

Videojuego 3D para Android: “3D Logic”

Memoria del proyecto

Autor: Antonio Jesús Ríos Guerrero

Fecha: 10/01/2012

Director: Lluís Pérez Vidal

Departamento del director: Lenguajes y Sistemas Informáticos

Titulación: Ingeniería informática

Centro: Facultat d'Informàtica de Barcelona (FIB)

Universidad: Universitat Politècnica de Catalunya (UPC) BarcelonaTech

Contenido

Contenido.....	2
1 Introducción	8
1.1 Motivación	8
1.2 Objetivo.....	9
1.3 Estructura del documento.....	9
2 Android y 3D Logic.....	12
2.1 Android.....	12
2.1.1 Historia de android.....	12
2.1.2 Características	13
2.1.3 Arquitectura	14
2.1.4 Cuota de mercado	18
2.2 3D Logic	21
2.2.1 3D Logic Original.....	21
2.2.2 3DLogic II - Stronghold of sage.....	22
2.2.3 iPhone 3D Logic	23
2.2.4 Cuby.....	24
3 Android para juegos	26
3.1 Conceptos básicos	26
3.2 Estructura de un proyecto.....	29
3.3 Manifest	32
3.4 GLSurfaceView	35
3.5 Pantalla táctil.....	38
3.6 Texturas.....	42
3.7 Sonido.....	44
3.8 Acelerómetro	46
3.9 Pantalla completa y actividad constante	48
4 Análisis.....	50
4.1 Análisis de requisitos	50
4.2 Casos de uso	52
4.2.1 Diagrama de casos de uso	52
4.2.2 Partida Nueva	53
4.2.3 Elegir Nivel.....	54
4.2.4 Continuar.....	55

4.2.5	Nivel Aleatorio.....	56
4.2.6	Zoom In/Out.....	57
4.2.7	Agitar el móvil	58
4.2.8	Pintar casilla	59
4.2.9	Mover el cubo	61
4.2.10	Cambiar idioma	62
4.2.11	Activar/desactivar vibración.....	63
4.2.12	Activar/desactivar sonido.....	64
5	Diseño e implementación del juego.....	65
5.1	Diagrama de clases general.....	66
5.2	Diagramas de secuencia.....	67
5.2.1	Interacción con el menú.....	68
5.2.2	Iniciar nivel	69
5.2.3	Pintar casilla	70
5.2.4	Siguiente nivel	74
5.2.5	Mover cubo	75
5.2.6	Zoom.....	76
5.2.7	Agitar móvil	77
5.2.8	Cambiar configuración	78
5.2.9	Fin del juego	79
5.3	Metodología de desarrollo.....	80
5.4	Herramientas de diseño y desarrollo	82
5.4.1	Eclipse.....	83
5.4.2	Android Development Tools Plugin (ADT).....	83
5.4.3	Android SDK.....	84
5.4.4	Control de versiones	84
5.4.5	Diagramas.....	85
5.4.6	Google Docs.....	86
6	Diseño de la interfaz de usuario.....	87
6.1	Menú Principal	88
6.2	Menú Jugar.....	90
6.3	Menú Configuración.....	92
6.4	Menú Ayuda	94
6.5	Menú Elegir Nivel	96

6.6	Pantalla de juego.....	98
6.7	Fin del juego	100
6.8	Diagrama de navegabilidad completo.....	103
6.9	Testeando la interfaz de usuario.....	104
7	Pruebas.....	105
7.1	Emulador	105
7.2	Android x86	106
7.3	Móviles reales	106
7.3.1	Galaxy S1	107
7.3.2	Nexus.....	108
7.3.3	Otros.....	109
8	Publicar el juego en el Android Market.....	110
8.1	Cuenta de desarrollador.....	110
8.2	Preparar la aplicación para ser publicada	111
8.3	Publicación	112
8.4	Reporte de errores	115
9	Planificación y costes	116
9.1	Planificación	116
9.2	Coste del proyecto	118
10	Conclusiones.....	122
10.1	Evaluación de objetivos.....	122
10.2	Dificultades.....	123
10.3	Posibles ampliaciones	124
10.4	Conclusiones personales	124
	Bibliografía	126
A.	Instalación del entorno de desarrollo	129
B.	Firmar aplicación Android en Eclipse.	132
C.	Clases del juego.....	134
D.	Creación de los elementos de la interfaz.....	168
E.	Diagramas en formato texto	177

