monkeys1 y monkeys2: son listas de elementos de tipo Monkey y contienen respectivamente los monos de la estancia uno y la estancia dos.
- puertas1 y puertas2: son listas de elementos de tipo Puerta y contendrán las puertas de la estancia uno y dos respectivamente.
- pinchosEstancia1: lista de elementos tipo Pinchos. Contiene los pinchos de la estancia 1.
- estancia1 y estancia2: de tipo Mapeado son los mapeados de las estancias uno y dos.

Variables para la lógica de Bluetooth:

- lista: es una lista de elementos tipo MensajeBT y contendrá los mensajes que se envían desde otra clase entre los dos dispositivos por Bluetooth.

- elservidor: de tipo boolean. Será true si el dispositivo está en la partida en modo servidor y false si está en modo cliente.

Otras variables:

- listener: de tipo WorldListener. Se usa para reproducir ciertos sonidos al ocurrir algún evento. Dentro de esta clase se implemente el interfaz WorldListener.
- state: de tipo int. Será el estado del juego.
- stateTime: de tipo float. Se usa para controlar que se realizan ciertas acciones en determinado tiempo.

Se ha creado la clase MensajeBT para que haga de puente entre la clase Mundo y la clase encargada de enviar los mensajes. En esta clase se contendrá la información necesaria para que el dispositivo que reciba el mensaje realice la acción correspondiente según el mensaje recibido.

Al crear la clase Mundo

El constructor de la clase posee un parámetro de entrada de tipo WorldListener. Lo primero que se hace al crear la clase Mundo será crear las instancias para las variables comentadas anteriormente. Algunas se crearán directamente en el constructor y otras a través de métodos auxiliares. Esos métodos auxiliares son:

- generarEstancia: Tiene un parámetro de entrada de tipo Mapeado. Inicializa la estancia que se le pasa como parámetro y genera su composición con su suelo y paredes.
- inicializaMonos: Inicializa las listas de monos en las estancias y las posiciones en las que se encuentran.
- inicializaPuertas: Inicializa las listas de puertas de las distintas estancias y las posiciones en las que se encuentran. También se establecerán las estancias que comunican las puertas.
- inicializaPinchos: Inicializa los pinchos de las estancias y las posiciones en las que se encontrarán.

También se crearán una serie de variables que definirán la posición inicial de los personajes y su dirección y animación iniciales.

Actualización de un personaje

El método *updatePersonaje* es el encargado de actualizar a cada personaje del juego. Se le pasará como parámetro el personaje y una variable de tiempo. Lo primero que hace este método es comprobar si el personaje está en estado PERSONAJE_STATE_EXPLODING, al que llega tras perder una vida. Si es así le restará una vida al personaje y se encargará de asignar la animación correspondiente a la explosión del personaje y animarlo y guardará un mensaje en la lista de mensajes para ser enviado por Bluetooth al otro dispositivo para informarle de que el personaje ha perdido una vida. También se encargará de volver a asignar al personaje la posición correcta tras su muerte llamando al método *asignaPosicion*.

Si el personaje no se encuentra en ese estado y está en el estado correspondiente a cuando se está moviendo, se encargará de mover al personaje comprobando que no salga de los límites de la estancia y no choque con otro personaje. También se encarga de realizar las acciones correspondientes cuando el personaje está saltando o cayendo.

Comprobando si los personajes chocan

Existe un método para comprobar si un personaje choca con otro para que el primero se pare y no traspase al segundo. Este método es el llamado *obstaculoPersonaje* que posee como parámetro de entrada el personaje que se está manejando y devolverá un valor booleano siendo verdadero si chocan y falso en caso contrario.

Actualización de los monos

El método encargado de actualizar los monos es *updateMonkeys.* Se encargará de actualizar la dirección en la que se mueve un mono de forma aleatoria y hacer que éste avance comprobando antes que no choca con ningún obstáculo y que no se sale de los límites de la estancia. También

guardará el mensaje correspondiente en la lista de mensajes para ser enviado al otro dispositivo para que actualice la posición de los monos.

Comprobando que un elemento dinámico no atraviese las paredes de una estancia

El método *sigPosValida* se encarga de comprobar que el objeto dinámico que se le pasa como parámetro no salga de los límites de la estancia que también se le pasa como parámetro. Devolverá un valor booleano, siendo true si la siguiente posición del objeto dinámico es válida o falso en caso de que la siguiente posición estuviera fuera de los límites según la dirección actual del objeto dinámico.

Comprobando las colisiones de un personaje

El método *checkCollisions* posee como parámetro de entrada un personaje y se encargará de comprobar las colisiones de ese personaje. Para ello usa tres métodos auxiliares:

- checkMonkeyCollisions: tiene como parámetro de entrada al personaje correspondiente y comprueba las colisiones del personaje con los monos de la estancia en la que se encuentra. Si choca con algún mono llamará a los métodos necesarios para restarle una vida y hacer que se reproduzca el sonido correspondiente.
- checkPuertaCollisions: se encarga de comprobar si el personaje que se le pasa como parámetro pasa por una puerta de la estancia en la que se encuentra. Si atraviesa la puerta se le asignará una nueva posición en la estancia correspondiente llamando al método auxiliar *asignaPosicion*.
- checkPinchosCollisions: también tiene como parámetro de entrada un personaje. Se encarga de comprobar si un personaje chocha con los pinchos. Si es así realizará las acciones necesarias para restar una vida al personaje y reproducir el sonido correspondiente a la muerte de un personaje.

