Agradecimientos

A mis padres, por su paciencia.

-No intentes doblar la cuchara, eso es imposible.

En lugar de eso, trata de comprender la verdad.-

-¿Qué verdad?-

-Que no hay cuchara-

Diálogo de Matrix sobre los límites de la programación.

Contenido

Capítulo 1. Introducción 1

Capítulo 2. Tecnologías y herramientas empleadas..... 7

2.1. Introducción 7

2.2. Entorno de desarrollo 7

2.2.1. Java como lenguaje 7

2.2.2. Java como plataforma 8

2.2.3. JDK (Java Development Kit) 9

2.2.4. Kit de Android SDK 9

2.2.5. Eclipse 10

2.2.6. Complemento ADT para Eclipse 11

2.2.7. Emuladores de código 11

2.2.8. Framework de Mario Zechner 12

2.2.9. Librería AndEngine 12

2.2.10. Clase DeviceListActivity de AOSP 13

2.2.11. Clases del actualizador de AndroCode 13

2.3. Bluetooth 13

2.4. Edición de imágenes 14

2.5. Edición de audio 15

2.5.1. WinUAE 15

2.5.2. Winamp y plug-in TFMX 16

2.5.3. Mp3Cut.net 16

2.5.4. Switch Sound File Converter 16

2.6. Emulador DosBox 17

2.7. Modelado UML 18

2.7.1. Diagrama de clases 20

2.7.2. Diagrama de estados 20
2.7.3. Diagrama de secuencia 21
Capítulo 3. Descripción de la aplicación 23
3.1. Introducción 23
3.2. Instalación 24
3.3. Pantallas y menús 25
3.3.1. Inicio 25
3.3.2. Menú principal 25
3.3.3. Menú Niveles 28
3.3.4. Dos jugadores 29
3.3.5. Opciones 32
3.3.6. Cómo jugar 34
3.3.7. Actualizar 34
3.4. Pantallas de la partida 36
3.4.1. Preparado 36
3.4.2. Esperando 1 37
3.4.3. Esperando 2 38
3.4.4. Victoria 39
3.4.5. Derrota 39
3.5. Instrucciones de juego 40
3.5.1. Modo Un jugador 40
3.5.2. Modo dos jugadores 40
3.6. Elementos asiduos 41
3.6.1. Engranajes 42
3.6.2. Pasillos y cruces 42
3.6.3. Siguiente bola 43
3.6.4. Reloj de arena 43

3.6.5. Número de bolas 44
3.6.6. Tiempo de bola 44
3.7. Elementos especiales 44
3.7.1. Teletransporte 45
3.7.2. Filtro 45
3.7.3. Pintura 46
3.7.4. Semáforo 46
3.7.5. Direccionador 47
3.7.6. Cruz de bolas 47
3.8. Niveles 48
3.9. Audios 51
3.10. Puntuación y estrellas 52
Capítulo 4. Diseño de la aplicación 53
4.1. Introducción 53
4.1.1. Marco software de la aplicación 53
4.2. Visión general de la aplicación 55
4.2.1. Herencia 56
4.2.2. Layouts 56
4.2.3. Hilos 57
4.2.4. División de la aplicación 57
4.2.5. Clases implementadas del bloque de menús 58
4.2.6. Clases implementadas del bloque del motor del juego 59
4.2.7. Diagrama de estados de la aplicación 61
4.3. Utilidades generales para la aplicación 63
4.3.1. Clases heredadas 63
4.3.2. Interfaz gráfica en el bloque de menús 70
4.3.3. Interfaz gráfica en el bloque del motor del juego 73

4.3.4. Lectura / escritura de datos 74
4.3.5. Gestor de sonidos 77
4.3.6. Interacción con el usuario 78
4.3.7. Interacción entre clases 79
4.3.8. Conexión a internet 80
4.3.9. Bluetooth 81
4.3.10. DeviceListActivity.java 82
4.3.11. VersionChequer.java 83
4.4. Permisos 83
4.5. Bluetooth 83
4.5.1. Pasos previos al emparejamiento 84
4.5.2. Funcionamiento 85
4.5.3. Modo servidor 86
4.5.4. Modo cliente 87
4.5.5. Implementación de la funcionalidad Bluetooth dentro del juego 88
4.5.6. Intercambio de mensajes Bluetooth. 89
4.6. Menús y sus elementos 98
4.6.1. Layouts y archivos xml de ayuda 98
4.6.2. Menú Inicio 99
4.6.3. Cómo jugar 100
4.6.4. Menú opciones 100
4.6.5. Rejilla de niveles. Menú niveles. 102
4.6.6. Menú dos jugadores 103
4.7. Guardar/Cargar datos 104
4.7.1. Ficheros 105
4.7.2. La clase Ajustes 105
4.8. Actualizar 106

4.9. Pantalla Splash 108
4.10. Elementos y recursos del videojuego 108
4.10.1. Bolas 108
4.10.2. Casillas 109
4.10.3. Recursos 112
4.10.4. Estructuras de los niveles 114
4.10.5. Entre los menús y el juego. Carga de recursos. 115
4.11. Lógica del videojuego 116
4.11.1. Lógica automática 117
4.11.2. Lógica automática multijugador 129
4.11.3. Interacción de usuario 131
4.11.4. Interactuación de usuario multijugador 136
4.11.5. Modelado gráfico 138
Capítulo 5. Plan de pruebas 145
5.1. Pruebas con emuladores 145
5.2. Pruebas con dispositivos reales 147
5.3. Testeadores 154
5.4. Evolución de las versiones 156
5.5. Análisis de resultado 160
Capítulo 6. Conclusiones y trabajo futuro 163
6.1. Conclusiones 163
6.2. Líneas futuras 166
Bibliografía 169
Referencias 171

Índice de Figuras

FIGURA 1-1. MÁQUINA PONG ORIGINAL 1

FIGURA 1-2. REVISTA MICROMANÍA ANALIZANDO EL LOG!CAL 3

FIGURA 1-3. NIVEL 4 DE LOG!CAL 4

FIGURA 1-4. NIVEL EN LA VERSIÓN PARA GAMEBOY COLOR 5

FIGURA 2-1. ECLIPSE EN PLENO FUNCIONAMIENTO 10

FIGURA 2-2. EMULADOR EN EJECUCIÓN. 11

FIGURA 2-3. ADOBE PHOTOSHOP EN FUNCIONAMIENTO. 15

FIGURA 2-4. SWITCH SOUND FILE CONVERTER. 17

FIGURA 2-5. DOSBOX EMULANDO EL VIDEOJUEGO LOG!CAL 18

FIGURA 2-6. DIAGRAME DE CLASES DEL PERSONAL UNIVERSITARIO. 20

FIGURA 2-7. DIAGRAMA DE ESTADOS DE UN SEMÁFORO. 21

FIGURA 2-8. DIAGRAMA DE SECUENCIA DE UN VIDEOJUEGO DE AJEDREZ. 21

FIGURA 3-1. ASPECTO DEL ACCESO A LA APLICACIÓN UNA VEZ INSTALADA. 24

FIGURA 3-2. CAPTURA DE LA INTRODUCCIÓN DE LA APLICACIÓN. 25

FIGURA 3-3. PANTALLA DEL MENÚ PRINCIPAL. 26

FIGURA 3-4. VENTANA EMERGENTE QUE APARECE AL PULSAR "SOBRE NOSOTROS" 27

FIGURA 3-5. MENSAJE DE CONFIRMACIÓN PARA ABANDONAR LA APLICACIÓN. 27

FIGURA 3-6. PANTALLA DEL MENÚ NIVELES. 28

FIGURA 3-7. VENTANA EMERGENTE PIDIENDO PERMISO PARA ACTIVAR EL BLUETOOTH DEL DISPOSITIVO. 29

FIGURA 3-8. MENSAJE MIENTRAS SE PROCEDE A LA ACTIVACIÓN DEL BLUETOOTH 30

FIGURA 3-9. NOTIFICACIÓN DE VISIBILIDAD. 30

FIGURA 3-10. MENSAJE DE SELECCIÓN DE MODO. 31

FIGURA 3-11. LISTA DE DISPOSITIVOS PREVIAMENTE EMPAREJADOS 31

FIGURA 3-12. SOLICITUD DE SINCRONIZACIÓN VÍA BLUETOOTH. 32

FIGURA 3-13. PANTALLA OPCIONES. 33

FIGURA 3-14. PANTALLA COMO JUGAR 34

FIGURA 3-15. PANTALLA ACTUALIZADOR. 35

FIGURA 3-16. ACTUALIZADOR CON LA APLICACIÓN ACTUALIZADA. 35
FIGURA 3-17. ACTUALIZADOR CON UNA VERSIÓN MÁS RECIENTE DE LA APLICACIÓN. 36
FIGURA 3-18. PANTALLA PREPARADO CON SÓLO LA ESTRUCTURA DEL JUEGO. 37
FIGURA 3-19. PANTALLA PREPARADO TRAS PAUSA. 37
FIGURA 3-20. PANTALLA ESPERANDO 1. 38
FIGURA 3-21. PANTALLA ESPERANDO 2 38
FIGURA 3-22. PANTALLA VICTORIA 39
FIGURA 3-23. PANTALLA DERROTA 39
FIGURA 3-24. PARTIDA EN MODO DOS JUGADORES. 41
FIGURA 3-25. NIVEL 1, ESTRUCTURA ÚNICAMENTE CON ELEMENTOS ASIDUOS. 41
FIGURA 3-26. IMAGEN DE UN ENGRANAJE POR SUPERAR 42
FIGURA 3-27. IMAGEN DE UN PASILLO VERTICAL. 42
FIGURA 3-28. IMAGEN DEL ELEMENTO QUE INDICA LA SIGUIENTE BOLA. 43
FIGURA 3-29. IMAGEN DEL RELOJ DE ARENA. 43
FIGURA 3-30. IMAGEN DEL CONTADOR DE BOLAS. 44
FIGURA 3-31. IMAGEN DEL TIEMPO DE BOLA. 44
FIGURA 3-32. NIVEL CON VARIOS ELEMENTOS ESPECIALES. 45
FIGURA 3-33. IMAGEN DEL TELETRANSPORTE. 45
FIGURA 3-34. IMAGEN DE UN FILTRO. 46
FIGURA 3-35. IMAGEN DE UNA PINTURA. 46
FIGURA 3-36. IMAGEN DEL SEMÁFORO. 47
FIGURA 3-37. IMAGEN DE UN DIRECCIONADOR. 47
FIGURA 3-38. IMAGEN DE LA CRUZ DE BOLAS. 48
FIGURA 3-39. NIVELES 1 A 9. 49
FIGURA 3-40. NIVELES 10 A 18. 49
FIGURA 3-41. NIVELES 19 A 27 50
FIGURA 3-42. NIVELES 28 A 36 50
FIGURA 3-43. NIVELES 37 A 45. 51
FIGURA 3-44. NIVELES 46 A 50. 51
FIGURA 4-1. RESUMEN DE LA ESTRUCTURA DE CAPAS DE ANDROID 54

FIGURA 4-2. ESTRUCTURA DE CAPAS OFICIAL DE ANDROID. 55
FIGURA 4-3. DIAGRAMA DE CLASES DEL BLOQUE DE MENÚS 59
FIGURA 4-4. DIAGRAMA DE CLASES DEL BLOQUE DEL MOTOR DE JUEGO. 61
FIGURA 4-5. DIAGRAMA DE ESTADOS DE LA APLICACIÓN. 62
FIGURA 4-6. CICLO DE VIDA DE LA CLASE ACTIVITY. 64
FIGURA 4-7. INTERFAZ GAME 65
FIGURA 4-8. CLASE SCREEN. 67
FIGURA 4-9. CLASE ANDROIDGAME 68
FIGURA 4-10. CLASE BASESPLASHACTIVITY. 69
FIGURA 4-11. FORMATO DE LOS MENSAJES PAUSA 90
FIGURA 4-12. FORMATO DE LOS MENSAJES REINICIAR 90
FIGURA 4-13. FORMATO DE LOS MENSAJES GAMEOVER 91
FIGURA 4-14. FORMATO DE LOS MENSAJES VICTORIA 91
FIGURA 4-15. FORMATO DE LOS MENSAJES ELNIVEL 91
FIGURA 4-16. FORMATO DE LOS MENSAJES VAMOS 92
FIGURA 4-17. FORMATO DE LOS MENSAJES BOLASIG 92
FIGURA 4-18. FORMATO DE LOS MENSAJES BOLAS55 93
FIGURA 4-19. FORMATO DE LOS MENSAJES GIRAR 94
FIGURA 4-20. FORMATO DE LOS MENSAJES BIZQ 95
FIGURA 4-21. FORMATO DE LOS MENSAJES BDER 95
FIGURA 4-22. FORMATO DE LOS MENSAJES BARR 95
FIGURA 4-23. FORMATO DE LOS MENSAJES BABA 95
FIGURA 4-24. FORMATO DE LOS MENSAJES CRUZ 96
FIGURA 4-25. FORMATO DE LOS MENSAJES BOLAENTRO55 96
FIGURA 4-26. FORMATO DE LOS MENSAJES BOLAENTRO00 97
FIGURA 4-27. FORMATO DE LOS MENSAJES MOV BUSCANDO REBOTAR 97
FIGURA 4-28. FORMATO DE LOS MENSAJES MOV BUSCANDO ENTRAR EN ENGRANAJE. 98
FIGURA 4-29. INFORMACIÓN EN JSON PROPORCIONADA. 107
FIGURA 4-30. TODOS LOS RECURSOS GRÁFICOS DE LA PARTE JUGABLE CON SUS NOMBRES 113
FIGURA 4-31. DIAGRAMA DE ESTADOS DE LA COMPROBACIÓN DE LA SUPERACIÓN DE ENGRANAJE. .. 124

Capítulo 1. Introducción

Desde que el videojuego “Pong”, cuya máquina puede verse en la Figura 1-1, irrumpiera con éxito en 1972 [1], el mercado del sector del videojuego se ha mantenido al alza. Las previsiones, según la firma PwC [2], son que los videojuegos abarcan un sector con proyección de futuro. Entre los años 2013 y 2016, el gasto mundial alcanzará los 83.000 millones de dólares, con una tasa de crecimiento interanual del 7.2%.

Figura 1-1. Máquina Pong original

AEVI (Asociación Española de Videojuegos) [3] extrae de ese informe, entre otras conclusiones, que el desarrollo de nuevas plataformas de juegos como tabletas y teléfonos inteligentes está contribuyendo a aumentar el número de jugadores casuales. Además aporta el dato en el que actualmente, los videojuegos representan el 75% de las aplicaciones desarrolladas para teléfonos móviles y tabletas.

Se consideran jugadores casuales a aquellos individuos que no están comprometidos propiamente a conseguir todos los objetivos posibles en un juego dado. Además, estos jugadores no suelen emplear mucho tiempo en cada juego, caracterizándose, en muchos casos, por jugar a muchos juegos, pero durante poco tiempo o en intervalos irregulares [4].

Los videojuegos que suelen jugar, denominados videojuegos casuales pueden tener cualquier tipo de mecánica de juego, y ser clasificados dentro de cualquier otro género, aunque han de coincidir en ciertos parámetros [5]:

- Que tengan Jugabilidad. Deben conseguir evitar todos los obstáculos que reducen la satisfacción del jugador.
- Atracción desde el primer minuto. Deben tener la capacidad de atraer la atención y el interés del jugador en los primeros minutos de juego.
- Estructura en mini-ciclos. El flujo debe estar estructurado generalmente como una secuencia de mini-ciclos, o niveles, que con el tiempo pueden remunerar el jugador inmediatamente.
- Duración larga. Deben ser diseñados para poder jugar con una experiencia tan convincente para hacer la decisión de interrumpir el juego o la opción de jugar con menos frecuencia, lo más difícil posible.
- Mantener el equilibrio. Siempre tiene que haber un equilibrio entre el aumento de la dificultad y la progresión de la habilidad del jugador.

Las primeras generaciones de videojuegos, es decir, lo que hoy se agrupa como videojuegos de genero arcade, cumplían la mayoría de estos requisitos en su época dorada. Tenían buena jugabilidad, atraían desde el primer minuto, mantenían el equilibrio y poseían una duración lo suficientemente larga para tener una buena experiencia.

Un videojuego tanto de genero árcade como de puzle, es el creado por la compañía alemana Rainbow Arts y lanzado en 1991 con el nombre Log!cal, también conocido como Logical, para las plataformas Commodore Amiga, Atari ST, Commodore 64 y PC.

Cuando fue lanzado, la revista Micromanía [6] comentó que era capaz de reunir en su desarrollo elementos tan imprescindibles como jugabilidad y originalidad. Bajo el epígrafe “lógicamente divertido”, lo catalogaba como una de las mejores secuelas surgidas a la estela de Tetris (exitoso videojuego que marcó una época en el género puzle [7]). Comentaba que estaba perfeccionado hasta en sus más mínimos detalles, incluido aspectos que por entonces se consideraban secundarios, como la música o los efectos sonoros. Terminaba

puntuándolo con un 8, destacando su adicción y originalidad. Las páginas que abarcan dicha información se pueden ver en la Figura 1-2.


PUNTO de mira

Karate a muerte en tu ordenador

SHADOW DANCER

Una idea increíblemente divertida

LOGICAL

Pura extravagancia

FLIP-IT & MAGNOSE

RECOMENDADOS

Figura 1-2. Revista Micromanía analizando el Log!cal

Otras revistas del sector como las estadounidenses Advaced Computer Entertainment, Amiga Computing y The One o como la sueca Hemmdatornytt también elogiaban el videojuego Log!cal puntuándolo de manera muy positiva [8].

El juego se basa en que el jugador intente rellenar los cuatro huecos de unos engranajes distribuidos por la pantalla de juego con bolas del mismo color, aunque habrá algunos casos que pida una distribución de color específica. Para poder transportar las bolas de un engranaje a otro se dispone de unas canalizaciones en cada pantalla.

El jugador dispone únicamente de dos acciones: girar el engranaje 90º en sentido horario o soltar la bola si dispone de una canalización en esa posición. Inicialmente se comienza con tres vidas, y se pueden perder si se agota el tiempo destinado a la fase o el tiempo destinado a recoger la última bola lanzada.

Como ayuda visual, se tiene un reloj de arena con el tiempo global restante, un marcador analógico con el número de bolas en circulación posibles que quedan, un indicador del color de la siguiente bola que aparecerá y la notificación del tiempo que queda para recoger la última bola lanzada mediante el rellenado por sombreado de la tubería inicial, la cual recorre todo el eje horizontal en la parte superior de la pantalla. Tanto el tiempo global como el tiempo por bola son variables de una fase a otra. Se puede ver la captura de pantalla de uno de los niveles del videojuego en la Figura 1-3.

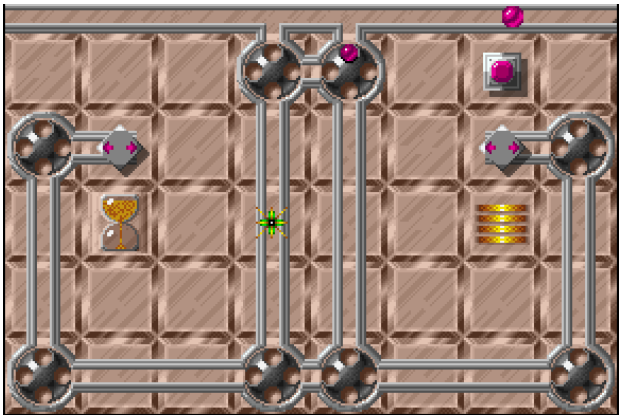

Figura 1-3. Nivel 4 de Log!cal

Para añadirle complejidad, diversos elementos se irán sumando a los ya citados: teletransportadores, que llevarán la bola de un lugar a otro de la pantalla; flechas de dirección, que obligaran a tomar un único rumbo a las bolas; filtros por color, que solo dejaran pasar las bolas del color que indiquen; botes de pintura, que cambiaran el color de la bola al que indica el bote; semáforos, que exigirá que se completen engranajes con colores en un orden determinado y hasta que no se satisfaga, los posibles engranajes ya completados no se marcarán, ni las bolas desaparecerán; y cruces de colores, que actúa restringiendo al igual que los semáforos, solo que esta vez exige un engranaje con unos colores concretos en posiciones concretas.

En 1999, SunSoft publicaría una versión para GameBoy Color con los mismos fundamentos, aunque con los niveles adaptados a la pantalla cuadrada de la videoconsola portátil, lo que dejaba menos espacio para poner elementos. [9] Esta peculiaridad se puede apreciar en la Figura 1-4. Posteriormente se han realizado distintas versiones por aficionados, siendo la más conocida Psychoballs, para Windows, cuyo autor confiesa que creó este videojuego, el cual posee la misma mecánica, porque Log!cal fue uno de sus juegos favoritos durante la época en el que se lanzó. [10]

Figura 1-4. Nivel en la versión para GameBoy Color

En este proyecto se desarrollará una versión de un jugador y otra multijugador –en concreto, para 2 jugadores- de este videojuego, adaptado a dispositivos con sistema operativo Android. Se parte del Log!cal, tanto a nivel de lógica del juego como de estética, tomando como referencia su aspecto gráfico y sonoro (el fondo, los engranajes, las bolas, los elementos extra…).

El apartado multijugador constará de dos pasillos, uno en la parte superior y otro en la inferior del resto de elementos. Los dos engranajes de las dos primeras filas, serán controlados de manera exclusiva por el dispositivo en modo servidor y los de las dos últimas filas por el dispositivo en modo cliente, siendo los engranajes de la franja central controlados de manera compartida.

Las bolas saldrían tanto por el pasillo superior como por el inferior y el nivel podría darse como perdido en caso de que cualquiera de los dos pasillos rebasase el tiempo por bola. De esta manera, la parte colaborativa es esencial para superar las fases en modo multijugador.

Los menús y la selección de niveles tendrá un aspecto más acorde a una aplicación para dispositivos móviles, dejando de lado elementos como las vidas, el sistema de puntuación global o el acceso a niveles mediante *password* y añadiendo una pantalla de selección de nivel, un sistema de puntuación en el que se recoge individualmente en cada nivel e instrucciones para poder comenzar. Sin embargo, con el aspecto gráfico de una partida, se busca la proximidad al original. Incluido la disposición de elementos en cada nivel.

El presente documento está estructurado en diversos capítulos para poder explicar con detalle todas las partes del proyecto:

1. Introducción: es el presente capítulo, en el que se hace una descripción y justificación del proyecto, así 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 de proyecto y estudio de posibles mejoras o ampliaciones.

Capítulo 2. Tecnologías y herramientas empleadas

2.1. Introducción

En este capítulo se describirán los elementos usados para crear la aplicación describiendo cada una de las herramientas software utilizadas y el uso que se le ha dado.

2.2. Entorno de desarrollo

Para conseguir un entorno de desarrollo adecuado para la realización de la aplicación, se deben aplicar varias plataformas que proporcionan distintas funciones y tener ciertos conocimientos.

2.2.1. Java como lenguaje

Java es un lenguaje y una plataforma creada por Sun Microsystems y actualmente propiedad de Oracle.

Java es un lenguaje en el cual los desarrolladores expresan código fuente. Su sintaxis proviene parcialmente de C y C++, con el objeto 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 Virtual Java (JVM), ésta sí dependiente del sistema operativo, presente en la gran mayoría de los sistemas.

- Seguridad: Esto es en parte debido 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 que están compuestas de procesadores físicos y de sistemas operativos, la plataforma Java consiste en una máquina virtual y un entorno asociado.

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

En nuestro proyecto se ha utilizado una edición especial de la plataforma Java creada por Google para hacer aplicaciones que se ejecutan sobre sus dispositivos Android. Esta edición es conocida como la plataforma Android [11]. Centrándonos en ella, podríamos decir que está compuesta básicamente de bibliotecas del núcleo Java (en parte basadas en Java SE) y una máquina virtual conocida como Dalvik. Este software colectivo se ejecuta sobre un núcleo de Linux especialmente modificado.

2.2.3. JDK (Java Development Kit)

Para el desarrollo de programas Java, se necesita el Kit de desarrollo Java SE (JDK), el cual 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 [12].

2.2.4. Kit de Android SDK

Es una herramienta básica compuesta por las herramientas basadas en la línea de comandos. Se necesita para crear, compilar y desarrollar proyectos Android. También cuenta 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 para originar dispositivos virtuales.

El gestor de SDK y AVD puede descargar e instalar varios tipos de componentes:

- Plataformas Android: Para cada versión oficial de Android existe una plataforma que se debe incluir 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: Los complementos o librerías externas y herramientas que no pertenecen a ninguna plataforma específica.

- Un controlador USB para Windows: Necesario para ejecutar y depurar la aplicación en un dispositivo físico si trabaja con el sistema operativo Windows.
- Ejemplos: Para cada plataforma hay una serie de ejemplos especialmente desarrollados para ella.
- Documentación: Copia local de la documentación de la última API del *framework* de Android.

2.2.5. Eclipse

Eclipse es un Entorno de Desarrollo Integrado (IDE - Integrated Development Environment) desarrollado en código abierto que se puede usar para generar aplicaciones con diferentes lenguajes de programación. Generalmente se utiliza en los desarrollos con Java y es el entorno de desarrollo recomendado por Google para el desarrollo de aplicaciones Android.

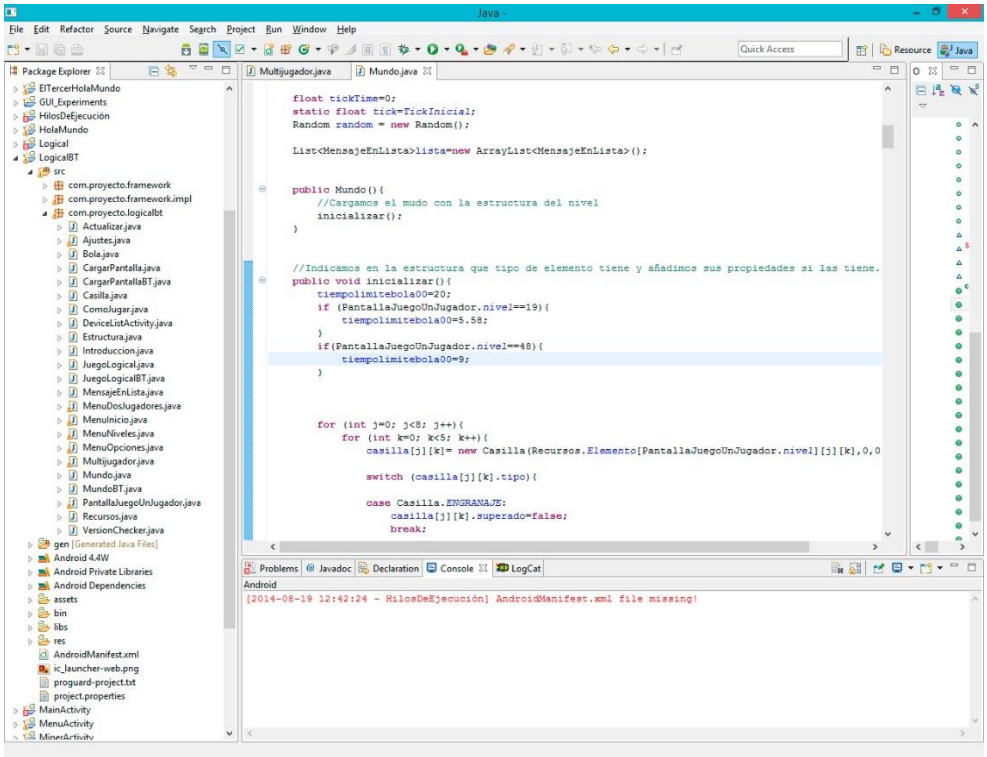

Figura 2-1. Eclipse en pleno funcionamiento

Desarrollada originalmente por IBM, esta plataforma conforma un entorno de desarrollo maduro, con una gran base de usuarios que contribuyen a crear

tutoriales, manuales y *plugins* para facilitar el desarrollo [13]. En la Figura 2-1 se puede observar el programa iniciado con parte de un código en pantalla.

2.2.6. Complemento ADT para Eclipse

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

2.2.7. Emuladores de código

Una buena forma de detectar y corregir errores de código antes de empezar a probar la aplicación sobre dispositivos móviles reales, es hacer uso de emuladores. Si bien no garantizan que las aplicaciones funcionen exactamente igual que en los dispositivos móviles y que no disponen de la capacidad de simular conexiones por Bluetooth. Sin los emuladores el desarrollo habría sido más lento y complicado.

Figura 2-2. Emulador en ejecución.

De las posibilidades que da el entorno de trabajo para crear un dispositivo virtual (emulador) ya sea porque viene con la instalación, o porque ofrece la posibilidad de descargarlo, se han realizado casi todas las pruebas en el emulador de un Nexus One con un Android 4.2 interno. En la Figura 2-2 se

observa al emulador ejecutando el nivel 1 de la aplicación en una fase temprana del desarrollo.

2.2.8. Framework de Mario Zechner

El capítulo 5 del libro Desarrollo de Juegos de Android de Mario Zechner trata de la implementación de las clases y las interfaces de un *framework* para juegos para codificar toda la mecánica de un juego en 2D de Android. Se decidió usar este *framework* para que ayudase al desarrollo del motor del juego. La 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 si se precisase.

Cabe destacar que Mario Zechner es también el creador de LibGDX [14], *framework* para el desarrollo de videojuegos multiplataforma, soportando actualmente Windows, Linux, Mac OS X, Android, iOS y HTML5 que es muy utilizado hoy en día.

2.2.9. Librería AndEngine

AndEngine [15] es un motor de juegos gratuito y de código abierto para Android desarrollado por Nicolas Gramlich. Permite desarrollar juegos en 2D y utiliza OpenGL para dibujar los gráficos en pantalla.

Algunas de las características que ofrece son:

- Ajuste automático de la imagen al tamaño de pantalla de diferentes dispositivos
- Motor de físicas
- Sistema de partículas
- Extensión para multijugador
- Manejo sencillo de *sprites* y texturas

AndEngine no tiene documentación, todo está explicado en un conjunto de ejemplos así como en el código fuente.

Aunque esta librería no se use para realizar el motor de nuestro proyecto, sí se utiliza una de sus clases en un momento dado.

2.2.10. Clase DeviceListActivity de AOSP

AOSP, es decir, Android Open Source Project [16], es un proyecto que trabaja en pos del desarrollo y manejo de Android desde el punto de vista de código abierto, libre de todo lo referente a fabricantes y sus especificaciones, e incluso, libre de lo que Google encierra dentro del mismo código de Android.

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.

De esa *release* se ha extraído la clase DeviceListActivity.java que ayudará al emparejamiento de dispositivos con Bluetooth.

2.2.11. Clases del actualizador de AndroCode

Androcode es un portal sobre información para desarrolladores de habla hispana. Desde Androcode se informa de novedades en herramientas y noticias sobre el mundo de Android y proporcionar tutoriales de Android.

En uno de estos tutoriales se crea un actualizador [17], por si no se quiere distribuir la aplicación en el Google Market, bien porque no se tenga cuenta de desarrollador o porque se esté distribuyendo una beta privada.

De este tutorial se ha utilizado una de sus clases para realizar las actualizaciones automáticas. Así se facilita la distribución a los usuarios para que testeen la aplicación con la última versión.

2.3. Bluetooth

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

Opera en la banda libre de radio ISM (banda industrial científico-médica) a 2’4 GHz. Su máxima transmisión de datos es de 1 Mbps. El rango de alcance

de Bluetooth depende de la potencia empleada en la transmisión. La mayor parte de los dispositivos que usan Bluetooth transmiten con una potencia nominal de salida de 0dBm, lo que permite un alcance de unos 10 metros en ambiente libre de obstáculos [18].

Para el acceso a Bluetooth, Android define su propia API en el paquete `Android.bluetooth` y requiere el emparejamiento previo de dos dispositivos antes de intercambiar datos [19].

2.4. Edición de imágenes

Para la edición de las imágenes que aparecen en el videojuego, ha sido necesario recurrir a un software específico para tal fin.

Adobe Photoshop en sus versiones iniciales trabajaba en un espacio (*bitmap*) formado por una sola capa, donde se podían aplicar toda una serie de efectos, textos, marcas y tratamientos. En cierto modo tenía mucho parecido con las tradicionales ampliadoras. En la actualidad lo hace con múltiples capas.

A medida que ha ido evolucionando, el software ha ido incluyendo diversas mejoras fundamentales, como la incorporación de un espacio de trabajo multicapa, inclusión de elementos vectoriales, gestión avanzada de color (ICM / ICC), tratamiento extensivo de tipografías, control y retoque de color, efectos creativos, posibilidad de incorporar plugins de terceras compañías, exportación para sitios web entre otros.

Photoshop se ha convertido, casi desde sus comienzos, en el estándar de facto en retoque fotográfico, pero también se usa extensivamente en multitud de disciplinas del campo del diseño y fotografía, como diseño web, composición de imágenes en mapa de bits, estilismo digital, fotocomposición, edición y grafismos de vídeo y básicamente en cualquier actividad que requiera el tratamiento de imágenes digitales. [20]

Los usos típicos y empleados en la elaboración de este proyecto incluyen la creación de gráficos y logos, el cambio de tamaño, recorte y modificación de fotografías digitales, la modificación de colores, la combinación de imágenes usando un paradigma de capas, la eliminación o alteración de elementos no deseados en imágenes o la conversión entre distintos formatos de imágenes. En

el ejemplo de la Figura 2-3 se puede observar al programa ejecutando los engranajes.

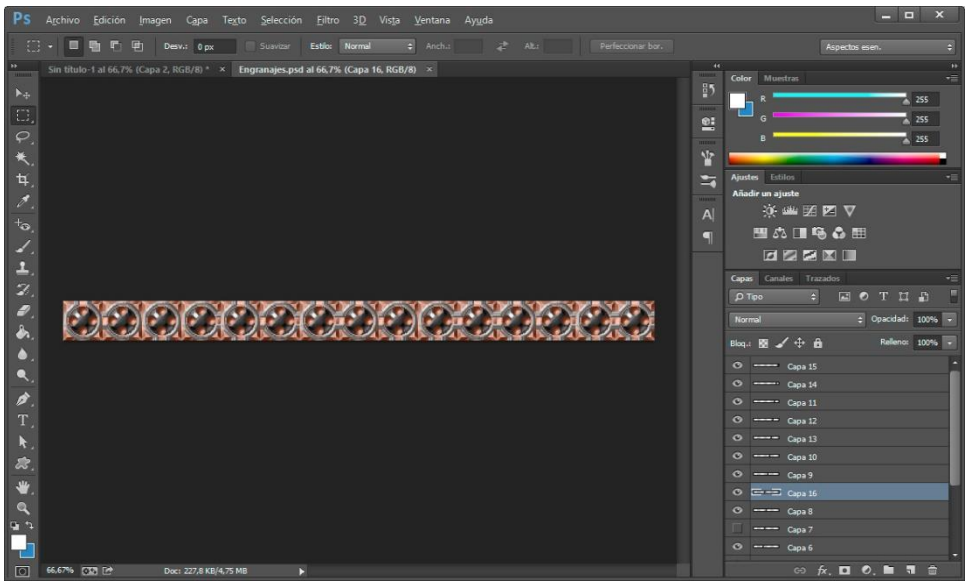

Figura 2-3. Adobe Photoshop en funcionamiento.

2.5. Edición de audio

La aplicación que se ha desarrollado precisaba reproducir efectos de sonido y varias melodías del videojuego original en el que se basa, “Log!cal”.

2.5.1. WinUAE

Unix Amiga Emulator (UAE) es un emulador libre, diseñado para hacer funcionar software escrito para la familia de computadores Commodore Amiga. La primera versión del programa se lanzó en 1995. Permite llevar a cabo instalaciones originales del *workbench* y usar todo tipo de programas de Amiga en él. Intenta producir una experiencia similar a la sensación de estar con un ordenador Amiga. El emulador utiliza el mismo sonido y muestra las mismas imágenes tal y como hacían los ordenadores originales.

WinUAE es la versión portable para el sistema operativo Windows de este emulador [21].

Si bien la experiencia que busca un emulador se obtuvo con el emulador que se presenta en la sección 2.6, este programa ha servido para poder capturar los archivos de efectos de sonido originales.

2.5.2. Winamp y plug-in TFMX

Las melodías del juego que se han podido extraer están en formato smpl y mdat. Mientras que el archivo smpl contiene todo el audio, el archivo mdat contiene la información que indica cómo ejecutar el audio de su archivo paralelo smpl. A los archivos extraídos se le indicaban un fragmento como introducción y otro fragmento más grueso que sonase a continuación y se repitiese durante todo el nivel. Android no reconoce este tipo de archivos.

Winamp es un reproductor multimedia popularizado por usar pocos recursos durante su ejecución y tener una interfaz de usuario sencilla y fácil de usar. Entre sus funciones está la aceptación de *plug-ins* para entrada y salida de audio, como el Procesamiento Digital de Señal para efectos de sonido (DSP).

El *plug-in* TFMX fue creado por el equipo de aficionados a videojuegos antiguos de la página checa AntiXChat2 [22]. Es capaz de leer archivos smpl y mdat en el Winamp aunque con el problema de que realiza lo que indica mdat, por lo que el audio no tiene fin, ya que una parte entra en modo repetición. Esto sucede también al exportarse desde Winamp, lo que deja unos archivos exportados con excesiva duración en la mayoría de los casos.

2.5.3. Mp3Cut.net

La página Mp3Cut.net [23] se basa, principalmente, en importar archivos de audio con formato mp3, mostrar su espectro gráficamente para que el usuario decida por dónde cortar dicho archivo y crear un nuevo archivo mp3 con las modificaciones.