Índice de figuras

Figura 2.1 Arquitectura Android	14
Figura 2.2 Cuota de mercado Android España.....	18
Figura 2.3 Cuota de mercado Smartphone a escala mundial	19
Figura 2.4 Cuota de mercado de las nuevas adquisiciones.....	19
Figura 2.5 Porcentaje de uso, versiones Android.....	20
Figura 2.6 3D Logic original	22
Figura 2.7 Historia 3D Logic II.....	23
Figura 2.8 iPhone 3D Logic	24
Figura 2.9 Cuby 3D Logic	25
Figura 3.1 Ciclo de vida de una actividad	28
Figura 3.2 Estructura de un proyecto en Eclipse.....	29
Figura 4.1 Diagrama de casos de uso	52
Figura 5.1 Diagrama de clases.....	66
Figura 5.2 Interacción con el menú (Diagrama de secuencia)	68
Figura 5.3 Iniciar nivel (Diagrama de secuencia).....	69
Figura 5.4 Pintar casilla (Diagrama de secuencia).....	71
Figura 5.5 Pintar casilla (Cambiar de color)	72
Figura 5.6 Pintar casilla (Completar camino)	72
Figura 5.7 Pintar casilla (Romper camino)	73
Figura 5.8 Pintar casilla (Completar nivel)	73
Figura 5.9 Siguiente nivel (Diagrama de secuencia).....	74
Figura 5.10 Mover cubo (Diagrama de secuencia).....	75
Figura 5.11 Zoom (Diagrama de secuencia)	76
Figura 5.12 Agitar móvil (Diagrama de secuencia).....	77
Figura 5.13 Cambiar configuración (Diagrama de secuencia).....	78
Figura 5.14 Fin del juego (Diagrama de secuencia).....	79
Figura 5.15 Emulador Android	84
Figura 6.1 Menú Principal	89
Figura 6.2 Menú Jugar.....	91
Figura 6.3 Menú Configuración	93
Figura 6.4 Menú Ayuda	95
Figura 6.5 Menú Elegir Nivel	97
Figura 6.6 Pantalla de juego	99
Figura 6.7 Fin del juego (modo espejo).....	101
Figura 6.8 Fin del juego	102
Figura 6.9 Diagrama de navegabilidad	103
Figura 7.1 Primera prueba en Nexus One	108
Figura 8.1 Publicación de la aplicación, paso 1.....	112
Figura 8.2 Publicación de la aplicación, paso 2	112
Figura 8.3 Publicación de la aplicación, paso 3.....	113
Figura 8.4 Publicación de la aplicación, paso 4	114
Figura 8.5 Publicación de la aplicación, paso 5	114

Figura 8.6 Informes de errores.....	115
Figura 9.2 Planificación (Gantt)	117
Figura 9.1 Planificación (Diagrama Gantt)	117
Figura B.1 Firmar aplicación, paso 1	132
Figura B.2 Firmar aplicación, paso 2-1	132
Figura B.3 Firmar la aplicación, paso 2-2	133
Figura B.4 Firmar aplicación, paso 3	133
Figura C.1 Clase A_3D_Logic	135
Figura C.2 Clase GameRenderer.....	140
Figura C.3 Clase GLWorld	144
Figura C.4 Clase Cube	151
Figura C.5 Clase Square	155
Figura C.6 Clase Game.....	156
Figura C.7 Clase Level	159
Figura C.8 Clase Textures	160
Figura C.9 Clase Animations	162
Figura C.10 Clase SoundEffect.....	164
Figura C.11 Clase Menu.....	166
Figura D.1 Elección de logotipo	170
Figura D.2 Elección del indicador de nivel	171
Figura D.3 Elección de fondos de pantalla	172
Figura D.4 Diseño de las casillas.....	173
Figura D.5 Botones de los menús.....	174
Figura D.6 Títulos y elementos del menú de configuración.....	174
Figura D.7 Botones de navegabilidad por los menús e indicador de nivel	175
Figura D.8 Pantalla de ayuda en los dos idiomas	175
Figura D.9 Pantalla de fin de juego en los dos idiomas.....	176