Comprobando si Head se encuentra sobre Heels

El método encargado de realizar esta comprobación ha sido el denominado *checkHeadOverHeelsCollision*. Este método se encargará de actualizar la variable booleana headOverHeels, poniéndola a verdadero en el caso de que Head esté situado encima de Heels y a falso en caso contrario. También se encargará de hacer que Head se mantenga en la altura correspondiente cuando esté sobre la cabeza de Heels y no caiga mientras lo tenga debajo.

Consiguiendo la corona

El método *checkCoronaCollisions* de la clase mundo es el encargado de comprobar si un personaje controlado por el usuario consigue la corona. Si un personaje toca la corona pondrá WORL_STATE_VICTORY como estado del juego.

Fin de la partida

El método *checkGameOver* comprobará si alguno de los personajes llega a tener cero vidas. En ese caso pondrá WORLD_STATE_GAME_OVER como valor del estado de juego.

Uniendo a Head y Heels

Existe un método en la clase Mundo que se encargará de unir a los dos personajes para dar como resultado al personaje Head over Heels. Ese método es *unirHOH*. En este método además se actualizan los valores de las posiciones en las que volverán a ser dibujados los personajes en el caso de que pierdan una vida. También se actualizará la animación inicial del personaje Head over Heels.

Separando a Head over Heels

El método de esta clase que se encarga de separar a Head y Heels es el llamado *separarHOH.* Este método asignará las posiciones de Head y Heels así como su dirección y animación.

Moviendo el mundo

Por último se pasará a comentar el método que realmente hace mover al mundo y debe ser llamado cada cierto tiempo (deltaTime). Ese es el método *update*. Lo primero que hace el método es comprobar si Head y Heels están unidos. Si lo están comprobará si el dispositivo esta en modo cliente o servidor. Si está en modo servidor actualizará al personaje Head over Heels. Posteriormente comprobará las colisiones de este personaje y después actualizará los monos. Después de hacer esto guardará un mensaje en la lista de mensajes para ser enviados por Bluetooth indicando la actualización del personaje.

Si los dos personajes están juntos pero el dispositivo está en modo cliente, animará al personaje Head over Heels en el caso de que el estado de este sea PERSONAJE_STATE_MOVE o el PERSONAJE_STATE_EXPLODING.

Si los dos personajes están separados se comprobará si el dispositivo está en modo cliente o servidor. Si está en modo servidor actualizará al personaje Head y si el personaje Heels se encuentra en el estado PERSONAJE_STATE_MOVE o PERSONAJE_STATE_EXPLODING, lo animará. Posteriormente se actualizarán los monos y después se comprobarán las colisiones de Head. Tras esto se guardará un mensaje en la lista de mensajes Bluetooth en el que se informará de la actualización del personaje Head.

Si el dispositivo está en modo cliente, cuando los dos personajes están separados lo primero que se hará será actualizar a Heels. Después se procederá a comprobar si Head se encuentra en el estado PERSONAJE_STATE_MOVE o PERSONAJE_STATE_EXPLODING animándolo en caso afirmativo. Tras ello se comprobarán las colisiones de Heels y se guardará un mensaje en la lista de mensajes Bluetooth para ser enviados al otro dispositivo indicando la actualización del personaje Heels.

Lo siguiente que se hará en el caso de que los dos personajes estén separados tanto en modo cliente como en modo servidor será comprobar si Head está sobre Heels.

Después de todo esto, y en todos los casos, tanto si están los dos personajes juntos como si no lo están, o si está el dispositivo en modo cliente o modo servidor, se comprobarán las colisiones de los personajes con la corona y también si alguno de los personajes ha perdido todas sus vidas.

4.9.2. Interacción del usuario

En la clase PantallaMultijugador se manejará toda la interacción del usuario. Se podría decir que esta clase es el corazón del juego ya que, además, es la que determina en qué estado se encuentra la partida y qué se debe hacer en ese estado. Esta clase también desarrolla todo el modelado gráfico.

Esta clase hereda de Screen. Eso quiere decir que tendrá cinco métodos por defecto que son: *update*, donde se marcaran las acciones a realizar; *present* que manejará todo el modelado gráfico y que se desarrollará en el apartado 4.9.3; *pause*, que marcará que acciones hacer en caso de interrupción del juego; *resume*, que hará las rutinas en el caso de que se retorne a la partida tras la interrupción; y por ultimo *dispose*, que elimina la clase y libera los recursos y en el que no se realizará ninguna modificación.

Lo primero que se hace en esta clase es definir una lista de variables y constantes. Se ha generado un nuevo tipo de variable que se ha denominado EstadoJuego y cuyos posibles valores son: Preparado, Funcionando, Pausado, GameOver, Salir y Victoria. Las variables globales de la clase son:

- estado: Variable de tipo EstadoJuego que representa el estado en el que se encuentra la partida.
- mundo: Es una variable de tipo Mundo y es a la que se llamará para que realice los cambios automáticos.
- worldListener: de tipo WorldListener y servirá para reproducir el sonido de muerte cuando un personaje pierda una vida.