Es una opción sencilla y con un buen resultado para acortar el exceso de audio que deja un archivo mp3 que ha sido creado a partir de unos archivos smpl y mdat con la opción vista en el apartado 2.5.2

2.5.4. Switch Sound File Converter

Aunque Android soporta diversos formatos de audio, la recomendación es usar archivos ogg vorbis. El formato ogg ofrece una mayor compresión de los archivos, lo que reduce su tamaño, sin conllevar pérdidas de calidad, ya que emplea un sistema de compresión inteligente que elimina las partes no audibles

para el oído humano y parte del ruido ambiente. Esto hace que se eliminen las distorsiones en las grabaciones antiguas. Ogg vorbis conserva mejor el sonido cuando no hay diferencia de tasas de bits, es decir que comparando un archivo de audio mp3 con un archivo ogg vorbis a idénticas velocidades de datos, el último ocupa menos espacio en disco y tiene la misma calidad de sonido. [24]

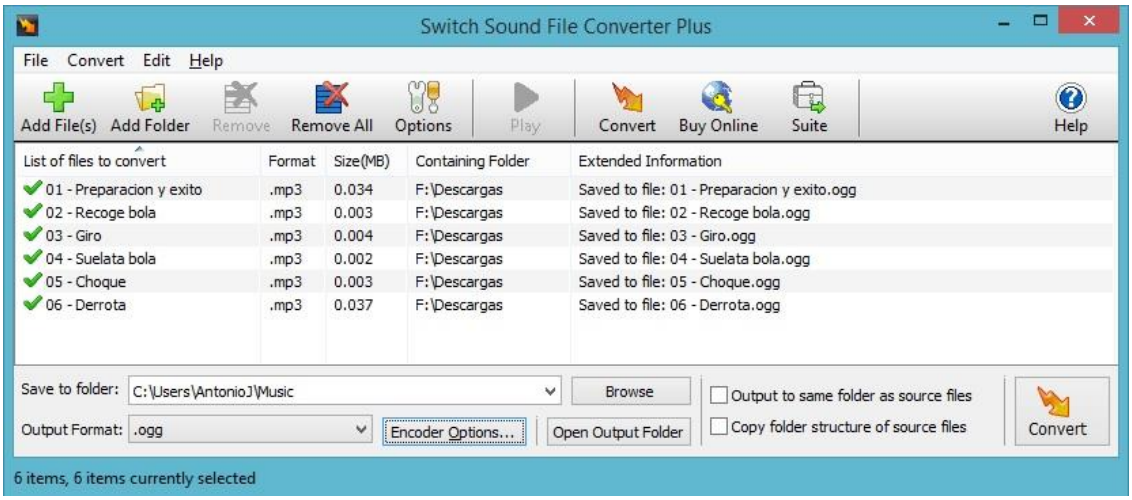

Figura 2-4. Switch Sound File Converter.

Debido a que la mayoría de los archivos de audio se tenían en formato mp3, se buscó un software que hiciese la conversión a ogg vorbis. El Switch Sound File Converter es un software que se utiliza para la conversión de archivos de audio pudiendo seleccionar tanto la tasa de bits como el formato de salida de una manera simple y eficiente. Podemos apreciarlo tras una conversión exitosa en la Figura 2-4.

2.6. Emulador DosBox

Para que se pueda conocer a fondo el videojuego Log!cal, además de las referencias vistas en el capítulo 1, se debe interactuar con la versión original. Si no se dispone de ninguna de las videoconsola ni cartucho con el juego, una opción es usar un emulador con los archivos originales del juego.

DosBox es un emulador de MsDOS en máquinas 286, 386 y 486 ideal para poder disfrutar de los juegos basados en DOS que hoy en día no funcionan en los ordenadores actuales y en sistemas operativos modernos. Además emula tarjetas gráficas antiguas como la Hercules y sobre todo tarjetas de sonido como

la Sound Blaster, la Adlib o la Gravis Ultrasound, con lo que podremos disfrutar de los juegos con su sonido original.

Figura 2-5. DosBox emulando el videojuego Log!cal

Se recomienda usarlo junto al *frontend* Boxer [25], que es capaz de modificar, entre otras opciones, la pantalla, los ciclos de CPU emulados, la memoria o la tarjeta de sonido e incluso puede instalar un modem virtual. La Figura 2-5 es la captura del videojuego siendo emulado por DosBox.

El juego emulado ha tenido dos funciones. La primera ha sido conseguir mayor conocimiento del juego, de su lógica de juego y de la estructura de los niveles. El segundo ha sido la obtención de las imágenes que usaremos en el desarrollo mediante captura de pantalla y posterior tratamiento en Photoshop.

2.7. Modelado UML

A la hora de afrontar el diseño de un sistema complejo, resulta conveniente realizar una planificación previa que permita visualizar el esqueleto del sistema. Para ello se creó el Lenguaje Unificado de Modelado (UML) que permite representar y modelar 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 de modelado, entendiendo como modelo, una simplificación de la realidad. El objetivo del modelado de un sistema es capturar las partes esenciales del mismo. Para facilitar este modelado, se realiza una abstracción y se plasma en una notación gráfica, lo que se conoce como modelado visual. Este modelado visual permite manejar la complejidad de los sistemas a analizar o diseñar y es independiente del lenguaje usado en la implementación.

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

Sus funciones son:

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

Un modelo UML está compuesto por tres clases de bloques de construcción:

- Elementos: son abstracciones de cosas reales o ficticias
- Relaciones: sirven para ligar los elementos entre si
- Diagramas: son colecciones de elementos con sus relaciones.

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

Los diagramas que se utilizarán más adelantes en el diseño de la aplicación son el de clases, de estados y de secuencia.

2.7.1. Diagrama de clases

Un diagrama de clases muestra un conjunto de clases, interfaces y colaboraciones, así como sus relaciones. Estos diagramas son los más comunes en el modelado de sistemas orientados a objetos. Abarcan la vista de diseño estática de un sistema, como se puede ver en el ejemplo del personal universitario en la Figura 2-6.

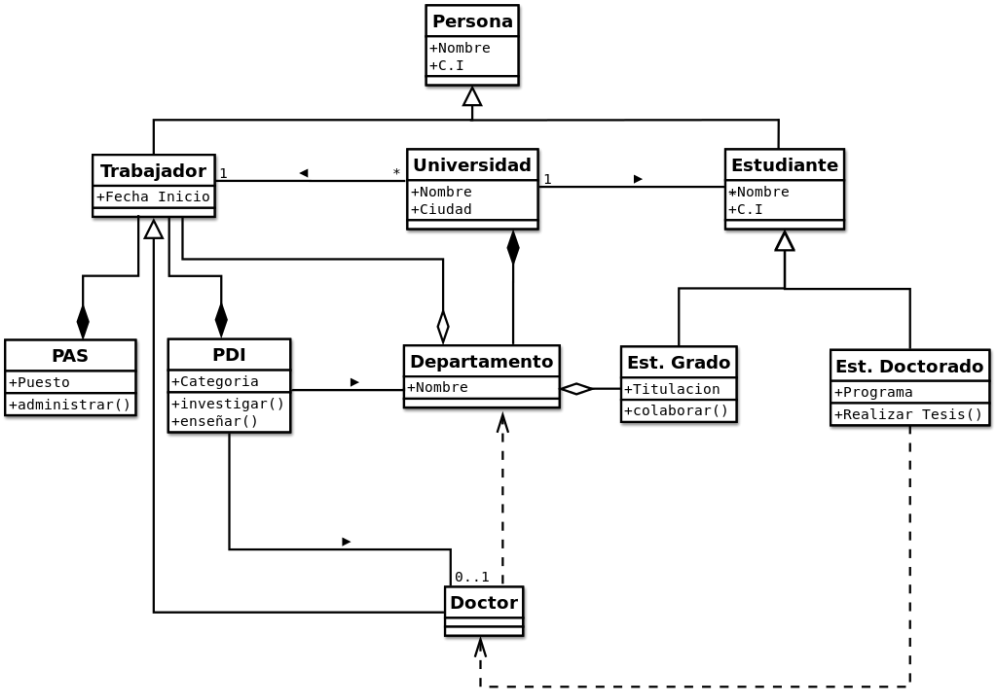

Figura 2-6. Diagrame de clases del personal universitario.

2.7.2. Diagrama de estados

Se utilizan para modelar los aspectos dinámicos de un sistema. Se pueden usar para modelar un caso de uso, una clase o un sistema completo. Estos diagramas pueden ser útiles para modelar la vida de un objeto, como se puede ver en la Figura 2-7.

La mayoría de las veces se utilizan para modelar el comportamiento de objetos reactivos. Un objeto reactivo es aquel para el que la mejor forma de caracterizar su comportamiento es señalar cuál es su respuesta a los eventos lanzados desde fuera de su contexto. Estos objetos tienen un ciclo de vida bien definido, cuyo comportamiento se ve afectado por su pasado.

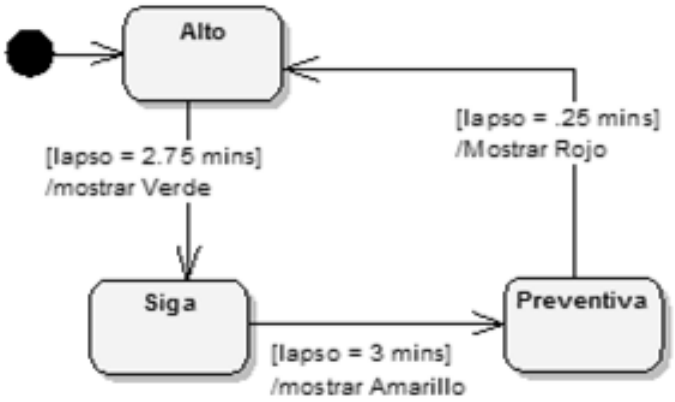

Figura 2-7. Diagrama de estados de un semáforo.

2.7.3. Diagrama de secuencia

Son efectivos para modelar la interacción entre objetos en un sistema. El diagrama de secuencia muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo y se modela para cada método de la clase.

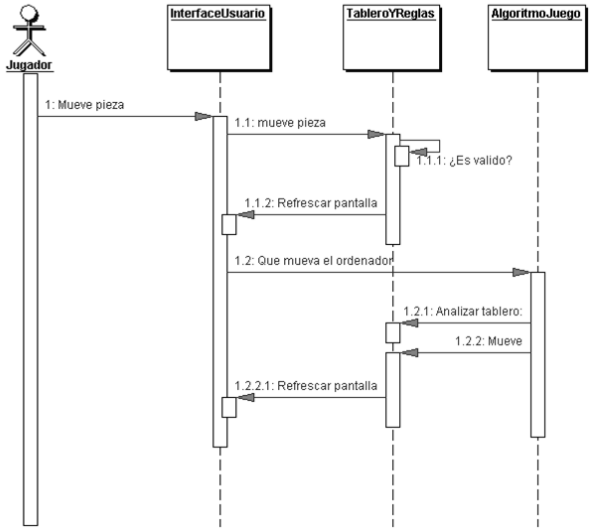

Figura 2-8. Diagrama de secuencia de un videojuego de ajedrez.

Este diagrama contiene detalles sobre la implementación, incluyendo los objetos y las clases que se usan para implementar el escenario, así como los mensajes intercambiados entre los objetos. Dichos mensajes se dibujan cronológicamente desde la parte superior del diagrama hasta la parte inferior. En la Figura 2-8 se puede ver un ejemplo de este tipo de diagramas.

Capítulo 3. Descripción de la aplicación

3.1. Introducción

En este capítulo se explicará las funciones y el desarrollo del juego desde la perspectiva del usuario, sin entrar en aspectos técnicos ni explicar el funcionamiento. En cierta medida, es un manual de usuario pormenorizado donde se explican todos los aspectos de la aplicación.

Comenzará con las posibilidades y el resultado de la instalación de la aplicación en un dispositivo. Posteriormente se hará un recorrido por cada una de las pantallas que conforman la aplicación, dando a conocer las opciones, imágenes y estados que el usuario podrá encontrarse durante la ejecución del programa.

Como se comentó en el capítulo de introducción, se han desarrollado dos modos de juego basados en el videojuego “Log!cal”. Después de presentar las pantallas y menús, se dedicará una sección donde se precisará con mayor detalle los aspectos que distinguen los dos modos de juego, así como una descripción minuciosa del mismo y la manera de proceder para jugar.

En la siguiente sección del capítulo, se especificarán y analizarán los elementos que integran el videojuego: los elementos básicos que se encuentran en cada pantalla, así como los elementos que van apareciendo en los sucesivos niveles que buscan incrementar la dificultad. Se presentarán los niveles del juego, la estructura de cada uno y los elementos que lo componen. Por último, se dedicará una sección al apartado sonoro, indicando efectos sonoros y melodías y cuándo son utilizados.

3.2. Instalación

Para la instalación de la aplicación se necesita un dispositivo con sistema operativo Android. Al no estar subida a ninguna plataforma de distribución de aplicaciones como Google Play o Aptoide, existen sólo dos maneras de hacer la instalación:

- Mediante el programa Eclipse, conectando directamente el dispositivo a un ordenador que posea el código y ejecutándolo en dicho dispositivo.
- Obteniéndose el archivo apk y ejecutándose, habiendo sido configurado el dispositivo para aceptar la instalación de aplicaciones de fuentes desconocidas.

Una vez realizada la instalación, si todo ha ocurrido de manera correcta, debe aparecer en el dispositivo un icono junto al nombre LogicalBT. Podemos observar el resultado dentro de un dispositivo Huawei Ascend P6 en la Figura 3-1. Todas las capturas de pantalla de la aplicación ejecutándose en un dispositivo que aparecen en las figuras de este documento, son obtenidas de dicho modelo de terminal.

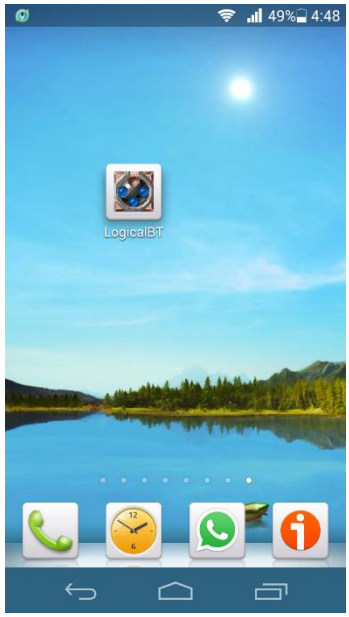

Figura 3-1. Aspecto del acceso a la aplicación una vez instalada.

3.3. Pantallas y menús

3.3.1. Inicio

Lo habitual al iniciar un videojuego, es encontrarte con un logotipo que haga referencia a los productores del juego. A veces contiene algún tipo de animación y en ocasiones contiene los créditos del equipo que ha diseñado el juego, pero esta introducción no suele durar más que unos pocos segundos.

La primera pantalla, que aparece cuando se inicia el juego, es una imagen en la que se ejecuta un zoom progresivo. En la imagen se puede leer el texto “el proyecto Log!cal” sobre un fondo difuminado.

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

En la Figura 3-2, que es una captura de esta pantalla, puede apreciarse como el fondo es una captura de pantalla del proyecto abierto en Eclipse a la que se le ha aplicado un desenfoque gaussiano.

Tras 3 segundos en esta pantalla, el juego pasa a la pantalla Menú principal.

3.3.2. Menú principal

La pantalla Menú principal posee un fondo realizado a partir de una captura de pantalla de un nivel del juego, como se puede apreciar en la Figura 3-3.

Figura 3-3. Pantalla del menú principal.

En él se distingue el título del juego “Log!cal BT” y los textos a pulsar para la ejecución de acciones:

- Un jugador
- Dos jugadores
- Opciones
- Cómo jugar
- Actualizar
- Sobre nosotros
- Salir

Si se pulsa sobre alguno de los textos, este texto se volverá azul hasta que el dedo sea despegado de la pantalla.

Las acciones que se realizan cuando se pulsa alguno de los textos son:

- *Un jugador* redirige a la pantalla Menú Niveles.
- *Dos jugadores* redirige a la pantalla Menú Niveles dos jugadores.
- *Opciones* redirige a la pantalla Opciones.
- *Cómo jugar* redirige a la pantalla Cómo Jugar
- *Actualizar* redirige a la pantalla Actualizar.
- *Sobre nosotros* crea una ventana emergente con los créditos del juego dando la opción de dirigirte a http://antonioandujar.es a través del navegador del dispositivo o de cerrar dicha ventana pulsando el botón

Cerrar creado junto a la ventana, o pulsando el botón *Volver* del dispositivo. Los créditos pormenorizados se pueden ver en la Figura 3-4.

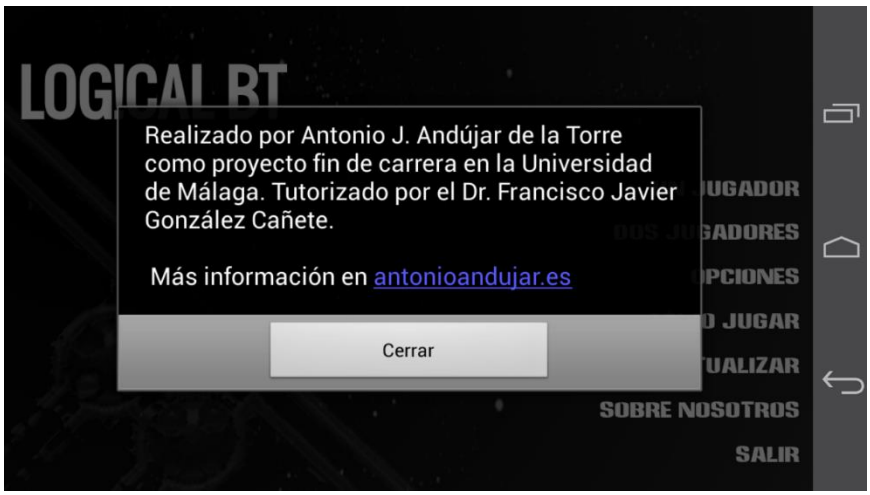

Figura 3-4. Ventana emergente que aparece al pulsar "Sobre nosotros"

- *Salir* crea una ventana emergente de confirmación en la que se da la opción de salir definitivamente del juego o de cancelar, en cuyo caso, al igual que si se pulsa el botón volver del dispositivo, se cerraría la ventana.

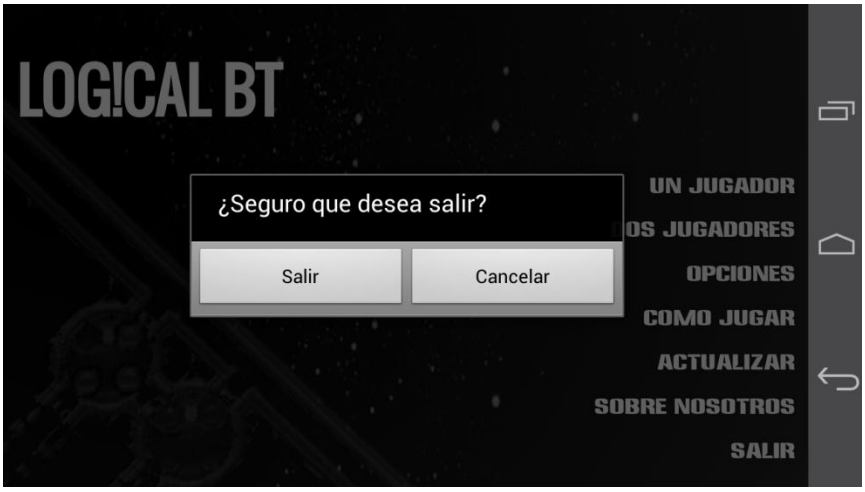

Figura 3-5. Mensaje de confirmación para abandonar la aplicación.

En caso de que se pulse el botón *Volver* del dispositivo dentro de esta pantalla sin que hubiese ninguna ventana emergente, la aplicación actúa como si se hubiese pulsado la opción *Salir*. Ambas maneras de salir de la aplicación dan previamente el mensaje que se aprecia en la Figura 3-5.

3.3.3. Menú Niveles

El menú niveles crea una matriz de iconos de los niveles disponibles. Si en el dispositivo se ha conseguido la victoria en alguno de los niveles con anterioridad, se verá reflejado en el icono del nivel correspondiente con una representación del máximo número de estrellas que se hubiese conseguido en dicho nivel, o en caso de tener restringido el acceso a ese nivel, un candado. Esta pantalla se actualizará cada vez que se provenga de la pantalla Menú principal o se vuelva del juego tras obtener una nueva puntuación máxima en algún nivel durante el transcurso de la partida.

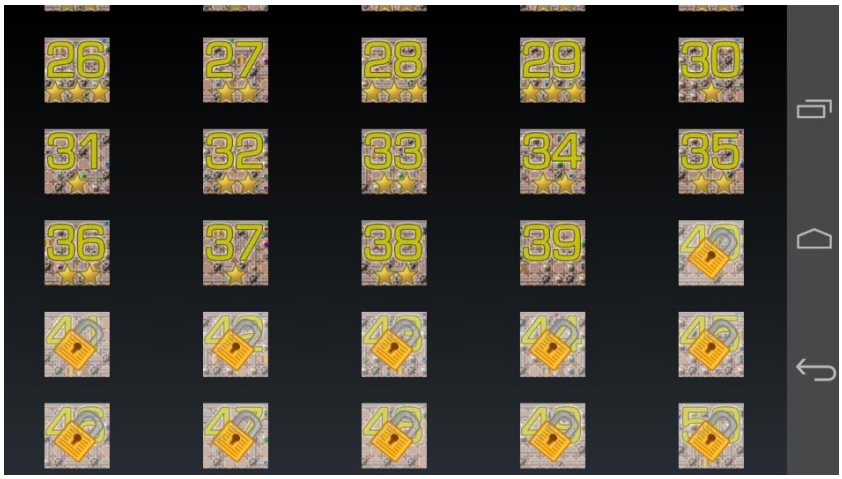

Figura 3-6. Pantalla del Menú Niveles.

Adicionalmente, en caso de haberse superado en el dispositivo la totalidad de los niveles, sonará en el dispositivo “eres un fenómeno”, fragmento del audio de un anuncio de GameBoy de los años 90 [26] en homenaje a dicha plataforma.

La disposición de los iconos está diseñada para mostrarse en 5 columnas, aunque el tamaño de estos lo decide cada dispositivo y, en caso de que no cupiesen, se verían menos columnas. Al sólo haber añadido las imágenes de los iconos en un solo tamaño, es posible que en dispositivos que exijan una definición alta, los iconos no posean un tamaño adecuado para distinguir sus elementos más pequeños.

Los iconos son una captura de pantalla parcial de cada nivel, con el número de nivel sobre ésta y las estrellas o candado si los hubiese. En la Figura 3-6 se puede observar el Menú niveles con el usuario habiendo superado hasta el nivel 38.

Al pulsar uno de los iconos que no tenga candado, la aplicación ejecutará el modo “un jugador” en el nivel seleccionado.

En caso de pulsar el botón *Volver* del dispositivo, la aplicación volverá a la pantalla Menú principal.

3.3.4. Dos jugadores

Al pulsar sobre dos jugadores, lo primero que hace la aplicación es detectar si el dispositivo dispone de Bluetooth. Si no dispone, lo notifica con una ventana emergente.

En caso de disponer de Bluetooth, pero éste estar desactivado, en la aplicación aparece una ventana emergente que da la opción de activarlo, como puede apreciarse en la Figura 3-7.

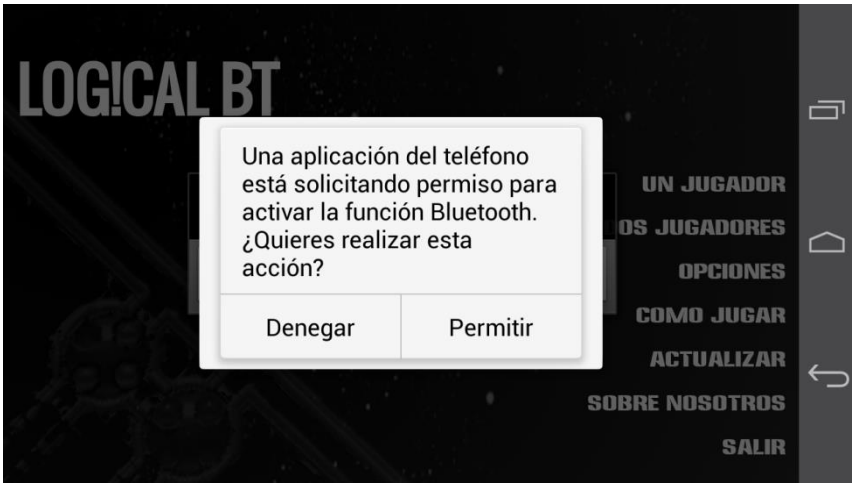

Figura 3-7. Ventana emergente pidiendo permiso para activar el Bluetooth del dispositivo.

En caso de permitir la activación del Bluetooth, la aplicación notificará cuándo se esté realizando dicha operación. Se puede observar en la Figura 3-8. Una vez realizada, las consecuencias de esta acción se notificarán en el dispositivo.

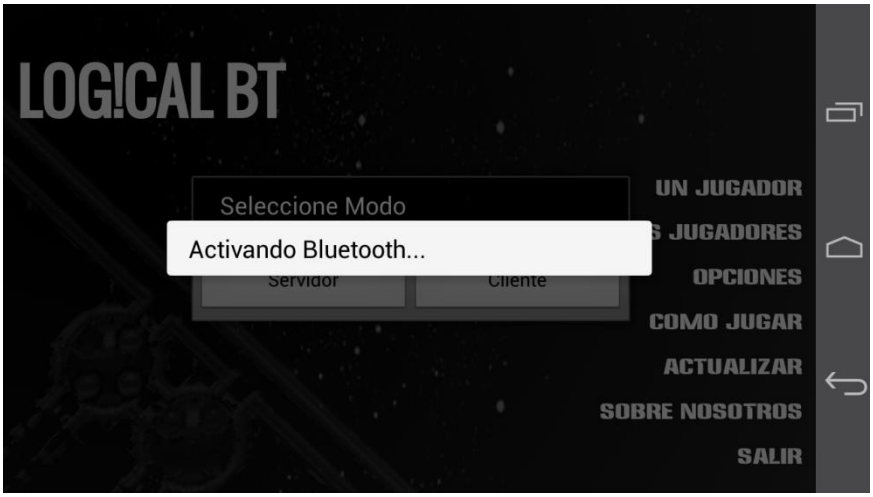

Figura 3-8. Mensaje mientras se procede a la activación del Bluetooth.

En algunos dispositivos la visibilidad será permanente y en otros dispondrá de 120 segundos. En el caso del Huawei Ascend P6, es de 120 segundos. La notificación del mensaje se puede ver en la Figura 3-9.

Figura 3-9. Notificación de visibilidad.

Con el Bluetooth activo, la aplicación dará dos opciones en una ventana emergente: ser el servidor o ser el cliente de la partida creada. Se puede observar en la Figura 3-10.

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

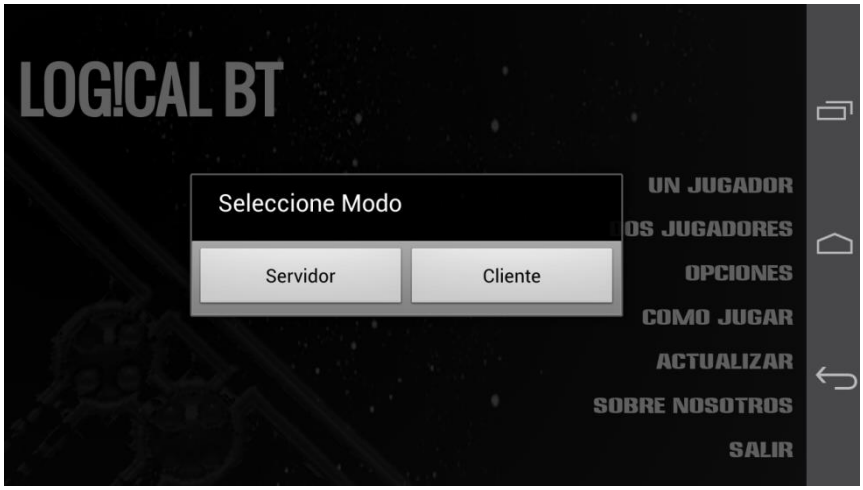

Figura 3-10. Mensaje de selección de modo.

En caso de pulsar el botón Cliente, la aplicación dará la opción de buscar dispositivos o seleccionar algún dispositivo emparejado anteriormente. Si el dispositivo que actúa como servidor está dentro del área de Bluetooth, aparecerá en una lista junto a los otros dispositivos con Bluetooth dentro de esa área. Si selecciona el dispositivo servidor dentro de esa lista y se aceptan las solicitudes de sincronización, aparecerá en el nivel seleccionado y se añadirá ese dispositivo a la lista de dispositivos emparejados. En la Figura 3-11 se pueden observar cómo se da la oportunidad de enlazar con distintos dispositivos previamente emparejados.

Figura 3-11. Lista de dispositivos previamente emparejados.

La solicitud de sincronización aparece en ambos móviles y, aunque lo usual 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 3-12 muestra el caso usual, poder aceptar la sincronización sin necesitar de aparcar la aplicación.

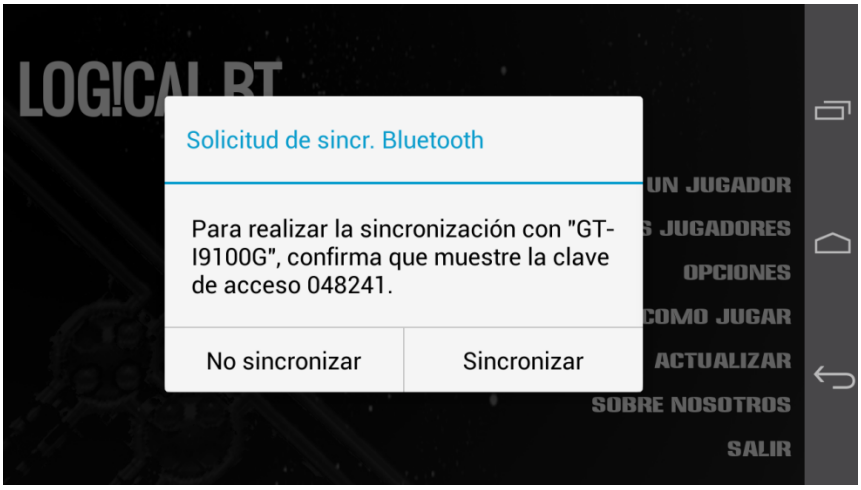

Figura 3-12. Solicitud de sincronización vía Bluetooth.

Tras ser aceptada por ambos dispositivos, el dispositivo en modo Cliente irá a la pantalla de partida Esperando 1, mientras que el dispositivo en modo Servidor entrará en una pantalla similar al Menú Niveles donde podrá seleccionar el nivel a jugar. Una vez seleccionado, se trasladara a la pantalla de partida Esperando 2.

3.3.5. Opciones

La pantalla Opciones muestra cuatro botones, una barra indicativa y una cajetilla con la posibilidad de introducir texto, cada uno con un texto de referencia. La Figura 3-13 es una captura de esta pantalla.

- *Botón Música:* Activa o desactiva la música del juego. Por defecto está activado. Se modifica pulsando el botón. En caso de ser modificado se verá reflejado en el botón y aparecerá momentáneamente un texto con el cambio introducido.
- *Botón Sonido:* Activa o desactiva los efectos de sonido del juego. Por defecto está activado. Se modifica pulsando el botón. En caso de ser modificado se verá reflejado en el botón y se mostrará momentáneamente un texto con el cambio introducido.

- *Barra de volumen:* Modifica porcentualmente el volumen de la música del juego. Por defecto tiene el 100% del volumen. Se modifica arrastrando el marcador dentro de la barra y se el cambio se ve reflejado numéricamente a su derecha.
- *Nombre del jugador:* Cajetilla donde se puede introducir el nombre de un jugador distinto para que así distintos usuarios puedan guardar sus progresos en un mismo dispositivo. Una vez introducido el texto, la aplicación sólo registrará los cambios con el botón *Cambiar al jugador.*
- *Cambiar al jugador:* Una vez introducido el nombre del jugador en la cajetilla correspondiente, si se pulsa este botón la aplicación cargará las puntuaciones y las opciones guardadas del usuario indicado. En caso de que se pulse con un nombre no usado anteriormente, se cargarán las opciones por defecto y tendrá las puntuaciones reiniciadas.
- *Salvar:* Vuelve al Menú Principal y guarda las modificaciones realizadas en las opciones. Esta acción se ve reflejada con un pequeño texto en el Menú Principal.

Figura 3-13. Pantalla Opciones.

La pantalla Opciones refleja los cambios introducidos una vez salvados y estos cambios, incluido el último usuario, serán cargados cada vez que se inicie el juego. En caso de no haber modificado las opciones, la pantalla reflejará los valores por defecto. En caso del nombre del usuario, el nombre por defecto será “Jugador”.

En caso de cargar el jugador habiendo escrito “Miyamoto” en la cajetilla Nombre del jugador, el usuario tendrá activado el *modo trampas* que le permitirá acceder a cualquier nivel en el menú niveles, independientemente de los niveles pasados. Este *modo trampas* ha sido creado para poder comprobar los niveles durante el desarrollo y la razón de usar el nombre Miyamoto es en homenaje a Shigeru Miyamoto, diseñador y productor de videojuegos [27]. El *modo trampas* permanecerá ligado al nombre del jugador anterior a cargarlo.

En caso de pulsar el botón *Volver* dentro de la pantalla Opciones, la aplicación pasará a Menú principal sin llegar a salvar los cambios introducidos.

3.3.6. Cómo jugar

La pantalla Cómo jugar carga una imagen con las indicaciones esenciales para comprender la metodología del juego y dispone de un texto “*Continuar”* que carga el Menú principal. En la Figura 3-14 se pude observar las instrucciones que se dan en esta pantalla.

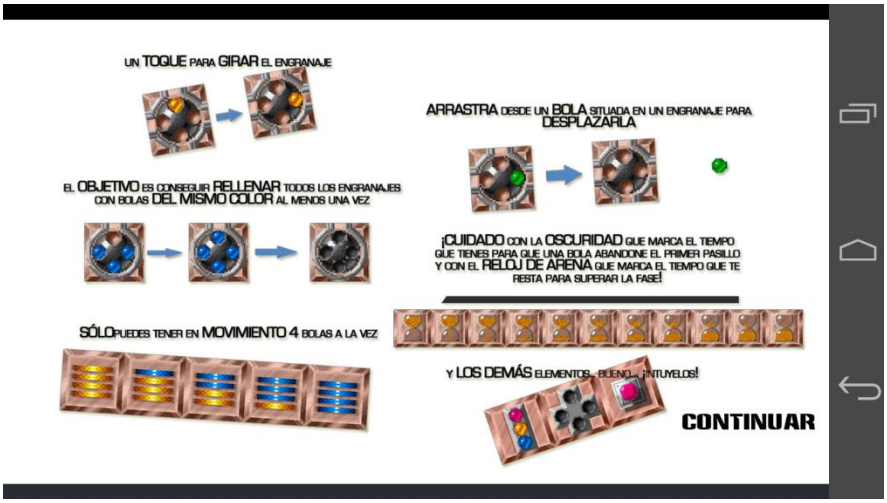

Figura 3-14. Pantalla Como jugar

En caso de pulsar el botón Volver del dispositivo, la aplicación regresa al Menú principal.

3.3.7. Actualizar

La pantalla actualizar, cuya captura puede verse en la Figura 3-15, dispone de un único botón de acción “Obtener datos” que, al pulsar, carga a través de internet el número de la última versión de la aplicación.

Figura 3-15. Pantalla Actualizador.

En caso de disponer de la última versión, se verá notificado en la pantalla, indicando que la aplicación está actualizada, como puede verse en la Figura 3-16.

Figura 3-16. Actualizador con la aplicación actualizada.

En caso de existir una versión más novedosa que la instalada, una vez dado al botón obtener datos, se reflejará en la pantalla la versión del dispositivo, la versión subida a internet y aparecerá un nuevo botón de acción que dará la posibilidad de descargar la última versión a través del navegador del dispositivo. La Figura 3-17 muestra un ejemplo de ello, con una versión disponible más novedosa que la instalada.

Si al pulsar obtener datos no hubiese conexión a internet o la obtención de datos fallase, se notificaría en el dispositivo a través de un mensaje que lo refleja.

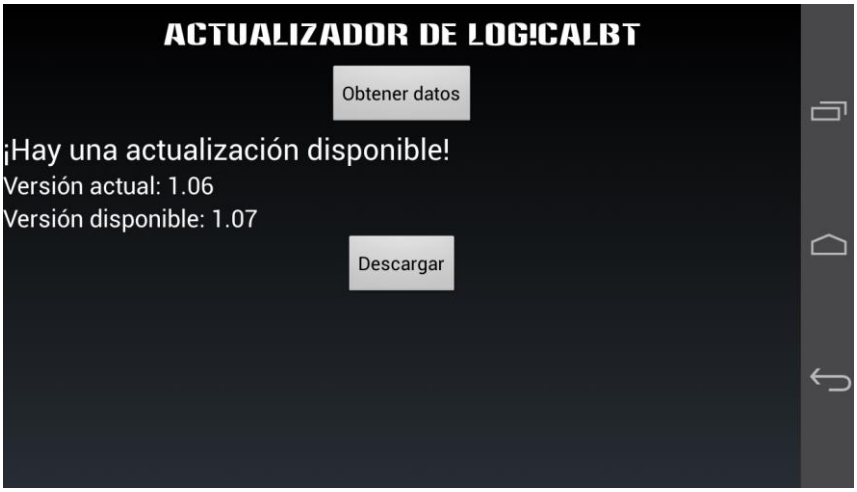

Figura 3-17. Actualizador con una versión más reciente de la aplicación.

En cualquier caso, si se pulsa el botón *Volver*, la aplicación cargará Menú principal.

3.4. Pantallas de la partida

Durante una partida, tanto de un jugador como de dos jugadores, pueden aparecer distintas pantallas. A continuación se dará una descripción detallada de cada una. En cada una de estas pantallas, si se pulsa el botón Volver del dispositivo, la aplicación volvería a la pantalla de donde procediese, es decir, o Menú Niveles o Menú Niveles Dos Jugadores.

3.4.1. Preparado

Una vez se inicia una partida de un jugador o de dos jugadores en modo cliente tras la pantalla Esperando 1, la primera pantalla que aparece tiene la estructura del nivel elegido y el mensaje “Preparado. Pulse la pantalla para continuar”. El juego permanecerá parado hasta que se pulse cualquier lugar de la pantalla y entonces será cuando comience la partida. Se puede observar esta pantalla en la Figura 3-18.

Figura 3-18. Pantalla Preparado con sólo la estructura del juego.