Fragментos de código

Código 3.1 3D Logic Manifest.....	33
Código 3.2 Render OpenGL básico.....	37
Código 3.3 Gestión táctil simple.....	39
Código 3.4 Gestión multi-táctil	41
Código 3.5 Cargar texturas en OpenGL.....	43
Código 3.6 Fondo de pantalla y otras texturas estáticas en escena	44
Código 3.7 Reproducción de efectos sonoros.....	46
Código 3.8 Gestión de acelerómetro (agitar el móvil).....	47
Código 3.9 WakeLock y quitar título	49
Código C.1 Asignar propiedades de la actividad principal	137
Código C.2 Pintar nuevo <i>frame</i> de pantalla	142
Código C.3 Draw (Pintado de escena 3D).....	146
Código C.4 Algoritmo de selección (por color).....	148
Código C.5 Procesar pintado de una casilla	153
Código C.6 Buscar camino entre generadores (DFS).....	154
Código C.7 Añadir nivel nuevo	157
Código C.8 Cálculo de nivel espejo.....	158

1 Introducción

En este primer capítulo se empieza explicando la motivación del proyecto, impulsada por el gran crecimiento actual de tecnología y aplicaciones móviles, en concreto Android.

A continuación se definen los principales objetivos de este proyecto en cuanto al análisis de la tecnología y el desarrollo del juego propuesto.

Y finalmente, una guía rápida de la estructura y contenido de este documento, con una breve descripción del contenido de los capítulos de la memoria.

1.1 Motivación

El desarrollo de videojuegos y OpenGL siempre me ha llamado la atención. En la carrera he cursado tres asignaturas sobre el tema, videojuegos, visualización e interacción gráfica y visualización avanzada. Y es un mundo en el que me gustaría seguir formándome.

Por otro lado, el mercado de las aplicaciones móviles está en constante expansión y ofrece muchas posibilidades. Sobre todo desde que apareció Android para revolucionar el mundo de los dispositivos móviles. Siendo este de software libre, código abierto y compatible con multitud de dispositivos, facilitando así el desarrollo de aplicaciones de terceros, cuando hasta la fecha cada sistema operativo (Windows Mobile¹, Symbian², iOS³) iba ligado a unos dispositivos concretos, con características diferentes, que dificultaban el desarrollo de aplicaciones. No es de extrañar, pues, que Android se esté convirtiendo en el sistema operativo con mayor cuota de mercado, como se explica más detalladamente en el punto 2.1 - *Android*.

Siguiendo estas premisas, y estando a punto de acabar todas las asignaturas de la carrera, no dudé en ponerme en contacto con Lluís Pérez Vidal al encontrar su propuesta de PFC para realizar un videojuego 3D para Android. Permitiéndome así poner en práctica los conocimientos adquiridos en la programación de videojuegos y OpenGL, al mismo tiempo que profundizaba en un área de interés personal, Android y sus posibilidades en el mercado de juegos 3D.

¹ Sistema operativo de Microsoft para móviles: http://en.wikipedia.org/wiki/Windows_Mobile

² Sistema operativo móvil de Nokia: <http://en.wikipedia.org/wiki/Symbian>

³ Sistema operativo móvil de Apple: <http://en.wikipedia.org/wiki/IOS>

1.2 Objetivo

Con este proyecto pretendo iniciarme en el desarrollo de videojuegos para Android en OpenGL. Investigando los principios básicos a tener en cuenta, así como los actuales inconvenientes del desarrollo para esta plataforma.