- comienza: Booleano que se activará cuando empiece la partida.
- music: Música para la partida
- pers: Personaje controlado por el usuario.
- pers2: Personaje no controlado por el usuario.
- pauseBounds: De tipo Rectangle y es la zona de la pantalla reservada para el botón de pausa.
- resumeBounds: De tipo Circle y es la zona de la pantalla del botón de volver a la partida en el menú de pausa.
- quitBounds: Zona de la pantalla para el botón para salir del juego en el menú de pausa. Es de tipo Circle.
- soundBounds: Zona de la pantalla para el botón para activar o desactivar el sonido. Es de tipo Circle.
- gameOverBounds: Zona de la pantalla para reiniciar la partida en la pantalla de Juego Terminado.
- jumpBounds: Zona de la pantalla del botón de salto.
- joinBounds: Zona de la pantalla del botón para unir o separar a los personajes.
- leftBounds: Zona de la pantalla para la dirección izquierda de la cruceta direccional.
- upBounds: Zona de la pantalla para la dirección arriba de la cruceta direccional.
- rightBounds: Zona de la pantalla para la dirección derecha de la cruceta direccional.
- downBounds: Zona de la pantalla para la dirección debajo de la cruceta direccional.
- lista: Lista de mensajes Bluetooth.
- objetosApintar: Lista de objetos a pintar por pantalla.

Posteriormente a las anteriores se han definido una serie de variables para la comunicación Bluetooth entre los dispositivos.

Preparativos, previo, pausa y salir

Lo primero que se hace la clase es cargar el recurso de la música de la partida de la clase Recursos. Si la opción de audio está activa la reproducirá.

Posteriormente sobrescribirá el método *muerte* del interfaz WorldListener para que reproduzca el sonido adecuado. Posteriormente se asignará la zona de pantalla para cada botón. Después de esto se pondrá el estado de juego a Preparado e inicializará la variable objetosApintar. Tras realizar esto se llamará a tres métodos: *cargarMundo*, *inicializarBT* y *empezarARecibirMensajes*.

El método *cargarMundo* inicia en la variable mundo una nueva instancia de la clase Mundo. Después indicará si es el servidor o el cliente. En caso de que sea el servidor se le asignará a pers el personaje Head y a pers2 el personaje Heels. En caso de que sea el cliente, pers será Heels y pers2 será Head.

Los métodos *inicializarBT* y *empezarARecibirMensajes* arrancarán la rutina para el envío de mensajes Bluetooth entre los dispositivos.

Cuando finalizan todas las rutinas previas se arranca el método *update*. A este método se le proporciona el valor del tiempo que ha pasado desde la última vez que fue llamado (deltaTime). Este método compara el estado en el que se encuentra el juego y llama al método que corresponda. Si el método que se arranca es el correspondiente al estado Funcionando se le proporciona el tiempo deltaTime.

El método *updatePreparado* es el que será llamado desde *update* si el estado es Preparado. Este método se mantiene a la espera hasta que el usuario toque la pantalla. Cuando esto ocurra, cambiará el valor de la variable llamada comienza a verdadero y enviará un mensaje para informar al otro dispositivo de que comienza la partida. También cambiará el estado a Funcionando.

El método que será el llamado desde *update* cuando se encuentre en el estado Pausado es *updatePausado.* Comprobará si el usuario ha pulsado la pantalla. Si esto ocurre identificará qué botón del menú de pausa se ha pulsado y realizará la acción correspondiente. Si se ha pulsado en resumeBounds pondrá el estado a Funcionando. En el caso de que se pulse sobre quitBounds se pondrá el estado del juego Salir. Si se pulsa sobre soundBounds se activará

o desactivará la música y guardará la nueva configuración de sonido en un fichero de texto.

El método *updateSalir* es el que se llama desde *update* cuando se encuentra el juego en estado Salir. Lo primero que hace este método es asignar las zonas de la pantalla correspondientes a dos botones. Si se pulsa el correspondiente a salir de juego se llamará al método *reiniciar* que se encargará de hacer las acciones necesarias para reiniciar la partida. Además, enviará un mensaje por Bluetooth para que el otro dispositivo también reinicie la partida. Si se pulsa el botón correspondiente a no salir de la partida se pondrá el estado de juego Pausado y se enviará un mensaje por Bluetooth para que el otro dispositivo ponga su estado a Pausado.

La partida

El método *updateFuncionando* es el encargado de actualizar la partida mientras se encuentre en estado Funcionando. Es el llamado desde *update* cuando se encuentra en ese estado. Este método lo primero que hace es comprobar si el personaje que está siendo controlado por el dispositivo se encuentra en estado PERSONAJE_STATE_EXPLODING. Si no lo está leerá los eventos de pulsado o levantado del usuario en la pantalla. Si el evento es de pulsado, identificará si se ha pulsado sobre alguno de los botones de acción, cruceta de dirección o pausado, y realizará la acción correspondiente. Si se pulsa sobre alguna de las direcciones de la cruceta de dirección establecerá la dirección del personaje, pondrá su estado a PERSONAJE_STATE_MOVE y le asignará la animación correspondiente. Si se pulsa sobre jumpBounds comprobará si el personaje controlado por el usuario está saltando o cayendo. Si no lo está pondrá el valor de la variable jumpin del personaje a true y se reproducirá el sonido correspondiente a un salto. Si se pulsa sobre joinBounds se comprobará si los dos personajes están unidos. Si lo están se comprobará si el personaje está cayendo o saltando y si no lo está hará las acciones necesarias para separar a los personajes y enviará un mensaje al otro dispositivo para informarle. Si no están juntos y el personaje Head está sobre Heels hará las acciones necesarias para unir a los personajes

y posteriormente enviará un mensaje al otro dispositivo para informar de esta acción.