Esta pantalla también aparece en caso de que el dispositivo salga del juego y vuelva a él pero el fondo no sólo refleja la estructura, sino todos los elementos en la posición que tenía la partida en el momento que se interrumpió la aplicación. Ningún elemento se moverá hasta que se restaure la aplicación y se pulse la pantalla. Puede notarse las diferencias mencionadas si se compara la Figura 3-19 con la Figura 3-18.

Figura 3-19. Pantalla Preparado tras pausa.

3.4.2. Esperando 1

Es una pantalla que únicamente aparece en el modo dos jugadores. Cuando uno de los dispositivos está en modo cliente, y tras aceptar la sincronización, aparecerá en esta pantalla hasta que el dispositivo en modo servidor elija nivel. Una vez se elija nivel, el dispositivo pasará a la pantalla

preparado. Podemos observar en la Figura 3-20 como el texto “Esperando. Otro jugador debe elegir nivel” está sobre una estructura de nivel sin ningún elemento.

Figura 3-20. Pantalla Esperando 1.

3.4.3. Esperando 2

Solo aparece en el modo que implica a dos jugadores. Al empezar el nivel, en caso de ser servidor o en caso de que, dentro de una partida, el otro dispositivo entre en modo pausa. Se permanecerá en esta pantalla hasta que en el otro dispositivo se pulse en su pantalla Preparado. No tiene ninguna interacción con el juego exceptuando el botón Volver del dispositivo, en cuyo caso sale a Menú Niveles y corta el enlace establecido con el otro dispositivo mediante Bluetooth. El mensaje que enseña esta pantalla puede verse en la Figura 3-21.

Figura 3-21. Pantalla Esperando 2

3.4.4. Victoria

En caso de superar el nivel, aparecerá la pantalla Victoria. Contiene un letrero con el texto “Victoria”, el número de puntos obtenidos, las estrellas conseguidas (las cuales dependen de los puntos obtenidos), un botón con el símbolo *repeat* que reinicia el nivel y un botón con una flecha que iniciará una partida en el nivel siguiente. Pulsar este último botón en el último nivel no reflejará acción alguna. Esta pantalla se muestra en la Figura 3-22.

Figura 3-22. Pantalla Victoria

3.4.5. Derrota

En caso de no superarse el nivel, aparecerá la pantalla Derrota, con un letrero indicativo con el texto Derrota y un botón con el símbolo *repeat* que reinicia el nivel. Esta pantalla se muestra en la Figura 3-23.

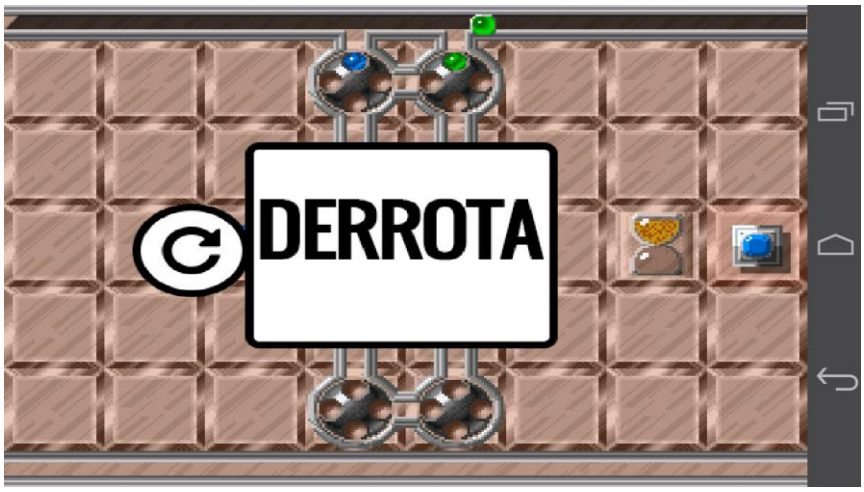

Figura 3-23. Pantalla Derrota

3.5. Instrucciones de juego

El videojuego admite dos tipos de partida: un jugador o dos jugadores (en modo cooperativo). En ambos casos el objetivo es completar cada engranaje con bolas del mismo color al menos una vez antes de que se termine el tiempo límite para completar el nivel. Los elementos extra que frustran esta posibilidad, así como la manera de salvar los impedimentos que esos elementos ponen, se detallan en la sección 3.7.

Para superar el nivel, se dispone de dos acciones:

- Girar engranaje: gira 90º en sentido de las agujas de reloj el engranaje controlado que es pulsado.
- Soltar bolas: suelta del engranaje la bola seleccionada hacia un lugar habilitado. Se ejecuta al arrastrar el dedo desde el engranaje hacia otra casilla con la dirección deseada.

Con estas dos acciones se ha de superar cualquier nivel en cualquiera de los modos.

3.5.1. Modo Un jugador

En este modo de juego un solo jugador debe intentar superar el nivel. Para ello se controlarán todos los engranajes. Aparecerán bolas sólo por el pasillo superior y se tendrá que conseguir ubicar dicha bola en algún engranaje antes de que el *tiempo por bola* se agote. Además, posee un tiempo límite para superar el nivel, que, en caso de ser superado, hará que se traslade a la pantalla derrota.

3.5.2. Modo dos jugadores

En el modo dos jugadores, quien hace de servidor controla los engranajes de las dos filas superiores. Quien hace de cliente los de las dos filas inferiores, mientras la fila central es controlada por ambos jugadores. Podemos apreciar una perspectiva de una partida en este modo en la Figura 3-24.

Figura 3-24. Partida en modo dos jugadores.

Aparecen bolas tanto por el pasillo superior como por el pasillo inferior, cada una con un límite de *tiempo por bola* independiente. Si uno de estos dos límites o el límite de tiempo establecido para superar el nivel es superado, hará que ambos jugadores pasen a la pantalla derrota. El color de las bolas que salen por los pasillos son independientes y así es reflejado en el elemento *siguiente bola*, analizado con más detenimiento en el apartado 3.6.3.

3.6. Elementos asiduos

Estos son los elementos comunes en la mayoría de los niveles desde el primer nivel. Son los que se encargan de crear una estructura básica o dar información. Los tres primeros niveles están compuestos únicamente de estos elementos. La Figura 3-25 muestra al primero de ellos.

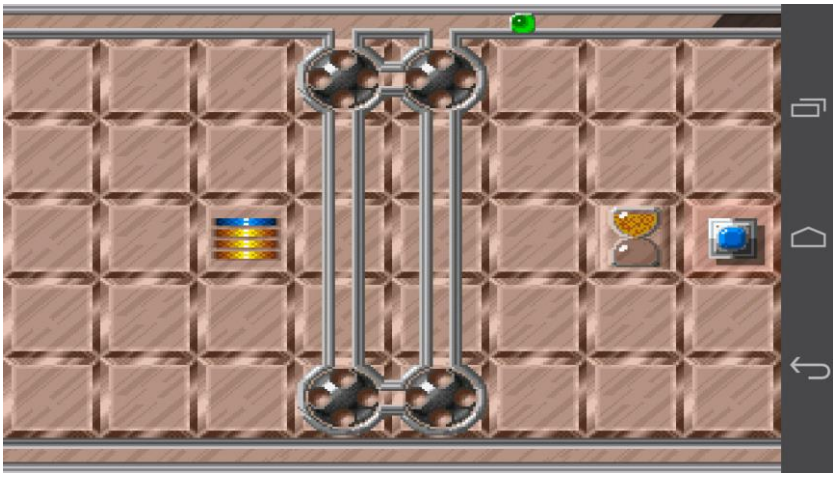

Figura 3-25. Nivel 1, estructura únicamente con elementos asiduos.

3.6.1. Engranajes

En los engranajes se cimienta la jugabilidad del videojuego. Es el único elemento activo, en el que se puede realizar la función de girarlo 90º en sentido de las agujas de reloj al tocarse en la pantalla o soltar alguna bola por algún lugar habilitado si se arrastra el dedo desde el engranaje hacia el elemento en el que se quiere soltar la bola. Uno de estos engranajes se pueden apreciar en la Figura 3-26.

Figura 3-26. Imagen de un engranaje por superar.

Un engranaje se completa, a no ser que algún elemento extra lo impida, al tener en cada una de sus cuatro casillas bolas del mismo color. En tal caso el engranaje se considerará completado, las bolas que contenía desaparecerán y el color de los huecos será gris desde ese momento.

3.6.2. Pasillos y cruces

Son los elementos que indican al usuario por dónde puede hacer pasar una bola. No alteran en ningún momento la dirección o color de la bola. Uno de los pasillos se puede ver en la Figura 3-27.

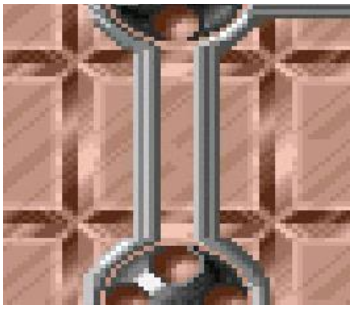

Figura 3-27. Imagen de un pasillo vertical.

3.6.3. Siguiente bola

Indica cuál será la siguiente bola que saldrá por el pasillo. En modo un jugador, lo indicará con una representación completa de una bola con el color que toque, mientras que en modo dos jugadores, la mitad superior indica el color de la bola que saldrá por el pasillo superior y la mitad inferior indica el color de la bola que saldrá por el pasillo inferior (Figura 3-28).

Figura 3-28. Imagen del elemento que indica la siguiente bola.

3.6.4. Reloj de arena

Indica el porcentaje del tiempo superado con respecto al tiempo límite del nivel. El reloj se va modificando cada vez que transcurre un 10% de ese porcentaje. Se puede ver un reloj con más del 90% del tiempo en la Figura 3-29.

Figura 3-29. Imagen del reloj de arena.

Llegar hasta el tiempo límite o hasta el tiempo por bola límite, que se explica en el apartado 3.6.6, son las dos únicas maneras de que se cargue la pantalla Derrota.

3.6.5. Número de bolas

El número de bolas que se puede tener en circulación son cuatro. Esto no incluye las que van por los pasillos superior e inferior. Este elemento es un contador que indica cuántas bolas hay en circulación y aporta una rápida referencia de cuántas quedan. Se puede apreciar este contador de bolas con la mitad de las bolas activas en la Figura 3-30.

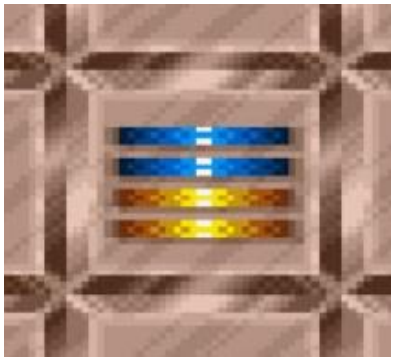

Figura 3-30. Imagen del contador de bolas.

3.6.6. Tiempo de bola

Cada uno de los pasillos por donde salen bolas tiene un tiempo límite para que la bola sea ubicada en un engranaje próximo al pasillo. Este tiempo se ve representado en la misma barra del pasillo, siendo éste oscurecido progresivamente, llegando al límite cuando la barra es oscurecida por completo. Si esto pasase, se cargaría la pantalla Derrota. En la Figura 3-31 se ve como el tiempo de bola, es decir, la parte oscurecida, ha avanzado algo en ese momento.

Figura 3-31. Imagen del tiempo de bola.

3.7. Elementos especiales

Estos elementos tienen como función aumentar la complejidad de los niveles y son añadidos poco a poco conforme van pasando los niveles. Podemos observar una estructura con varios de estos elementos en la Figura 3-32.

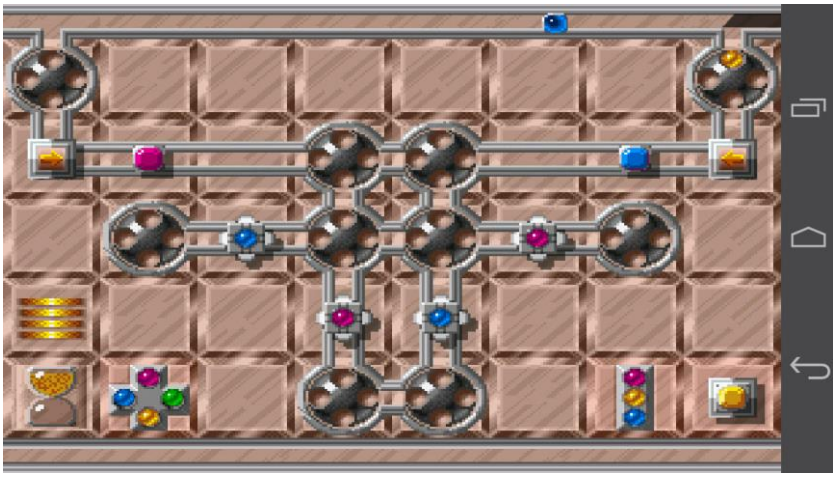

Figura 3-32. Nivel con varios elementos especiales.

3.7.1. Teletransporte

El teletransporte es un elemento que requiere más de uno para poder ser utilizado. Si una bola entra en uno de ellos, aparecerá en otro distinto, es decir, en otras coordenadas pero con la misma dirección. Si la bola es metida en el teletransporte que antes fue usado como destino, aparecería en el teletransporte origen. Funciona con todos los sentidos por los que la bola pueda entrar. Un teletransporte vertical se puede apreciar en la Figura 3-33.

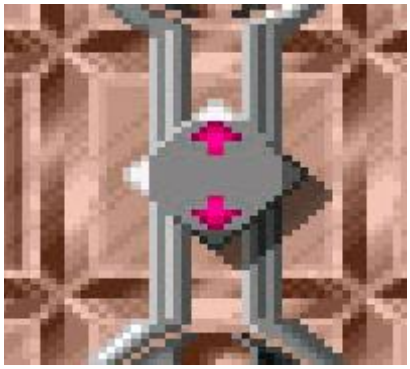

Figura 3-33. Imagen del teletransporte.

3.7.2. Filtro

El filtro deja pasar solamente a las bolas de color que indica, haciendo rebotar, es decir, cambiar la dirección 180º, a las bolas que entren en él y no tengan el color que indica. Uno de estos filtros, en modo vertical y filtrando a dejar pasar sólo las bolas amarillas se puede ver en la Figura 3-34.

Figura 3-34. Imagen de un filtro.

3.7.3. Pintura

La pintura modifica el color de las bolas una vez atraviesa por el centro al color que indica el elemento, dejándolo tal cual si el color coincide sin modificar ninguna otra de sus características. La imagen de una pintura horizontal que vuelve las bolas de color rosa se puede apreciar en la Figura 3-35.

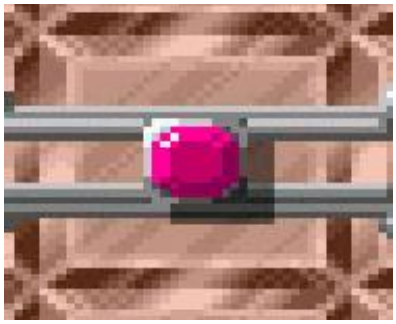

Figura 3-35. Imagen de una pintura.

3.7.4. Semáforo

El semáforo impide a un engranaje que se dé por completado a no ser que sea completado por el color que indique. Una vez completado un engranaje con el primer color, analizará la estructura para ver si hay alguno del segundo color y si encuentra alguno, dará el engranaje como completado y buscará para el tercer color. Conforme vaya completando los colores pedidos, irán desapareciendo esos colores del elemento hasta quedar vacío, dejando desde entonces de interferir en el juego. Se puede ver al semáforo en la Figura 3-36.

Figura 3-36. Imagen del semáforo.

3.7.5. Direccionador

El direccionador cambia la dirección de la bola cuando ésta llega al centro del elemento con la dirección que indica la flecha visible. En caso de coincidir direcciones, la bola no sería modificada. La Figura 3-37 representa un direccionador donde a cuyas bolas que vengan por la izquierda o por arriba, se les cambiará la dirección para que fuesen hacia arriba.

Figura 3-37. Imagen de un direccionador.

3.7.6. Cruz de bolas

La cruz de bola exige un patrón concreto en cualquiera de los engranajes. Han de coincidir tanto color, como posición de las bolas. Hasta no realizar dicho patrón, ningún engranaje se podrá considerar como completado. Este patrón nunca tendrá las cuatro bolas del mismo color. Podemos comprobar como en la Figura 3-38 se encuentra uno de estos patrones seleccionado aleatoriamente y como cumple la regla de no exigir las cuatro bolas del mismo color

Figura 3-38. Imagen de la cruz de bolas.

En caso de coincidencia con el semáforo en la estructura de un nivel, la cruz de bolas tiene prioridad, teniendo que realizarse antes el patrón de la cruz de bolas. Una vez resuelto el patrón, el juego comprueba automáticamente si se puede completar algún otro engranaje.

A diferencia del semáforo, en caso de completarse, volverá a restringir los engranajes con un nuevo patrón transcurrido un tiempo. Esto ser verá reflejado mostrándose vacío cuando no tenga, y lleno con el patrón cuando éste sea exigido.

3.8. Niveles

El juego se compone de un total de 50 niveles, cuya dificultad se incrementa a medida que se avanza a través de ellos. La dificultad depende tanto de los elementos empleados, como del número de ellos utilizados y su colocación.

Todas las estructuras de los niveles se encuentran reflejadas en las figuras que van desde la Figura 3-39 hasta la Figura 3-44.

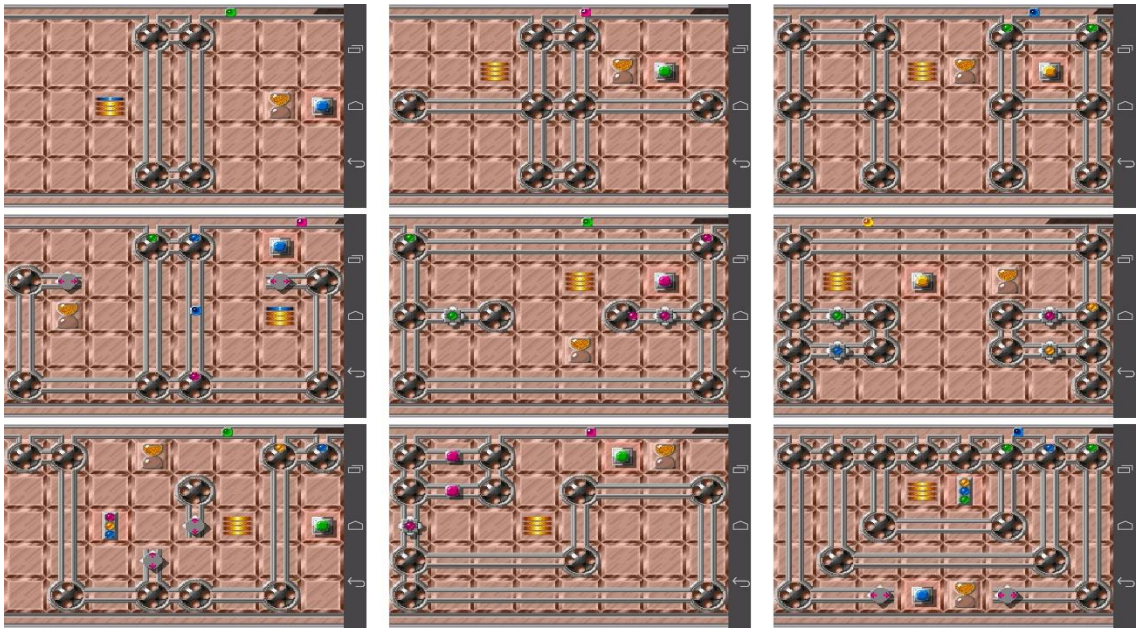

Figura 3-39. Niveles 1 a 9.

Figura 3-40. Niveles 10 a 18.

Figura 3-41. Niveles 19 a 27

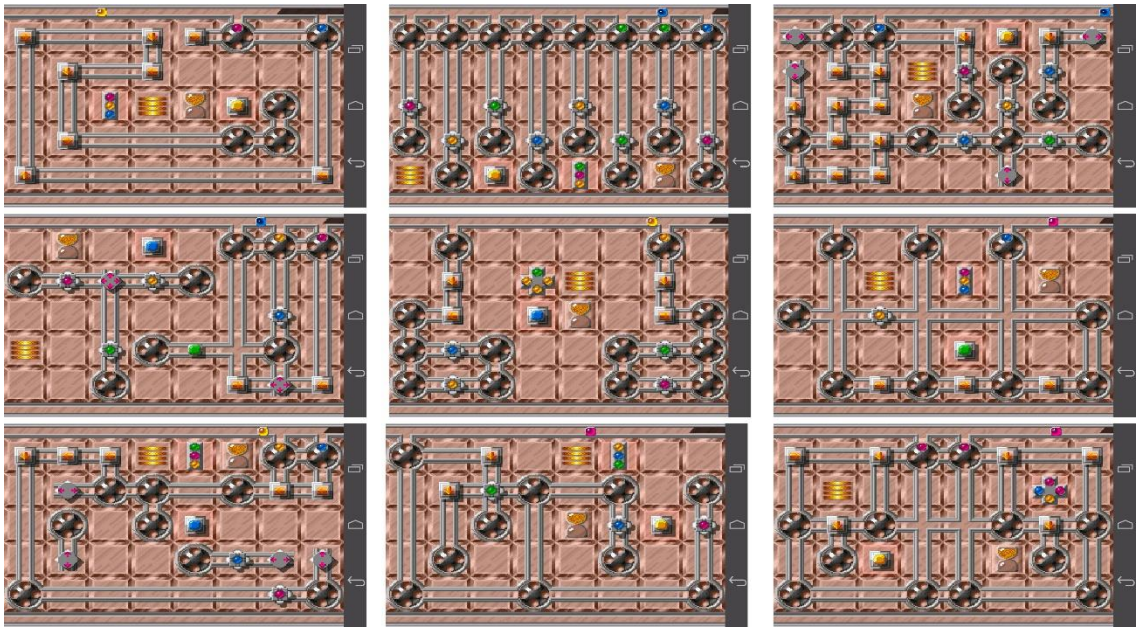

Figura 3-42. Niveles 28 a 36

Figura 3-43. Niveles 37 a 45.

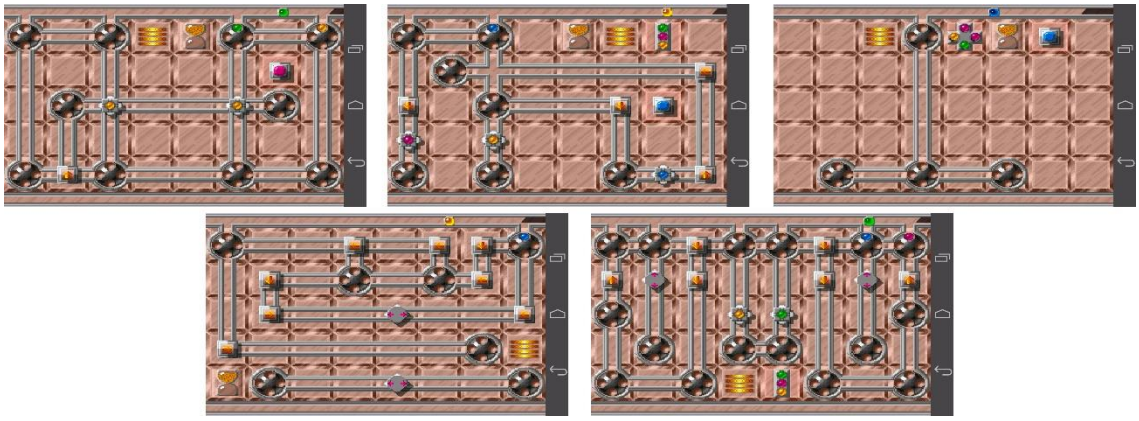

Figura 3-44. Niveles 46 a 50.

3.9. Audios

La aplicación dispone de efectos de sonido y de música, que pueden ser deshabilitados y cuyo volumen es modificable desde la pantalla de opciones.

La aplicación dispone de cuatro melodías que son cambiadas cada cuatro niveles de forma cíclica. Por ejemplo, la primera melodía aparece en los niveles del 1 al 4 y del 17 al 22.

Estas melodías aparecen al cargarse por primera vez en un nivel la pantalla Preparado o la pantalla Esperando 2, y continua hasta la Victoria o Derrota, en cuyo caso el volumen de la melodía pasa al 10%; o hasta que la

partida se interrumpe volviendo al Menú Niveles, en cuyo caso la melodía se interrumpe.

Los sonidos que contiene el juego durante una partida suenan únicamente si la opción Sonidos está activada.

Durante el desarrollo de una partida hay 5 sonidos distintos implementados:

- Un sonido de explosión cada vez que un engranaje se dé por completado.
- Un sonido metálico cada vez que una bola rebote contra un engranaje que tenga lleno el lugar al que correspondería esa bola.
- Un sonido cada vez que una bola salga de un engranaje.
- Un sonido cada vez que una bola entre en un engranaje.
- Un sonido cada vez que se gire un engranaje.

Además dispone de un sonido melodioso cuando se carga la pantalla Victoria y otro sonido melodioso distinto cuando se carga la pantalla Derrota.

Adicionalmente e independientemente de si el sonido está habilitado o no, se carga el sonido explicado en el apartado 3.3.3.

3.10. Puntuación y estrellas

Una vez conseguida la victoria, la aplicación ejecuta un pequeño algoritmo en el que intervienen el tiempo transcurrido y las bolas que en ese momento poseen los engranajes restantes para calcular la puntuación. La máxima puntuación teórica, aunque no real, es de 2000 puntos.

Las estrellas que aparecen dependen de dicha puntuación, dando tres estrellas a las puntuaciones que superen los 999 puntos, dos estrellas a las que se encuentren entre 500 y 999 puntos, ambas incluidas, y una estrella a las victorias con menos de 500 puntos.

Capítulo 4. Diseño de la aplicación

4.1. Introducción

En este capítulo se explica cómo se ha implementado la aplicación para obtener la funcionalidad descrita en el capítulo anterior. Para ello, se indicará de qué manera se han organizado las diferentes clases que componen el programa y cómo se han implementado para contribuir a la funcionalidad global del mismo. Se comenzará por introducir, mediante una pequeña descripción, el marco software donde se sitúa la aplicación desarrollada.

4.1.1. Marco software de la aplicación

La aplicación Android de videojuego se tendrá que ejecutar en el entorno proporcionado por el dispositivo. En esta aplicación software, la interacción con el usuario se realizará a través del teclado, la pantalla y el altavoz del terminal. El dispositivo responderá a las acciones realizadas por el jugador mediante teclado o pulsando la pantalla táctil, mostrando los resultados de dicha interacción y del desarrollo del juego a través de la pantalla y el altavoz (si el sonido estuviera habilitado).

La tecnología Android es un modelo de capas para conseguir un alto grado de portabilidad de las aplicaciones. Esta distribución permite acceder a las capas más bajas mediante el uso de librerías para que así el desarrollador no tenga que programar a bajo nivel las funcionalidades necesarias para que una aplicación haga uso de los componentes de hardware de los dispositivos moviles. Cada una de las capas utiliza elementos de la capa inferior para realizar sus funciones. Un resumen de estas capas lo podemos ver 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 el resto de las capas de la arquitectura. El desarrollador no accede directamente a esta capa, sino que debe utilizar las librerías disponibles en capas superiores.

Las bibliotecas nativas de Android, también llamadas librerías, están escritas en C o C++ y compiladas para la arquitectura hardware específica del teléfono. Éstas normalmente están implementadas por el fabricante, quien también se encarga de instalarlas en el dispositivo antes de ponerlo a la venta. El objetivo de las librerías es proporcionar funcionalidad a las aplicaciones para tareas que se repiten con frecuencia, evitando tener que codificarlas cada vez y garantizando que se llevan a cabo de la forma “más eficiente”.

El entorno de ejecución de Android no se considera una capa en sí mismo, dado que también está formado por librerías. Aquí encontramos las librerías con las funcionalidades habituales de Java así como otras específicas de Android.

El componente principal del entorno de ejecución de Android es la máquina virtual Dalvik. Las aplicaciones se codifican en Java y son compiladas en un formato específico para que esta máquina virtual las ejecute. La ventaja de esto es que las aplicaciones se compilan una única vez y de esta forma estarán listas para distribuirse con la total garantía de que podrán ejecutarse en cualquier dispositivo Android que disponga de la versión mínima del sistema operativo que requiera la aplicación.

La siguiente capa, el *framework* de aplicaciones, está formada por todas las clases y servicios que utilizan directamente las aplicaciones para realizar sus

funciones. La mayoría de los componentes de esta capa son librerías Java que acceden a los recursos de las capas anteriores a través de la máquina virtual Dalvik.

La capa superior de esta pila software la forman, como no podría ser de otra forma, las aplicaciones. En este saco se incluyen todas las aplicaciones del dispositivo, tanto las que tienen interfaz de usuario como las que no, tanto las nativas (programadas en C o C++) como las administradas (programadas en Java), tanto las que vienen de serie con el dispositivo como las instaladas por el usuario. Toda esta estructura se puede ver más detenidamente en la Figura 4-2.

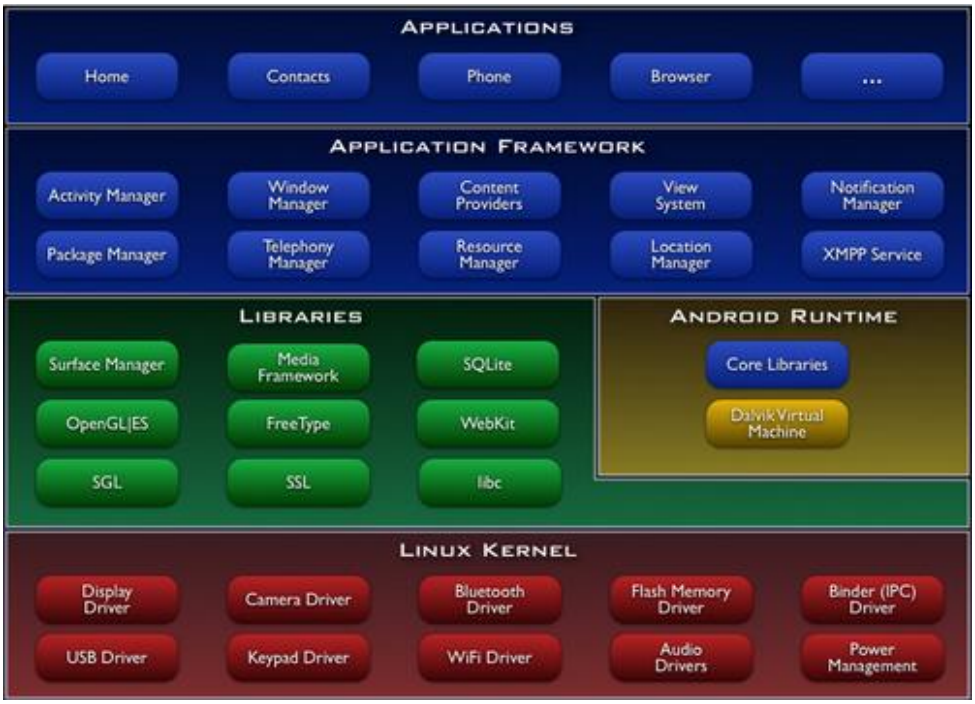

Figura 4-2. Estructura de capas oficial de Android.

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

4.2. Visión general de la aplicación

Para establecer un punto de referencia desde el que tener una visión de conjunto sobre la misma, es decir, por un lado, la distribución e identificación de las clases implementadas, así como las relaciones entre ellas, y por otro, un diagrama que represente de forma esquemática el funcionamiento global del

sistema, conviene conocer algunas características sobre las clases dentro de Android.

4.2.1. Herencia

Uno de los grandes atractivos de la programación orientada a objetos es la facilidad que nos ofrece para la reutilización de código. La programación puede llegar a ser muy repetitiva. La reutilización de código consiste en no volver a escribir una y otra vez los mismos algoritmos para resolver situaciones similares. La clave para conseguir esto en Java, y por consiguiente, en Android, va a ser la herencia.

La idea consiste en definir una nueva clase, que herede de una clase padre y que añada el nuevo atributo. De esta forma podremos añadir nuevas funcionalidades sin tener que modificar la clase de partida. Incluso es posible que ni siquiera se disponga del código de una clase para poder heredar de ella.

La herencia consiste en crear la nueva clase a partir de otra que llamaremos padre. La clase hija va a heredar los atributos y métodos de la clase padre y podrá crear nuevos atributos y métodos para completar su comportamiento. La clase hija también puede volver a definir los métodos de la clase padre.

4.2.2. Layouts

Los *layouts* son elementos no visuales destinados a controlar la distribución, posición y dimensiones de los controles que se insertan en su interior.

Al desarrollar para Android se nos anima a declarar las actividades, con sus correspondientes elementos, en ficheros XML independientes del código. De esta manera se gana en flexibilidad y claridad.

En estos archivos se describe cómo se agrupan los elementos de la actividad utilizando etiquetas XML de un modo muy similar al utilizado en el desarrollo web. Las herramientas de desarrollo interpretan este fichero y generan el código necesario.

Las ventajas de este esquema son:

- Simplicidad y claridad en la descripción de las interfaces gráficas
- Es fácil y rápido cambiar la disposición gráfica
- Se separa el código de la aplicación de la representación gráfica de la interfaz

Sigue siendo posible generar la interfaz escribiéndolo en el código de la clase y esto es una necesidad cuando la interfaz es dinámica y los elementos se deban generar al vuelo durante la ejecución del programa.

4.2.3. Hilos

El *software* en el que se ejecuta un dispositivo Android está basado en eventos. Esto significa que el flujo de ejecución es conducido por eventos de entrada. Estos eventos pueden ocurrir en cualquier orden y en cualquier instante de tiempo.

La ejecución del *software* basado en eventos se basa generalmente en una “cola de eventos”. En esta cola se almacenan los eventos de entrada en el orden en que llegan. Para cada cola, un hilo, en un bucle infinito, lee los eventos de la cola, uno tras otro, y llama a las rutinas de manejo de eventos correspondientes.

Las rutinas pueden ser ejecutadas en un mismo hilo. En ese caso, los eventos son ejecutados uno detrás de otro, es decir, de forma secuencial. Otra alternativa es ejecutar cada rutina en un hilo separado. En este caso, las rutinas se ejecutan simultáneamente, o en paralelo. Esta forma de trabajar tiene una ventaja: el hilo encargado de la cola de eventos puede atender rápidamente nuevos eventos, al no tener que esperar a que termine la ejecución de cada tarea.

4.2.4. División de la aplicación

Para la realización de la aplicación se dividió la estructura en dos grandes bloques. El primero, al que se denominó “bloque de menús” contiene toda el área de menús. La mayoría de 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 un *layout* asociado.

El segundo bloque, denominado “bloque del motor del juego” contiene todas las clases que crean la parte “jugable” de la aplicación. Muchas de las clases heredan de alguna clase del *framework* descrito en el apartado 2.2.8. Todo el apartado gráfico es diseñado en una de las clases que la componen.

La clase Ajustes pertenece a ambas, ya que es posible modificarla tanto consiguiendo puntuaciones altas en el bloque del motor de juego, como cambiando algunas opciones en el bloque de menús.

4.2.5. Clases implementadas del bloque de menús

Las clases que integran el bloque de menús implementan toda la preparación previa a una partida, como puede ser el establecimiento de la conexión Bluetooth para dos jugadores, la selección del nivel a jugar o la modificación de algunas opciones sonoras.

Las clases son:

- Introducción: inicia la aplicación y muestra una pantalla de inicio.
- MenuInicio: implementa el menú inicio.
- MenuNiveles: desarrolla el menú niveles y es la puerta al bloque del motor del juego.
- MenuDosJugadores: hija de MenuNiveles. Desarrolla el menú niveles de dos jugadores, se encarga también del establecimiento de la conexión Bluetooth. Dentro 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.
- MenuOpciones: crea el menú Opciones para la modificación de algunos ajustes.
- Ajustes: guarda y carga ajustes del juego.
- Actualizar: carga la pantalla de actualización del juego

- VersionChecker: ayuda a la clase Actualizar a buscar y descargar versiones más recientes de la aplicación.
- ComoJugar: crea la pantalla Como jugar.

La manera en que estas clases están relacionadas se representa en el diagrama de clases de la Figura 4-3.

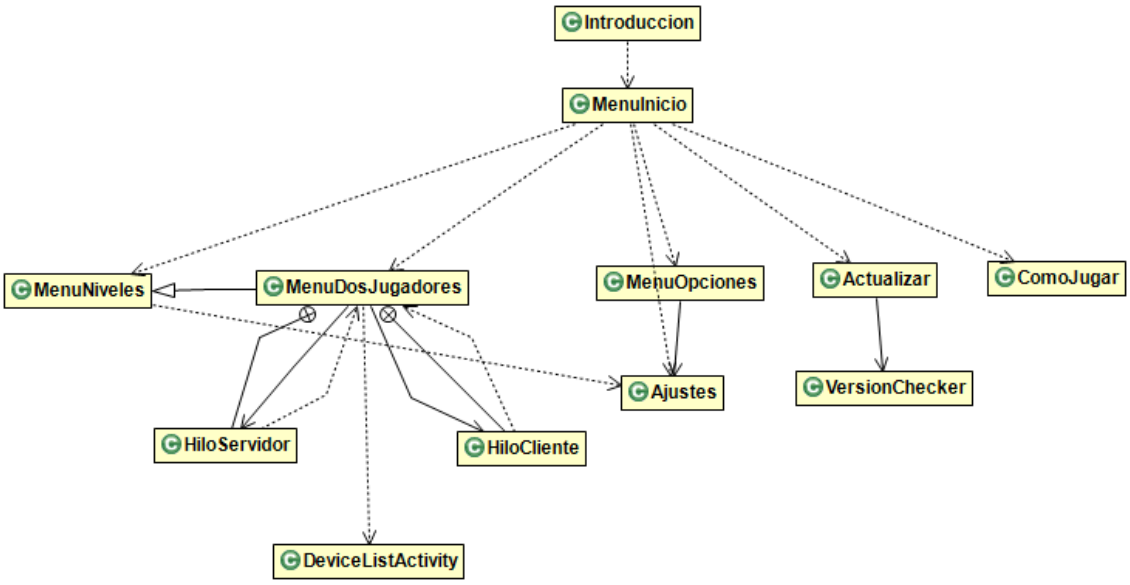

Figura 4-3. Diagrama de clases del bloque de menús

Cada una de estas clases, al igual que las del bloque siguiente, se detallará y se analizará detenidamente en las siguientes secciones.