Para ello, se ha elegido desarrollar el juego tipo puzzle 3D Logic explicado en el apartado [2.2 - 3D Logic](#). Mediante el cual se intentará sacar el mayor partido posible de las opciones de un móvil de última generación: efectos sonoros, vibración, pantalla táctil, gráficos de alta resolución, bases de datos, detector de movimiento, etc. Tratando además de que sea compatible con el mayor número de versiones del sistema operativo Android, así como con diferentes resoluciones y modelos de móvil disponibles en el mercado.

El objetivo final, si es posible, es publicar el juego en el Android Market, para que cualquier persona con un móvil Android pueda descargarlo y jugar. Colaborando así directamente con la comunidad de Android y ofreciendo a la gente diversión, como guinda final al trabajo realizado.

1.3 Estructura del documento

A continuación se facilita un breve resumen del contenido de los diferentes capítulos del documento, con el objetivo de facilitar la lectura del mismo.

En el capítulo [2 - Android y 3D Logic](#) se encuentra una introducción a los dos principales elementos que componen el proyecto. Explicando primero las características del sistema operativo Android, y a continuación el tipo de juego que se va a desarrollar, junto con ejemplos de las alternativas ya existentes en el mercado.

El capítulo [3 - Android para juegos](#) es una guía de los elementos básicos para desarrollar juegos, en concreto 3D, para Android. Se inicia el capítulo con una introducción a los elementos básicos en toda aplicación Android, y a continuación se ofrece una lista de características útiles para quien pretenda desarrollar un juego en esta plataforma.

En el capítulo [4 - Análisis](#) se definen los requisitos del juego. Leyendo este apartado nos podemos hacer una idea de qué hará el juego y qué propiedades tendrá. Además se muestra una lista detallada de los casos de uso que puede iniciar un usuario.

El apartado *5 - Diseño e implementación del juego* nos muestra cómo funciona internamente el juego. Cuenta con un diagrama de clases para entender la estructura completa y las relaciones entre los diferentes elementos. Seguido de los diagramas de secuencia que muestran el flujo interno de información para cada caso de uso relevante. Además, se explica la metodología de desarrollo, en la que podemos encontrar en detalle el proceso seguido en cada etapa del desarrollo, con sus respectivas dificultades y puntos clave. Finalmente se expone una lista de las herramientas usadas durante el diseño e implementación del juego.

En todo juego, uno de los puntos clave es la interacción con el usuario. Por eso el capítulo *6 - Diseño de la interfaz de usuario* explica en detalle cómo se interactúa con la aplicación. Se muestran todas las pantallas del juego y se explican los elementos clave de cada una. Además cuenta con un diagrama de navegabilidad que muestra claramente cómo se accede a cada una de las pantallas. Por último, se explica el proceso seguido para poner a prueba la interfaz de usuario, y elementos que la componen, ante usuarios reales.

En el apartado *7 - Pruebas* se encuentra una memoria de la experiencia personal probando el juego en diferentes entornos y dispositivos. Se exponen las ventajas e inconvenientes encontrados al usar el emulador oficial. Una alternativa a tener en cuenta. Y a continuación los problemas encontrados al probar el juego en diferentes móviles reales.

Una vez acabado y testeado el juego, lo ideal es publicarlo, compartiéndolo así con todos los usuarios de Android que deseen entretenerte con este tipo de juegos, ¿cómo lo hacemos? Aquí entra en juego el capítulo *8 - Publicar el juego en el Android Market*, que detalla los pasos a seguir para publicar oficialmente una aplicación Android.

En el capítulo *9 - Planificación y costes* se muestra la planificación que se ha seguido para llevar a cabo el proyecto. Añadiendo también un cálculo del coste aproximado del proyecto en caso de haberse tratado de un proyecto de empresa.