Posteriormente, el dispositivo comprueba si se ha pulsado sobre la zona de pauseBounds. En caso de que sea así, se pondrá el estado de la partida a Pausado y llamará al método *enviarPausado* que se encargará de enviar un mensaje al otro dispositivo para que se ponga en pausa. Además, pondrá a los dos personajes (el controlado por el usuario y el no controlado por éste) su estado a PERSONAJE_STATE_STOP.

Después este método comprobará si ha ocurrido un evento de levantado en la pantalla y este ha ocurrido fuera de la zona de joinBounds y jumpBounds, es decir, fuera de los botones de acción. En caso afirmativo pondrá al personaje controlado el estado PERSONAJE_STATE_STOP.

Tras comprobar los eventos táctiles de la pantalla, el método llamará al método *update* de la clase mundo pasándole como parámetro deltaTime. A continuación comprobará si el estado de mundo es WORLD_STATE_VICTORY o WORLD_STATE_GAME_OVER poniendo el estado de la partida a Victoria o GameOver respectivamente. Lo último que hace el método *updateFuncionando* es llamar al método *enviarMensajesMundo* que se encargará de enviar al otro dispositivo todos los mensajes guardados en la lista de mensajes Bluetooth de mundo en ese momento y limpiar esa lista.

Victoria o derrota

El método *updateGameOver* será llamado cuando el estado de la partida sea GameOver. Lo primero que realiza es enviar un mensaje al otro dispositivo para que este actualice el estado de la partida a GameOver. Después reconocerá los eventos táctiles de la pantalla. Si se pulsa sobre gameOverBounds reiniciará la partida llamando al método *reiniciar* y enviará un mensaje al otro dispositivo para que reinicie la partida.

Por otro lado, el método *updateVictoria* será ejecutado cuando el estado de la partida sea Victoria. Este método reconocerá si el usuario pulsa sobre la zona gameOverBounds y si esta zona ha sido tocada reiniciará la partida llamando al método *reiniciar* y enviará un mensaje al otro dispositivo para que reinicie la partida.

4.9.3. Modelado gráfico

En la clase PantallaMultijugador se implementa todo el apartado gráfico de juego. El método *present* se encargará de decidir lo que hay que pintar en la pantalla. Primero llamará al método *PintarMundo* al que le pasa como parámetros la variable mundo y deltaTime. El método *PintarMundo* primero cargará en una variable los gráficos del juego. Posteriormente dibujará el fondo negro y el marco de la pantalla. Después dibujará los iconos de los personajes junto a sus vidas. Tras esto, dibujará la estancia actual del personaje controlado por el dispositivo llamando al método *pintarMapeado* que se encargará de esto. Posteriormente guardará todos los objetos a dibujar en una lista en el orden en el que estos tienen que ser dibujados en pantalla. Cuando termine de cargar los objetos en la lista, los dibujará llamando al método *pintarObjetos* que es el encargado de dibujarlos. El método *pintarObjetos* identificará la clase del objeto a dibujar y después pintará en pantalla el recurso correspondiente.

Pintura de los estados

Después de realizar esto, el método *present* identificará el estado del juego y llamará al método correspondiente según ese estado.

Si se encuentra en el estado Preparado se llamará al método denominado *pintarPreparado* que cargará en una variable los gráficos del juego y dibujará por pantalla el recurso correspondiente.

Si se encuentra en el estado Funcionando se ejecutará el método denominad *pintarFuncionando* que cargará en una variable los gráficos del juego y dibujará por pantalla la cruceta de dirección, los botones de acción y el botón de pausa.

En el caso de que se encuentre en estado Pausado se llamará al método *pintarPausado* que cargará en una variable los gráficos del juego y pintará el recurso correspondiente.

Si se está en el estado Salir se llamará al método *pintarSalir* que cargará en una variable los gráficos del juego y dibujará el recurso correspondiente.

El método *pintarGameOver* será llamado cuando se encuentre el juego en el estado GameOver. Este método cargará en una variable los gráficos del juego y pintará el recurso correspondiente para este estado y el botón que será pulsado para reiniciar la partida.

El método *pintarVictoria* funciona de la misma forma que el método anterior. Será llamado cuando el juego se encuentre en estado Victoria.

Capítulo 5. Plan de pruebas

En este capítulo serán detalladas las comprobaciones realizadas para verificar el correcto funcionamiento de la aplicación desarrollada. Para realizar dichas pruebas se han utilizado varios terminales. En concreto se han utilizado los terminales de Samsung: Galaxy Young GT-S6310, Galaxy S 2 GT-I9100G y Galaxy Tab 2 7.0 GT-P3110.