4.2.6. Clases implementadas del bloque del motor del juego

Las clases implementadas en este bloque controlan tanto el aspecto visual como el desarrollo de la partida y los elementos que la componen. Las principales clases tienen su clase hija para el modo multijugador.

Las clases que componen este bloque son:

- JuegoLogical: arranca el juego en modo un jugador y crea el hilo que se mantendrá escuchando la interacción del usuario con la pantalla.
- JuegolLogicalBT: idéntico a JuegoLogical solo que proviene de la estructura multijugador.
- CargarPantalla: carga las imágenes y los archivos sonoros en Recursos, además de cargar los ajustes y la estructura que se usarán en el juego.

- CargarPantallaBT: heredera de CargarPantalla para el modo de dos jugadores.
- Estructura: tiene implementado cómo irán los elementos en cada uno de los niveles.
- Recursos: lista los recursos que se van a utilizar.
- Ajustes: se cargan las opciones y se guardan los puntos si se mejoran las puntuaciones.
- PantallaJuegoUnJugador: crea toda la interfaz gráfica y controla la interacción del usuario con la aplicación.
- Multijugador: heredera de PantallaJuegoUnJugador que además, 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.
- MundoBT: heredero de Mundo para el apartado multijugador.
- MensajeEnLista: pila que sirve de puente para que los eventos de MundoBT puedan ser enviados por Bluetooth en Multijugador.
- Casilla: tipos y propiedades de los elementos que componen una estructura.
- Bola: propiedades de las bolas que se mueven en la partida.

La manera en que estas clases están relacionadas se representa en el diagrama de clases de la Figura 4-4. Se aprecia cómo, partiendo de la clase JuegoLogical o JuegoLogicalBT, se llega a la clases principales del juego PantallaJuegoUnJugador o Multijugador respectivamente, y cómo estas se relacionan con Mundo o MundoBT, clases encargadas de toda la lógica automática.

La relación entre los dos bloques viene con el arranque de la clase JuegoLogical en MenuNiveles y el arranque del JuegoLogicalBT en MenuNivelesDosJugadores, además de compartir la clase Ajustes.

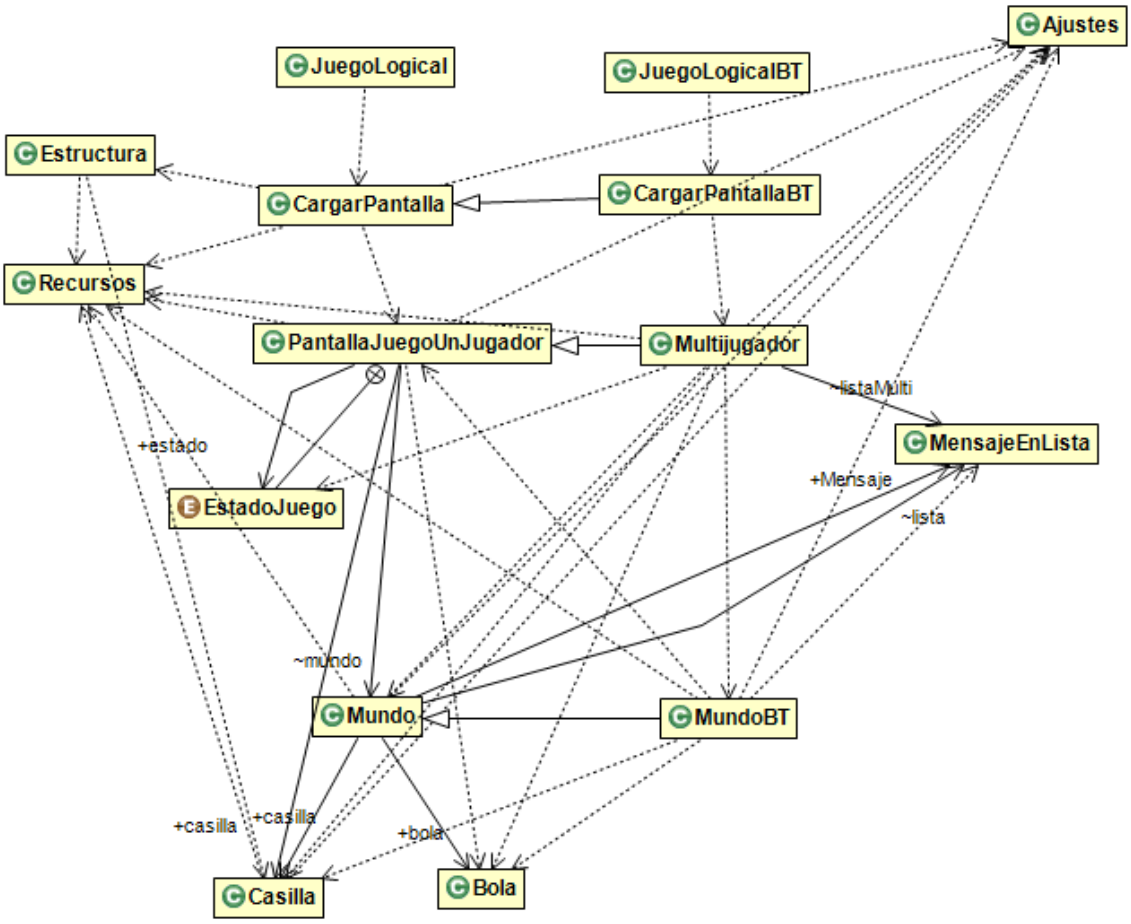

Figura 4-4. Diagrama de clases del bloque del motor de juego.

4.2.7. Diagrama de estados de la aplicación

El diagrama de estados representa, de manera esquemática, el funcionamiento global de la aplicación y es el que se muestra en la Figura 4-5. Básicamente, muestra de forma simplificada la descripción del funcionamiento realizada en el capítulo 3.

A partir del menú principal se puede acceder a los distintos menús. En el de opciones, se pueden modificar distintos parámetros, como el jugador, en cuyo caso se actualizara el menú con sus parámetros, o los ajustes de sonido. El de ajustes da la posibilidad de descargar mediante un navegador externo la nueva versión si la hubiera. El cómo jugar simplemente muestra su pantalla para volver al menú principal. El menú niveles o el de dos jugadores son la puerta al juego. De todos los niveles se puede volver al menú principal usando la tecla *Volver*.

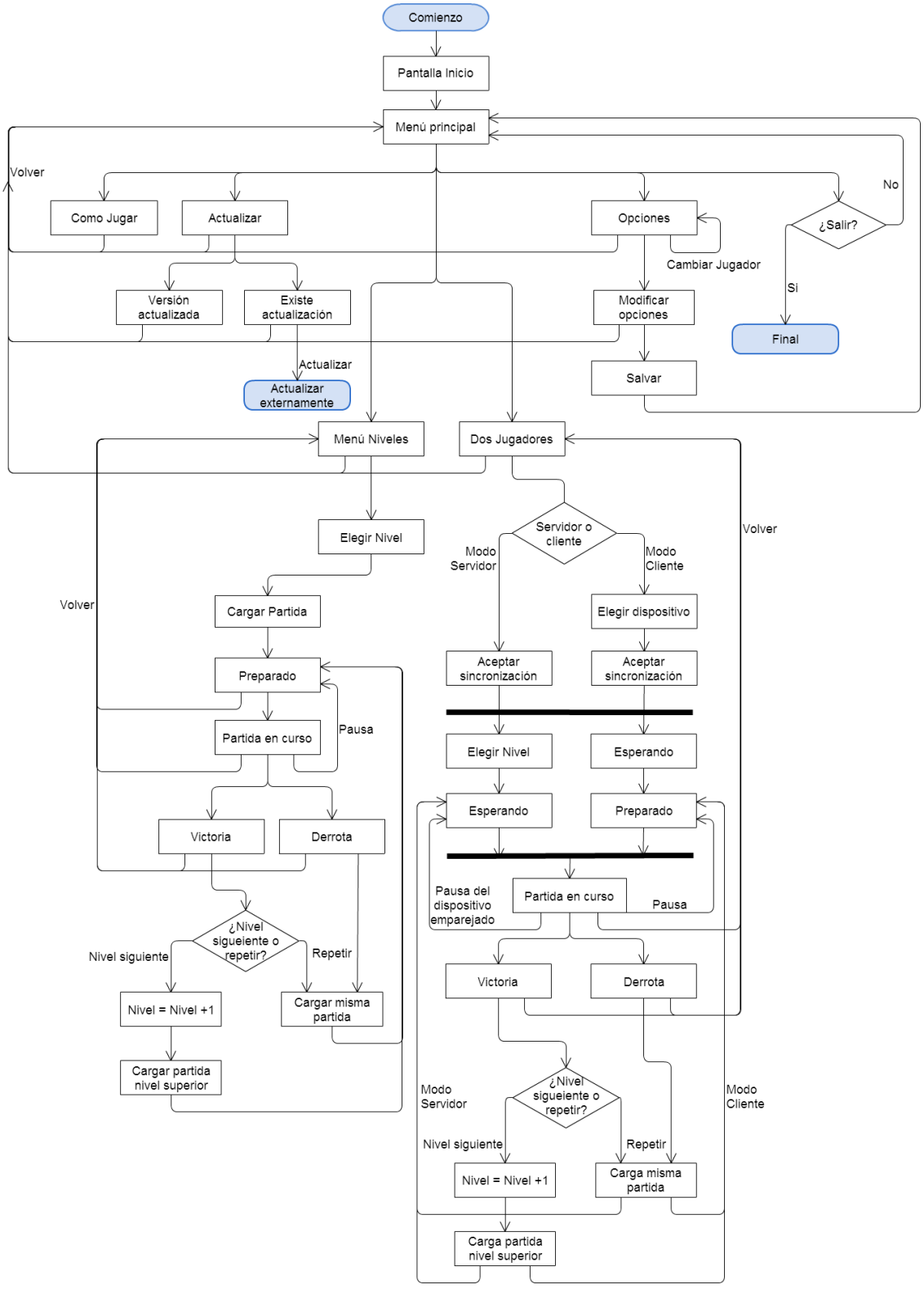

Figura 4-5. Diagrama de estados de la aplicación.

Si se desea jugar una partida de un jugador, se hace a través del menú niveles. Una vez seleccionado el nivel se carga la partida y pasa a la pantalla preparado. El juego transcurrirá hasta conseguir la victoria o la derrota, a no ser

que se pause. Si se ha conseguido la victoria, se dará la posibilidad de cargar una nueva partida con el mismo nivel o con el superior. En caso de derrota, únicamente se da cargar una nueva partida con el mismo nivel. En cualquier momento, si se usa el botón *Volver*, regresará al menú niveles.

En caso de que se desee jugar una partida multijugador, ha de hacerse a través del menú diseñado para dos jugadores. Una vez seleccionado el nivel, se debe elegir entre ser servidor o cliente. Si se decide ser cliente, tendrá que elegir el dispositivo al que enlazarse. Una vez aceptada la sincronización se cargará la partida de manera similar a la partida para un jugador, salvo que un dispositivo accede a la pantalla esperando cuando el otro esté en la pantalla preparado.

Para salir de la aplicación, se debe usar la opción de salir en el menú principal. La aplicación realizará una pregunta de confirmación y en caso de respuesta afirmativa, procederá a ello.

4.3. Utilidades generales para la aplicación

Bajo este epígrafe se incluyen un elenco de clases e interfaces de utilidad para todo el código, como las clases heredadas por las clases implementadas, las herramientas que dan formato gráfico y sonoro a la aplicación o los mecanismos de lectura/escritura de datos.

4.3.1. Clases heredadas

A continuación se comentan los aspectos básicos de las clases e interfaces padre de algunas de las clases que se han implementado.

Activity

Esta clase es la base de cualquier aplicación Android con interfaz de usuario, es decir, si tiene interfaz de usuario tendrá al menos una clase Activity, o más bien una clase que hereda de Activity.

Cada Activity está conformadas por dos partes: la parte lógica y la parte gráfica.

La parte lógica es una archivo .java que es la clase que se crea para poder manipular, interactuar y colocar el código de esa actividad.

La parte gráfica son los *layouts* que se comentaron en la sección 4.2.2. Tiene todos los elementos que estamos viendo de una pantalla declarados con etiquetas parecidas a las del HTML, es decir, que el diseño de una aplicación en Android se hace similar a una página web; XML es un primo de HTML.

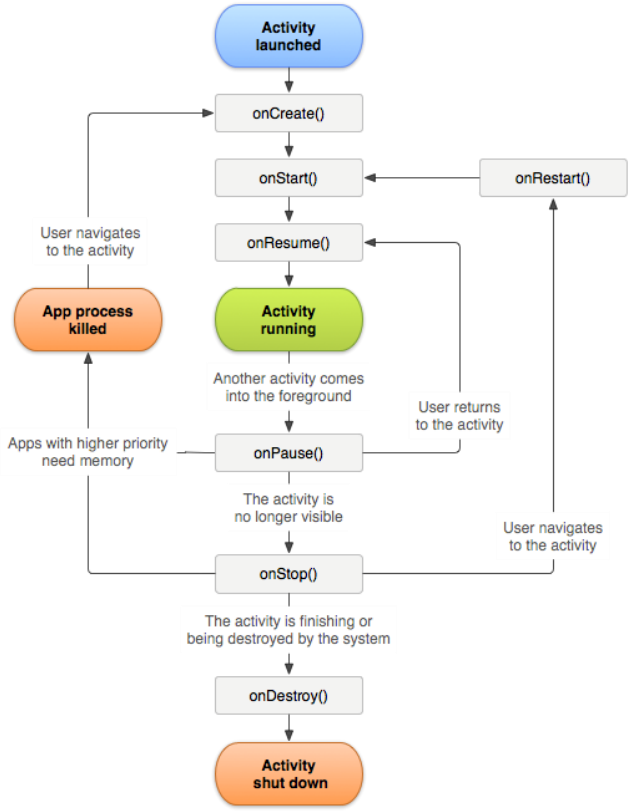

Figura 4-6. Ciclo de vida de la clase Activity.

Una Activity debe implementar por lo menos un método, el método `onCreate`, que es en donde se crea la actividad o podemos decir que es donde se le da vida. En una parte de él se enlaza la parte lógica con la parte gráfica.

Sobre su ciclo de vida Activity, se debe saber 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 estuviera ejecutándose antes siempre estará por debajo en la pila y no volverá a primer plano mientras la nueva Activity exista. En la Figura 4-6 se puede ver cuál es el ciclo de vida de este tipo de clases.

Para comenzar una Activity desde otras clases, usaremos los métodos `startActivity`, el cual lanza una nueva Activity sin esperar resultado al

finalizarse; y `startActivityForResult` que lanza una Activity de la que esperas un resultado una vez esta finalice. Cuando esta segunda Activity termina, el método `onActivityResult` dentro de la primera Activity será llamado con los datos proporcionados.

Game

La interfaz Game se incluye dentro del framework de Mario Zechner, comentado en el capítulo 2.2.8 y lo usamos con asiduidad en el bloque del motor del juego. Su implementación busca:

- 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 ejecución principal.
- Guardar un registro de la pantalla principal y comunicarle que se actualice y presente su contenido cada vez que se repita el bucle principal.
- Transferir cualquier evento de la ventana que se envíe 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

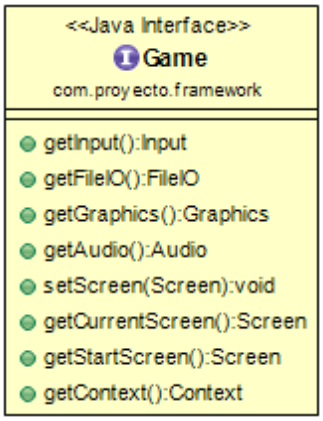

Figura 4-7. Interfaz Game

La implementación Game se encargará 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.

El método `setScreen` permite definir la pantalla Screen (que se define en este documento un poco más adelante) de Game. Este método y los que devuelven ejemplos de los módulos de bajo nivel sólo se implementarán una vez, junto con la creación del hilo de ejecución interno, el gestor de la ventana y la lógica que se empleará con el bucle principal, la cual pedirá constantemente a la pantalla que se actualice y muestre su contenido en el dispositivo. El método `getCurrentScreen` muestra la pantalla que está activa.

Más adelante se describe la clase abstracta AndroidGame para implementar la interfaz Game. Esta clase implementará todos los métodos a excepción de `StartScreen`. Con este método se obtiene una instancia de la primera pantalla del juego.

Adicionalmente, se ha modificado el interfaz original dado por el framework, añadiendo el método `getContext` para poder interactuar mejor en métodos comunes de los dos bloques.

Screen

La clase abstracta Screen también se incluye dentro del framework de Mario Zechner. Se creó abstracta en lugar de una interfaz para que se pueda implementar algo de contabilidad y así escribir menos código cuando se fuese a implementar.

Es capaz de acceder a módulos de bajo nivel de Game para reproducir sonido, dibujar el contenido de la pantalla y leer y escribir datos en los archivos.

Cada vez que sea necesario, configura una nueva pantalla Screen. Esto permite 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` se encargarán de actualizar el estado de la pantalla y presentarlo en el dispositivo. Se llamarán una única vez en cada una de las repeticiones del bloque principal.

A los métodos `pause` y `resume` se les llamará cuando se detenga y retome el juego. Esta acción llevará a cabo una instancia que será aplicada a la pantalla el Screen que este activo.

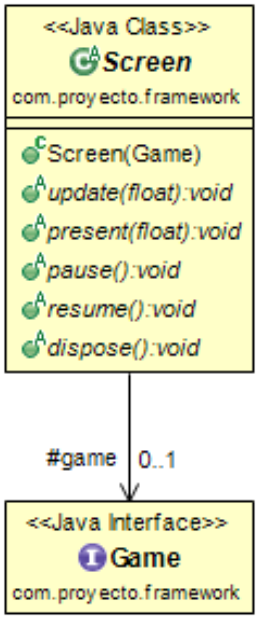

Figura 4-8. Clase Screen.

El método `dispose` eliminará el objeto Screen actual y liberará todos los recursos del sistema que tuviese reservados. Podemos observar con detenimiento la clase en la Figura 4-8.

AndroidGame

También pertenece al framework de Mario Zechner. Es una clase heredera de Activity y que implementa la interfaz Game, creada con el objeto de sólo ser llamada una vez y que de ésta se puedan obtener todos los elementos que se necesiten.

Es una clase algo compleja, como puede verse en la Figura 4-9. Lo que destaca para este proyecto de esta clase, lo encontramos en el método `onCreate`. Hace que se trabaje a modo pantalla completa con la orientación que requiramos y con un tamaño originalmente de 320x480, que en nuestra aplicación cambiamos a 424x576 y que en cada dispositivo es adaptado a su resolución. 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 buenas ideas implementadas en esta clase se encuentra en el método `setScreen`, heredada del interfaz Game. Comunica 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 sea igual a cero, a esta nueva instancia se le pedirá que retome la actividad y que se actualice, asignando el miembro Screen a la nueva pantalla Screen.

Métodos que se usarán muy a menudo como `getGraphics` no precisa mucha explicación. Devuelve la instancia al elemento que ha generado la llamada. Este elemento siempre será una de las implementaciones Screen del juego.

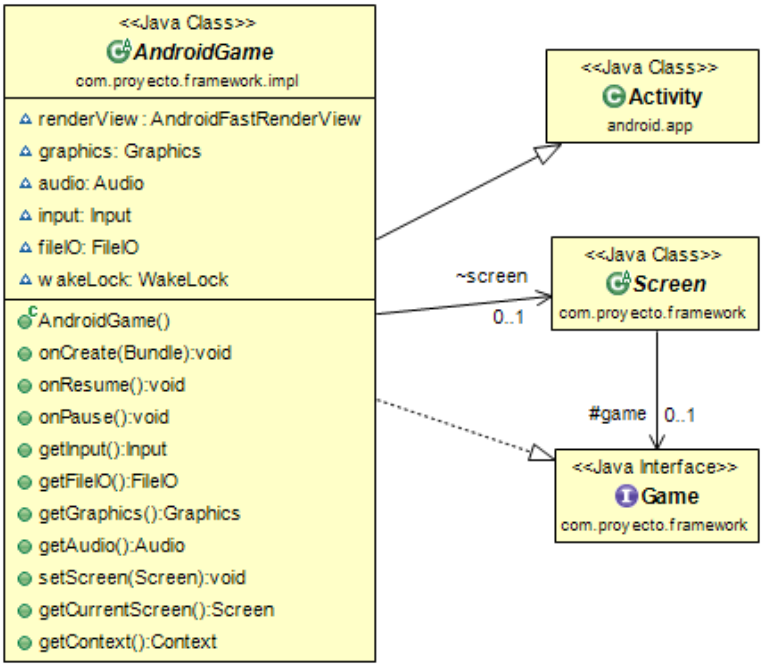

Figura 4-9. Clase AndroidGame

BaseSplashActivity

La clase BaseSplashActivity pertenece a la librería AndEngine, que se mencionó en el capítulo 2.2.9. Esta clase es heredera de otra clase de la librería, que a su vez es heredera de otra clase más de la librería la cual es heredera de la clase Activity, como podemos comprobar en la Figura 4-10. Esto quiere decir que es una clase que es capaz de actuar por sí misma.

Esta clase se usa para crear una *Splash screen*. Las *Splash screens* en Android son esas imágenes que aparecen al iniciar una aplicación ocupando

toda la pantalla durante un corto espacio de tiempo mostrando habitualmente información de marca.

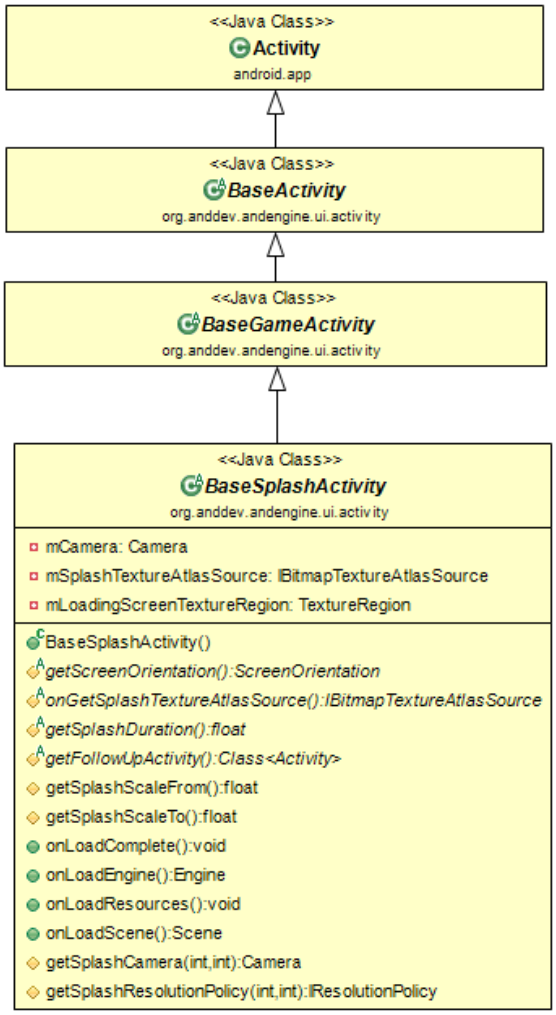

Figura 4-10. Clase BaseSplashActivity.

En su método `getScreenOrientation`, se indica qué tipo de orientación se quiere tener. En `onGetSplashTextureAtlasSource` la imagen que se usará para el efecto, en `getSplashDuration` el tiempo de duración y en `getSplashScaleFrom` la escala inicial de la imagen.

Además de estos métodos, se destaca el método `getFollowUpActivity`, que hará arrancar la siguiente clase una vez terminado el tiempo que se definió.

4.3.2. Interfaz gráfica en el bloque de menús

Widget

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

1) TextView

Muestra un texto al usuario y opcionalmente permite a estos editarlos. La clase TextView es un completo editor de texto, pero normalmente (la clase básica) está configurada para no permitirles editar.

2) EditText

EditText es un pequeño espacio sobre TextView, el cual se configura él mismo como editable.

Su método `setText` establece el texto que la clase debe mostrar y también si se guarda ese texto o no dentro de un buffer o si es o no editable por el usuario.

3) Toast

Un *toast* es una ventana emergente que contiene 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. El usuario estaría, probablemente, centrado en otra cosa. La idea es que sea lo menos intrusivo posible, mientras se enseña al usuario la información que se quiera mostrar.

El principal método de la clase que vamos a usar es `makeText`, que hace el toast estándar con el mensaje a mostrar.

4) Button

Representa un botón “pulsable”. Estos botones pueden ser presionados o pulsados por el usuario para que se realice una acción.

5) ToggleButton

Muestra un botón con el estado seleccionado/deseleccionado a través de un indicador “luminoso”. Por defecto, es acompañado con el texto “ON” o “OFF”.

6) SeekBar

SeekBar es una clase heredara de ProgressBar. ProgressBar crea una barra de progreso. La clase SeekBar, además de crear esta barra, añade una marca a la barra de progreso que puede ser movida. El usuario puede tocar esta marca y moverla a la izquierda o a la derecha para seleccionar el nivel dentro de esa barra de progreso.

`SeekBar.OnSeekBarChangeListener` es un interfaz que notifica cuándo la barra de progreso es modificada. Estos cambios pueden ser hechos por el usuario táctilmente o usando las flechas del teclado o igualmente los cambios pueden ser hechos de manera programada.

7) AdapterView

Como ya se ha mencionado, algunos grupos de vistas tienen UI. Estos objetos tipicamente son subclases de AdapterView. Algunos ejemplos son GridView y ListView. Estos objetos hacen dos tareas en común: llenar el esquema de datos y manipular las selecciones del usuario.

Llenar el esquema con datos es típicamente hecho a través de la asociación de la clase con un interfaz Adapter que toma los datos de alguna parte: una lista que el código suministra o el resultado de una consulta a una base de datos del dispositivo. El ListAdapter es un interfaz heredado de Adapter que hace de puente entre ListView y los datos que llenan la lista.

Manipular las selecciones del usuario es hecho al asignar al interfaz `AdapterView.OnItemClickListener` de la clase un objeto escuchador de eventos. Con ello se captura los cambios que el usuario hace en su selección.

8) GridView

Muestra figuras en una rejilla de dos dimensiones con la posibilidad de hacer *scroll* en ella. Las figuras de esa rejilla vienen del ListAdapter asociado con ella. ListAdapter se describe dentro del apartado dedicado a AdapterView.

9) ListView

Una vista que muestra elementos en una lista vertical con *scroll*. Los elementos provienen de un ListAdapter asociado a esta ventana.

10) ImageView

Muestra una imagen como un icono. La clase ImageView puede cargar imágenes de varias fuentes y se encarga de escoger la medida de la imagen en el dispositivo. Así puede usarse con cualquier layout. Proporciona algunas opciones como la escala o el tono.

11) BaseAdapter

Es la base común de las implementaciones comunes para un Adapter que puede es usado tanto en ListView (implementando un interfaz ListAdapter) como en Spinner.

12) ArrayAdapter

ArrayAdapter es un BaseAdapter al que se le puede relacionar con una gran variedad de objetos. Por defecto, esta clase espera que el recurso proporcionado haga referencia a un único TextView pero se puede utilizar un layout más complejo, para provocar un diseño mayor.

Text

El paquete Text proporciona clases que se utilizan para hacer o realizar un seguimiento de texto y su manera de abarcar la pantalla.

1) SpannableString

Esta es la clase es utilizada cuando se requiere un contenido inmutable pero al que los objetos puedan ligarse y desligarse.

2) method.LinkMovementMethod

Un método para hacer que, si hay dentro de un texto un link cliqueable, continúe de esa manera en un buffer de texto o en los métodos con *scroll*.

3) util.Linkify

Linkify coge un trozo del texto para convertirlo en un link cliqueable. Usado en este proyecto para direcciones webs.

Graphics

El paquete Graphics ubicado dentro de la librería Android proporciona herramientas gráficas de bajo nivel que permiten manejar directamente lo que se quiere dibujar en la pantalla.

1) Typeface

Usado en este proyecto para seleccionar la fuente tipográfica de algunos textos que aparecerán en los menús.

El método `createFromAsset` crea una nuevo tipo de letra a partir de una fuente tipográfica que se ha de especificar.

App

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

1) AlertDialog

Muestra un texto emergente con uno, dos o tres botones. Cada botón puede relacionarse con una acción. Se usará 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 acción que indiquemos.

4.3.3. Interfaz gráfica en el bloque del motor del juego

Graphics

El *framework* de Mario Zechner usa dos interfaces para emplear con los módulos, aunque a nosotros sólo utilizaremos el interfaz Graphics. Las imágenes

que queramos usar se cargarán en un buffer virtual del componente de la UI donde se dibuja la imagen. Con esto, el interfaz es capaz de:

- Cargar imágenes almacenadas en el disco y guardarlas en la memoria para dibujarlas más tarde.
- Dibujar las imágenes que hayamos cargados al inicio en el buffer, pudiendo dibujar la imagen completa o bien parte de ella en una posición específica.

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

El método `drawPixmap` dibuja partes de una imagen cargada en las coordenadas especificadas. Además se puede recortar la imagen ya que se ha de especificar qué lugar de la imagen se cuenta como el inicio de ésta y cuál sería el tamaño a partir de dicho inicio.

También se ha de destacar la enumeración en una lista llamada PixmapFormat de los distintos formatos que se podrán usar: ARGB8888, ARGB4444 o RGB565

4.3.4. Lectura / escritura de datos

IO

1) BufferedReader

Esta clase es usada para hacer clases Reader de bajo nivel de una manera eficiente y fácil de usar. Los BufferedReaders leen grandes cantidades de un archivo a la vez, y mantienen esta información en el buffer. Cuando preguntamos por el siguiente carácter o la siguiente línea de información, la respuesta es recuperada del buffer, lo que minimiza el número de veces que se tiene que leer desde el archivo, lo cual sería una operación más lenta. La clase BufferedReader provee métodos como `readLine`, el cual nos permite leer la siguiente línea de caracteres de un archivo.

2) BufferedWriter

Esta clase es usada 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 provee un método llamado `newLine` el cual crea separadores de línea específicos de la plataforma de manera automática.

3) InputStreamReader

Una clase para cambiar un *stream* de bytes a un *stream* de caracteres. Los datos leídos de la fuente son convertidos 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 convertidos a caracteres conforme se van necesitando. Este buffer tiene un tamaño de 8KB.

4) OutputStreamWriter

Una clase para cambiar un *stream* de caracteres a un *stream* de bytes. Los datos escritos en el *stream* de salida son convertidos a bytes por un convertidor que puede ser provisto o se puede usar el convertidor por defecto. El convertidor por defecto es obtenido por el "convertidor de archivos" proporcionado por el sistema. OutputStreamWriter contiene un buffer de bytes para ser escritos en el *stream* de salida y convertir los caracteres conforme se van necesitando. El tamaño del buffer es de 8KB.

5) OutputStream

Clase que representa un *stream* de bytes de salida

6) PrintWriter

PrintWriter es un objeto que nos permite imprimir representaciones formateadas de una salida de *stream* de texto.

El método `println` presenta la cadena que se carga y hace que lo siguiente a escribir se haga en una nueva línea. El método `flush` se asegura que todos los datos pendientes son enviados al objetivo.

content

1) Context

Proporciona un interfaz con toda la información global sobre el entorno de una aplicación. Esta 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, así como las convocatorias de las operaciones a nivel de aplicación, tales como el lanzamiento de las actividades, la difusión y recepción de los *Intents*, etc.

Integer

1) parseInt

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

String

1) trim

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

2) split

Divide una cadena usando la expresión que se le especifique. En este proyecto será el símbolo coma el con el que se marque el final de una sección de la cadena.

Boolean

1) parseBoolean

Analiza la línea especificada como un valor booleano.

4.3.5. Gestor de sonidos

Media

1) MediaPlayer

MediaPlayer es una clase que puede ser usada para el control de las acciones visuales o sonoras de archivos de audio y video y de *streams*.

Audio

Para reproducir los varios sonidos de manera simultánea, lo mejor es guardarlos en memoria para poder reproducirlos sobre la marcha. Con esta interfaz se encuentra la manera de cargarlos y de controlar la reproducción de los archivos de audio precargados y la reproducción del *stream* del sonido que no se quiere cargar.

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

1) newSound

Toma el nombre de archivo como argumento y lo relaciona con la interfaz Sound. Tras ello, se puede ejecutar en él el método `play` indicando el volumen al que quiere ejecutar el sonido.

2) newMusic

Toma el nombre de archivo como argumento y lo relaciona con la interfaz Music. En esta ocasión, sus métodos permiten la reproducción del *stream* de música, pausarlo o detenerlo completamente y definir un bucle de reproducción. En él, cuando se llega al final del archivo de audio, se volverá a reproducir desde el principio. Además se puede definir el volumen.

Adicionalmente, se ha creado en el interfaz Music el método `restart` que permite volver al principio del *stream* del archivo de audio. Esto servirá para reiniciar la música cuando se reinicie un nivel.

4.3.6. Interacción con el usuario

View

1) View

Esta clase representa el bloque básico de construcción para los componentes de la interfaz de usuario. Una clase View ocupa un área rectangular en la pantalla y es responsable tanto del dibujado en ella como de registrar el evento de ser pulsado. Esta es la clase base para los widgets, de los que hemos hablado antes, que son usados para crear componentes UI interactivos, como los botones.

2) OnclickListener

Es la interfaz definida para que avise siendo invocada cuando una View sea cliqueada.

3) KeyEvent

Es un objeto usado para reportar eventos que producen el presionar una tecla o botón.

4) MotionEvent

Es un objeto usado para reportar eventos de movimientos (de ratón, de dedo, de boli…). Estos eventos de movimientos pueden contener tanto movimientos absolutos como relativos y de otros tipos, dependiendo del tipo de dispositivo.

Content

1) DialogInterface

Interfaz usado para permitir la creación de un dialogo que ejecute algún código cuando un ítem del dialogo sea cliqueado.

4.3.7. Interacción entre clases

IO

1) IOException

Señala un error general de entrada/salida. Los detalles del error pueden ser especificados si se llama al constructor de éste.

OS

1) Bundle

La clase Bundle sirve para contener tipos primitivos y objetos de otras clases. Con esta clase se pueden pasar datos entre distintas Activity.

2) Handler

Un Handler implementa un mecanismo de paso de mensajes entre hilos de forma que permite enviar mensajes desde un hilo a otro.

Util

3) List

Una List es una lista de elementos que mantiene el orden. Cada elemento de la lista tiene un índice. A cada elemento se puede acceder por ese índice, siendo el primero de ellos cero. Normalmente, las List permiten elementos duplicados, no como Sets, cuyos elementos han de ser únicos.

4) ArrayList

ArrayList es una implementación de List, creada como una matriz. Todas las operaciones opcionales de List, incluyendo la adición, eliminación y sustitución de elementos son compatibles.

Content

1) Intent

Un Intent es una descripción abstracta de una operación a realizar. En nuestra aplicación lo usaremos junto a los métodos ya descritos `startActivity` o `startActivityForResult` para comenzar una nueva clase Activity.

2) Pm.ActivityInfo

Proporciona la información que puede saberse de una Activity de una aplicación. Esta información es recolectada del AndroidManifest.xml.

Lang

1) Runnable

Representa un comando que puede ser ejecutado. Se usa normalmente para ejecutar un código en un hilo diferente.

4.3.8. Conexión a internet

Net

1) Uri

Uri es una referencia a una cadena de texto similar a cualquiera de las direcciones web que utilizamos en nuestro navegador.

El método `parse` analiza la cadena de texto en lenguaje URI y lo traduce a una clase Uri.

4.3.9. Bluetooth

util

1) UUID

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

bluetooth

1) BluetoothAdapter

Representa el adaptador Bluetooth local, es decir, el dispositivo físico. Ésta clase es el punto de entrada para toda la infraestructura Bluetooth de nuestra aplicación. Con ella podemos descubrir otros dispositivos Bluetooth activos cerca de nosotros, acceder a la lista de dispositivos emparejados, instanciar un objeto de la clase BluetoothDevice usando una MAC conocida o crear sockets para comunicar con otros dispositivos.

El método estático `getDefaultAdapter` devuelve una referencia del adaptador local Bluetooth del dispositivo que ejecuta la aplicación.

El método `listenUsingRfcommWithServiceRecord` crea un socket de servidor con un nombre y un Identificador Único Universal (UUID - Universally Unique Identifier) determinados. El nombre y el UUID serán registrados en el SDP (Service Discovery Protocol). El SDP proporciona a los clientes la información necesaria para descubrir un servicio.

2) BluetoothDevice

Representa a un dispositivo Bluetooth remoto. Se utiliza para solicitar una conexión con dicho dispositivo a través de un BluetoothSocket o también solicitar información acerca del dispositivo, como puede ser el nombre, la MAC, clase, etcétera.

El método `createRfcommSocketToServiceRecord` es el que crea un socket de cliente listo para ser conectado al socket de servidor con el UUID que se proporcione.

3) BluetoothServerSocket

Para poder establecer una conexión, uno de los dispositivos debe asumir el rol de Servidor (conexión pasiva) y otro el rol de Cliente (conexión activa). BluetoothServerSocket representa a un *socket* escuchando en el lado de la aplicación servidora.

Cuando el cliente solicita una conexión, un objeto de la clase BluetoothServerSocket se encarga de retornar un objeto de la clase BluetoothSocket que representará al servidor, siempre y cuando la conexión haya sido aceptada.

4) BluetoothSocket

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

Las llamadas a sus métodos `getInputStream` y `getOutputStream` devuelven los flujos de información que se pueden utilizar para la transmisión de datos.

Net

1) UnknowHostException

Es una clase que se ejecuta cuando no puede realizarse la tarea dada con la dirección del host proporcionado. Con ello se puede crear alguna rutina para que, en caso de error, la aplicación realice una actividad distinta a si hubiese tenido éxito.