Para acabar, el apartado *10 - Conclusiones* ofrece una reflexión en la que se tienen en cuenta tanto los objetivos cumplidos en cuanto al desarrollo del juego, como los conocimientos adquiridos a lo largo de todo el tiempo dedicado al proyecto. Y se marca una lista de puntos clave en caso de desear ampliar las funcionalidades del juego.

Las últimas páginas se completan con la bibliografía, en la que se pueden encontrar los libros, artículos y webs consultados. Tanto para el análisis inicial, como para la resolución de errores que surgieron durante el desarrollo y pruebas de la aplicación.

Además, a modo de apéndices, se ha incluido información más técnica sobre las diferentes clases java del juego ([C - Clases del juego](#)); una guía de técnicas básicas en GIMP, junto con comentarios sobre los diseños finales y descartados de la interfaz de usuario ([D -Creación de los elementos de la interfaz](#)); y una guía de instalación del entorno de desarrollo Android ([A - Instalación del entorno de desarrollo](#)).

2 Android y 3D Logic

En este apartado se encuentra una introducción a los dos principales elementos que componen el proyecto. Se empieza explicando el origen y características principales del sistema operativo Android. Y a continuación se analiza el tipo de juego que se va a desarrollar, mostrando también las alternativas ya existentes en el mercado.

2.1 Android

Android es una plataforma software y un sistema operativo para dispositivos móviles (y *tablets*, *netbooks*, entre otros aparatos electrónicos e incluso electrodomésticos) basado en un núcleo de Linux. Fue creado por Google y la *Open Handset Alliance*⁴. Pero Android es mucho más que un sistema operativo, es también un lenguaje de programación y un *framework* para desarrollo de aplicaciones. Es decir, que al conjunto de todos esos elementos se lo llama Android. Y no solo es desarrollado por la Open Handset Alliance, sino que también colaboran muchísimas personas alrededor del mundo, ya que es un proyecto de Software Libre y cualquiera puede descargarse el código fuente desde la página de Android y modificarlo a su antojo. Tal y como se hace con los sistemas GNU/Linux de escritorio.

La sintaxis del lenguaje de programación es Java, pero es importante destacar (y marcar la diferencia) que no es Java, ya que no implementa todas las bibliotecas.

2.1.1 Historia de android

Allá por el mes de julio de 2005, Google adquirió una pequeña empresa Californiana llamada Android Inc al Ingeniero de Software Andy Rubin con la intención de desarrollar un sistema operativo móvil que pudiera responder a las expectativas que el mundo de los dispositivos móviles comenzaba a despertar.

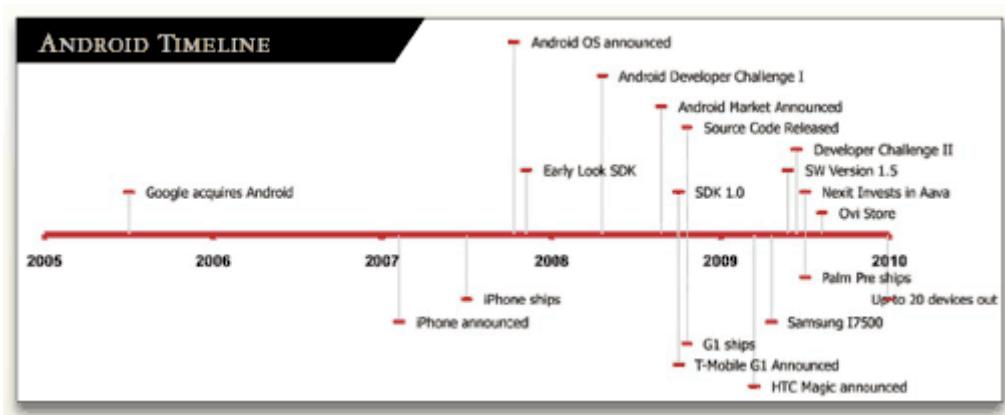
De esta forma comenzaron los trabajos para que, en noviembre de 2007, se anunciara oficialmente la creación de la *Open Handset Alliance* (una alianza mercantil formada por múltiples empresas que desarrollan estándares abiertos para dispositivos móviles, entre ellas Android) que vendría acompañada del lanzamiento oficial del Android Software Development