5.1. Pruebas en los terminales

Antes de desarrollar la funcionalidad Bluetooth de la aplicación las pruebas se realizaban con un solo terminal. Tras añadir esta funcionalidad, siempre se han requerido dos terminales, ya que para poder comenzar la partida un terminal necesita estar emparejado con otro. Las comprobaciones que se relatan a continuación han sido realizadas en los tres terminales anteriormente comentados y emparejándolos dos a dos con todas las combinaciones posibles.

Comprobaciones realizadas:

- El arranque de la aplicación es correcto.
- Aparece la pantalla de introducción con normalidad, con la imagen de fondo deseada y con el botón para pasar a la siguiente pantalla con su imagen de apareciendo de forma correcta.
- El botón funciona correctamente y al pulsarlo se pasa a la pantalla de selección de personaje.
- Al pasar a la pantalla de selección de personaje, aparece la imagen de fondo correcta y el cuadro de diálogo de selección de personaje con los textos e imagen correctos.

- Si entrar en la pantalla de selección de personaje el Bluetooth del dispositivo está desactivado, aparecerá un mensaje para solicitar la activación del Bluetooth.
- Si al aparecer este mensaje se pulsa sobre el botón con el texto “Si”, el Bluetooth del dispositivo se activa correctamente y desaparece el cuadro de diálogo para activar el Bluetooth.
- Si se pulsa el botón con el texto “No” el mensaje desaparece.
- Si al entrar en la pantalla de selección de personaje el Bluetooth está activo, aparece la imagen de fondo correcta y sobre esta, el cuadro de diálogo para seleccionar personaje con la configuración correcta.
- Se pulsa sobre el botón con el texto “HEAD” y desaparece el cuadro de diálogo para seleccionar personaje y se mantiene a la espera.
- Se pulsa sobre el botón con el texto “HEELS” y aparece un cuadro de diálogo con una lista de dispositivos a emparejar.
- Se pulsa en ese cuadro de diálogo sobre el botón buscar y se rastrea en busca de nuevos dispositivos para emparejar que tengan el Bluetooth activo.
- Se selecciona un dispositivo de la lista y aparece un mensaje en la pantalla de los dos dispositivos para confirmar el emparejamiento.
- Se pulsa el botón aceptar en los dos dispositivos y se entra en el juego apareciendo la pantalla dibujada de forma correcta y el mensaje que solicita que se pulse la pantalla para comenzar la partida.
- Mientras no se pulsa la pantalla permanece a la espera.
- Si se toca la pantalla de uno de los dos dispositivos comienza la partida.
- Al comenzar la partida todos los elementos están pintados de forma correcta, tanto elementos del juego como los controles y en la pantalla donde hay un mono, éste se mueve correctamente.

- Se comprueba que el mono no sale de los límites de la estancia ni pasa sobre la corona y cambia de dirección de forma aleatoria con la frecuencia deseada. Cambia de dirección al chocar contra algún límite de la estancia o contra la corona.
- Se pulsa sobre el botón de pausa y en los dos terminales aparece el menú de pausa con los dibujos correctos y en el formato correcto.
- Se pulsa sobre el botón de activar o desactivar sonido dentro del menú de pausa y el sonido se activa o desactiva de forma correcta.
- Se pulsa sobre el botón de volver a la partida y la partida continua en los dos terminales desapareciendo en los dos el menú de pausa.
- Se pulsa sobre las distintas direcciones de la cruceta direccional y el personaje controlado se mueve de forma correcta.
- Mientras se mantiene el botón de una dirección el personaje controlado continúa moviéndose en esa dirección.
- Si deja de pulsarse un botón de la cruceta direccional el personaje se para.
- Se comprueba que los personajes se paran al chocar con las paredes de la estancia pero sigue animado de forma correcta.
- Se comprueba que los dos personajes pasan de una estancia a otra al pasar por una puerta y se dibuja por pantalla siempre la estancia en la que se encuentra el personaje controlado por el dispositivo.
- Si los dos personajes están en la misma estancia, en los dos terminales aparecen la misma pantalla.
- Los movimientos que se hacen con un personaje en un dispositivo aparecen reflejados en el otro dispositivo de la misma forma cuando los dos personajes están en la misma estancia. Las pantallas de los dos dispositivos están totalmente sincronizadas.
- Se pulsa sobre el botón de salto en los dos dispositivos y los dos personajes saltan correctamente.

- Se comprueba que Head salta más que Heels y el segundo es más rápido que el primero.
- Cuando se controla a Head el dibujo del personaje que se encuentra junto al número de vidas que éste posee aparece a color y el dibujo de Heels en tonos grises.
- Cuando se controla a Heels el dibujo de Heels junto a sus vidas aparece a color y el dibujo de Head junto a sus vidas en tonos grises.
- Cuando un personaje encuentra como obstáculo, mientras está andando, al otro personaje, se para. Los personajes no pueden traspasarse.
- Si Head salta sobre Heels permanece sobre él.
- Cuando Head está sobre Heels y se mueve, Head cae al suelo cuando no se encuentra situado sobre Heels.
- Se pulsa el botón para unir personaje cuando Head está sobre Heels y los dos personajes se unen correctamente.
- Se pulsa el botón de separar cuando los dos personajes están juntos y se separan correctamente.
- Cuando los dos personajes están juntos se comprueba que el usuario que juega en modo cliente no puede manejar a ninguno.
- Se comprueba que el usuario que juega en modo servidor puede manejar al personaje Head over Heels cuando Head y Heels están unidos.
- Todos los controles del personaje Head over Heels funcionan correctamente.
- Se ha comprobado que Head over Heels no sale de los límites de la estancia y se para al chocar con una pared.
- Comprobado que Head over Heels pasa por las puertas y cambia de estancia de forma correcta.
- Cuando Head y Heels están unidos los dibujos que los representan junto a sus vidas están los dos en color.
- Se ha comprobado con los tres personajes controlables que al chocar contra los pinchos pierden una vida y que esto se refleja

de forma correcta en todos los casos en el número de vidas que aparecen en la pantalla.