4.3.10. DeviceListActivity.java

Activity que aparece como un diálogo. Se encarga de obtener y mostrar los dispositivos a los que la aplicación puede conectarse y conseguir la dirección de aquel dispositivo al que el usuario desee conectarse. Además discrimina entre aquellos dispositivos ya enlazados (que muestra inicialmente) con los nuevos

dispositivos descubiertos, para los cuales hay que realizar una búsqueda. Muestra una lista de los aparatos vinculados y los descubiertos.

4.3.11. VersionChequer.java

Es una clase auxiliar que ayuda a la clase Actualizar a realizar su labor. Se usarán cada uno de sus métodos para conseguir información tanto de la versión instalada como la versión más actualizada subida en la red y para comparar ambas versiones.

4.4. Permisos

Las aplicaciones Android recogen los permisos se le otorgan a la aplicación y algunas características generales en AndroidManifest.xml. En el proceso de instalación de la aplicación, se informa al usuario dichos permisos que se concederán. El usuario puede no estar de acuerdo en otorgárselos cancelando la instalación.

Los permisos que se usan en esta aplicación son:

- WAKE_LOCK: Permite el acceso a los bloqueos de energía para mantener el procesador durmiendo o mantener la pantalla apagada. En la aplicación se usa para que el dispositivo permanezca siempre activo durante una partida.
- BLUETOOTH: Permite la conexión entre aplicaciones y dispositivos pareados por Bluetooth.
- BLUETOOTH_ADMIN: Permite a una aplicación buscar y parear dispositivos Bluetooth.
- INTERNET: Permite a una aplicación abrir sockets. En la aplicación se necesita para poder acceder a internet y así obtener datos de la versión más actualizada.

4.5. Bluetooth

Como ya se comentó en el segundo capítulo, Bluetooth es una tecnología de radio de corto alcance (aproximadamente unos 10 metros en un ambiente libre de obstáculos), que permite conectividad inalámbrica entre dispositivos

remotos y que opera en la banda libre de radio ISM (banda industrial científico-médica) a 2,4 GHz. También se señaló que Android permite el desarrollo de aplicaciones que utilicen Bluetooth usando su propia API en el paquete `Android.bluetooth`.

El primer aspecto importante a la hora de diseñar la funcionalidad multijugador es fijar el protocolo de comunicación sobre el que se va a sustentar la estructura. El enfoque seguido será de una estructura cliente-servidor sobre Bluetooth limitando el número de clientes a uno solo, es decir, el modo multijugador sólo tendría dos jugadores y se usará el protocolo de RFCOMM (Radio Frequency Communication) que establece las sesiones de comunicación utilizando las direcciones Bluetooth de los dos puntos terminales. Estas sesiones están ligadas a cada par único de direcciones Bluetooth de los dispositivos que se conectan.

4.5.1. Pasos previos al emparejamiento

Se crea una variable general para el tipo BluetoothAdapter y, una vez seleccionado el modo dos jugadores, la asociamos con el adaptador de Bluetooth mediante el método `BluetoothAdapter.getDefaultAdapter`.

Se realiza la comprobación de si el dispositivo tiene Bluetooth comprobando que la variable a la que hemos asociado el adaptador no ha resultado nula. En caso de que lo sea, indicamos con un *toast* que el dispositivo no soporta Bluetooth.

A continuación se comprueba que el Bluetooth esté activado con la respuesta booleana de la característica `isEnabled` de la variable. En caso de que no esté activado, se lanzará un `Intent` para dar la posibilidad de activarlo a través de la función `ACTION_REQUEST_ENABLE`. Además se hará el dispositivo visible en su entorno para que pueda ser detectado por otros dispositivos y que así le envíen una solicitud de vinculación.

Realizado esto, se seleccionará momentáneamente el nivel cero como el nivel a jugar y se creará un diálogo mediante `AlertDialog` preguntando si se desea jugar en modo servidor o en modo cliente.

4.5.2. Funcionamiento

Emparejar dispositivos

Antes de que los datos puedan ser transmitidos entre dos dispositivos a través de Bluetooth, los dispositivos deben estar emparejados. El emparejamiento de dos dispositivos, una vez tengan el Bluetooth activado y visible, requiere de los siguientes pasos:

- Uno de los dispositivos, en el caso de nuestra aplicación el dispositivo Cliente, debe explorar su entorno para ver si hay otros dispositivos Bluetooth. Por esto, encontrará todos los dispositivos detectables, entre ellos el dispositivo Servidor.
- El dispositivo Cliente debe enviar una solicitud de vinculación con el dispositivo Servidor. La solicitud entrante se muestra en ambos dispositivos junto a una clave de enlace generada por el sistema que pueden aceptar o rechazar. Si ambos dispositivos han aceptado la solicitud, los dispositivos 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 significa que un programa puede leer datos entrantes de un *socket* y escribir los datos de salida del mismo sin necesidad de saber el funcionamiento interno de la red. Los datos se transmiten siempre a través de un par de *sockets*: Los datos escritos en el socket del cliente se pueden leer desde el *socket* del servidor y viceversa. Por lo tanto, los dos *sockets* se pueden considerar como los extremos de un canal de datos entre el cliente y el servidor.

La transmisión de datos a través de un par de *sockets* Bluetooth en Android requiere los siguientes pasos:

- Si el dispositivo actúa como servidor, la aplicación debe crear un "*socket* servidor" con un UUID y esperar una solicitud de conexión de un cliente.

- Si el dispositivo actúa como cliente, la aplicación debe crear un "*socket* de cliente" y enviar una solicitud de conexión al *socket* de servidor, identificándole por su UUID.
- El servidor debe aceptar la solicitud. Después de esta aceptación, el cliente y el servidor están conectados. Ahora comparten un canal RFCOMM para la siguiente transmisión de datos. El servidor crea un nuevo *socket* para esta conexión.
- El cliente y el servidor deben obtener los flujos de entrada y salida de sus respectivos *sockets*. Todos los datos que el cliente escriba en su flujo de salida aparecerán en el flujo de entrada del servidor y viceversa. Por lo tanto, el par de *sockets* se puede utilizar para la transferencia de datos de manera bidireccional; es decir, la transferencia de datos se puede realizar en ambas direcciones.

4.5.3. Modo servidor

Si es seleccionado el modo servidor, la rutina del diálogo que nos preguntaba el modo, hará crear un hilo llamado HiloServidor que escuchará las conexiones entrantes y pasará el *socket* a la clase Multijugador donde la conexión *socket* será establecida.

Este hilo comienza creando un objeto temporal que después será asignado al *socket* servidor. El objeto temporal utilizará del adaptador el método `listenUsingRfcommWithServiceRecord` con la UUID y el nombre de la aplicación que también es usada en el código cliente. Una vez creado el *socket* en el objeto temporal, lo pasamos al *socket* servidor. Tras esto el hilo se dará como creado y volverá a la rutina del diálogo.

La rutina será la que hará arrancar el hilo creado. El cual se mantendrá a la escucha hasta que ocurra una excepción y el *socket* es devuelto. Cuando la conexión *socket* es devuelta, se establecerá dicho *socket* en la clase Multijugador e informará a una variable booleana dentro de esa clase, que es el servidor. Tras ello, pasará a la clase MenuDosJugadores para que se seleccione el nivel.

La clase MenuDosJugadores es hija de MenuNiveles. Sirve para seleccionar el nivel a jugar y carga los recursos del juego hasta terminar en la clase Multijugador en la cual ya se ha trasladado el socket correspondiente.

4.5.4. Modo cliente

Se activa cuando se pulsa en la opción modo cliente en el dialogo creado al pulsar dos jugadores. Esta opción lanza la actividad definida en la clase DeviceListActivity (explicado anteriormente). Para lanzarla utiliza el método `startActivityForResult` que espera 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. Si los resultados dados indican que se terminó bien la Activity y que el paquete de datos proporcionados no está vacío, extraerá de esos datos la dirección MAC del dispositivo remoto y usará esa dirección para crear una variable de la clase BluetoothDevice.

A continuación creará un hilo llamado HiloCliente que intente conectarse con el otro aparato y pasar el *socket* a la clase Multijugador cuando la conexión *socket* esté establecida de manera similar al modo servidor.

En este caso, para conectar el *socket* de Bluetooth con el aparato cuya dirección fue dada, se usa el método de la variable BluetoothDevice llamado `createRfcommSocketToServiceRecord` con el UUID de la aplicación.

Una vez creado el hilo, el método `onActivityResult` arrancará dicho hilo el cual lo primero que hace es cancelar la búsqueda de nuevos dispositivos, ya que haría que la conexión se ralentizase mucho. A continuación tratará de conectar el dispositivo con el *socket*. Esto se bloqueará hasta el éxito o que lance una excepción. Si hay éxito, establecerá dicho *socket* en la clase Multijugador e informará a una variable booleana dentro de esa clase, que es el cliente. Tras ello, se arrancan la clase para cargar recursos con el fin de llegar a la clase Multijugador y así comenzar el juego.

4.5.5. Implementación de la funcionalidad Bluetooth dentro del juego

En la clase Multijugador se ha cargado previamente tanto el *socket* como el modo en el que entra el dispositivo mediante el método creado en ésta llamado `setBluetoothSocket`. Este método lo único que hace es crear en una variable estática, llamada mSocket, de la clase `BluetoothSocket` con el *socket* importado y en una variable estática booleana si se actúa como servidor o cliente.

Una de las primeras cosas que hace esta clase, la Multijugador la cual es hija de PantallaJuegoUnJugador, es arrancar el método `inicializarBT` que se ha creado en ella. Con este método se crearán dos objetos. El primero de tipo BufferedReader se le hará la correspondencia con el flujo de datos a leer del *socket*. El segundo de tipo PrintWriter con el flujo de datos a escribir en el *socket*.

Lo siguiente que hace la clase Multijugador es crear un nuevo hilo llamado EmpezarRecibirMensajes que ejecuta mientras esté funcionando la rutina necesaria para leer los mensajes recogidos del flujo de datos a leer del socket.

Para entender los pormenores de cómo se realizan el envío y la recepción de mensaje, se debe leer cómo es el modelo de los mensajes Bluetooth que se describe en la siguiente sección.

Cuando se invoca el método `RecibirMensajes`, se crea una variable *string* al que se le da el valor de la línea leida del objeto BufferedReader. A la variable se le ejecuta el método `trim` y el resultado de este método se establece como el nuevo valor de la variable.

El hilo EmpezarRecibirMensajes realiza una rutina siempre que esté funcionando que consiste en crear una matriz de *String* de siete casillas, meter en esa matriz el resultado del método `RecibirMensajes` donde cada dato se meterá en una de las casillas mediante el método `split` y trasladar esa matriz al método `RutinaMensajeRecibido` a la vez que se arranca.

En la rutina del método `RutinaMensajeRecibido` se compara lo recibido en la primera casilla de la matriz con unos caracteres, y cuando coincide con uno, se ejecuta la rutina asignada para la coincidencia con dichos caracteres.

Para enviar mensajes se usa el método `EnviarMensaje` con un *String* asociado. Este método simplemente introduce el *String* en el flujo de salida y ejecuta el método `flush` a dicho flujo. Hay que asegurarse que el String asociado es consecuente con el modelo de mensaje Bluetooth para la correcta lectura de éste.

4.5.6. Intercambio de mensajes Bluetooth.

Mientras que el envío de mensajes se puede realizar en cualquier momento de la rutina general invocando al método `EnviarMensajes`, recibirlos es consecuencia de un hilo que se crea y arranca al inicio y comprueba una y otra vez si el flujo de entrada contiene algo.

Modelo del mensaje Bluetooth

El mensaje que se envía a través del socket establecido contiene únicamente una cadena *String*. Esta cadena estará dividida mediante comas con máximo de siete secciones.

En la primera sección se introducirán los caracteres que indican la rutina que se quiere que se ejecute. En el resto de secciones se introducirán los datos para que la rutina llegue a buen puerto. Estos datos son introducidos en su equivalente *String* y así se reciben, por lo que si se trata de datos *Integer* o *Boolean*, habrá que usar el método `parseInt` para poder leer los datos como *Integer* o `parseBoolean` para poder leerlos como *Boolean.*

Mensajes de sincronización y cambios de estado

Se necesitan transmitir y recibir distintos datos o momentos para poder sincronizar el inicio, la interrupción o la salida del juego. Estos datos son incluidos dentro de las secciones cuya cabecera marcan la acción.

1) Se pausa la partida

Este mensaje se envía cuando un dispositivo cambia su estado a Pausado sin añadir más datos. Actúa sólo si el dispositivo que recibe el mensaje está en estado Funcionando dentro de la partida haciendo que la variable que permite

restaurar la partida en pausa sea negativa, es decir, no tenga el dispositivo que recibe el mensaje la oportunidad de hacer que se restaure, sino que se quede esperando a recibir otro mensaje (el mensaje “vamos”) para continuar jugado. Tras cambiar esta variable, cambia el estado de Funcionando a Pausado. La Figura 4-11 muestra el aspecto que presenta el mensaje pausa.

pausa						

Figura 4-11. Formato de los mensajes pausa

2) Se reinicia una partida tras victoria o derrota

Una vez concluida una partida tras haber conseguido la victoria o bien habiendo sido derrotado, en ambos dispositivos se da la posibilidad de reiniciar el nivel. Y si se ha conseguido la victoria, de iniciar el siguiente nivel. El primer dispositivo que pulse alguno de los botones que lleva a esas acciones enviará el mensaje reiniciar. Este mensaje contiene el nivel en el que se quiere comenzar la nueva partida, es decir, el mismo si se reinicia el nivel o el siguiente si se consiguió la victoria y se desea avanzar al siguiente nivel.

La rutina lo que realiza cuando recibe este mensaje, es cargar el número del nivel a jugar y comenzar el método `Reiniciar` dentro de la misma clase, que hará que la partida comience de nuevo. En la Figura 4-12 se puede observar el esquema de este mensaje.

reiniciar	nivel					

Figura 4-12. Formato de los mensajes reiniciar

3) Derrota

Una vez el juego pasa al estado GameOver, el dispositivo envía este mensaje para comunicar dicho cambio de estado. Este mensaje es de refuerzo ya que con los últimos ajustes implementados, es muy complicado que el motor del juego de un dispositivo de como derrotado a un dispositivo y a otro no, pero se ha decidido mantener para reforzar este cambio de estado.

Cuando se recibe el mensaje, el cual no tiene más datos, la rutina comprueba si el dispositivo que lo recibe tiene el estado Funcionando, y si lo

está, realiza los pasos necesarios para que su estado pasos para cambiar el estado a GameOver. En la Figura 4-13 se puede ver la estructura del mensaje.

gameover						

Figura 4-13. Formato de los mensajes gameover

4) Victoria

Una vez el juego pasa al estado Victoria se debe sincronizar la puntuación ya que los relojes internos pueden hacer que varíe el montante total de puntos. Para evitar esto, el servidor mandará su puntuación al cliente en la segunda sección de un mensaje cuya primera sección será victoria. El cliente, al recibir el mensaje establecerá dicha puntuación como la suya propia. En la Figura 4-14 se puede ver la estructura del mensaje.

victoria	puntuación					

Figura 4-14. Formato de los mensajes victoria

5) Carga del nivel de la partida

Como se ha explicado anteriormente, el dispositivo en modo cliente entra en el juego en el estado Esperando con el nivel cero cargado, es decir, con una estructura vacía. Permanece así hasta que el dispositivo en modo servidor selecciona el nivel a jugar, el cual comunicará mediante un mensaje Bluetooth el nivel seleccionado.

Una vez recibido el mensaje que contiene “elnivel” en la primera casilla, el dispositivo cargará dicho nivel, inicializará el mundo con el nuevo nivel a jugar e informará mediante el cambio de una variable booleana que el nivel seleccionado por el dispositivo en modo servidor está cargado. Se puede observar en la Figura 4-15 el formato de este mensaje.

elnivel	nivel					

Figura 4-15. Formato de los mensajes elnivel

6) Se inicia la partida

Si al iniciar una partida, el dispositivo está en modo servidor o si en mitad de una, el otro dispositivo invoca el estado Pausa, el dispositivo se queda en

estado Esperando sin poder salir de éste a no ser que abandone la partida. En el otro dispositivo, en cuanto se toque la pantalla, se envía un mensaje con vamos en la primera sección para dar el momento de arrancar de nuevo y así pasar al estado Funcionando en ambos dispositivos. La estructura de este mensaje se puede observar en la Figura 4-16.

vamos						

Figura 4-16. Formato de los mensajes vamos

7) Colores de la cola de bolas

Cuando el dispositivo en modo cliente está en el estado Preparado y, por consiguiente, el dispositivo en modo servidor está en el estado Esperando, la estructura del nivel en ambos dispositivos está cargada, pero los elementos que han de realizarse de forma aleatoria no son los mismos.

Por ello, cuando se pulsa la pantalla en el dispositivo en modo cliente, se envían varios mensajes al otro dispositivo antes dar comienzo a la partida. El primero de ellos es el que tiene en su primera sección “bolasig”. En las siguientes cuatro secciones tiene los colores de la cola de bolas que están esperando salir por la fila superior, en la sexta sección tiene el color de la bola que saldrá por dicha fila y en la séptima sección, el color de la bola que saldrá por la fila inferior. Esta estructura se puede observar con más detenimiento en la Figura 4-17.

bolasig	Bola siguiente	Bola siguiente2	Bola siguiente3	Bola primeriza	Color bola 0	Color bola 5

Figura 4-17. Formato de los mensajes bolasig

El dispositivo que se encuentra en modo servidor, recibe este mensaje y superpone los colores recibidos en las variables correspondientes del motor de juego.

Tras enviar este primer mensaje, el dispositivo en modo cliente envía un segundo mensaje cuya primera sección contiene “bolas55” con las siguientes cuatro secciones con los colores de la cola de bolas que saldrán por la fila inferior. La Figura 4-18 muestra la estructura de este mensaje.

El dispositivo en modo servidor recibe este mensaje y superpone los colores de la cola de bolas de la fila inferior por los dados en el mensaje.

bolas55	Bola siguiente55	Bola siguiente255	Bola siguiente355	Bola primeriza55		

Figura 4-18. Formato de los mensajes bolas55

Tras esto, el dispositivo en modo cliente comprobará si la estructura posee cruz de bolas, y si es así actuará en consecuencia. Esto se verá en una sección más adelante. Una vez enviado todos estos mensajes, enviará el mensaje vamos con el que se dará el inicio a la partida en sí, como se ha visto en una sección anterior.

Mensajes de información durante la partida

Durante el transcurso de una partida, en ambos dispositivos se realizan acciones o transcurren eventos que deben comunicarse para la correcta sincronización del juego.

Se debe tener en cuenta que se ha implementado el juego de manera que el dispositivo en modo servidor puede mover las dos filas superiores de engranajes, el dispositivo en modo cliente puede mover las dos filas inferiores y la fila central es movida por ambos.

Dentro del motor del juego, el dispositivo en modo servidor controla el pasillo superior y las bolas creadas para que salgan por este pasillo, así como la interacción de las bolas con los engranajes en las 3 filas superiores. El dispositivo en modo cliente controla el pasillo inferior, las bolas creadas para que salgan en este pasillo y la interacción de las bolas con los engranajes en las 2 filas inferiores.

La estructura vista como un elemento de la estructura, es un rectángulo de cinco filas y ocho columnas cuya disposición en la pantalla es traducida por el motor del juego con la información de su fila y columna. Las bolas se mueven a través de las coordenadas de la pantalla que hemos diseñado, es decir, entre 576 pixeles de ancho y 424 pixeles de alto.

1) Se gira un engranaje

Si en un dispositivo se pulsa sobre un engranaje con el que puede interaccionar para girarlo, enviará un mensaje al otro dispositivo con “girar” en la

primera sección, el número de fila del engranaje a girar en la segunda sección y el número de columna en la tercera sección.

El dispositivo que recibe el mensaje, invocará el método `girarengranaje` de la clase Casilla que se establecerá en el motor de juego, aportando la fila y la columna. Este método es el mismo que se invoca si el que gira el engranaje es el usuario del dispositivo, por lo que el resultado es transparente para el motor del juego. En la Figura 4-19 se puede ver la estructura del mensaje.

girar	fila	columna				

Figura 4-19. Formato de los mensajes girar

2) Se suelta una bola

Si en un dispositivo se pulsa sobre un engranaje con el que se pueda interaccionar arrastrando el dedo hacia otra casilla, si hay bola situada en esa dirección y hay un pasillo o algún otro elemento por el que la bola pueda pasar, soltará dicha bola por ese elemento en la dirección que lo aleje del engranaje original.

Las consecuencias de la acción del usuario una vez comprobada su validez transcurren en el método `bolasale` de la clase Mundo al que hay que indicar la columna y fila del elemento y la bola a extraer. Con esta premisa, el dispositivo cuyo usuario formalice esta acción envía un mensaje al otro dispositivo con el número de fila y columna en el que se situa engranaje en las secciones segunda y tercera, dejando la primera para la rutina a seguir, que será distinta dependiendo de la bola que se extraiga.

Si la elegida es la bola de la izquierda, la primera sección tendrá el código bizq como puede comprobarse en su estructura de la Figura 4-20. Si es la de la derecha, usará el código bder. Se puede ver en la Figura 4-21. Si es la de arriba, el código usado será barr, ver la Figura 4-22.Y si es el de debajo, el código baba será el que indique que la bola debe tomar dirección hacia abajo como se puede observar en la Figura 4-23. Estos cuatro mensajes, tienen una rutina independiente pero idéntica en lo referido llamar al método `bolasale` con la fila y columna recibida y diferente al indicar la bola a extraer a ese método.

bizq	fila	columna				

Figura 4-20. Formato de los mensajes bizq

bder	fila	columna				

Figura 4-21. Formato de los mensajes bder

barr	fila	columna				

Figura 4-22. Formato de los mensajes barr

baba	fila	columna				

Figura 4-23. Formato de los mensajes baba

3) Se supera una cruz de bolas

El elemento cruz de bolas consiste en un patrón de colores en los lugares de un engranaje que los usuarios deben conseguir para poder resolver el resto de engranajes sin esta restricción. Este patrón es aleatorio con la única condición de que no posea en los cuatro huecos el mismo color. Además, una vez resuelto y transcurrido un tiempo, un nuevo patrón aleatorio volverá a aparecer con las mismas características que el primero.

Antes de comenzar una partida, durante la preparación de la estructura, el dispositivo observa si la estructura posee cruz de bolas. Si es así, crea un patrón aleatorio y activa una variable booleana. El dispositivo cliente en estado Preparado, una vez se pulse la pantalla, transmitirá al dispositivo en modo servidor los colores de la cola de bolas y a continuación, el patrón de la cruz de bolas, si es que lo tiene, justo antes de cambiar la variable booleana que indica que todo está cargado y pasar al estado Funcionado.

En caso de que la cruz se haya resuelto, se crea un nuevo patrón que permanecerá a la espera. El dispositivo en modo servidor enviará un mensaje con los datos de dicho patrón para que una vez pase el tiempo establecido, entre de nuevo la cruz de bolsas con el mismo patrón en ambos dispositivos.

Este mensaje con el patrón de la cruz de bolas tiene en su primera sección el código cruz, en la segunda el color de la bola de arriba, en la tercera el color de la bola de la derecha, en el de la cuarta el de la bola de debajo y en el de la

quinta el color de la bola de la izquierda. Podemos ver la estructura de este mensaje en la Figura 4-24.

El dispositivo que recibe el mensaje comprueba si el juego ha comenzado o está aún en modo preparación. Si está preparándose, sobrescribe los colores del patrón y al no estar dibujado aún, este cambio es invisible para el usuario. Si ha comenzado, rellena el patrón de la cruz de bolas que está en espera de que transcurra el tiempo para activarse.

cruz	Color bola cruz 0	Color bola cruz 1	Color bola cruz 2	Color bola cruz 3		

Figura 4-24. Formato de los mensajes cruz

4) La bola del pasillo inferior entra en un engranaje

Si la bola que transcurre por el pasillo inferior entra en un engranaje, la cola con los colores de las bolas para salir por dicho pasillo, avanzara una posición. La bola que sale cogerá la bolasiguiente55, bolasiguiente55 cogerá el color de bolasiguiente255, bolasiguiente255 el de bolasiguiente355 y bolasiguiente355 el de bolaprimeriza55. El nuevo color de bolaprimeriza55 será elegido aleatoriamente.

Debido a esta aleatoriedad en el color del final de la cola, en el motor se activa un booleano que hará que la clase Multijugador envíe, en caso de ser el dispositivo en modo cliente, un mensaje donde la primera sección contiene bolaentro55 y las dos siguientes secciones tienen los colores de los dos últimos elementos de la cola dada. Estos serán asignado a los del dispositivo en modo servidor una vez reciba el mensaje. El primero será el color nuevo que fue escogido de forma aleatoria, y el segundo es únicamente por precaución, ya que deberá ser el mismo. La estructura del mensaje se puede ver en la Figura 4-25.

bolaentro55	Bola primeriza 55	Bola siguiente3 55				

Figura 4-25. Formato de los mensajes bolaentro55

5) La bola del pasillo superior entra en un engranaje

De manera análoga al apartado anterior pero con el pasillo superior, así como siendo el dispositivo en modo servidor el que envía el mensaje que tiene en la primera sección, en este caso, bolaentro00 y en la segunda, el color de

bolaprimeriza. En este caso no necesita refuerzo. La estructura del mensaje se observa en la Figura 4-26.

bolaentro00	Bola primeriza					

Figura 4-26. Formato de los mensajes bolaentro00

6) Una bola en movimiento rebota tras chocar con un engranaje ocupado

Para una mejor sincronización, el motor de juego hará sólo rebotar o encajar bolas en determinados engranajes en función del por el modo en el que esté el dispositivo. De esa manera se aprovechan esas situaciones para sincronizar la posición de dicha bola. El motor del juego meterá en una lista todas estas acciones, indicando determinadas características de ellas. La clase Multijugador irá sacando una a una las acciones y enviando mensajes informando de ellas con las características presentadas.

Para el caso de la bola rebotando ante un engranaje lleno, se envía un mensaje cuya primera sección es Mov, la segunda, un booleano indicando que es un rebote, la tercera, el número de bola que rebota, la cuarta la dirección que tenía antes de rebotar, la quinta la coordenada x de la bola en el momento de rebotar y la sexta la coordenada y. Esta estructura se puede apreciar en la Figura 4-27.

Con estos datos, el dispositivo que recibe el mensaje realiza tres acciones: cambia la dirección de la bola a su contraria, ajusta la coordenada x de la bola a la que le proporciona el mensaje y ajusta la coordenada y de la bola a la que le ha proporcionado el mensaje.

Mov	Rebota	Numero de bola	Dirección	Coordenada x	Coordenada y	

Figura 4-27. Formato de los mensajes Mov buscando rebotar

7) Una bola en movimiento entra en un engranaje libre

Si lo que hace la bola es encajar en un engranaje, realizará acciones similares a las del rebote, aunque se introducirá en la lista distintas características. Estas características serán enviadas a través de un mensaje al otro dispositivo.

En el mensaje, la primera sección contendrá Mov y la segunda tendrá el booleano de rebote como falso, para así seleccionar la rutina en la que la bola entra en el engranaje. La tercera sección contiene el número de bola, la cuarta la dirección en la que entra, la quinta la fila del engranaje y la sexta la columna del engranaje. Se puede ver estas opciones en la Figura 4-28.

Al recibir el mensaje, el dispositivo ejecuta una rutina para incluir la bola en su posición del engranaje, comprobar si ese engranaje se ha completado y cambiar el estado de la bola como no activa.

Mov	No rebota	Numero de bola	Dirección	Fila	Columna	

Figura 4-28. Formato de los mensajes Mov buscando entrar en engranaje.

4.6. Menús y sus elementos

Los menús del juego se han realizado usando los layouts de cada clase para que manejasen las posiciones, creando una lista de valores en string.xml que indica los textos se deben poner, creando un estilo en styles.xml y usando la clase para las acciones, relaciones y otros detalles.

Se ha utilizado este formato para hacer el menú principal, el menú de opciones, el cómo jugar, el menú ajustes, el menú niveles y el menú niveles de dos jugadores. Estos dos últimos tendrá una explicación más detallada más adelante y la actuación en el menú ajustes será desarrollado en el apartado 4.8.

4.6.1. Layouts y archivos xml de ayuda

El layout de cada clase marca la disposición de cada texto, botón o imagen que se muestre. Cada objeto debe poseer una identificación única. Si necesita algún elemento externo, como puede ser el fondo de una imagen, debe indicarse la ruta de dicho objeto. Para cada objeto se puede seleccionar la altura, anchura, orientación, márgenes y posición dentro de unos límites. También puede asociarse estas características a las mismas características de un objeto padre.

En los textos además se le puede seleccionar un estilo tipográfico, el cual será concretado en la clase styles.xml. También se puede seleccionar

individualmente todas las características de este estilo como el tamaño del texto, el peso, si es cliqueable, si se quiere en itálica o negrita o el color.

A estos textos se ha de añadir el texto a mostrar. Para ello se ha utilizado una asociación individual a un elemento propio dentro de la clase string.xml donde se han definido cada texto a mostrar para cada elemento texto del layout.

Con los botones, es en el propio layout donde se selecciona el tipo. Se han seleccionado tanto botones clásicos como botones con luz simulada o barras de selección. En algunos, además, se le ha añadido un texto asociado para que sea representado dentro del botón.

4.6.2. Menú Inicio

Lo primero que se hace es ajustar la orientación a modo apaisado. Esto se hará en cada uno de los menús. Tras ello, se cargan los ajustes guardados y así poder conocer las opciones del usuario para el resto de la partida.

Como el menú es heredado de Activity, la presentación de éste se forjará en `onCreate`. Se carga la disposición del menú que hemos configurado en el layout asociado, se seleccionan las fuentes que se usarán para la escritura de los textos, se asocian los textos a poner con el elemento del layout, se indica qué fuente usar para cada elemento, se hace que todos los textos hagan algo al ser tocados, en este caso llamando al método creado más adelante `MenuTextTouchListener`, que cambia el color del texto elegido a un azul mientras se tenga el dedo pulsado sobre éste; y por último se hace que los textos que se desean entren dentro del método `onClick` cuando son cliqueados, es decir, cuando son pulsados y dejados de pulsar.

El método `onClick` indica qué hacer en caso de que uno de los distintos elementos del menú sea pulsado. En caso de que se pulse sobre el texto Un jugador, Opciones o Cómo Jugar, se avanzará a las clases menú Niveles, menú Opciones o Cómo Jugar recurriendo a `Intent` y a `startActivity` y finalizando el menú inicio tras ello. En caso de que se pulse sobre Dos jugadores, comenzará toda la rutina explicada en el capítulo 4.5 dedicado al Bluetooth. En caso de pulsar en Sobre nosotros, se ejecutará el método `HacerDialogoCreditos` y si se pulsa sobre Salir, el método `Salir`.

El método `Salir` crea una ventana emergente con dos opciones y un texto en el que se pide la confirmación para salir. Esta venta es creada con un `AlertDialog`. En caso de que se cancele, simplemente se sale del método. En caso de que se salga, se finalizará la clase MenúInicio sin haber arrancado antes otra, por lo que se saldrá del juego.

`HacerDialogoCreditos` se crea de manera parecida. En este caso, en el `AlertDialog` sólo se utiliza un botón y usamos `Linkify` para poder acceder a la página especificada en los créditos en un navegador externo.

Por último se modifica el método `onKeyDown` para que, en caso de que se pulse el botón volver, se ejecute el método Volver. Este método será creado en cada uno de los menús, aunque en el resto simplemente se arrancará el MenuInicio con `Intent` y `StartActivity` y se finalizará el menú en el que es pulsado.

4.6.3. Cómo jugar

La pantalla Cómo jugar recurre a todos los mecanismos explicados en el apartado sobre el Menú Inicio. Usamos una imagen de fondo creada con Photoshop y le añadimos un texto cliqueable para salir de esta pantalla arrancando de nuevo el menú inicio. Todo de manera similar al apartado previo.

4.6.4. Menú opciones

Aunque la creación y visualización de textos, las características de la pantalla y la interacción del botón volver sea similar que el menú inicio, el añadir botones y su correspondencia con el cambio de los ajustes de un partida hace que se deba explicar el menú opciones un poco más a fondo.

Toda la rutina y los parámetros, excepto el método `onKeyDown`, se crearán dentro del método `onCreate` de la Activity. Eso incluye lo que se hacen al pulsar distintos elementos ya que, a diferencia del menú inicio que se creaba un `onClick` general y luego se definían acciones para cada caso, aquí se irá definiendo elemento a elemento incluyendo su propio `onClick`.

Este menú contará con seis elementos. Un botón con luz para la opción de música y otro para el sonido, una barra para el volumen, una casilla para introducir texto y un botón asociado para modificar las opciones dependiendo de ese texto y un botón para salvar todas las opciones y cambios realizados.

Para que las modificaciones de las opciones sólo se guarden si se pulsa el botón salvar, lo que se hace es crear un objeto tipo Ajustes. Después se cargarán en dicho objeto todos los datos de las opciones guardadas.

La creación del botón música parte del layout asociado, donde ya se ha definido su disposición como botón de luz. Una vez se asocia un objeto con su parte en el layout, basta con llamar al método `toggle` para que tenga la posibilidad de crear el efecto y obtenga sus características como botón de luz. Se comprueba si los ajustes iniciales daban la música como encendida. En tal caso, se le asigna que el botón tenga luz y la palabra escrita ON y en caso contrario, estará desprovisto de luz y pondrá OFF sobre él. Estas características irán itinerando según se pulse sobre el botón. Lo siguiente que se crea son las acciones del botón. Al pulsar el botón se comprueba si previamente estaba encendido o apagado, si estaba encendido se apagará cambiando el booleano musicaEncendida en el objeto tipo Ajustes y se creara un mensaje emergente indicando “Música OFF” con un *toast*. En caso de que este apagado el botón, también se modificará, solo que al contrario, el booleano y el mensaje indicará “Música ON”.

El botón para la opción del sonido se hace de manera idéntica al botón para la opción de música, salvo los textos y el booleano a cambiar, que será sonidoEncendido.

La barra en la que se escoge el volumen es enteramente creada por la clase SeekBar. Se establece el elemento en el layout correspondiente con el objeto definido y se le indica que el progreso corresponda a la variable volumen dentro del ajuste. La única modificación sobre las acciones de la barra en caso de interactuar con ella es que si se cambia el progreso, repercuta en la variable volumen y que ese nuevo progreso sea escrito en la barra junto al símbolo %.

Un texto editable solo es un objeto distinto a los mencionados anteriormente, pero su manera de proceder es para que sea cargado y que se

tenga visualmente en la pantalla es la misma. Este objeto se relaciona con una variable dentro del objeto tipo Ajustes. Serán las propias librerías de Android las que muestren el teclado cuando se quiera editar y la aplicación solo atenderá al resultado establecido.

El botón cambiar usuario hará que con ese nuevo texto se carguen las opciones para un usuario que modificase las opciones anteriormente que tuviese el nombre del texto introducido. En caso de nunca haber sido introducido anteriormente ese texto como nombre de usuario, la aplicación cargará las opciones por defectos. En concreto, lo que hace este botón al ser pulsado es leer el texto puesto, guardar el nombre como nombre de usuario, reiniciar puntuaciones y opciones, cargar las opciones del nuevo nombre de usuario y reiniciar el menú opciones. Además, crea un *toast* informando que el jugador es cargado o, en caso que el nombre introducido fuese Miyamoto, que las trampas han sido añadidas.

Tras crear el botón salvar estableciendo la relación entre el elemento del layout y un objeto botón, se seleccionan las acciones que se realizarán en caso de ser pulsado. Estas acciones será la guarda de los datos, salir al menú inicio a través de un `Intent` y un `stratActivity`, realizar un *toast* que confirme visualmente al usuario que las opciones se han guardado y finalizar el menú opciones.

4.6.5. Rejilla de niveles. Menú niveles.

En las declaraciones se crean dos listas repetidas pero con distinto nombre direccionando los iconos de los niveles a dibujar. Esto servirá como la base de la rejilla con los iconos de cada nivel.

Para hacer que visualmente cada icono tenga las estrellas correspondientes a las puntuaciones, el candado en caso de que no se pueda jugar a ese nivel o nada en caso de que se pueda jugar pero aún no se haya superado, se crea el método `CreaRejillaEstrellada`. Este método carga las puntuaciones máximas que se guardan en los ajustes y compara las puntuaciones una a una. Si es mayor a cero, cambia el icono de la segunda lista creada con anterioridad por el dibujo de una estrella, si es mayor de 499 por dos

estrellas y si es mayor de 999 por tres estrellas. En caso de que no sea mayor de 0, es decir, el nivel no haya sido superado y sea el primer nivel en no ser superado, apunta dicho nivel como el nivel limite que se ha pasado y carga el icono correspondiente a ese nivel con la imagen de la pantalla. En caso de que no sea mayor que cero pero ya haber habido algún nivel que no fuese superado con anterioridad, cargará en esa segunda lista el icono de un candado. Adicionalmente se va sumando en una variable los niveles pasados.

También se crea el método `AdaptarImagen` que es herencia de la clase BaseAdapter. De éste, destacar que se selecciona la primera lista que se creó y en ningún momento se modificó, como fondo de cada icono y la segunda lista modificada con estrellas y candados como imagen por delante de ese fondo.

En el método `onCreate` se llama al método `CreaRejillaEstrellada`, se establece el layout y se relaciona su elemento estilo rejilla con un objeto al que más adelante lo hacemos adaptable a la pantalla cargando sobre él el método `AdaptarImagen`.

Como guiño, si se ha llegado a superar todos los niveles, se carga un sonido con MediaPlayer. A continuación se indica qué hacer si se cliquea un elemento de la rejilla en el método `onItemClick` dando la posición del elemento pulsado.

En caso de que la posición corresponda a un nivel al que se pueda jugar, se establecerá el nivel llamando al método `setMap` de PantallaJuegoUnJugador y se lanzará la clase JuegoLogical. En caso de que no se pueda jugar, será indicado con un *toast*.

Si tras la partida se vuelve a esta pantalla directamente, se ha modificado el método `onResume` para que se vean reflejado los cambios, en caso de que hubiese, en los iconos de los niveles llamando de nuevo al método `onCreate`.