⁴ <http://www.openhandsetalliance.com/>

Kit (SDK). Esta primera versión supuso tanto un punto de partida como una forma de evaluar la aceptación del mercado, por lo que hasta mediados de agosto de 2008 no apareció la segunda versión, conocida como Android 0.9 SDK. Tras las comprobaciones oportunas, a finales del mes de septiembre lanzaron oficialmente la versión Android 1.0 SDK (*Release 1*).

Aunque los inicios fueran un poco lentos, debido a que se lanzó antes el sistema operativo que el primer móvil, rápidamente se ha colocado como el sistema operativo de móviles más vendido del mundo, situación que se alcanzó en el último trimestre de 2010.

En febrero de 2011 se anunció la versión 3.0 de Android, con nombre en clave Honeycomb⁵, que está optimizada para tabletas en lugar de teléfonos móviles. Por tanto Android ha transcendido de los teléfonos móviles a dispositivos más grandes.



2.1.2 Características

Entre sus características principales destacan:

- **Conektividad:**
 - Navegador integrado, basado en el motor open Source WebKit⁶
 - Dependiendo del hardware del teléfono: Telefonía GSM, Bluetooth, EDGE⁷, 3G, WI-FI.
- **Software:**
 - Gráficos optimizados con OpenGL ES 1.0, y a partir de la versión Android 2.2 OpenGL ES 2.0.
 - SQLite para el almacenamiento de datos estructurados (Base de datos)
 - Entorno de desarrollo: emulador, *debugger*.

⁵ <http://www.xatakandroid.com/fichas/sistema-operativo/android-3-0-honeycomb>

⁶ <http://www.webkit.org/>

⁷ http://es.wikipedia.org/wiki/Enhanced_Data_Rates_for_GSM_Evolution

- **Hardware y multimedia:**

- Máquina virtual *Dalvik*⁸: Base de llamadas de instancias muy similar a Java.
- Soporte multimedia con formatos comunes de audio, video e imágenes planas (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- Dependiendo del hardware del teléfono: Cámara, GPS, brújula, acelerómetro, pantalla táctil/multitáctil.

2.1.3 Arquitectura

Antes de empezar con el desarrollo de aplicaciones en Android es importante conocer cómo está estructurado este sistema operativo. A esto le llamamos arquitectura, y en el caso de Android está formada por varias capas que facilitan al desarrollador la creación de aplicaciones (*Figura 2.1*). A continuación explicamos cada una de las capas, de abajo hacia arriba.