- Se comprueba que si Head o Head over Heels saltan por encima de los pinchos consigue esquivarlos si lo hace de forma correcta. Si el salto se queda corto y cae sobre los pinchos pierde una vida.
- Si el personaje Head over Heels choca contra los pinchos o contra el mono, tanto Head como Heels pierden una vida.
- Se comprueba que cuando un personaje choca contra el mono pierde una vida.
- Se comprueba que, cuando se maneja a Head over Heels y se está en un salto, los personajes no se separan si se pulsa el botón para separarlos.
- Se comprueba que cuando un personaje pierde una vida aparece la animación correspondiente a la explosión del personaje.
- Cuando un personaje pierde una vida y aún le quedan vidas, vuelve a aparecer dibujado en la posición correcta.
- Si uno o los dos personajes pierden todas sus vidas se comprueba que el juego termina y aparece la pantalla correcta y se reproduce el sonido correcto cuando el audio del juego está activo.
- Se ha probado que todos los audios del juego funcionan correctamente si el sonido está activado en los momentos correctos. Cuando se pulsa el botón de salto suena el audio de salto y cuando un personaje pierde una vida suena el audio correcto. La música se inicia al iniciar la partida si el audio está activo.
- El sonido que debe sonar al pulsar el botón de pausa durante la partida, o algún botón del menú de pausa cuando el audio está activo, se reproduce correctamente.
- Si se consigue la corona durante la partida aparece la pantalla de Mundo Liberado y se reproduce el sonido correcto cuando el audio está activo.

- Tanto en la pantalla de Juego Terminado como en la de Mundo Liberado, aparecen los botones correctos y al ser pulsados se reinicia la partida de forma correcta.
- En el menú de pausa se pulsa el botón para reiniciar la partida y el juego se reinicia de forma correcta.
- En el caso de coincidir varios sonidos, suenan a la vez.
- Se comprueba que al salir y volver a entrar en el juego se guarda la última configuración de audio activa.
- Cuando el personaje controlado es Head over Heels y este cambia de una estancia a otra, en el dispositivo que no controla al personaje también se produce el cambio de pantalla y puede ver en todo momento los movimientos del personaje.

5.2. Análisis de resultado

Las pruebas realizadas en la última versión compilada de la aplicación muestran la ejecución esperada. Se han probado todos los controles de todos los personajes en distintos dispositivos y funcionan correctamente. La sincronización entre los dos dispositivos funciona correctamente y los elementos móviles durante la partida se mueven de forma fluida en la mayoría de las ocasiones. Se han observado ciertas ralentizaciones y movimientos bruscos en los movimientos de los personajes y el mono en algunos casos.

Por otro lado, al haber implementado las colisiones en el juego con esferas, a veces las colisiones no son del todo precisas ya que hay algunos elementos que no son muy similares en su forma a una esfera.

En el dispositivo Samsung GT-S6310, al buscar dispositivos para emparejarse, en la lista aparecen dispositivos repetidos.

Capítulo 6. Conclusiones y trabajo futuro

6.1. Conclusiones

La aplicación realizada en este proyecto de fin de carrera es un videojuego multijugador basado en el original para un solo jugador Head over Heels, que fue desarrollado por Ocean Software Ltd. en 1987. Se ha realizado para dispositivos móviles con sistema operativo Android, y para la funcionalidad multijugador se ha utilizado la tecnología Bluetooth. Se ha tomado como referencia las características gráficas y sonoras de la versión de este videojuego para PC que realizaron Jorge Rodríguez Santos y Santiago Acha Jiménez con algunas modificaciones y añadiendo algunos elementos nuevos para adaptarlo para ser jugado en dispositivos móviles con pantalla táctil. El videojuego realizado es exclusivamente multijugador y en concreto se ha implementado para dos jugadores. No es un videojuego completo, sino un prototipo con dos estancias y en el que se pueden controlar tres personajes.

Se ha implementado de forma que un dispositivo funcione como servidor en la partida y el otro como cliente. El dispositivo en modo servidor comenzará manejando a Head y el cliente a Heels, y al unirse los dos personajes, el personaje resultante será manejado por el dispositivo servidor.

Todos los elementos que aparecen en los niveles aparecen en el videojuego original. Para este proyecto cada una de las estancias posee una puerta que comunica a las dos. En una estancia se ha situado una serie de pinchos para demostrar el comportamiento de las colisiones de los personajes con obstáculos estáticos. Mientras tanto, en la otra estancia se ha situado un mono que irá desplazándose aleatoriamente por la estancia. Este mono se ha añadido con el fin de probar las colisiones de los personajes controlados por el

usuario con un enemigo dinámico. Además, se ha añadido una corona en una de las estancias para que el juego termine al conseguirla.