4.6.6. Menú dos jugadores

El menú dos jugadores es una clase que hereda de MenuNiveles, por lo que sirve cada parte explicada anteriormente con la modificación de que lanza la clase JuegoLogicalBT en caso de que se quiera jugar a algún nivel permitido.

4.7. Guardar/Cargar datos

La carga y guarda de datos de la aplicación será desarrollada en la clase Ajustes. La intención es guardar las puntuaciones y opciones de cada usuario en archivos individuales dentro de la memoria del dispositivo asignada a la aplicación.

Android proporciona para ello el método `openFileOutput`, que recibe como parámetros el nombre del fichero y el modo de acceso con el que queremos abrir el fichero. Este modo de acceso puede variar entre MODE_PRIVATE para acceso privado desde nuestra aplicación (crea el fichero o lo sobrescribe si ya existe), MODE_APPEND para añadir datos a un fichero ya existente, MODE_WORLD_READABLE para permitir a otras aplicaciones leer el fichero, o MODE_WORLD_WRITABLE para permitir a otras aplicaciones escribir sobre el fichero. Los dos últimos no deberían utilizarse dada su peligrosidad. De hecho, han sido declarados como obsoletos en la API 17.

Este método, el `openFileOutput`, devuelve una referencia al *stream* de salida asociado al fichero (en forma de objeto `FileOutputStream`), a partir del cual ya podremos utilizar los métodos de manipulación de ficheros tradicionales del lenguaje java y en este caso se usará el mismo tratamiento que se hizo con los mensajes a enviar por Bluetooth. Se convierte este *stream* a un `OutputStreamWriter` para escribir una cadena de texto al fichero. Esa cadena de texto contendrá todas las opciones que queremos guardar separados entre ellas por comas.

Por otra parte, leer ficheros desde la memoria interna se hace de manera similar, y se procede de forma análoga, con la única diferencia de que utiliza el método `openFileInput` para abrir el fichero, los métodos de lectura de java.io para leer el contenido y todos los datos son pasados a una lista con el método `split` antes de ser cargados en las variables individuales.

4.7.1. Ficheros

Con estas opciones se ha creado un fichero de texto en la memoria interna. Android almacena por defecto los ficheros creados en una ruta determinada, que en este caso seguirá el siguiente patrón:

/data/data/paquete.java/files/nombre_fichero

En el caso de esta aplicación, la ruta de las opciones por defecto sería:

/data/data/com.proyecto.LogicalBT/files/Opciones6.txt

El archivo contendrá una única cadena con varios datos separados por comas.

4.7.2. La clase Ajustes

La clase Ajustes es una clase en la que se han escrito los métodos para la carga y guarda de los datos y la modificación de estos datos de manera correcta antes de cualquier guarda.

Lo primero es crear variables con los valores por defecto que se cargan en caso de abrir la aplicación por primera vez, incluida una lista de cien *Integers* para las máximas puntuaciones, y algunas variables de apoyo.

El primero de los métodos creados es `anadirPuntuación` que compara la puntuación que se otorga con la puntuación dentro de la lista con la posición de un nivel concreto y en caso de ser superior, sobrescribe el elemento en dicha posición e indica en una variable de apoyo que se ha realizado un cambio. Esta variable sirve para indicar al menú niveles que debe refrescar los iconos.

El siguiente método es `reiniciarPuntuaciones` que se encarga de volver a coger todas las variables de las opciones y poner sus valores por defecto. Este método es utilizado en el menú opciones cuando se cambia de usuario, justo antes de cargar sus valores guardados si los tuviese.

El método `getDataNombre` utiliza lo descrito anteriormente para cargar del fichero Nombre2.txt el nombre de usuario y si está activo el modo trampas. El método `getData` utiliza ese nombre de usuario que previamente ha podido ser cargado para crear el nombre del archivo con los valores de partida a cargar.

Este método carga el fichero Opciones+nombre+.txt a no ser que el nombre sea Jugador, entonces carga Opciones6.txt. Esta excepción se creó para que los dispositivos de los testeadores guardasen las puntuaciones de antes de implementarse la posibilidad de cambiar usuario. Las opciones que se cargan son si la música y el sonido se quieren encendidos, el volumen de la música y las puntuaciones máximas conseguidas en los distintos niveles.

Una vez se introduce un nuevo nombre, si se quiere guardar, se hace utilizando el método `putDataNombre`. Este método distingue si el nombre que se quiere guardar es Miyamoto, en tal caso activará el modo trampas y hará que el nombre a guardar fuese el nombre que previamente estuviese puesto. Tras ello, y de la manera explicada anteriormente, guardará una cadena con el nombre y el booleano del modo trampas en el archivo Nombre2.txt. De esta misma manera, y con las mismas excepciones del método `getData` se guardarán el resto de opciones con `putData` en un archivo en el que se usara el nombre de usuario para crear su nombre de manera similar al archivo que se carga en `getData`.

4.8. Actualizar

Para facilitar la obtención de la última versión de la aplicación por parte de los testeadores, se decidió implementar un apartado que permitiera actualizar de una manera sencilla.

Este apartado se basa en el actualizador de la página web androcode.es quienes lo presentan junto a un tutorial. Consta de una clase principal con layout asociado y una clase auxiliar, que es la importante.

La clase auxiliar VersionChecker posee el método `getData` que debe ser llamado antes que cualquier otro método, pues es el encargado de conectarse con internet, recoger, ayudándose del método `downloadHttp`, la información que se proporciona sobre la última versión y trasladarla a las variables. Esto se hace gracias a que se ha introducido en la cadena una dirección web donde poder obtener un archivo con esa información. Más adelante se describe ese archivo. Este método se ha modificado añadiendo un booleano que se modifica si no tiene éxito en conseguir dicha información.

El método `downloadHttp` sirve para leer el archivo de información. Es el encargado de conectarse a la red, descargar el archivo y convertirlo a *String*. Los demás métodos de esta clase se dedican a devolver algún dato de la información recogida.

El archivo que se ha colgado en internet debe estar escrito en formato JSON. En él se debe describir el número y nombre oficial de la última versión y la dirección desde la que se puede recoger. En la Figura 4-29 podemos observar el documento de texto con la información que se subió a internet.

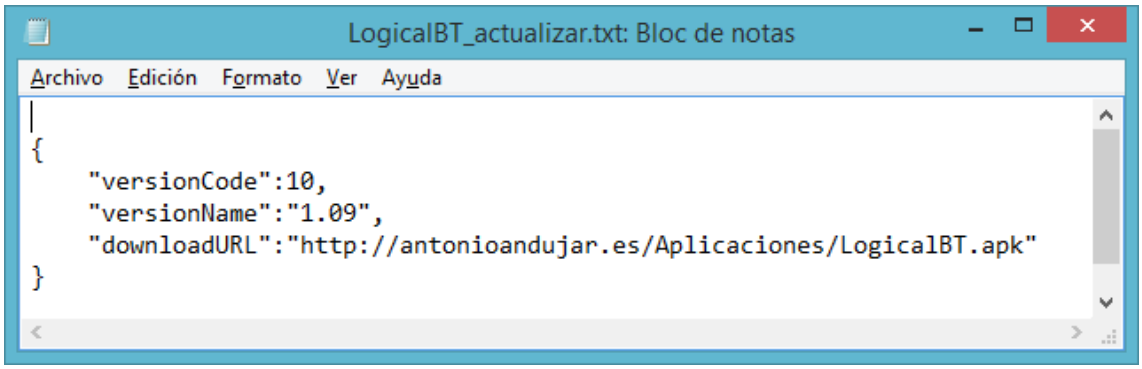

Figura 4-29. Información en JSON proporcionada.

A la clase Actualizar se llega a través del menú inicio. Hereda de la clase Activity y en él se crea un objeto Handler para la actualización de la interfaz desde el hilo en segundo plano, un objeto Runnable llamado finishBackgroundDownload que se ejecutará desde el hilo principal cuando acabe la obtención de datos en segundo plano y otro objeto Runnable llamado backgroundDownload encargado de descargar los datos en un hilo en segundo plano.

FinishBackgroundDownload lo que hace es comprobar si hay una nueva versión disponible y modificar lo que se muestra en pantalla dependiendo del resultado.

BackgroundDownload usa la clase de apoyo para obtener los datos de Internet y ejecuta en el objeto Handler el objeto finishBackgroundDownload cuando se haya acabado la descarga y así actualizar el interfaz.

En el método `onCreate` buscamos que la apariencia coincida con la de la aplicación realizando al comienzo una rutina muy parecida al mismo método del menú inicio o el menú opciones. Además se configura el primer botón para

que, si es pulsado, cree un nuevo hilo para la descarga de datos utilizando backgroundDownload. Si el segundo botón es pulsado, se lanza un `Intent` con el enlace de la descarga y dejando que el dispositivo Android se encargue de la descarga de la aplicación actualizada.

4.9. Pantalla Splash

La pantalla que introduce al usuario a la aplicación se forma en la clase Introducción y es hereda de BaseSplashActivity. Para que esta sea la primera pantalla en verse, se ha indicado así en el AndroidManifest.xml. Superponiendo algunos valores en los métodos explicados en uno de los apartados del punto 4.3.1, se hace que la orientación de la pantalla sea horizontal, que la imagen que se use sea el archivo splash.png, que la duración de la transición sea de 2 segundos, que el grado de zoom inverso a implementar sea el deseado y que la siguiente pantalla a mostrar sea el menú inicio.

4.10. Elementos y recursos del videojuego

Los elementos que integran una partida del juego se componen de un cantidad máxima de seis bolas activas a la vez y de cuarenta casillas donde cada una tiene la opción tener una actividad entre una lista de veintitrés. Estas casillas varían en cada uno de los cincuenta niveles implementados.

Por ello se decidió crear clases de apoyo que contuviesen la base para meter la información con métodos que ayudasen a la variación de sus características.

4.10.1. Bolas

La información relativa a las bolas activas que aparecen en el videojuego se recoge en la clase Bola. En ella se definen cuatro constantes para hacer más comprensible trabajar con la variable dirección. A estas cuatro constantes llamadas ARRIBA, DERECHA, ABAJO e IZQUIERDA se les asigna un número del cero al tres. Además, a la clase se le asignan distintas características en forma de variables: dirección, coordenada x, coordenada y, color y si se

encuentra activa la bola. Las cuatro primeras variables son tipo *Integer* y la última tipo *Boolean*.

Se ha especificado que en el método `Bola` se introduzcan todos los parámetros necesarios de la bola, haciendo que cada variable adquiera un valor concreto especificado al llamar al método. El otro método de la clase es `Avanzar`, donde se comprueba si la bola está activa, y si lo está, hace que una de las coordenadas aumente o disminuya según la dirección que tiene la bola.

4.10.2. Casillas

En el elemento Casilla se guarda toda la información relevante de cualquiera de los elementos que pueden encontrarse en cualquiera de los cuarenta espacios de cada nivel.

Lo primero que se hace es definir las constantes. A cada tipo de elemento le corresponderá un número *Integer*, del cero al veintidós. Sus asignaciones han sido ideadas para recopilar en números consecutivos los elementos que no actúan con el resto de elementos, los elementos que actúan sólo con otros elementos que se encuentren en su plano horizontal, los que pueden actuar tanto con los elementos que se encuentren en su plano horizontal como con los que se encuentren en su plano vertical y en la última serie, los elementos que actúan sólo con los de su plano vertical.

Esto ayudará a que a los elementos con características comunes se les puedan llamar al seleccionar un rango en lugar de llamar uno a uno. Así se simplifican códigos.

Los elementos definidos, por orden numérico asignado son:

0= NADA. Es un elemento vacío.
1= CONTADORBOLAS. Representa el número de bolas activas.
2= RELOJARENA. Representa el tiempo de la partida.
3= CRUZ. Muestra un patrón de bolas a conseguir.
4= SEMAFORO. Muestra colores para que sean resueltos en orden.
5= SIGUIENTEBOLA. Muestra la siguiente bola que saldrá.
6= PASILLOHORIZONTAL. Por donde puede pasar la bola.

7= TELETRANSPORTEHORIZONTAL. Si la bola llega a este elemento, aparecerá en otro similar pero en otras coordenadas.

8= PINTURAHORIZONTAL. Las bolas que pasan por él, terminan con el color que indican.

9= FILTROHORIZONTAL. Las bolas cuyo color no coincida rebotan y cambian su dirección a la contraria.

10= ENGRANAJE. Recoge bolas activas. Es el elemento clave del juego como ya se explicó anteriormente.

11= PINTURACRUZ. Igual que PINTURAHORIZONTAL pero las bolas pueden venir de cualquier dirección.

12= PASILLOCRUZ. Igual que PASILLOHORIZONTAL pero las bolas pueden venir de cualquier dirección.

13= FILTROCRUZ. Igual que FILTROHORIZONTAL pero las bolas pueden venir de cualquier dirección.

14= TELETRANSPORTECRUZ. Igual que el elemento con el número 7 TELETRANSPORTEHORIZONTAL pero las bolas pueden venir de cualquier dirección.

15= DIRECCIONADORDER. Cambia la dirección de la bola que pasa por su centro a DERECHA, puede venir de cualquier dirección.

16= DIRECCIONADORIZQ. Cambia la dirección de la bola que pasa por su centro a IZQUIERDA, puede venir de cualquier dirección.

17= DIRECCIONADORARR Cambia la dirección de la bola a ARRIBA, puede venir de cualquier dirección

18= DIRECCIONADORABA Cambia la dirección de la bola a ABAJO, puede venir de cualquier dirección

19= TELETRANSPORTEVERTICAL. Igual que el elemento con el número 7 TELETRANSPORTEHORIZONTAL pero las bolas solo pueden venir desde debajo o desde arriba.

20= PASILLOVERTICAL. Igual que PASILLOHORIZONTAL pero las bolas sólo pueden venir desde debajo o desde arriba.

21= FILTROVERTICAL. Igual que FILTROHORIZONTAL pero las bolas sólo pueden venir desde debajo o desde arriba.

22= PINTURAVERTICAL. Igual que PINTURAHORIZONTAL pero las bolas sólo pueden venir desde debajo o desde arriba.

Lo siguiente a definir son los colores que usan varios de los elementos para su interactuación con las bolas. Se deja el cero como vacío y se define ROJO, AMARILLO, AZUL y VERDE como cuatro constantes *Integer* con números del uno al cuatro respectivamente.

A continuación se definen las características de cada casilla aunque no todas serán usadas por cada elemento.

- tipo: *Integer* donde se selecciona el tipo de la casilla definido anteriormente.
- bolaarriba, boladerecha, bolaabajo, bolaizquieda: *Integers* que definen los colores de las bolas cuando el elemento lo necesite. En caso de que solo necesite un color, como los filtros o la pintura, únicamente usará bolaarriba.
- superado: booleano que indica si un elemento tipo engranaje ha sido superado. A todos los demás elementos se le dará esta variable como verdadera, de esa manera revisar si se ha superado una fase se realizará comprobando si no queda ningún elemento con la variable como falsa.
- teleHORx, teleHORy: posición del teletransporte al que enviar la bola. Lo usan tanto los teletransportes horizontales como los tipo cruz.
- teleVERx, teleVERy: posición del teletransporte al que enviar la bola. Lo usan tanto los teletransportes verticales como los tipo cruz.

Los métodos que definimos que ayudan a cambiar o establecer las variables de cada elemento se centran en el elemento tipo engranaje, ya que es el que mayor acción tendrá en cualquiera de las partidas.

El método `Casilla` introduce de golpe las características tipo, bolaarriba, boladerecha, bolaabajo, bolaizquierda y superado dentro del objeto Casilla que lo llame.

El método `GirarEngranaje` está ideado para mover las bolas del interior de un engranaje en sentido de las agujas del reloj. Haciendo uso de una variable temporal, crea unas relaciones entre los elementos que definen los colores de las bolas que posee para que esto suceda.

El método `Comprobacion` tiene como tarea hacer la comprobación de si el engranaje ha sido superado en condiciones normales, y en caso de que sea así actuar en consecuencia. Esa comprobación se hace asegurándose de que el engranaje no tiene huecos vacíos y que todos sus huecos poseen bolas del mismo color. Una vez comprobado, hace que todos sus huecos se den como vacíos, que se dé el engranaje como superado y que, en caso de tener el sonido encendido, suene un sonido parecido a una explosión.

4.10.3. Recursos

En lo referente al juego, se usan dos tipos de recursos: sonoros y gráficos. Los sonoros son extraídos del videojuego original con el emulador WinUAE, cambiados a formato MP3 con Winamp y el *plug-in* TFMX, recortados a través de la página MP3cut.net y pasados a formato OGG mediante el programa Switch Sound File Converter. Son guardados en la carpeta assets. Estos archivos se podrían dividir en música y sonidos. Los sonidos son:

- choque.ogg: se usa para el momento en que una bola choca contra la parte de un engranaje ocupado probando que esta rebote.
- click.ogg: usada en el momento en que el usuario extrae una bola de un engranaje.
- completado.ogg: se usa para el momento que un engranaje se da por completado.
- game_over.ogg: suena siempre que el usuario sea derrotado.
- Giro.ogg: se usa para la acción en la que se giran los engranajes.
- Recoge.ogg: usado en el momento que una bola se establece en un engranaje.
- Victorious.ogg: suena siempre que el usuario obtiene la victoria.

En cuanto a los archivos sonoros de música, tenemos cuatro para la música durante la partida llamados Levelmusica01.ogg, Levelmusica02.ogg, Levelmusica03.ogg y Levelmusica04.ogg y uno para la pantalla de espera en el modo de dos jugadores cuando el dispositivo en modo cliente se encuentra esperando a que el dispositivo en modo servidor elija nivel.

Los archivos gráficos también se guardan en la carpeta assets. Son tipo JPG o PNG, la mayoría extraídos mediante una captura de pantalla del videojuego original modificándolos cuando era necesario con Photoshop. Algunos otros han sido creados directamente con Photoshop. La Figura 4-30 muestra todos los recursos gráficos del juego.

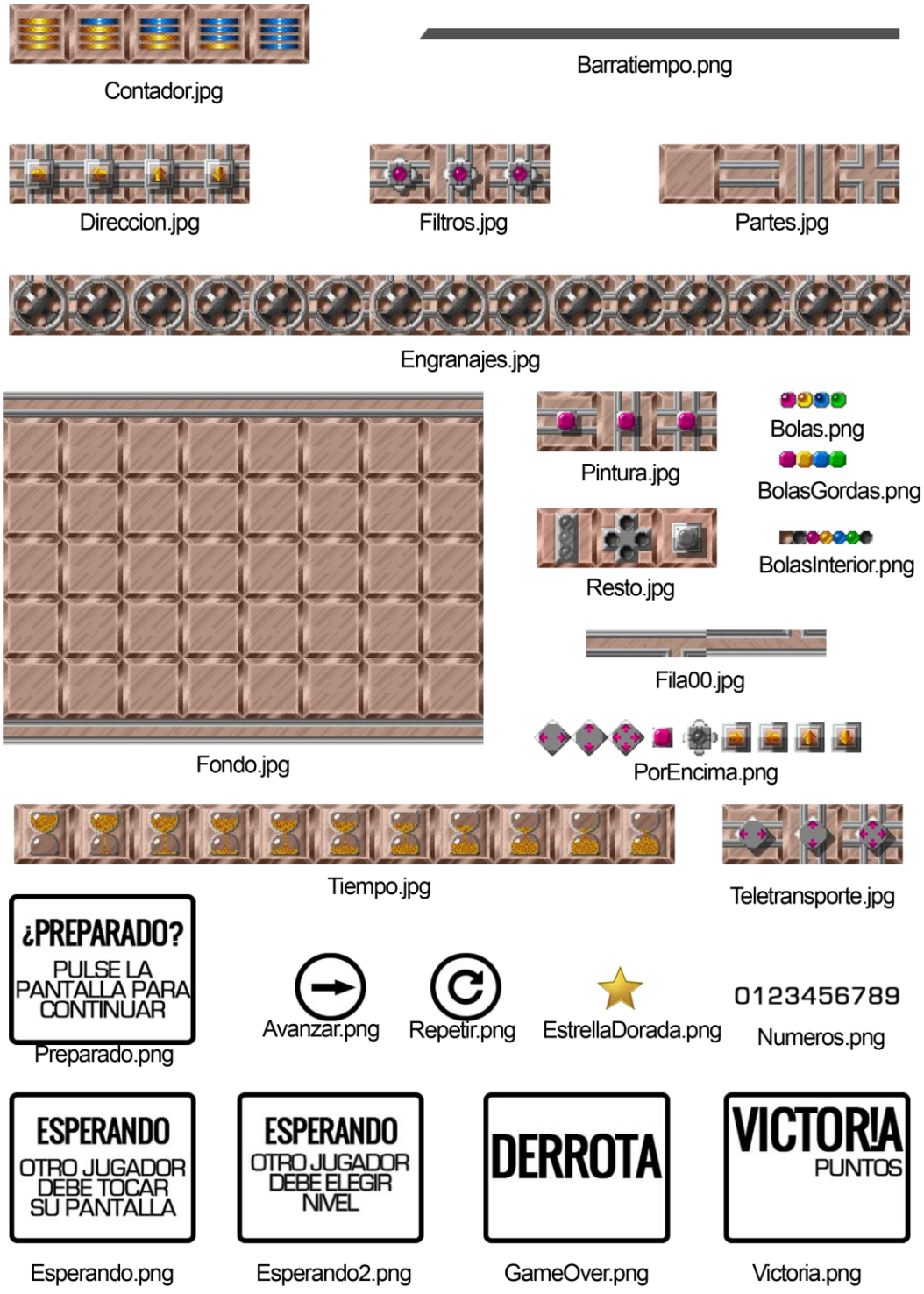

Figura 4-30. Todos los recursos gráficos de la parte jugable con sus nombres

Todos estos recursos se almacenarán en la clase Recursos y así se facilita el acceso a ellos. Esta clase consiste en tener muchos objetos públicos y estáticos que se encargan de contener los `Pixmap`, `Sound` y `Music` de los recursos. Hay un objeto estático para cada imagen y sonido que se ha cargado en la carpeta assets. Además también se guarda un *Integer* que recoge el nivel máximo implementado, en este caso 50, y dos matrices de *Integer*. La primera de tres dimensiones a la cual llamamos Elemento servirá para listar cada elemento de cada nivel. Por ello la amplitud de las dimensiones son del número del nivel máximo por el número de columnas por el número de las filas. La segunda matriz se crea porque hay determinados elementos que necesitan alguna característica extra de inicio, no bastando únicamente con saber el tipo que es. Esta matriz, llamada ElementoExtra, es del nivel máximo por el número de columnas por el número de filas por cuatro.

4.10.4. Estructuras de los niveles

La clase Estructura se ha creado para especificar cada elemento que constituye un nivel. Se basa únicamente en el método `Iniciar` que comienza inicializando toda la matriz Elemento de la clase Recursos con casillas tipo NADA, es decir, con ceros.

A continuación va nivel por nivel indicando a los elementos que sean distintos de NADA en la estructura de cada nivel qué tipo de casilla son. En caso de que la casilla sea tipo filtro o pintura, se usará la matriz ElementoExtra, donde se indicará en la misma posición, y dentro de ella, en la posición cero de la última dimensión, el color que se desea filtrar o pintar. Con los semáforos se hará igual, pero usando tres de esos elementos en orden del color que se desea que se tenga que completar primero hasta el último color que se debería completar para quitar la restricción del semáforo. Con los teletransportes se usaran sus dos primeros elementos para indicar la posición del otro teletransporte donde aparezca las bolas una vez atraviese su centro. En caso de ser un teletransporte con forma de cruz, usará los cuatro elementos para indicar dichas coordenadas, primero con las bolas que vengan en sentido horizontal, y luego con las bolas que vengan en sentido vertical.

Con indicar estos parámetros el motor del juego ya creará el nivel y lo desarrollara tanto en su apartado gráfico como en su lógica interna.

4.10.5. Entre los menús y el juego. Carga de recursos.

Ya se ha comentado anteriormente cómo comenzar una partida ya sea desde el modo de un jugador como del multijugador. En este apartado veremos 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 selecciona nivel en la clase MenuNiveles será arrancar la clase JuegoLogical, que será la responsable de generar y reproducir la primera pantalla del juego. Se ha derivado de AndroidGame e implementado el método `getStartScreen` con el que obtiene una instancia de la clase CargarPantalla que se comenta más adelante. Esta acción permite iniciar el juego junto con todos los elementos necesarios para que funcione, desde la configuración de diferentes módulos de sonido, gráficos, entrada del usuario y entrada y salida de datos en un archivo, hasta el inicio del hilo de ejecución del bucle principal.

La clase CargarPantalla se consigue a partir de la clase Screen. Se implementa un constructor que obtiene una instancia de Game que se recibe del constructor de la superclase gracias a que fue llamada usando el método `getStartScreen`.

A continuación, se implementa el método `update` que se emplea para cargar los recursos y la configuración del juego. Para los recursos se usa el método `Graphics.newPixmap` con los que se generan nuevos Pixmap en los objetos definidos en la clase Recursos. Se especifica el formato del color que tienen los Pixmap. El de fondo será RGB565, mientras que el resto de imágenes será ARGB444. De esta forma se conserva memoria y se incrementa la velocidad de juego. Las imágenes originales se guardan como archivos PNG y JPG con formatos RGB888 y ARGB8888. También se crean los efectos de sonido y la música y se guardan en los objetos correspondientes de la clase Recursos.

Se cargan los ajustes que nos dará la configuración del juego recuperándolos de la memoria del dispositivo como ya se ha explicado anteriormente, con el método `getData` de la clase Ajustes. También se carga la estructura de todos los niveles en los objetos que eran matrices de *Integers* definidos en la clase Recursos al llamar al método `Iniciar` de la clase Estructura.

Por último, se llama al método `siguientePantalla`, que se crea un poco más adelante. Este método inicia una pantalla de transición a una Screen llamada PantallaJuegoUnJugador que es la clase donde se desarrolla todo el peso del juego.

En caso de que se quiera jugar una partida Multijugador se hace todo de forma muy parecida. Se inicia la clase JuegoLogicalBT que es exactamente igual que JuegoLogical, salvo que ésta termina llamando a la clase CargarPantallaBT. CargarPantallaBT hereda todo de CargarPantalla y sólo hace un cambio, dentro del método `siguientePantalla`, que es llamar a la clase Multijugador en lugar de PantallaJuegoUnJugador.

4.11. Lógica del videojuego

El desarrollo del motor del videojuego se realiza todo en unas pocas clases, si bien muy atareadas durante el transcurso de una partida. La primera clase llamada se ayudará de una clase que contendrá la mayoría de la lógica automática. La usará para fijar los elementos del nivel al inicio y la irá llamando cada cierto tiempo (*deltaTime*) para que haga que todos los elementos realicen la acción automática que les corresponda.

El modelado y la presentación gráfica lo realizará la primera clase cargando el estado de ese momento y dibujándolo en pantalla cada periodo de tiempo.

4.11.1. Lógica automática

Las variables

Se desarrolla en la clase Mundo y lo primero que hace es desarrollar una larga lista de variables. Estas variables se pueden dividir según su función para poderlas explicar mejor al saberse el marco en el que actuarán.

Variables principales:

- casilla. Es una matriz del número de columnas por el de filas con elementos de clase Casilla. En ella se establecerán los elementos del nivel a jugar.
- bola. Es una lista de diez elementos de clase Bola donde se registrarán las bolas que transcurren por la pantalla. La primera casilla será usada para la bola que transcurre por el pasillo superior; desde la segunda hasta la quinta, se rellenarán cuando se esté en modo un jugador o si las bolas son creadas en el dispositivo servidor; desde la sexta a la novena se rellenara con bolas creadas desde el dispositivo cliente y la décima y última está guardada para la bola que transcurre por el pasillo inferior.
- numerobolas. Establece el número de casillas que serán usada en la lista bola. Cinco para modo un jugador y diez para dos jugadores.
- bolamin. Establece la primera posición de la lista donde se registran las bolas dentro de la estructura creadas por el dispositivo.
- bolamax. Establece la última posición de la lista donde se registran las bolas dentro de la estructura creadas por el dispositivo.
- jugadorunico. *Boolean* activo si está en modo un jugador.

Variables de estado:

- gameOver. *Boolean* que indica si el usuario ha sido derrotado.
- Victoria. *Boolean* que indica si el usuario ha conseguido superar el nivel.
- Pausa. *Boolean* que se activa cuando la partida está pausada.

Variables sobre el tiempo:

- tiempobola00. En este *float* se guarda el tiempo que ha transcurrido desde la última vez que entró una bola nueva por el pasillo superior.
- tiempobola55. En este *float* se guarda el tiempo que ha transcurrido desde la última vez que entró una bola nueva por el pasillo inferior.
- tiempolimitebola00. Sirve para establecer el tiempo límite que tiene el jugador para conseguir quitar las bolas del pasillo superior e inferior.
- tiempo. Se utiliza para guardar el tiempo transcurrido en el nivel.
- tiempolimite. Se establece el límite de tiempo del usuario para superar el nivel.
- TickInicial. Tiempo que indica cada cuánto tiempo ha de avanzar una bola y otras acciones.
- tickTime. Variable que se usara para asegurarse que se realizan ciertas acciones en determinado tiempo.
- tick: Variable marcada por TickInicial que será la manejada dentro de los métodos.

Variables sobre la cola de bolas:

- bolasiguiente. Guarda el color de la siguiente bola en salir por el pasillo superior.
- bolasiguiente2. Guarda el color siguiente en establecerse como bolasiguiente.
- bolasiguiente3. Guarda el color siguiente en establecerse como bolasiguiente2.
- bolaprimeriza. Guarda el color siguiente en establecerse como bolasiguiente3.
- bolasiguiente55. Guarda el color de la siguiente bola en salir por el pasillo inferior.
- bolasiguiente255. Guarda el color siguiente en establecerse como bolasiguiente55.
- bolasiguiente355. Guarda el color siguiente en establecerse como bolasiguiente255.

- bolaprimeriza55. Guarda el color siguiente en establecerse como bolasiguiente355

Variables sobre el semáforo:

- semaforo1. Guarda el primer color que el semáforo quiere restringir.
- semaforo2. Guarda el segundo color que el semáforo quiere restringir.
- semaforo3. Guarda el tercer color que el semáforo quiere restringir.
- semaforo. Indica cuando hay un semáforo y aun no se han superado todos los colores.

Variables sobre la cruz de bolas

- haycruz. Indica si existe una cruz de bolas en el nivel.
- cruzllena. Indica si la cruz de bolas tiene un patrón activo.
- Cruz. Lista con las cuatro bolas del patrón de la cruz de bolas.
- BolasparaCruz. Lista con el siguiente patrón que se establecerá en la lista Cruz.
- tiempocruz. Tiempo que lleva la cruz de bolas con el último patrón resuelto y sin nuevo patrón puesto.
- nuevabolascruz. Indica si ya se han establecido los colores en BolasparaCruz.

Variables para la puntuación

- puntuacion. Es donde se guarda la puntuación total cuando se consigue la victoria.
- bolasquedan. Indica las bolas que quedan dentro de los engranajes. Será utilizado para calcular la puntuación.

Variables para el modo multijugador

- bolaentro00. Avisa que la bola del pasillo superior entró en un engranaje.
- bolaentro55. Avisa que la bola del pasillo inferior entró en un engranaje.

- elservidor. *Boolean* que refleja si el dispositivo está en modo servidor o cliente.
- Mensaje. Variable de la clase MensajeEnLista, que se verá más adelante,
- lista. Crea una lista mediante el método `List` de objetos de clase MensajeEnLista.

Al crearse la clase

Lo único que hace la clase Mundo al ser llamado es ejecutar la rutina de su método `inicializar` para cargar el mundo con la estructura y parámetros del nivel.

En el método `inicializar` se indican los parámetros, característica y estructuras de los niveles cargándolos además con valores iniciales a los elementos que podrían verse heredados de alguna otra partida.

Lo primero que hace este método es comprobar el nivel en el que se encuentra y seleccionar el tiempo límite para que el jugador tenga que encajar en algún engranaje la bola del pasillo. Lo normal es que sea de 20 segundos, pero el nivel 19 tiene un límite de 5.58 segundos y el nivel 48 de 9 segundos para equiparar la dificultad del nivel al original.

La siguiente acción que realiza es la composición de la pantalla. Cogemos la variable casilla, que era una matriz de 8 por 5 de elementos Casilla donde se pueden definir todos los espacios de la partida. Para cada elemento de la matriz usamos el método `Casilla` para establecer el tipo que especifica la matriz Elemento de la clase Recurso que cargamos gracias a la clase Estructura, la fila, la columna, las bolas de dentro como vacías y el booleano superado como verdadero.

Tras esto se vuelve a mirar el tipo otorgado a casilla y se realizaran acciones pudiendo sobrescribir algunas variables de las que se acaban de establecer. En caso de ser engranaje se da superado como falso. Para los teletransportes, se igualan las variables teleHORx, teleHORy, teleVERx y teleVERy con la fila o columna que se escribió en sus correspondientes elementos de la matriz ElementoExtra. Si es pintura o filtro se iguala la variable

de la Casibolaarriba con el ElementoExtra donde introdujimos el color que debe caracterizarlo. En caso de ser semáforo, se igualará las variables semaforo1, semaforo2 y semaforo3 a los tres elementos correspondientes con los colores que se querían poner de la matriz ElementoExtra. Además se hará que el booleano semaforo sea verdadero, lo que más adelante servirá para indicar que hay un semáforo activo. Por último se crea una rutina especial en caso de ser un elemento tipo cruz de bolas. Lo primero que se hace es poner la booleana haycruz como verdadera, después ejecuta los métodos `bolasparacruz` y `rellenacruz` que describiremos más adelante.

Para finalizar este primer método, `incializar`, se crea una rutina que inicialice también las bolas así como la lista de las siguientes bolas. Para ello coge la lista de elementos de clase Bola que se llamó bola y ejecuta en cada una el método `Bola` donde se ha de destacar que se pone la booleana activa de cada bola como falsa. Después se hace que bolasiguiente, bolasiguiente2, bolasiguiente3 y bolaprimeriza valgan un número aleatorio entre el 0 y el 3 que indicarán los colores de la cola de bolas listas para salir. Para terminar con este método se llama al método `nuevabola`.

La bola del pasillo

El método `nuevabola` se implementó para crear una nueva bola en el pasillo. Ejecuta en la primera Bola de la lista bola el método `Bola` indicando sus nuevas características: como su dirección de movimiento, hacia la izquierda; como coordenadas, las de inicio; como color, el que había en la variable bolasiguiente; y como booleano activa, verdadero, indicando así que esa bola está en movimiento.

Tras ello hace que bolasiguiente coja el color de bolasiguiente2, bolasiguiente2 de bolasiguiente3, bolasiguiente3 de bolaprimeriza y a bolaprimeriza se le asigna un nuevo color de forma aleatoria al que se seleccione un número entero aleatorio entre 0 y 3, ambos incluidos. Después hacemos que el booleano bolaentro00 sea verdadero.

El siguiente método escrito es `bolasale` que junto al método `bolasale2`, se utilizan en caso de que el usuario saque una bola de algún

engranaje. A bolasale se la ejecuta dándole la posición del engranaje y la dirección que debería tomar la bola que se desea sacar. Lo primero que hace es comprobar que no se ha superado el límite de bolas activas a la vez. En caso de que no se haya superado, buscará la primera bola de la lista bola dentro de su parte asignada (que lo limita bolamin y bolamax) que no esté activa. Con ello se llama a bolasale2 proporcionándole la posición del engranaje y la dirección dada y el lugar, dentro de la lista, de la bola que no estaba activa.

El método `bolasale2` comprueba con los datos dados si hay bola en la parte del engranaje que se quiere sacar. Si es así, asigna a la bola su color, su dirección, la pone activa y le adjunta las coordenadas que corresponden a la posición del engranaje y de su localización en él en el momento de salir. Hecho esto, se reproduce el sonido click si está habilitado sonidoEncendido y se vacía la bola de la que se extrajo el color dentro del engranaje correspondiente.

El método `bolafila00check` comprueba si alguna bola del pasillo superior entra en el engranaje, y en caso de que lo haga, actúa en consecuencia. Tras asegurarse que le corresponde comprobar dicha fila con las variables jugadorunico o elservidor, comprueba si la bola que avanza por el pasillo está justo arriba y en el centro de un engranaje con el espacio de la bola de arriba vacío. Si es así desactiva la bola poniendo su booleano activa como falso y ejecuta el método `bolaentro` indicando que es la bola 0, la columna del engranaje, que la fila es la 0 y que la posición es arriba. Antes de terminar este método, se hace una comprobación adicional. Si la bola ha llegado al principio o al final del pasillo, se cambia la dirección al sentido opuesto, para que así se produzca un rebote.

La cruz de bolas.

El método `bolasparacruz`, que será llamado solo si existe un elemento tipo cruz de bolas dentro del nivel, busca poner nuevos colores en una especie de lista de espera para la cruz de bolas. Lo hace activando el booleano que indica que hay nuevas bolas en la cruz y añadiendo en cada elemento de la lista BolasparaCruz colores de bola aleatorios asegurándose que los cuatro elementos no tengan el mismo color.

El método `rellanacruz` coge la lista que se modificó en `bolasparacruz` y la traslada a la lista Cruz, además de poner como verdadero el booleano que indica que la cruz está llena.

Comprobación de la superación de un engranaje

Los métodos `compsemaforo`, `compruebacruz`, `compruebatodos` y `compruebasemaforo` se fueron haciendo a medida que se fueron añadiendo restricciones a la comprobación simple de un engranaje de si tenía todos sus espacios con las bolas del mismo color. El semáforo restringe el color teniendo que resolverse por orden, pero la complicación viene cuando se ha resuelto uno, ya que ha de hacer una comprobación para ver si el siguiente color se ha resuelto o, en caso de estar ya resuelto, si se puede dar por completados más engranajes. La cruz de bolas solo deja dar como completados los engranajes que tienen un patrón, por lo que no solo hay que hacer comprobación cada vez que se introduce una nueva bola en un engranaje, sino también cuando un engranaje se gira. Si mezclamos estos dos elementos la cruz de bolas tiene prioridad sobre el semáforo pero toda la comprobación se vuelve todavía más compleja del método inicial, donde solo se miraban si los elementos eran engranajes y si estaban completos llamando al método `Comprobacion` para cada casilla. Para tener un mejor entendimiento de esta parte, se puede observar la Figura 4-31.