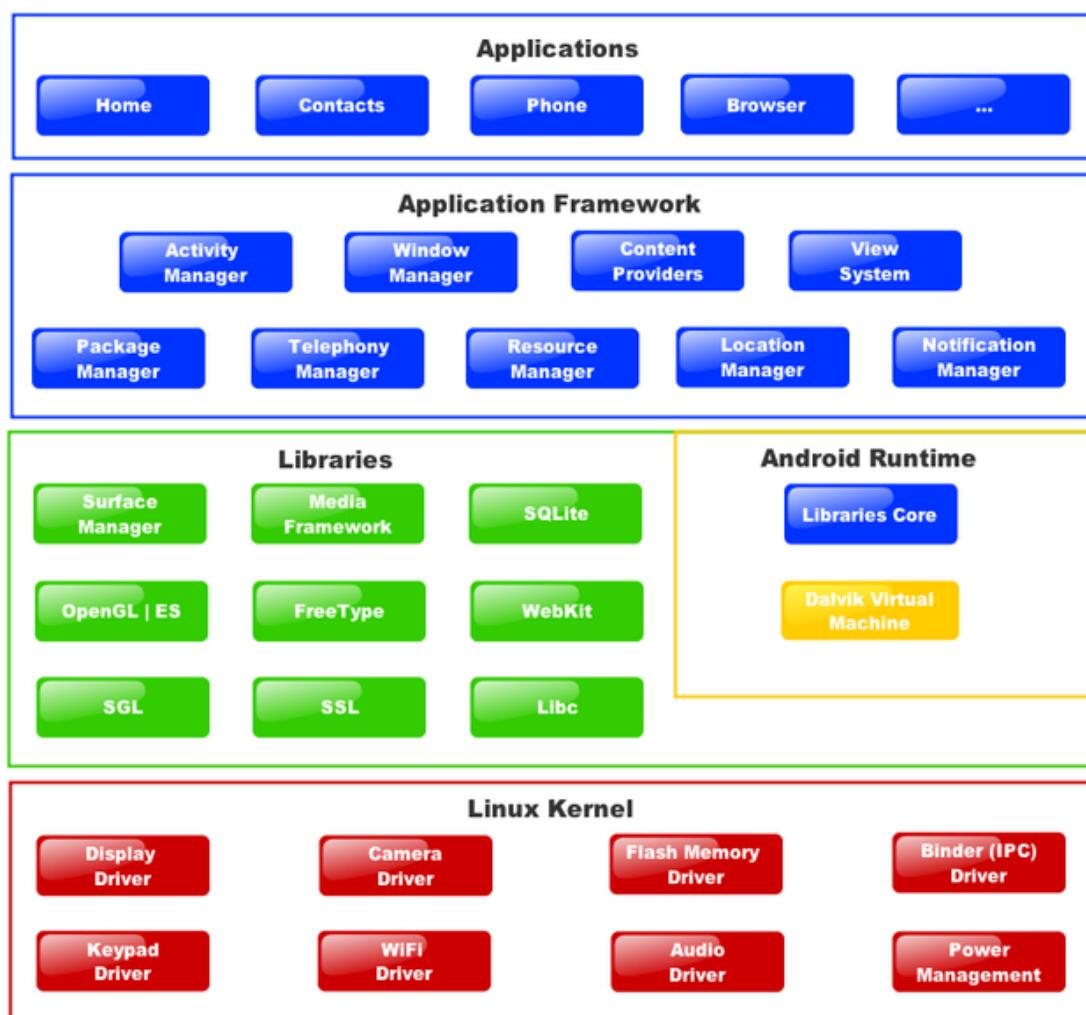


Figura 2.1 Arquitectura Android

⁸ <http://es.wikipedia.org/wiki/Dalvik>

1. Kernel de Linux:

El núcleo del sistema operativo Android está basado en el *kernel* de Linux versión 2.6, similar al que puede incluir cualquier distribución de Linux, como Ubuntu, solo que adaptado a las características del hardware en el que se ejecutará Android, es decir, para dispositivos móviles.

El núcleo 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. De esta forma también nos evitamos el hecho de quebrarnos la cabeza para conocer las características precisas de cada teléfono. Si necesitamos hacer uso de la cámara, el sistema operativo se encarga de utilizar la que incluya el teléfono, sea cual sea. Para cada elemento de hardware del teléfono existe un controlador (o *driver*) dentro del *kernel* que permite utilizarlo desde el software.

El *kernel* también se encarga de gestionar los diferentes recursos del teléfono (energía, memoria, etc.) y del sistema operativo en sí: procesos, elementos de comunicación (*networking*), etc.

2. Librerías:

La siguiente capa que se sitúa justo sobre el *kernel* la componen las bibliotecas nativas de Android, también llamadas librerías. Están escritas en C o C++ y compiladas para la arquitectura hardware específica del teléfono. Estas normalmente están hechas 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”.

Entre las librerías incluidas habitualmente encontramos OpenGL (motor gráfico), Bibliotecas multimedia (formatos de audio, imagen y video), Webkit (navegador), SSL (cifrado de comunicaciones), FreeType (fuentes de texto), SQLite (base de datos), entre otras.