Se han añadido controles táctiles en la pantalla para controlar los personajes: una cruceta para controlar el movimiento del personaje en cuatro direcciones, un botón para hacer saltar al personaje y un botón para unir y separar a los personajes aunque éste solo tendrá funcionalidad en el dispositivo que controla a Head al iniciar la partida. El personaje puede moverse y saltar en cuatro direcciones sin salir de los límites que son delimitados por las paredes de las estancias. Podrá pasar de una estancia a otra atravesando las puertas.

Se han implementado animaciones para los tres personajes y para el mono. Entre esas animaciones se ha añadido una que simula la desintegración de un personaje cuando pierde una vida. Además, se han incluido una serie de efectos sonoros que se reproducirán al ocurrir determinados eventos. También cuenta la aplicación con una melodía. Los efectos de sonido y música sonarán si está habilitada la opción de sonido. Esta opción puede activarse o desactivarse desde el menú de pausa.

Los menús, botones, ventanas y los diferentes mensajes de texto que aparecen en la aplicación se han diseñado con un aspecto de dibujo animado y con un estilo de representación más acorde con las aplicaciones actuales que las que presentaba el videojuego original. También se ha diseñado un inventario en el marco de pantalla que aparece durante la partida algo diferente al que aparecía en el videojuego original. En él se han situado unos iconos de los personajes junto a sus vidas en una plataforma. Se han añadido varias plataformas con la finalidad de tener la posibilidad de añadir en un futuro objetos que otorguen nuevas habilidades a los personajes cuando los consigan como en el videojuego original.

Se ha empleado un lenguaje de alto nivel a la hora de elaborar el código de la aplicación. En concreto se ha utilizado el lenguaje Java para Android. Al programar la aplicación con un lenguaje de estas características se facilita la tarea de confeccionar el código, ya que este se compone de instrucciones

parecidas al lenguaje humano. Como contrapartida, este lenguaje se aleja del lenguaje máquina con los inconvenientes que esto acarrea.

Con la realización de este proyecto mediante el lenguaje escogido se ha podido conocer y profundizar sobre las técnicas de programación para dispositivos móviles, programación orientada a objetos y programación de videojuegos.

Se ha usado un protocolo de comunicación a través de la tecnología Bluetooth para satisfacer las necesidades de comunicación entre dos terminales. En concreto se ha utilizado el protocolo RFCOMM. Se ha desarrollado un mecanismo de sincronización entre los dos dispositivos sobre este enlace. Dicho mecanismo se ha implementado para la interactividad entre los dos usuarios que juegan una partida.

La metodología utilizada para el desarrollo del proyecto ha consistido en implementar sucesivas versiones que fueran añadiendo progresivamente nuevos elementos y funcionalidades al juego. En un primer momento controlado todo desde un mismo dispositivo. Se implementaron los tres personajes que en un principio podrían moverse en las cuatro direcciones libremente sobre un fondo negro sin ser animados. También se incluyó la cruceta direccional para manejarlos. Se añadió un menú provisional para seleccionar qué personaje se deseaba controlar para ir probando sus funciones a medida que se le iban añadiendo. Posteriormente se añadieron las animaciones, la funcionalidad de salto y el botón para saltar. Se dibujó la estancia y los elementos que aparecerían en ella, implementando la transición entre estancias, la forma de interactuar entre los personajes y todos los elementos de la partida y se añadió un nuevo botón para unir y separar a los personajes. Posteriormente se añadieron la condición de victoria al conseguir la corona, la de game over al perder todas las vidas y el estado Preparado y Pausado con su menú correspondiente.

Tras ello se desarrollaron las pantallas previas al comienzo de la partida y la forma de establecer el emparejamiento entre los dispositivos. Posteriormente se adaptó el código para que cada dispositivo pudiera controlar

a un personaje y que fuese el dispositivo que actuaba como servidor de la partida el que controlase al personaje Head over Heels.

Se definieron los mensajes que se intercambiarían los dispositivos y las rutinas que se llevarían a cabo al recibir cada mensaje.

A medida que se fueron añadiendo elementos y funcionalidades se fueron probando y depurando estos. Con la aplicación terminada se realizó una serie de pruebas para comprobar que todo el conjunto funcionaba correctamente.

6.2. Líneas futuras

Ya que en este proyecto se ha desarrollado un prototipo, entre las posibilidades de ampliación está la de completar la aplicación con todos los elementos y pantallas del videojuego original. Además, durante el desarrollo de este proyecto se ha comprendido que en la elaboración de un videojuego, por sencillo que éste sea, se pueden considerar infinidad de variantes y alternativas, según elementos que se quieran tener en cuenta y funcionalidades que se quieran añadir, de manera que deja la puerta abierta a multitud de posibilidades y nuevas características que el programador deberá decidir incluir o no, en función de su complejidad o si se alejan o no del videojuego que se quiera conseguir realizar.