El método `compsemaforo` recibe el número de la bola que acaba de entrar en un engranaje y la posición de dicho engranaje. Si no hay semáforo o está vacío y no hay cruz de bolas o está vacía, realiza una comprobación del engranaje llamando al método `Comprobacion` dentro del elemento con la posición dada. Si tiene la cruz de bolas llena, ejecuta el método `compruebacruz` en la posición del engranaje y si no está la cruz de bolas llena, mira si la bola añadida tiene el color que exige resolver el semáforo y si es así lanza el método `compruebasemaforo` en la posición dada.

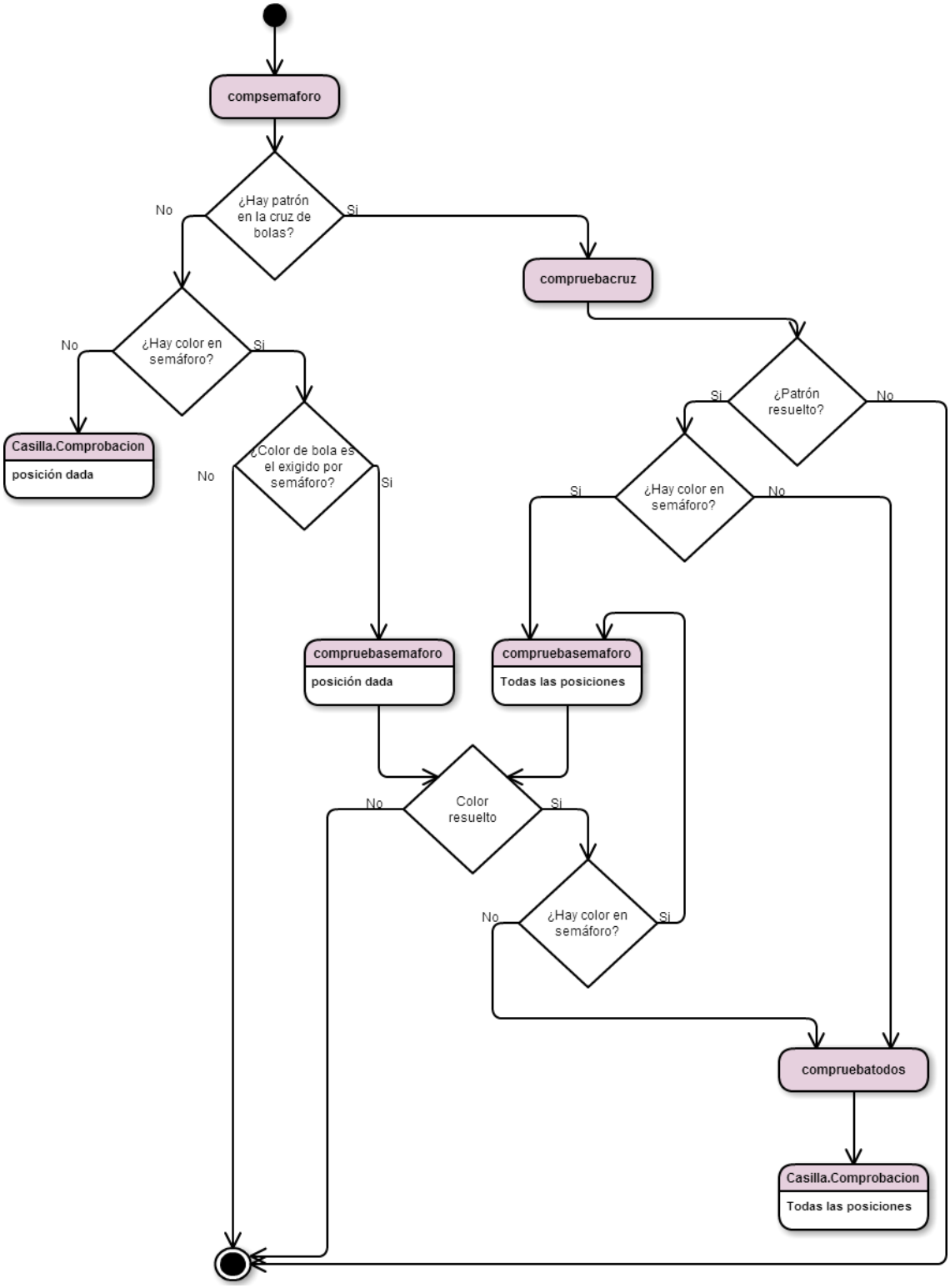

Figura 4-31. Diagrama de estados de la comprobación de la superación de engranaje.

`Compruebacruz` compara una a una las casillas del patrón que exige la cruz de bolas con las del engranaje en la posición dada. Si es así, vacía el engranaje y lo marca como superado, vacía la cruz de bolas, marca que la cruz de bolas está vacía, ejecuta el método `bolasparacruz` y, en caso de estar

activado, hace que suene el sonido completado. Tras ello, observa si hay semáforo y no está vacío. Si es así va casilla por casilla comprobando si es un engranaje y su espacio de arriba tiene el mismo color que exige el semáforo. Si ocurre eso en algún caso, arranca el método `compruebasemaforo` en esa posición. En caso de que la comprobación marcase que no había semáforo o que estaba vacío, se ejecuta el método `compruebatodos`.

Al método `compruebasemaforo` se llega dando una posición de un engranaje y habiéndose asegurado que la bola de arriba de coincide con el color que en eso momento restringe el semáforo y que no hay restricciones por parte de una posible cruz de bolas. Por ello lo primero que hace es comprobar si está completado llamando al método `Comprobacion` del engranaje en la posición dada.

Tras ello mira si el engranaje está vacío, lo cual sería síntoma de que el engranaje ha sido superado y si es así, observa en cuál de los colores que restringen ha sido completado:

- Si es el primero, da paso a la siguiente restricción y hace un barrido de todas las casillas del nivel mirando si los elementos son engranajes y si su espacio superior tiene el color de esa nueva restricción. En caso de que así sea, vuelve a llamar al método Comprobación y de manera similar a lo antes escrito, comprueba si ha sido resuelto el engranaje o no. En caso de que sea resuelto, se actualiza la restricción del semáforo y se realiza un nuevo barrido de manera idéntica pero con el tercer y último color del semáforo. En caso de que éste también sea resuelto, da el semáforo como resuelto y hace un último barrido comprobando si cualquiera de los engranajes tiene los cuatro colores idénticos en sus huecos.
- Si es el segundo el color realiza la misma rutina anteriormente escrita partiendo del momento en que el segundo color se supera.
- Si es el tercer y último color, se da al semáforo como completado y llama al método `compruebatodos`.

El método comprueba todos hace un barrido de cada posición del nivel haciendo ejecutar el método `Comprobacion` en cada elemento tipo engranaje que encuentra.

Comprobación y movimiento de las bolas

Al método `bolaentro` se llama cuando una bola entra en un engranaje. Es llamado indicándole el número de la bola, la posición del engranaje, la dirección que tenía la bola y si hay que enviar algún mensaje. La información de esta última variable sólo se utiliza en la parte multijugador. Lo que hace en el modo un jugador es comprobar qué dirección tenía la bola y equipara la posición del hueco del engranaje con el color de la bola. Hace sonar el sonido recoge si está habilitado y arranca la rutina para comprobar si el engranaje está completado llamando a `compsemaforo` y dando el número de bola y la posición del engranaje. Tras ello desactiva la bola y comprueba si esta bola venía del pasillo superior. En tal caso reinicia el tiempo del pasillo y ejecuta el método `nuevabola` para que haya una nueva bola en el pasillo.

El método `bolareboto` es ejecutado cuando una bola rebota contra un engranaje que tenga el hueco donde debería situarse lleno. Sabiendo el número de la bola y la dirección, lo que hace es cambiar la dirección a la opuesta y ejecutar el sonido choque si están habilitado los sonidos.

Para la comprobación de dónde están las bolas y su respuesta en la interactuación con algún elemento se ha creado el método `bolacualquieracheck`. Este método coge una a una cada bola que no se encuentra en pasillos y comprueba si está activa. Si lo está, saca la posición de la casilla donde se encuentra a través de sus coordenadas. Con ello, comprueba el tipo de elemento que hay en su posición y actúa en consecuencia:

- Si está en un engranaje, primero se asegura que es un jugador único, y si no, que es una de las filas de engranaje que maneja. El dispositivo servidor maneja las tres filas superiores y el cliente las dos inferiores. En caso afirmativo comprueba si las coordenadas de la bola encajan con alguno de los huecos, si es así mira si el hueco está vacío. Una respuesta afirmativa hará que se ejecute el

método `bolaentro` y una negativa el método `bolareboto` con los parámetros que necesite.

- Si está en un teletransporte comprueba si la bola, contando con la dirección en la que avanza, está justo a falta de un movimiento para llegar a las coordenadas del centro de la casilla. Si es así establece en la bola nuevas coordenadas. Éstas serán en el centro de la casilla cuya posición se estableció como una característica para las casillas tipo teletransportes.
- Si está en un filtro comprueba si la bola, contando con la dirección en la que avanza, está justo antes de lo que se podría considerar núcleo central de la casilla. Es decir, le quedarían varios avances para llegar al centro. Si es así, compara el color de la bola con el del filtro y en caso de no coincidir, cambia la dirección de la bola por la opuesta.
- Si está en una pintura comprueba si las coordenadas de la bola coinciden con el centro de la casilla y en caso afirmativo establece el color de la bola con el color de la pintura.
- En caso de ser un direccionador comprueba si la bola está en el centro y en caso afirmativo establece la dirección de la bola con la dirección a la que apunta el direccionador.
- En caso de ser un pasillo no hace nada.

El método `Avanzar` provoca el avance de la bola. Se fija si la bola está activa y en caso de estarlo comprueba su dirección y suma o resta uno a las coordenadas de esta dependiendo de esa dirección.

Consiguiendo la victoria

El método `Ganando` es el que realiza todo lo referido a la comprobación de si se han conseguido los parámetros para la victoria y sus consecuencias en el juego. Al principio, inicializa una variable temporal como falsa y comprueba casilla a casilla si la variable superada del elemento es verdadera. En cuanto hay una casilla, presumiblemente tipo engranaje, que no haya sido superado, coloca esa variable temporal como verdadera. En caso de que esa variable no sea verdadera, es decir, que no quede ningún engranaje no superado, y que no se

haya producido la rutina que se va a relatar con anterioridad, el método realiza una serie de acciones.

La primera es pausar la música, y después, en caso de tener el sonido encendido, hace sonar una melodía que indica la victoria. Lo siguiente a realizar es averiguar la puntuación. Para ello hace una búsqueda de todas las casillas para ver cuál de ellas es tipo engranaje, y en ellas, que huecos no están vacíos. Si quedan más de diez bolas dentro de los engranajes, se cuentan como diez. Tras ello calcula la puntuación restando al tiempo límite, el tiempo transcurrido y dividiéndolo por el tiempo límite. Es decir, averigua el tiempo porcentaje de tiempo que le quedaba. Esto lo multiplica por mil, le suma otros mil, y le resta el número de bolas que quedaban en los engranajes multiplicado por cien. Es decir, el tiempo que queda da como máximo mil puntos y las bolas que quedan dan como máximo mil puntos. La puntuación será la suma de ellas. Tras ello pone como afirmativa la booleana que indica la victoria sobre el nivel.

Moviendo el mundo

Hay dos métodos implementados en esta clase que no se han explicado, aparte del método que moverá el mundo. Principalmente están realizados para ser llamados ahora pero realizar acciones significativas en el modo multijugador. El primero de estos métodos en `Comprobacionfilas` que simplemente llama al método `bolafila00check`. El segundo es `tiempo55` que no hace nada.

El método `update` es lo que realmente hace mover al mundo y debe ser llamado cada cierto tiempo (deltaTime). Lo primero que hace es ver si las variables que indican victoria, derrota o pausa son afirmativas, en ese caso el método no hará nada más. A continuación suma a las variables tickTime, tiempobola00 y tiempo la variable que se da al iniciar el método, deltaTime. Esta variable marca el tiempo transcurrido desde la última vez que este método fue llamado por lo que si se van sumando los deltaTime que se van dando, se puede tener un cómputo del tiempo transcurrido.

Si en el nivel hay cruz de bolas, pero ésta está vacía, se añade también deltaTime a la variable tiempocruz. Cuando este llegue a los noventa segundos, ejecutará el método `rellenacruz` y reiniciará tiempocruz a cero. Esto provoca

que una vez se logre superar el patrón de una cruz de bolas, se establezca un nuevo patrón a superar a los noventa segundos.

Lo siguiente es comprobar si el tiempo transcurrido desde la última vez que una bola del pasillo se superó ha llegado al límite o si el tiempo que se lleva jugando en el nivel ha superado el tiempo límite total de la partida. Esto se hace comparando tiempobola00 con tiempolimitebola00 y tiempo con tiempolimite. Si esto sucede, se da la partida como perdida. Ello hará que la variable que indica derrota se establezca verdadera, que se pause la música y que, en caso de estar habilitado, suene la música de la derrota. Tras ello se ejecuta el método `tiempo55` pero no tiene consecuencias en el ámbito de un solo jugador.

Como deltaTime es demasiado tiempo y no es un valor fijo, se crea un bucle que realizará una acción repetidamente varias veces. Para ello se ira comparando la variable tickTime con el pequeño valor de tick. El valor tick especifica cada cuánto tiempo se quiere realizar una acción. El valor de tickTime recoge no solo lo transcurrido desde la última vez que se llamó al método (dato que proporciona deltaTime), sino también el tiempo que sobró en el último bucle. Mientras tickTime sea superior a tick no saldrá de este bucle. El bucle comienza restando a tickTime el valor de tick; tras ello hace una comprobación de las bolas llamando a los métodos `Comprobaciónfilas` y `bolacualquieracheck`. Posteriormente, hace avanzar a las bolas ejecutando el método `Avanzar` con cada una de ellas. Para finalizar el bucle, comprueba si se ha conseguido la victoria llamando al método `Ganando`. Todo esto, se mira en conjunto, lo que hace es que las bolas avancen una unidad cada tiempo tick y que se midan las consecuencias de ese avance.

4.11.2. Lógica automática multijugador

La lógica automática para el modo multijugador no solo requiere de acciones especiales, también debe tener en cuenta que algunas acciones, ya sean idénticas al modo de un jugador, modificadas o añadidas, deben de ser trasladadas al otro dispositivo con un mensaje vía Bluetooth. Por ello hay varios booleanos que actúan como resortes pero como no es suficiente, se crea la clase MensajeEnLista que hará de puente entre la clase MundoBT y la clase encargada de enviar los mensajes.

La clase MensajeEnLista se usa para trasladar si una bola ha rebotado en un engranaje o se ha acoplado a un engranaje mediante el booleano rebota. Con ello, la posición de ese engranaje, la dirección de la bola y cuál de las bolas es. Con el método `MensajeEnLista` introducimos de golpe todos estos elementos.

La clase MundoBT hereda todo lo descrito en Mundo variando algunos de sus métodos, cambiando por completo otros y creando un par nuevos.

Al método `inicializar` se le hace cambiar la cantidad de espacios en la lista bola que usa a diez y, en caso de ser dispositivo en modo cliente, la posición de la lista bola mínima y máxima que puede usar para la bolas que active él al sacarlas de un engranaje. Tras ello ejecuta toda la rutina original del método. Una vez concluida, carga cada variable relacionada con la cola de bolas a salir por el pasillo inferior un color aleatorio y llama al método `nuevabola55`.

El método `nuevabola55` es uno de los nuevos métodos implementados y crea de forma pareja al método `nuevabola00`, una nueva bola en el pasillo inferior. Esta bola será registrada en el último espacio de la lista bola.

El método `bolafila55check`, también nuevo, comprueba, tras cerciorarse que es el dispositivo en modo cliente, si la bola que transcurre por el pasillo inferior entra dentro de algún hueco inferior vacío de un engranaje. También hace rebotar la bola en caso de que está llegue a algún extremo del pasillo. Todo de manera similar al método `bolafila00check`.

En `bolareboto` y `bolaentro` ahora sí se utiliza el booleano que se introduce al llamar a esos métodos. Sirve para indicar que se debe introducir un nuevo mensaje en la lista que será utilizada para enviar información vía Bluetooth. Es decir, si se ejecuta alguno de estos métodos desde la rutina del mundo, se creara un mensaje con las mismas características introducidas al llamar al método y se agrega a la lista. Si se ejecuta alguno de estos métodos por haber recibido un mensaje vía Bluetooth, no se creara mensaje ni se enviara a ninguna lista pero se ejecutará el resto de la rutina de igual manera.

Aparte de este cambio, en el método `bolaentro` se introduce que se inicialice la variable tiempobola55 a cero y se ejecute el método nuevabola55 si la bola que entra en un engranaje es la del pasillo inferior.

El método `tiempo55` que en la clase Mundo estaba vacío, se rellena ahora con una rutina que marcará la derrota si se llega al tiempo límite en el pasillo inferior. Al tiempobola55 se le suma deltaTime y el resultado se compara con el límite establecido para los pasillos. Si ha sido superado, se activa el booleano de la derrota, se pausa la música y se ejecuta el sonido de derrota si procede.

El último cambio se realiza en `Comprobacionfilas`, que mientras anteriormente llamaba a `bolafila00check` para ejecutar la rutina que comprobaba el pasillo superior, ahora, además, también llamará a `bolafila55check`, comprobando así ambos pasillos.

4.11.3. Interacción de usuario

Toda la interacción del usuario en el juego se maneja en la clase PantallaJuegoUnJugador. 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 y desarrolla todo el modelado gráfico.

PantallaJuegoUnJugador hereda de Screen. Eso significa que tiene cinco métodos por defecto: `update`, donde se marcaran las acciones a realizar; `present` que manejara todo el modelado gráfico y que se desarrollará en el punto 4.11.5; `pause`, que marcará qué acciones hacer en caso de interrupción del juego; `resume`, que realizara las rutinas en caso del que se vuelva al juego tras la interrupción; y `dispose`, que elimina la clase y libera los recursos y en el que no se hará modificación alguna.

Lo primero que se hace en PantallaJuegoUnJugador es definir constantes y variables. Se crea un nuevo tipo de variable, llamado EstadoJuego, cuyas posibles valores son Preparado, Funcionando, Pausado, GameOver y Victoria. Las variables globales de la clase son:

- estado. Es una variable tipo EstadoJuego que marcará el estado en el que se encuentre la partida.
- mundo. Variable de la clase Mundo donde se cargará el nivel a jugar y al que se llamará para que realice los cambios automáticos.
- viejapuntuacion. Usada para la puntuación en caso de Victoria.

- puntuación. Usada para la puntuación en caso de Victoria.
- nivel. Guardará el número del nivel a jugar.
- lugarbarra. Marcará hasta qué lugar ha avanzado la barra que marca el tiempo que queda para que una bola sea quitada del pasillo superior.
- tiempo. Guarda el tiempo de la partida.
- comienza. Booleano que se activará en cuanto la partida comience.
- dedopulsado. Booleano que actúa de resorte para evitar el efecto de un doble toque.
- dedoj. Guardará qué fila se ha pulsado por última vez.
- dedjok. Guardará qué columna se ha pulsado por última vez.
- bolasgastadas. Cuenta las bolas activas.
- música. Música para este nivel.
- musicanivel. Recurso temporal que ayuda a seleccionar la música.
- casilla. Guarda la lista de elementos que componen el nivel.

Preparativos, previo y pausa

Lo primero que hace la clase es cambiar el estado a Preparado. Tras ello, calcula la música que sonará durante la partida cogiendo el resto de la división del nivel entre dieciséis. Cómo en el videojuego original, la música irá cambiando cada cuatro niveles. A la hora de establecer la música hay una particularidad, y es la elección de la sintonía elegirnivel en caso de que el nivel sea cero, circunstancia que sólo se da en el apartado multijugador, mientras el dispositivo cliente está esperando la elección del nivel por parte del dispositivo servidor. Una vez se tiene la música seleccionada, se establece que haga un bucle, se le pone el volumen que se ha podido modificar en el menú opciones y, en caso de tener como opción la música encendida, se reinicia con restart, rutina que se tuvo que implementar en el framework de Mario Zechner para que la sintonía se pudiera reiniciar tras ser pausada. Tras todo ello se llama al método `cargarMundo`.

El método `cargarMundo` inicia en la variable mundo una nueva clase tipo Mundo. Esto hará arrancar el nivel como se vio en el apartado 4.11.1 y se llamará

a esa variable cada vez que haya que introducir nuevos datos en la lógica automática.

La elección del nivel que se hace en el menú niveles se realiza gracias al método `setMap`. Cuando este método es llamado, se le proporciona el número de nivel que se quiere jugar. El método lo único que hace es trasladar a la variable nivel el número que representa dicho nivel.

Como se ha descrito con anterioridad, cuando finalizan todas las rutinas se arranca el método `update`. A este método se le proporciona el valor del tiempo transcurrido desde la última vez que fue llamado (deltaTime). Lo primero que hace este método es recoger en una lista de eventos, todos los toques y arrastres de dedos que ha realizado el usuario en el dispositivo. Tras ello, hace que se vacíen esos datos de donde los ha recogido, para que no se los vuelvan a proporcionar, y que se ponga a recoger las nuevas acciones del usuario.

Tras ello, compara el estado en el que se encuentra el juego y llama al método correspondiente, proporcionando al método llamado la lista de eventos (que puede estar vacía). Si este método es el que se arranca cuando el estado está en Funcionando, además de la lista, se le proporciona también el tiempo deltaTime.

El método `updatePreparado` es al que se llama desde `update` si el estado es Preparado. Busca mantenerse en espera hasta que el usuario toca la pantalla por lo que el método no realiza acción si al ser llamado la lista de eventos que le acompaña está vacía. En caso de no estarlo, marca comienza como verdadero y cambia su estado a Funcionando.

El método `updatePausado` hace lo mismo que el método anterior aunque en este caso no activa ningún booleano. Si la lista que le acompaña no viene vacía, cambia el estado a Funcionando.

La partida

Los métodos `maxK` y `minK`, establecidos como ayuda a `updateFuncionando` cuando se está en modo multijugador, devuelven un

Integer con el número de fila máximo y mínimo dentro de una estructura con el que el usuario puede interactuar.

El método `updateFuncionadno` busca que se reaccione a las acciones del usuario, además de hacer avanzar el mundo. Para ello extrae uno a uno los eventos de la lista que se le ha otorgado y va ejecutando una rutina con cada una hasta que la lista quede vacía. La rutina averigua si el evento se refiere a que el dedo ha tocado la pantalla o si el dedo ha sido levantado dejando de pulsarla. En caso de que se trate de arrastre, el evento es ignorado. Si la ha pulsado, saca las coordenadas donde se ha pulsado, la cuales son una característica del evento, y comprueba si se han realizado en las coordenadas que corresponderían a un engranaje con el que se pudiese interactuar. Aquí es donde se usarían los métodos `minK` y `maxK`. Si coincide con algún engranaje valido, añade la posición de dicho engranaje en unas variables y activa un booleano que hará de resorte para evitar un posible efecto de doble toque. Si lo que se ha hecho es quitar el dedo y pasa la barrera de seguridad que impide el efecto doble toque, se usa el evento para ver en qué coordenadas se ha producido y compararlo con lugares válidos para que tenga alguna influencia. Si se ha producido en el marco que correspondería al mismo engranaje donde se pulsó, se llama al método `GirarEngranaje` junto a la posición. Para saber si el jugador ha arrastrado el dedo hacia un lugar valido con el objetivo de sacar una bola tenemos los otros cuatro casos. Si se ha producido a la derecha o izquierda del engranaje y el elemento adyacente, en su derecha o izquierda respectivamente, es un elemento horizontal o tiene forma de cruz, es decir, su valor es mayor de cinco y menor de diecinueve, llamará al método `BolaDerecha` o `BolaIzquierda` junto a la posición del engranaje desde el que se pulsó el dispositivo. En caso de que se haya separado el dedo arriba o abajo y tengan elementos verticales o con forma de cruz, es decir, cuyos valores sean mayor de nueve, ejecutar `BolaArriba` o `BolaAbajo` junto a la posición del engranaje desde el que se empezó el arrastre.

Tras haber comprobado cada evento de la lista, ejecuta el método `update` de la variable mundo dándole además el deltaTime. Después, busca si ha habido algún cambio en mundo chequeando distinas variables booleanas de mundo. Si ahora está activa gameOver, cambia el estado a GameOver, si está

activa Pausa, cambia el estado a Pausado y si está activado Victoria, ejecuta el método `anadirPuntuacion` de la clase Ajustes para comprobar si la puntuación es mayor que otras veces, llama a `putData` de la clase Ajustes para guardar esa mejor puntuación y cambia el estado a Victoria.

El método `GirarEngranaje` ejecuta el método de mismo nombre pero de la clase Casilla dentro del engranaje de mundo que este en la posición que se la ha dado. Tras ello ejecuta el sonido giro si procediese y, en caso de tener una cruz de bolas en el mundo y que esté llena, arranca el método `compruebacruz` dentro de mundo con la posición dada.

Los métodos `BolaIzquierda`, `BolaDerecha`, `BolaArriba` y `BolaAbajo` arrancan el método `bolasale` dentro de mundo con la posición dada y la dirección hacia la que debe salir la bola.

Victoria o derrota

El método `updateGameOver` es el llamado cuando el estado es GameOver. Su rutina parte de coger uno a uno los eventos dela lista y comprobar si en alguno se ha separado el dedo del dispositivo dentro de un marco que en el apartado visual se representa con el símbolo *repeat*. En caso de que coincida, se cambia al estado Preparado y se ejecuta el método `NuevaPartida`.

El método `updateVictoria` realiza una rutina muy parecida a `updateGameOver` solo que tiene dos marcos donde se realiza acción en caso de que el dedo se separe en ese lugar. El primero, representado gráficamente con el símbolo *repeat* hace lo mismo que el anterior método, el nuevo, cuya representación es una flecha de avance, sólo se puede pulsar si el nivel no es el máximo nivel implementado. En caso de que no lo sea y se pulse, sumara uno al nivel actual, pondrá el estado como Preparado y arrancará el método `NuevaPartida`.

El método `NuevaPartida` reinicia la clase PantallaJuegoUnJugador llamándolo de la misma manera que se había accede a él a través de la clase CargarPantalla.

4.11.4. Interactuación de usuario multijugador

Para entender este capítulo no sólo se debe recoger la información del anterior capitulo, también se tiene que tener en cuenta lo referido en el capítulo del Bluetooth. En varios momentos se comenta que se enviará un mensaje usando el método `EnviarMensaje`. Para ver este método algo más pormenorizado o la reacción que tendrá el dispositivo emparejado al recibir el mensaje, hay que ver dicho capítulo.

Lo que se añade adicionalmente a todo lo visto, se hace en la clase Multijugador, que es heredera de PantallaJuegoUnJugador. Se añaden pocas variables nuevas, lugarbarra55 que es pareja a lugarbarra y que se utiliza en el apartado gráfico; listaMulti que recogerá la lista de los mensajes que la lógica automática pide enviar, es decir, una lista con variables tipo MensajeEnLista; primero, booleana que indica que dispositivo arranca en modo servidor y cuál en modo cliente; y cargado, una variable booleana que se activa cuando un nivel es cargado.

La clase, a la que ya se ha accedido con anterioridad en el establecimiento del emparejamiento Bluetooth, empieza cargando todo lo cargado en PantallaJuegoUnJugador, después llamará a `inicializarBT` y arrancará un nuevo hilo en `EmpezarRecibirMensajes`.

Cuando se ejecuta el método `cargarMundo`, en este caso arranca la variable mundo como un nuevo MundoBT. Además se indica a mundo que no es un jugador único y si es el servidor o el cliente. El método `update` no se varía aunque sí los métodos a los que este deriva.

El método `updatePreparado` ahora empieza chequeando si el dispositivo es el servidor y aún no ha sido cargado Si es así envía el nivel a jugar al cliente y da el nivel, o más bien, la parte de trabajo en la configuración del nivel, como cargado. Si es el cliente, le han pasado el nivel, es decir, está también cargado y el usuario toca la pantalla, enviara un mensaje al servidor con ambas colas de bolas que tenga, las bolas preparadas en los pasillos y, en caso de que haya cruz de bolas, el patrón que lo compone. Todo esto con los mensajes bolasig, bola55 y cruz. Tras ello, envía el mensaje “vamos”, da la partida como comenzada y cambia el estado a Funcionando.

En caso de que la partida se pause por alguna interrupción en el juego, en esta ocasión se envía un mensaje “pausa” antes de cambiar el estado a Pausado. El método `updatePausado` espera a que en el dispositivo que mantiene la variable booleana “comienza” como verdadera, es decir, el dispositivo que invocó la pausa, se toque la pantalla para enviar el mensaje “vamos” al otro dispositivo y cambiar su estado a Funcionando.

Para entender la primera parte del método cuando el estado es Funcionando, se debe saber cómo se han variado los métodos de apoyo ya que el método en si no cambia.

El método `GirarEngranaje` mandará ahora un mensaje para que el otro dispositivo gire el engranaje en la posición indicada. Después continuará con la misma rutina que un solo jugador.

Los métodos para soltar bolas, es decir, los métodos `BolaIzquierda`, `BolaDerecha`, `BolaArriba` y `BolaAbajo` comprueban ahora, antes de hacer nada, si las bolas activas son menos de cuatro. Si es así, busca entre los huecos que tiene designado en la lista de bolas, alguno que no contenga bolas activas. Una vez encontrado, enviará un mensaje que es específico para cada dirección, con el número de hueco dentro de la lista de bolas y la posición del engranaje que se desprenderá de la bola. Tras ello ejecuta el método `bolasale2` dentro de mundo para poner la bola en movimiento.

Además cuando se llame a `maxK` para saber cuál es la máxima fila con la que puede interactuar el dispositivo, en este caso se hará que sea la tercera si eres servidor y la última si eres cliente. Si se llama a `minK`, para la mínima fila, se hará que el servidor pueda interactuar desde la primera, y el cliente desde la tercera. Esto hará que las dos primeras filas sean exclusivas para el servidor, las dos últimas exclusivas para el cliente y la del medio sea de uso compartido.

Con todo esto, el método `updateFuncionando` lo primero que hace es ejecutar toda la rutina de PantallaJuegoUnJugador. Después iniciará una rutina distinta que comienza con el borrado la lista para los mensajes de esta clase. A ella se le añaden todos los mensajes de la lista del mundo y tras ello se borra la lista del mundo. A continuación, si la lista de la clase no está vacía, se irá

extrayendo mensaje tras mensaje para ser enviado y así las bolas reboten o sean introducidas en los engranajes en ambos dispositivos.

Tras ello, si el dispositivo es servidor, mirará si hay nuevas bolas preparadas para ser introducidas en un futuro en la cruz de bolas. Si es así, se las comunicará al cliente y dará el patrón como visto.

Lo siguiente que hace el método es mirar si es servidor y se cargó una nueva bola para el pasillo superior o si es cliente y se cargó una nueva bola para el pasillo inferior. En ambos casos se informará al nuevo dispositivo sobre el hecho y se le comunicará cuál es la nueva bola de la cola de bolas.

Para el método `updateGameOver`, la única modificación implementada es enviar un mensaje “gameover” antes de realizar la misma rutina que con un solo jugador.

En el método `updateVictoria` el servidor enviará la puntuación al cliente antes de continuar con la rutina.

En los dos últimos métodos vistos, hay variaciones cuando se llama al método `NuevaPartida`. Ahora se enviará un mensaje de reinicio junto al nivel que se quiere y se ejecutará el método `Reiniciar`.

El método `Reiniciar` se ha creado porque si se volvía a cargar la clase se perdía la conexión Bluetooth. Por ello, este método realiza todos los pasos reiniciando las variables, pasando al estado Preparado, seleccionando la música del nivel y reiniciándola y cargando llamando a `cargarMundo` para que reinicie mundo también.

4.11.5. Modelado gráfico

Todo el apartado gráfico del juego, al menos de la parte de un jugador, se implementa en la clase PantallaJuegoUnJugador. El método `present` se encargará de decidir lo que hay que pintar realizando una primera rutina para todos los estados y ejecutando después un método dependiendo del estado. En la primera rutina se carga en una variable los gráficos del juego con todas sus clases. Se ha de destacar el método de esta variable `drawPixmap` donde se le proporciona el recurso que se quiere pintar, las coordenadas x e y donde se

quiere situar, las coordenadas x e y dentro del recurso donde se quiere empezar a recortar y lo alto y ancho que se quiere recortar del recurso. De esa manera pueden “pegarse” en cualquier parte de la pantalla cualquier parte rectangular de alguno de los recursos anteriormente contados. La rutina continúa pintando el fondo con el recurso Fondo y ejecutando `PintarMundo` dándole a este método la variable mundo que contiene todos los datos del nivel. Después, dependiendo del estado ejecutará `pintarPreparado`, `pintarFuncionando`, `pintarPausado`, `pintarGameOver` o `pintarVictoria`.

`PintarMundo` es la rutina principal del apartado gráfico. Pintará la estructura de los niveles y todos los elementos activos con unos pocos métodos de apoyo.

Elementos en las casillas del nivel

Se carga en la matriz casilla cada elemento con sus características actualizadas. Tras ello se va mirando elemento por elemento del nivel. Sabiendo la posición se saben las coordenadas donde empezar a pintar cada elemento lo que ayudará a pintarlo. Dependiendo del tipo de éste, se comenzará una rutina u otra.

En caso de ser Engranaje, se dará pie a toda una serie de algoritmos para saber qué tipo de engranaje se debe pintar ya que, dependiendo de si tiene en alguna de las posiciones adyacente un elemento con el que interactuar o no, tendrá un tipo u otro de dibujo. Los algoritmos, identificaran cuál de los dieciséis engranajes le corresponde. Si pertenece a la última fila de los elementos, llamará a `Engranaje55` con el engranaje que le corresponde y el tipo de engranaje que le correspondería si el pasillo inferior estuviera activo, es decir, si estuviera en modo dos jugadores. Este método le devolverá uno de los dos tipos de engranaje introducidos. Entre medias, cada vez que se llame a un engranaje de la fila superior, modificará el pasillo para que refleje gráficamente que la bola que pase por encima de un engranaje pueda pasar a éste. También se llamará al método `pintarfila55` para que actué en caso multijugador. Tras el cálculo, dibujará el engranaje y tras ello, una a una las bolas del engranaje y, en caso de estar

vacío y estar el engranaje superado, un hueco con el color más oscurecido que si no estuviese superado.

En caso de ser pasillos, simplemente se elige el cuadrado a pintar dentro del recurso y se pinta en las coordenadas correspondientes a la posición del elemento.

Para el contador de bolas, se cuenta el número de bolas activas restando la de los pasillos. Con esa cuenta se selecciona qué momento del recurso que guarda al contador pintar, y de ese modo, el usuario tiene una referencia gráfica de cuántas bolas tiene en movimiento en cada momento.

Si es tipo siguiente bola, se pinta el elemento, el cual está vacío, y se averigua si la partida ha comenzado. Si es así se arrancará el método `pintarbola` que veremos más adelante.

Si es el reloj de arena, se calcula el porcentaje de tiempo que queda. Ese resultado sirve para seleccionar por dónde empezar a recortar de la imagen cargada en los recursos. Varía cada 10%. Conforme más tiempo pase, más arena se colará en el reloj.

Para el caso de que sea semáforo, se pinta el elemento con sus tres colores vacíos y después se van pintando sobre cada uno de los huecos los colores que correspondan. Estos colores se obtienen de mundo. En caso de que algún color haya sido superado, el mundo refleja el color como vacío, por lo que el semáforo pintará un hueco en esa posición.

En los casos de cualquiera de los teletransportes, la rutina se complica un poco. Se averigua desde qué direcciones es factible que le vaya a venir una bola, ya que no porque venga desde una dirección, no puede venir desde la opuesta. La técnica usada es parecida a la de los engranajes. Una vez sabido desde qué lados vienen, se dibuja la pintura recortando el pasillo del lado desde la que no es factible que una bola venga. Este recorte se hace desde el núcleo de la figura hasta el final del elemento. El resto del elemento se dibuja igual que cualquier otro.

Si los elementos son pinturas o filtros, se pintan seleccionando el recurso que lo contengan y la parte a partir de la cual recortar dependiendo de si son

horizontales, verticales o en forma de cruz. El color y algunas otras modificaciones se verán más adelante.

Con los direccionadores pasa igual que con los teletransportes. Se busca ver desde qué lugares puede proceder una bola mirando los elementos que le rodean. Cuando se averigua, se dibuja recortando en consecuencia de manera que solo aparezca el núcleo y los pasillos desde los que pueda venir la bola.

En caso de ser una cruz de bolas, se pinta la cruz vacía y después se pinta una a una cada bola del patrón si la partida ha comenzado y si la cruz está llena. En caso de no ser así, se dejará la cruz como vacía.

Otros elementos del nivel

Para dibujar el tiempo por bola, se calcula el lugar proporcional de que correspondería en el pasillo a la barra oscuridad. Sabiendo que el pasillo tiene una medida de 575, se usa ese dato para averiguar en qué posición debe comenzar a dibujarse la barra. Además se llama al método `pintarbarra55`, aunque dentro de este no se hace nada cuando se ejecuta en modo un jugador.

Para dibujar las bolas, se mira una a una las bolas una vez ha comenzado el juego. Si la bola esta activa se pinta usando sus características. Sus coordenadas x e y dará la posición y el color se utilizará para saber a partir de qué punto recortar en el recurso.

Tras ello lo que se hace es volver a mirar uno a uno el tipo de elemento que hay en cada casilla para pintar, si fuera necesario, el núcleo central para crear el efecto que la bola pasa por debajo de dicho núcleo. Esto se hace con los direccionadores, teletransportes, pinturas y filtros. Además en los dos últimos tipos se pintará también el color del elemento llamando a un recurso adicional y pintándolo después de pintar el núcleo.

Métodos de apoyo

El método `Engranaje55` recibe dos números que representan dos posibles engranajes. En la clase PantallaJuegoUnJugador este método devuelve

el valor del engranaje como si el último pasillo no se jugase y si se llama dentro de la clase Multijugador, el valor como si el último pasillo estuviese activo.