Una línea futura sería pulir la versión actual. Por ejemplo, mejorar la salida de la aplicación, las colisiones entre los elementos del juego, el diseño y comportamiento de las puertas para que fuesen dibujadas con los marcos más dentro de las estancias como en el juego original y que el personaje pudiese chocar, por ejemplo, con los marcos de las puertas. Esta funcionalidad se empezó a implementar pero se descartó por cierta complejidad a la hora de dibujar correctamente el personaje y la puerta. Algunas veces el dibujo del personaje se superponía sobre el de la puerta cuando no debía ser así.

El juego se ha desarrollado exclusivamente para dos jugadores, por lo tanto, una posibilidad de ampliación sería añadir un modo para un solo jugador

como el videojuego original. Se le podrían incluir todos los mundos y niveles del juego original, variaciones de estos o incluso añadir algunos nuevos que el jugador tendría que superar.

Se podrían incluir numerosos enemigos distintos, con habilidades distintas y algunos más complejos de superar que otros. También podrían añadirse diferentes tipos de obstáculos que hubiese que superar de diversas formas y diseñar las pantallas de forma que el jugador tendría que resolver una serie de puzles para conseguir superarlas.

Otra mejora sería añadir nuevos artefactos que los personajes pudieran encontrar por las estancias para mejorar sus habilidades o concederles unas nuevas, ya sea de forma temporal o indefinida. Por ejemplo, se le podría añadir una pistola que el personaje encontrara por la pantalla y que tuviera que ir recogiendo balas para recargarla. Además los personajes podrían encontrar elementos que les otorgaran más vidas.

Además de los obstáculos o enemigos que aparecen en las estancias, podrían añadirse una serie de objetos que ayudasen a los personajes. Por ejemplo muelles que impulsaran a un personaje para poder alcanzar un objeto o lugar, teletransportadores para viajar automáticamente entre una estancia y otra…

También existe la posibilidad de incluir diseños distintos para las estancias. Por ejemplo, se podrían incluir estancias con varias plantas o algunas más grandes que no se pudieran dibujar de golpe enteras en la pantalla y existiera una cámara que siguiera al personaje por la pantalla.

Se podría añadir un argumento que podría ser el del juego original u otro diferente, incluyendo nuevos personajes secundarios con los que interactuar o que nos ofrecieran información para superar los niveles. También podrían agregarse nuevos personajes con nuevas habilidades para ser controlados por el usuario y nuevos modos de juego como por ejemplo un multijugador en el que un jugador se enfrentase a otro de alguna forma. Existen multitud de posibilidades en este sentido.

Respecto a los menús y opciones, también se podrían incluir cosas nuevas. Por ejemplo un tutorial antes de empezar la partida al cual se accediera desde una opción del menú principal, la posibilidad de configurar más opciones del juego como la dificultad, por ejemplo, y no solamente el audio, añadir nuevos botones y diseños para los menús para acceder a los contenidos que se fuesen incluyendo, incluir una opción para descargar actualizaciones, etc.

Por otro lado, también podría existir la posibilidad de que el usuario diseñase sus propios niveles con la ayuda de algún editor de niveles o descargarlos mediante actualizaciones. Con esta opción se alargaría la vida del producto, ya que, el usuario contaría con los niveles originales del juego, los diseñados por él mismo y los que se hubiera descargado. También podrían pasarse niveles diseñados desde un dispositivo a otro por Bluetooth.

Bibliografía

- Mario Zechner, “Desarrollo de juegos Android”, Anaya Multimedia, 2012
- Jeff Friesen “Java para desarrollo Android”, Anaya Multimedia, 2011
- Grady Booch, James Rumbaugh, Ivar Jacobson, “El lenguaje unificado de modelado”, Adison-Wesley, 2006
- Página oficial para desarrolladores de Android http://developer.android.com/intl/es/index.html
- Foro de preguntas y respuestas de programadores http://stackoverflow.com/

Referencias

Todas las páginas web referidas han sido revisadas en Noviembre de 2015

[1] The International Arcade Museum, web dedicada a mantener una amplia base de datos de máquinas recreativas http://www.arcade-museum.com/game_detail.php?letter=&game_id=9074

[2] EAE Business School, estudio sobre el mercado del videojuego en España en el año 2014, http://www.eae.es/news/2015/01/26/el-mercado-del-videojuego-en-espana-movio-763-millones-de-en-2014-con-un-crecimiento-del-31-respecto-al-2013

[3] Estudio sobre el mercado de videojuegos para móviles, http://gamedustria.com/2015/el-mercado-de-videojuegos-para-moviles-superara-al-de-consolas-en-2015

[4] Página web de la empresa Newzoo, http://www.newzoo.com/

[5] Contexto histórico del videojuego Head over Heels original, http://www.headoverheels2.com/drupal/hoh

[6] Página web sobre la versión del videojuego Head over Heels creado por Jorge Rodríguez Santos y Santiago Acha Jiménez, http://www.headoverheels2.com/drupal/index.php

[7] Página desde donde se han descargado los gráficos de Davit Masiá para el videojuego, http://www.headoverheels2.com/drupal/downloads

[8] Libro Desarrollo de Juegos de Android de Mario Zechner de la editorial Anaya Multimedia.

[9] Página de la Wikipedia sobre GIMP, https://es.wikipedia.org/wiki/GIMP

[10] Página de descarga del videojuego Head over Heels de Jorge Rodríguez Santos y Santiago Acha Jiménez, http://www.headoverheels2.com/drupal/downloads