El método `pintarbola` dibuja en las coordenadas dadas la siguiente bola utilizando uno de los recursos. En caso de llamarse en la clase Multijugador dibuja la mitad superior de bolasiguiente y la mitad inferior de bolasiguiente55, ambas proporcionadas por el mundo. De esta manera refleja en un mismo lugar las siguientes bolas a jugar de ambos pasillos.

El método `pintarfila55` no hace nada en caso de llamarse desde la clase PantallaJuegoUnJugador y pinta la representación de un pasillo con una escapatoria hacia un engranaje en caso de llamarse desde la clase Multijugador.

El método pintarbarra55 tampoco hace nada si se llama desde la clase utilizada para un solo jugador. En caso de que sea una partida de dos jugadores, se calcula el lugar proporcional desde el que le correspondería dibujarse a la oscuridad en el pasillo inferior con respecto al tiempo que queda para extraer la bola de dicho pasillo a algún engranaje. Ese cálculo se utiliza para pintar la oscuridad.

Pintura de los estados

El método `pintarFuncionando` no hace nada para ningún modo ya que todo el juego de este estado es pintado con `pintarMundo`.

El método `pintarGameOver` pinta, para ambos modos, un jugador y multijugador, un símbolo que representa un botón para la repetición del nivel y un cartel que informa de la derrota.

El método `pintarVictoria` pinta, para ambos modos, un símbolo que representa un botón para la repetición de nivel, otro para jugar el nivel siguiente y un cartel que informa de la victoria.

El método `pintarPausado` pinta un cartel que indica que se toque la pantalla para empezar para el modo un jugador y para el modo multijugador, es decir, llamado desde la clase Multijugador, si es el dispositivo que ha pausado la partida. En caso de que sea el otro dispositivo el que ha pausado la partida, dibuja un cartel indicando que espere.

El método `pintarPreparado` dibuja un cartel comunicando que está todo preparado y que presione una tecla si se llama desde PantallaJuegoUnJugador. Si se llama desde la clase Multijugador, comprueba que es el cliente y tiene el nivel, entonces dibuja el mismo cartel. Si es el servidor, indica que debe esperar a que el otro dispositivo toque la pantalla y si es el cliente pero no ha recibido el nivel, dibuja un cartel en el que pide esperar a que seleccione el nivel el servidor.

Capítulo 5. Plan de pruebas

En este capítulo se detallarán las comprobaciones realizadas a la aplicación desarrollada para verificar su correcto funcionamiento. Para efectuar dichas comprobaciones se ha tomado un conjunto significativo de muestras, abarcando las pruebas desde el emulador y las realizadas con móviles de distintos fabricantes, hasta el reclutamiento de varios testeadores para que realizasen partidas y así notificar posibles errores a reparar.

5.1. Pruebas con emuladores

Durante el desarrollo de la aplicación se ha empleado el emulador del Nexus One que se crea con el programa Eclipse, para probar las versiones iniciales, antes de pasar a ejecutar el programa sobre dispositivos reales. Se ha utilizado principalmente para comprobar la lógica del juego desde los primeros pasos hasta un resultado satisfactorio de lo que podría entenderse como una partida de un solo jugador.

El emulador no permite la simulación de comunicación Bluetooth por lo que, una vez comprobado el funcionamiento de varios parámetros que comprendían el comienzo de la aplicación, la interactuación del usuario con el dispositivo en una partida y la lógica automática, se pasó a las pruebas con dispositivos reales.

La versión de la aplicación que se utilizaba en esta fase ni siquiera está registrada ya que era una versión muy precaria a lo que podría ser el resultado final. Los menús se establecían de manera gráfica con las mismas técnicas o clases que el juego. Aunque se visualizaban cuatro apartados, sólo estaba activo

“un jugador” y la habilitación o deshabilitación del sonido, lo que llevaba a que el único sonido implementado sonase o no. En un jugador sólo se disponía de un nivel donde no se trabajaba aún con el tiempo, por lo que no había manera de perder.

Con todo ello, se pudo comprobar que:

- El arranque de la aplicación era satisfactorio
- Las acciones del usuario tenían repercusión en la aplicación
- La aplicación podía pasar de una pantalla a otra
- La aplicación no realizaba acción alguna si el usuario pulsaba fuera del área designada para que se realice una acción
- El nivel se cargaba correctamente
- Los elementos se dibujaban en las posiciones indicadas
- La partida permanecía en estado Preparado hasta que el usuario pulsa la pantalla
- La bola tenía un color aleatorio
- La bola se movía de forma fluida
- El contador de bolas reflejaba las bolas a contar
- Los engranajes actuaban de forma correcta cuando se giraban
- Las bolas eran sacadas correctamente de los engranajes
- Las bolas se establecían correctamente en los engranajes seleccionados
- Las bolas rebotaban correctamente
- No se podían sacar bolas cuando el límite de bolas activas era superado.
- Los parámetros de victoria se activaban correctamente
- El efecto sonoro sonaba en el momento correcto cuando estaba habilitado
- El efecto sonoro no sonaba cuando estaba deshabilitado
- El engranaje superado hace desaparecer las bolas
- El engranaje que ya ha sido superado cambia el color de sus huecos

5.2. Pruebas con dispositivos reales

Una vez se empezó a desarrollar la conectividad Bluetooth dentro de la aplicación, las pruebas con emuladores no eran posibles, por lo que se procedió a hacer pruebas con dispositivos reales. Para ello se usaron un dispositivo móvil Huawei Ascend P6 y un Samsung Galaxy S2. La dinámica de las pruebas incluía los reportes de fallos de los testeadores, intentando reproducirlos en los terminales, repararlos, comprobar dicha reparación en los terminales y pedir la confirmación al testeador que se encontró el error que ya no volvía a tenerlo.

Las comprobaciones que a continuación se relatan han sido realizadas en distintas versiones, aunque es con la última versión con la que se puede asegurar que todas las comprobaciones son positivas.

Comprobaciones de arranque y menús:

- El arranque de la nueva aplicación es correcto
- La pantalla Splash se inicia con normalidad
- El tiempo, la imagen y el zoom de la pantalla Splash es la especificada
- De la pantalla Splash se pasa al menú inicio correctamente
- El menú inicio tiene la imagen de fondo deseada
- La distribución del menú es el deseado
- Se usa correctamente la tipografía incluida
- Los textos configurados como pulsables se pueden pulsar
- Al pulsar un texto, cambia de color
- Pulsar un jugador lleva al menú niveles
- Pulsar dos jugadores lleva a la rutina multijugador
- Pulsar opciones lleva al menú opciones
- Pulsar cómo jugar lleva al menú cómo jugar
- Cómo jugar muestra la imagen indicada
- Pulsar continuar en cómo jugar lleva al menú inicio
- Presionar la tecla volver en cómo jugar lleva al menú inicio
- Pulsar actualizar lleva al actualizador

- Pulsar sobre nosotros hace aparecer un dialogo con el texto especificado
- El dialogo de sobre nosotros se cierra con el botón cerrar
- El dialogo de sobre nosotros se cierra si se presiona el botón volver
- En el dialogo de sobre nosotros, si se pulsa sobre la página personal, se carga el navegador con esa dirección web.
- Pulsar salir abre una pregunta de confirmación con dos opciones
- Presionar el botón volver en el menú inicio arranca la pregunta de confirmación con dos opciones
- Si se pulsa el botón salir de la confirmación de salida, la aplicación se cierra
- Si se pulsa el botón cancelar de la confirmación de salida, se cierra el dialogo de la confirmación de salida

Comprobaciones del menú opciones:

- El botón de música se carga correctamente con la selección guardada
- El botón de sonido se carga correctamente con la selección guardada
- Los botones sonido y música cambian cuando son pulsados
- La barra de volumen se carga con el volumen guardado
- La barra de volumen registra cambio cuando se desplaza la marca
- La casilla del nombre del jugador se carga correctamente con el nombre guardado.
- En caso de no haber cambiado el nombre del jugador, la casilla se carga con el nombre Jugador
- Pulsar el botón cambiar al jugador informa del cambio de jugador y se cargan los ajustes del jugador guardado.
- En caso de cambiar a un jugador nuevo, los ajustes que aparecen son los de por defecto
- En caso de cambiar al jugador “Miyamoto” informa que se establece el modo trampas y vuelve a poner el nombre del antiguo jugador.

- Pulsar sobre salvar hace volver al menú inicio informando de que se han guardado los cambios.
- Pulsar el botón volver arranca el menú inicio

Comprobaciones del actualizador:

- El actualizador se carga con el botón obtener datos
- Pulsar para obtener datos con la última actualización instalada da como resultado el mensaje la aplicación está actualizada.
- Pulsar para obtener datos sin la última actualización instalada informa de la versión instalada, la versión más actual y hace aparecer un botón para descargar la nueva versión
- Pulsar el botón para bajar la nueva versión hace que se descargue el archivo LogicalBT.apk con la nueva versión al dispositivo
- Pulsar para obtener datos sin conexión a internet informa de error en la conexión
- Pulsar el botón volver arranca el menú inicio

Comprobaciones del menú niveles:

- Se carga correctamente el menú niveles
- Se pueden visualizar 50 elementos, uno para cada nivel
- Los niveles superados reflejan las estrellas obtenidas en la partida
- El primer nivel que no se ha superado no tiene estrellas ni candado
- Los niveles no superado, excepto el primero de ellos, tienen un candado
- No se puede jugar a los niveles con candados y así es notificado
- Con el modo trampas activo, no hay candados y se puede jugar a cualquier nivel
- Si se han superado todos los niveles suena la melodía “eres un fenómeno”
- Las estrellas y niveles pasados se actualizan tras volver de una partida
- Al ir a un nivel habilitado arranca una partida
- Al pulsar el botón volver arranca el menú inicio

Comprobaciones del juego:

- Todas las comprobaciones realizadas en el motor del juego con el emulador siguen funcionando
- Las estructuras de cada uno de los 50 niveles son las correctas
- Los dibujos de los elementos de las estructuras creados, incluidos sus recortes, son los correctos.
- Las mejoras gráficas introducidas en la versión 1.05 son efectivas
- La partida pasa al estado Pausa si es interrumpida
- El estado Pausa pasa al estado Funcionando si se presiona la pantalla
- En el estado Preparado no aparecen bolas ni colores en el contador o en la bola de cruces.
- El elemento siguiente bola refleja la bola que aparecerá a continuación.
- Los elementos teletransporte transporta la bola a otra localización
- Los elementos direccionadores cambian la dirección de la bola
- Los elementos pintura cambian el color de la bola
- Los elementos filtro restringen el paso de las bolas que no tengan el color indicado
- El elementos semáforo hace que los engranajes no se completen hasta no resolverse los colores
- Tras resolverse un color del semáforo, se busca si alguno de los engranajes restantes tiene un engranaje que se pueda dar por completado con la nueva restricción
- El elemento cruz de bolas hace que los engranajes no se completen hasta resolver el patrón
- Tras resolverse la cruz de bolas se busca si alguno de los engranajes restantes tiene un engranaje que se pueda dar por completado con la nueva restricción
- El elemento cruz de bolas tiene patrones aleatorios
- El elemento cruz de bolas crea un nuevo patrón que aparece 90 segundos después de haberse superado el antiguo patrón

- Si coinciden la cruz de bolas con el semáforo en una partida, se impone la restricción de la cruz de bolas
- El elemento reloj de arena refleja el tiempo que queda de la partida
- Si se supera el tiempo de partida, el estado cambia a GameOver
- En el pasillo superior se refleja el tiempo por bola transcurrido con siendo oscurecido
- Si se supera el tiempo por bola limite el estado cambia a GameOver
- El tiempo por bola se reinicia cada vez que una bola del pasillo entra en un engranaje
- El tiempo por bola es reducido en los niveles 19 y 48
- Si se superan todos los engranajes en cualquiera de los niveles, se pasa al estado Victoria
- La pantalla Victoria refleja la puntuación y las estrellas correctas
- La puntuación es guardada
- Si se pulsa el botón de reinicio en la pantalla Victoria se reinicia el nivel y se cambia a estado Preparado
- Si se pulsa el botón de avanzar en la pantalla Victoria se avanza un nivel y se cambia a estado Preparado
- Si se pulsa el botón de reinicio en la pantalla GameOver se reinicia el nivel y se cambia a estado Preparado
- Las pantallas en estado Preparado, Victoria, GameOver o Pausado se muestran correctamente
- Al pulsar el botón volver arranca el menú niveles
- Es capaz de superar pruebas de stress con satisfacción

Comprobaciones de conectividad multijugador:

- Arranca la posibilidad de conectar el Bluetooth si no lo tiene disponible
- En caso de declinar conectar el Bluetooth se cierra la ventana
- En caso de aceptar conectar el Bluetooth informa de que tiene una visibilidad de 120 segundos y continúa la rutina multijugador
- Si se tiene conectado el Bluetooth pregunta si se desea ser servidor o cliente

- Si se selecciona servidor se espera a que un cliente se conecte
- Si se selecciona cliente se abre la ventana para buscar dispositivos a los que emparejarse o seleccionar dispositivo anteriormente emparejado
- Si se buscan dispositivos se encuentran dispositivos con Bluetooth activado
- Si se selecciona dispositivo emparejado anteriormente que esté en modo servidor esperando, se establece emparejamiento
- Si se selecciona un dispositivo encontrado que esté en modo servidor esperando, se establece emparejamiento
- En caso de establecer emparejamiento, el dispositivo en modo cliente pasa al juego con el estado de Esperando
- En caso de establecer emparejamiento el dispositivo en modo servidor para al menú niveles de dos jugadores
- El menú niveles de dos jugadores funciona correctamente de manera similar al menú niveles
- El menú niveles de dos jugadores refleja las estrellas conseguidas en modo multijugador
- Si en cualquier momento se pulsa el botón volver, se cierran las ventanas emergentes

Comprobaciones del juego multijugador:

- El dispositivo en modo cliente arranca en estado Esperando con la pantalla Esperando2 y no puede hacer nada hasta que el dispositivo modo servidor elija nivel.
- La pantalla Esperando2 no refleja elemento alguno
- El dispositivo en modo servidor arranca en estado Esperando con los elementos del nivel cargados y se mantiene a la espera de que el dispositivo en modo cliente esté listo
- El dispositivo en modo cliente carga el nivel una vez lo ha seleccionado el servidor
- El dispositivo en modo cliente es quien inicia el nivel tras haber cargado los elementos

- Las colas de bolas y la cruz de bolas reflejan los mismos colores y patrones en todo momento en ambos dispositivos.
- El pasillo inferior entra en juego
- El tiempo de bola del pasillo inferior refleja bien el tiempo transcurrido
- Cada dispositivo solo puede usar los engranajes asignados
- El elemento siguiente bola refleja bien ambas bolas siguientes
- Ambos dispositivos pasan al estado GameOver a la vez
- Ambos dispositivos pasan al estado Victoria a la vez
- Ambos dispositivos repiten nivel o avanzan nivel en cuanto uno de los dos pulsa el botón para realizar esa acción
- Si un dispositivo interrumpe el juego, ambos dispositivos se pausan hasta que el primer dispositivo vuelva y confirme que está listo
- Las partidas transcurren con normalidad salvo si hay perdida de paquetes de datos
- En caso de que un dispositivo cierre conexión, el otro dispositivo interrumpe partida.

Comprobaciones de sonido:

- En caso de sonido habilitado suenan los sonidos y en caso de deshabilitado no.
- En caso de música habilitada suena la música y en caso de deshabilitada no.
- La música que se carga en cada nivel es la que le corresponde.
- La música que se carga en la pantalla Esperando 2 del modo multijugador es la que le corresponde
- Todos los sonidos suenan cuando la acción se produce
- Pasar al estado GameOver o Victoria interrumpe la música y hace sonar el sonido correspondiente
- Si se reinicia el nivel también se reinicia la música
- SI se avanza el nivel comienza la música correspondiente desde el principio.
- En caso de coincidir varios sonidos, suenan a la vez

5.3. Testeadores

Los testeadores han sido una parte fundamental para la evolución del proyecto. Su labor no sólo se ha limitado a comprobar que las partidas transcurrían de una manera correcta en sus dispositivos móviles, sino que también se les ha pedido que hicieran cosas aparentemente ilógicas para así poder encontrar fallos algo escondidos del diseño o del código. El reporte de las evoluciones con el juego y de los fallos, sugerencias e impresiones ha sido siempre fluido y constante.

Tal fue el impacto de los testeadores en el proyecto que se metió la sección Actualizar dentro del proyecto para que pudiesen obtener de manera simple la última versión y así hacer reportes de la versión de la aplicación más actualizada.

A continuación se listan los testeadores que más tiempo y recursos han dedicado a probar la aplicación y los dispositivos desde los que lo han realizado:

- Marta López Ramírez
- Adela Isabel Fernández Anta
- Francisco Javier González Cañete
- Antonio López de la Casa
- María López Ramírez
- Antonio Andújar Ortuño
- Elisa de la Torre Ruiz
- Elisa Andújar de la Torre
- Llanos Andújar de la Torre
- Rafael Andújar de la Torre
- Rubén Montes Serna
- Julián Enrique Ramón Vigo
- Pablo Navas Romero
- Lidia Ortega Rodríguez
- José Illán Hernández
- María Órpez Zafra
- Eusebio Gallardo Estrella

- Rosa Frías González
- Fermín Lozano Rodríguez
- Sonia Batista Lozano
- Alberto Rivas Ruiz
- Noemi Salvador Rojo
- Abel Bermúdez Román

Los dispositivos donde se ha probado la aplicación han sido:

- Samsung Galaxy Nexus
- Samsung Galaxy S2
- Samsung Galaxy S3
- Samsung Galaxy S4 Mini
- Samsung Galaxy S4
- Samsung Galaxy Tab 10.1
- Samsung Galaxy Tab 3 Lite
- Samsung Galaxy Trend Plus GT 57580
- Samsung Galaxy Note 10.1
- Samsung Ace S5830
- Asus Transformer Pad 10.1
- Sony Xperia J
- Huawei Ascend P6
- Huawei Ascend 330
- Miui Red Rice
- LG Nexus 4
- LG Nexus 5
- Wiko Darknight
- THL w200s
- Zopo C2

5.4. Evolución de las versiones

Versión 1.00

- Esta versión se utilizó para las pruebas de conectividad y de lógica interna sin prestar mucha atención a la apariencia final.

Versión 1.01

- Primera versión para presentar en la que hay tres niveles para jugar, solo un par de sonidos implementados y donde el menú inicial ya está desarrollado y consta de los apartados “un jugador”, “dos jugadores”, “opciones”, “sobre nosotros” y “salir”.

Versión 1.02

- Tiene 16 niveles.
- Nuevos iconos para los niveles.
- Activado y confeccionado "cómo jugar".
- Link a página personal en "sobre nosotros".
- Incluidas 4 canciones. Se cambian cada 4 niveles.
- Nuevo icono de aplicación.
- Las puntuaciones son guardadas, y en caso de superar una fase con 3 estrellas, el icono en el menú niveles se modifica.
- Implementados todos los objetos excepto la cruz de bolas.

Versión 1.03

- Reparado fallo de la versión anterior por el cual no todos los dispositivos podían ver el "cómo jugar".

Versión 1.04

- Reparado error por el que si se jugaba una partida a una pantalla y se terminaba, la siguiente empezaba sin música.
- Reparado error en el que si se deshabilitaba la música en las opciones, se empezaba a jugar y se le daba a salir de la partida (botón hacia atrás), la aplicación fallaba.
- En esta versión ya hay 1 y 2 estrellas. Las estrellas se dan dependiendo de los puntos: más de 1000 puntos, 3 estrellas; más

de 500, dos estrellas... Los puntos se dan con el porcentaje del tiempo restante con el total *1000 (Máximo 1000 puntos) + 1000 puntos - bolas que te queden en los engranajes *100. El mismo sistema que el juego original.

Versión 1.05

- Arreglados los problemas de sonido.
- Arreglados problemas en modo individual que interrumpían el programa.
- Mejora gráfica.
- Añadido sonido de rebote y modificado sonido de engranaje completado.
- Modificación de la lista de bolas siguientes, haciéndola ahora doble, una para cada salida, con su adaptación gráfica.
- Metido actualizador: comprueba si se usa la última versión del juego, y si no es así, se dará la posibilidad de bajarlo de una página web.
- Añadido pregunta de comprobación al salir y al pulsar la tecla "volver" en el menú principal.

Versión 1.06

- 14 niveles más, lo que ha requerido la implementación de la cruz de bolas y la implementación de prioridades cuando se juntan la cruz de bolas y el semáforo.
- Arreglado el problema del sonido quitado. La reparación del anterior no terminó de guardarse.
- Arreglado el sistema de puntuación, que daba menos puntos cuando en el nivel aparecían ciertos elementos.

Versión 1.07

- Añadidos hasta 50 niveles
- Implementada el teletransporte en cruz.
- Reparado el error que causaba al soltar bola hacia abajo en los engranajes de la última fila.

- Reparado el nivel 24
- La música cambia cada 4 niveles.
- Añadida la restricción de jugar más de un nivel que no se ha pasado.
- Añadida la posibilidad de cambiar jugador en opciones. Al cambiar el nombre, se reinician los niveles y estrellas. Si se vuelve al nombre anterior, vuelven los antiguos niveles y estrellas.
- Si se pone “Miyamoto” se guarda el modo trampas en el jugador y se podrá acceder a todos los niveles.
- Añadidos iconos hasta el nivel 50.
- Implementada nueva forma de poner iconos. Antes se hacían 4 iconos para cada nivel (con 0, 1 2 y 3 estrellas). Se ha implementado una nueva manera de superponer estrellas y candados al icono en sí. Se ahorra un espacio considerable.
- Añadido título en Opciones.
- Reparado Actualizar
- Implementada la cruz de bolas en dos jugadores
- Reparado el contador de bolas y la limitación que reflejaba en modos un jugador y multijugador.
- Implementado un refresco del menú niveles cuando se vuelve del juego. Así aparece con las estrellas actualizadas.
- Modificado completamente, aunque manteniéndose a la espera de pruebas intensivas, la manera de interactuar el modo multijugador, pues no soportaba un test de stress... las bolas se mueven a veces a velocidades distintas, eso hacía caber la posibilidad de que un engranaje cambiase una vez pasada una bola mientras que en otro móvil entrase. Se ha modificado completamente haciendo que el motor del juego solo interactúe con determinados engranajes si eres servidor o cliente y envíe un mensaje Bluetooth cuando algún evento pase. Todo pensado también para que los errores de sincronización no se noten al estar el jugador fijándose normalmente en su parte de la pantalla.

Versión 1.08

- Reparado el fallo en el que en Multijugador presentaba color de bolas distintas a veces.
- Reparado el fallo en el que, tras victoria, si se empezaba otro nivel con música distinta, la del nivel anterior seguía sonando (aunque en un volumen inferior).
- Reparado el fallo en el que si se pasaban los 50 niveles, el juego te impedía jugar a otro nivel que no sea el 1.
- Cambiado el tiempo por bola del nivel 48 al que debía de haberse modificado desde un principio.
- Reparado fallo en el modo multijugador donde la partida empezaba sin problemas aunque los dos jugadores eligiesen niveles diferentes. Luego era injugable porque aparecían cosas incoherentes en pantalla. Este fallo ha sido solucionado, aunque se cambió la manera de acceder al multijugador. Tras esta versión el dispositivo modo cliente se queda esperando a que el dispositivo modo servidor elija nivel.
- Sonidos sustituidos por los originales o añadidos.

Versión 1.09

- En caso de fallar la conexión a Internet lo notifica al intentar conseguir datos en actualizar.
- Cambiado su nombre en los créditos a Francisco Javier.
- Al usar Miyamoto, el usuario accede al modo trampas hasta que se cambie el nombre. Si se cambia el nombre de usuario se vuelve al modo normal y si se restablece el nombre del usuario anterior (con el que se pudo acceder al modo trampas), el modo trampas estará desconectado.
- Se han vuelto a poner los sonidos Victorius y Game Over.
- Se ha reparado el mal funcionamiento de la música tras repetir nivel o tener un nivel con la misma melodía. Además se ha implementado la función reiniciar para que la música sea reiniciada al empezar un nivel.

- En modo dos jugadores, se ha añadido una melodía mientras el jugador modo cliente espera a que el jugador modo servidor elija nivel.

Versión 1.10

- Sincronización de la puntuación obtenida pues variaba en unos pocos puntos entre un dispositivo y otro.
- Las puntuaciones, y por tanto, estrellas y nivel máximo conseguido entre los apartados multijugador y de un jugador serán distintos a partir de esta versión.
- Reparado error por el que el sonido GameOver no sonaba en el dispositivo modo cliente si perdía el servidor.
- Reparado error por el que no se ejecutaba el sonido si el otro dispositivo giraba un engranaje.
- Implementada nueva manera de listar las bolas, haciendo la lista más grande donde el dispositivo servidor podrá introducir bolas en una parte de ésta y el dispositivo cliente en otra parte distinta.

Versión 1.11

- Sustituidos varios "*bucles for*" por algoritmos simples.

Versión 1.12

- Reparado error que daba al denegar la conexión del Bluetooth
- Reparado fallo donde el dispositivo en modo cliente guardaba las puntuaciones obtenidas en modo dos jugadores como en el archivo de modo un jugador.

5.5. Análisis de resultado

Las pruebas realizadas en la última versión muestran una ejecución perfecta en el modo un jugador. Se han probado y superado todos los niveles tanto en las pruebas como por algunos testeadores. Las partidas son fluidas y no se han detectado errores o fallos.

La partida multijugador sin embargo presenta algunos inconvenientes. Las correcciones en el desfase a veces son visualmente apreciables aunque no

son significativas. Los testeadores que no están pendiente de ello no lo han llegado a apreciar ya que cada dispositivo realiza esas correcciones en la parte que no controla el usuario.

Tras varias horas de partidas, en un momento se ha producido un paquete de datos que tenía la labor de hacer rebotar una bola. Al no recibir ese rebote, en un dispositivo la bola sale de la estructura provocando error y que se anule la partida. Este problema no se ha vuelto a reproducir a pesar de intentarlo y llevar el dispositivo a un modo de stress.

Capítulo 6. Conclusiones y trabajo futuro

6.1. Conclusiones

El videojuego multijugador desarrollado en el presente proyecto se basa en el original “Log!cal” para un solo jugador, desarrollado por Rainbow Arts. En particular se trata de una adaptación para dispositivos móviles, tomando como referencia las características gráficas y sonoras y el funcionamiento general del videojuego (manejo, giros, desplazamientos, tiempo…). No obstante, el carácter multijugador de la versión implementada abre una nueva posibilidad en lo que a modo de juego se refiere. Concretamente se han desarrollado dos posibles modos de juego, uno para un solo jugador y otro para dos jugadores de forma cooperativa que, si bien introduce pequeñas variantes con respecto a la funcionalidad del videojuego original, respeta el planteamiento general y la idea de juego del mismo conservando el modo de manejar todos los entramados. En ambos modos de juego se han desarrollado 50 niveles de los 99 que dispone el juego original, implementando cada uno de los elementos del juego.

Cada modo de juego mencionado sirve para que el jugador en modo individual o los jugadores en modo multijugador intenten resolver un nivel seleccionado de los cincuenta disponibles, cada uno de ellos con su propia distribución de los elementos que lo componen. La aplicación almacenará la mejor puntuación que alguna vez se haya registrado en cada nivel y en cada modo y será representada con un número de estrellas a la hora de presentar los niveles.

Cada elemento implementado dentro del nivel ha sido inspirado por el videojuego original, al igual que la estructura del mapa. En la versión

multijugador se ha desarrollado la inclusión de un pasillo inferior con nuevas bolas y con una disponibilidad parcial a la hora de interactuar con los engranajes pero complementaria entre los dos dispositivos para que los jugadores deban jugar de forma cooperativa. Los elementos que componen los niveles del modo de juego son los engranajes, pasillos, contadores de bola, tiempo por bola, reloj de arena, semáforo, cruz de bolas, siguiente bola, filtros, pinturas, direccionadores y teletransportes. En definitiva, se han implementado todos los elementos que venían en el videojuego original.

Los menús, ventanas y los diferentes mensajes de texto antes de la partida, se muestran con un estilo de representación más acorde con las aplicaciones actuales que las que disponía el videojuego original, dando también más opciones y buscando una manera de interactuación más dinámica.

La aplicación dispone de una serie de efectos de sonido y melodías que sonarán en el terminal si se habilitan las opciones de sonido y música. Esta activación se puede hacer mediante el menú de opciones. Los efectos de sonido reproducen fielmente el apartado sonoro del videojuego original. Las melodías también son las compuestas en el videojuego original. Además se dispone de la posibilidad de cambiar de jugador para que las puntuaciones sean guardadas de manera distinta para los distintos usuarios de un mismo dispositivo.

A la hora de elaborar el código de la aplicación se ha empleado un lenguaje de alto nivel, en concreto Java para Android. El hecho de programar la aplicación mediante un lenguaje de estas características facilita la tarea de confeccionar el código, ya que éste se compone de instrucciones más fáciles de entender por el ser humano, pero como contrapartida, se aleja del lenguaje máquina con los inconvenientes que eso lleva. Aun así, como es lo establecido por Google para la creación de aplicaciones para dispositivos Android, la elección de escoger lo establecido hace más fácil la inclusión de la aplicación por parte de usuarios no especializados.

La realización de este proyecto mediante dicho lenguaje ha permitido conocer y profundizar sobre técnicas de programación propias de lenguajes concebidos para dispositivos con capacidades limitadas y presentación en pantallas con tamaño variable, como es el caso de los dispositivos móviles y las tabletas.

Por otro lado, se ha utilizado un protocolo de comunicación a través del enlace Bluetooth que satisficiera las necesidades de comunicación entre ambos terminales, en concreto, el protocolo RFCOMM. Para poder mantener la coherencia en la ejecución del juego en los dos dispositivos, se ha requerido desarrollar un mecanismo de sincronización sobre este enlace. Dicho mecanismo, además, se ha implementado en aras de la interactividad del juego, de forma que interfiera lo menos posible con la ejecución del mismo.

La metodología empleada para desarrollar este proyecto ha consistido en implementar sucesivas versiones que fueran incluyendo progresivamente complejidad y nuevos elementos al desarrollo, considerando desde el primer momento el modo multijugador y la correspondiente necesidad asociada de estar en comunicación con el dispositivo remoto.

Se ha partido desde el modelo más básico, consistente en un pasillo superior y cuatro engranajes conectados por pasillo. Únicamente implementando como elemento extra el contador de bolas y la limitación que representa. Se añadió el transcurrir de la bola por el pasillo superior y su interacción con los elementos mencionados. Tras ello la aleatoriedad del color de la bola, la condición de victoria y el estado de Preparado.

Tras ello se comenzó la creación de los menús que llevarán a jugar la partida y la manera de establecer un emparejamiento entre dispositivos. Se crearon hasta tres niveles con los mismos elementos y se introdujo el modo multijugador con las variantes características de éste. Se comenzó a medir el tiempo por bola y el tiempo total y con ello las condiciones de derrota y los elementos que reflejaban dicho tiempo.

Se comenzó a definir los mensajes que se intercambiarían los dispositivos y las rutinas a realizar una vez recibidos. También se definió el estado Pausado y la guarda y carga de opciones y puntos.

Los resultados de las siguientes acciones que se realizaron van recogidas en las sucesivas versiones en el apartado 5.4 y además siempre se buscó las mejoras, tanto visual como funcionalmente, de las características que ya se habían realizado.

6.2. Líneas futuras

Durante el desarrollo de este proyecto se ha puesto de manifiesto que en la elaboración de un videojuego, por sencillo que sea, se pueden considerar distintas alternativas y variantes, según qué elementos se desee tener en cuenta y añadir finalmente, de manera que deja la puerta abierta a multitud de posibilidades y nuevas características que el programador debe decidir si incluirlas o no, en función de su complejidad o de si se alejan o no del concepto o la esencia del propio videojuego.

Además, la propia naturaleza multijugador del videojuego implementado permite jugar con nuevas posibilidades y nuevos modos de juego. Además del juego cooperativo, se podrían haber definidos otros modos de juego que aprovecharan igualmente esta particularidad, como, por ejemplo, un modo en el que ambos jugadores se enfrentasen, donde resolver un engranaje haría que una nueva bola se lanzase por el pasillo del contrario, pudiéndose además enviar bolas al contrario y obteniendo la victoria si el contrario llega al límite del tiempo por bola. Otra posibilidad era que cada usuario cree un modelo de mapa, que se envíe al contrario y ver quien lo resuelve antes. Esto además se podría hacer de forma asíncrona, lo que habría evitado los desajustes que a veces suceden en una partida síncrona.

Como posibles mejoras del videojuego implementado, se incluirían la realización de los 99 niveles del videojuego original y de algunos propios. El código esta realizado para que la inclusión de nuevos niveles sea lo más sencilla posible. Sólo habría que establecer los elementos, cambiar el número de la variable que refleja el último nivel implementado y añadir iconos a la rejilla que muestra los niveles.

También dista del videojuego original un par de animaciones, una especie de explosión al completarse un engranaje y el conteo de arena en el elemento reloj de arena. La realización de éstas no es compleja con todo lo ya realizado. La animación de la explosión arrancaría cuando se completase el engranaje y conforme pase el tiempo iría variando. Para el reloj de arena, se debería hacer una animación que variase de manera constante con un periodo de tiempo muy pequeño. Aunque simplemente fuesen pixeles bajando hacia abajo en una

columna dentro del reloj de arena, la ilusión del juego original da cierto dinamismo a la partida.

Algo que también beneficiaría al juego sería la inclusión de un menú a la izquierda de la pantalla mientras transcurre la partida, que de la posibilidad directamente de pausar la partida, reiniciar nivel o salir al menú de niveles. Habría que desplazar todos los elementos y crear una pantalla mayor, pero merecería la pena ya que el usar la tecla volver da la sensación de un elemento algo artificial en un juego puramente táctil.

Otra línea a mejorar sería la salida de una partida multijugador. Actualmente si se da a la tecla volver, se vuelve al menú mientras el otro jugador está en pausa, y este no sale hasta que se rompe el enlace creado para esa partida. Eso se refleja en el dispositivo como una pérdida de conexión. Modificar la manera de interrumpir la partida para que se envíe un mensaje antes de desconectarse y cuidar el aspecto con el que los dispositivos muestran esa desconexión es algo factible a mejorar.

Por otra parte, también se podría permitir al jugador la posibilidad de desarrollar su propio nivel con la ayuda de un editor de niveles. Esta opción alargaría la vida útil del producto, si bien ahora cada jugador dispondría de cincuenta de los mapas originales, los que cree él mismo y los que haya podido confeccionar cualquier otro jugador con el que desee iniciar una partida.

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
- Jesús Tomás, Vicente Carbonell, Carsten Vogr, Miguel García, Jordi Bataller, Daniel Ferri, “El gran libro de Android Avanzado”, Marcombo, 2014
- 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 Septiembre de 2014

[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] Informe de la consultora PWC sobre la industria del entretenimiento http://www.pwc.es/es/publicaciones/entretenimiento-y-medios/assets/semo-2013-2017.pdf

[3] Página de la AEVI (Asociación española de videojuegos) http://www.aevi.org.es/index.php

[4] Página en la Wikipedia sobre los jugadores de videojuegos http://es.wikipedia.org/wiki/Jugador_de_videojuegos

[5] Neomobile es el Grupo de Comercio Móvil que permite a las empresas digitales de monetizar sus productos y servicios. En el blog de la empresa dan consejos a los creadores de aplicaciones http://www.neomobile-blog.com/es/como-crear-videojuego-casual/

[6] Revista Micromanía, Nº 38 Segunda época, Julio 1991.

[7] Página en la Wikipedia sobre el Tetris http://es.wikipedia.org/wiki/Tetris

[8] Página sobre referencias en prensa del videojuego Log!cal en Hall of Light, una amplia base sobre videojuegos de la videoconsola Amiga http://hol.abime.net/906/review

[9] Referencia en la base de videojuegos gamesdbase sobre la versión para GameBoy Color del videojuego Log!cal http://gamesdbase.com/game/nintendo-game-boy-color/logical.aspx

[10] Página sobre Psychoball dentro del portal de su desarrollador, Intermediaware. http://www.intermediaware.com/games/psychoballs

[11] Página oficial para los desarrolladores de Android http://developer.android.com/index.html

[12] Capítulo 1 del libro de Jeff Friesen “Java para desarrollo Android”, Anaya Multimedia,2011

[13] Página oficial del software Eclipse https://www.eclipse.org/home/index.php

[14] Comunidad de Libgdx, quienes desarrollan un *framework* para desarrolladores de videojuegos en multiplaforma sobre dispositivos móviles. http://libgdx.badlogicgames.com/

[15] Sitio web del motor AndEngine http://www.andengine.org/

[16] Página oficial de AOSP http://source.android.com/

[17] Tutorial de actualizador de AndroCode http://androcode.es/2012/03/aplicacion-auto-actualizable-sin-market/

[18] Página en la Wikipedia sobre Bluetooth http://es.wikipedia.org/wiki/Bluetooth

[19] Capítulo 4 del libro de Jesús Tomás, Vicente Carbonell, Carsten Vogr, Miguel García, Jordi Bataller, Daniel Ferri, “El gran libro de Android Avanzado”, Marcombo, 2014

[20] Página en la Wikipedia sobre Adobe Photoshop http://es.wikipedia.org/wiki/Adobe_Photoshop

[21] Página oficial de WinUAE http://www.winuae.net/

[22] Plugin desarrollado por AntiXChat 2, comunidad checa de aficionados a videojuegos antiguos http://ax2.old-cans.com/tfmx/

[23] Cortador de audio online http://mp3cut.net/es/

[24] Página en la Wikipedia sobre Ogg http://es.wikipedia.org/wiki/Ogg

[25] Tutorial en Juegomanía, portal de videojuegos que emplea especial atención a los juegos antiguos, para ejecutar el DosBox http://www.juegomania.org/syt/tutorial.htm

[26] Anuncio de Gameboy de los años 90 alojado en youtube https://www.youtube.com/watch?v=QbV6CospXv4

[27] Página en la Wikipedia sobre Shigure Miyamoto http://es.wikipedia.org/wiki/Shigeru_Miyamoto