3. Entorno de ejecución:

Como podemos apreciar en el diagrama (*Figura 2.1*), 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 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.

Cabe aclarar que *Dalvik* es una variación de la máquina virtual de Java, por lo que no es compatible con el bytecode Java. Java se usa únicamente como lenguaje de programación, y los ejecutables que se generan con el SDK de Android tienen la extensión .dex que es específico para *Dalvik*, y por ello no podemos correr aplicaciones Java en Android ni viceversa.

4. Framework de aplicaciones:

La siguiente capa 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*. Siguiendo el diagrama encontramos:

- Activity Manager: Se encarga de administrar la pila de actividades de nuestra aplicación así como su ciclo de vida.
- Windows Manager: Se encarga de organizar lo que se mostrará en pantalla. Básicamente crea las superficies en la pantalla que posteriormente pasarán a ser ocupadas por las actividades.
- Content Provider: Esta librería es muy interesante porque crea una capa que encapsula los datos que se compartirán entre aplicaciones para tener control sobre cómo se accede a la información.
- Views: En Android, las vistas son los elementos que nos ayudarán a construir las interfaces de usuario: botones, cuadros de texto, listas y hasta elementos más avanzados como un navegador web o un visor de Google Maps.
- Notification Manager: Engloba los servicios para notificar al usuario cuando algo requiera su atención mostrando alertas en la barra de estado. Un dato importante es que esta biblioteca también permite jugar con sonidos, activar el vibrador o utilizar los LEDs del teléfono en caso de tenerlos.

- Package Manager: Esta biblioteca permite obtener información sobre los paquetes instalados en el dispositivo Android, además de gestionar la instalación de nuevos paquetes. Con paquete nos referimos a la forma en que se distribuyen las aplicaciones Android, estos contienen el archivo .apk, que a su vez incluyen los archivos .dex con todos los recursos y archivos adicionales que necesita la aplicación, para facilitar su descarga e instalación.
- Telephony Manager: Con esta librería podremos realizar llamadas o enviar y recibir SMS/MMS, aunque no permite reemplazar o eliminar la actividad que se muestra cuando una llamada está en curso.
- Resource Manager: Con esta librería podremos gestionar todos los elementos que forman parte de la aplicación y que están fuera del código, es decir, cadenas de texto traducidas a diferentes idiomas, imágenes, sonidos o layouts. En un post relacionado a la estructura de un proyecto Android veremos esto más a fondo.
- Location Manager: Permite determinar la posición geográfica del dispositivo Android mediante GPS o redes disponibles y trabajar con mapas.
- Sensor Manager: Nos permite manipular los elementos de hardware del teléfono como el acelerómetro, giroscopio, sensor de luminosidad, sensor de campo magnético, brújula, sensor de presión, sensor de proximidad, sensor de temperatura, etc.
- Cámara: Con esta librería podemos hacer uso de la(s) cámara(s) del dispositivo para tomar fotografías o para grabar vídeo.
- Multimedia: Permiten reproducir y visualizar audio, vídeo e imágenes en el dispositivo.
-

5. Aplicaciones:

En la última capa se incluyen todas las aplicaciones del dispositivo, tanto las que tienen interfaz de usuario como las que no; las nativas (programadas en C o C++) y las administradas (programadas en Java); las que vienen pre-instaladas en el dispositivo y aquellas que el usuario ha instalado.

En esta capa encontramos también la aplicación principal del sistema: Inicio (Home) o lanzador (*launcher*), ya que es la que permite ejecutar otras aplicaciones mediante una lista, mostrando diferentes escritorios donde se pueden colocar accesos directos a aplicaciones o incluso *widgets*, que son también aplicaciones de esta capa.

2.1.4 Cuota de mercado

Basándonos en las estadísticas ofrecidas por *StatCounter* en Enero 2012 [20], Android cuenta con una cuota de mercado del 49.11% en España, *Figura 2.2*. Siendo así el sistema operativo móvil más popular de este país, seguido de iOS (37.8%) y SymbianOS (6.74%).

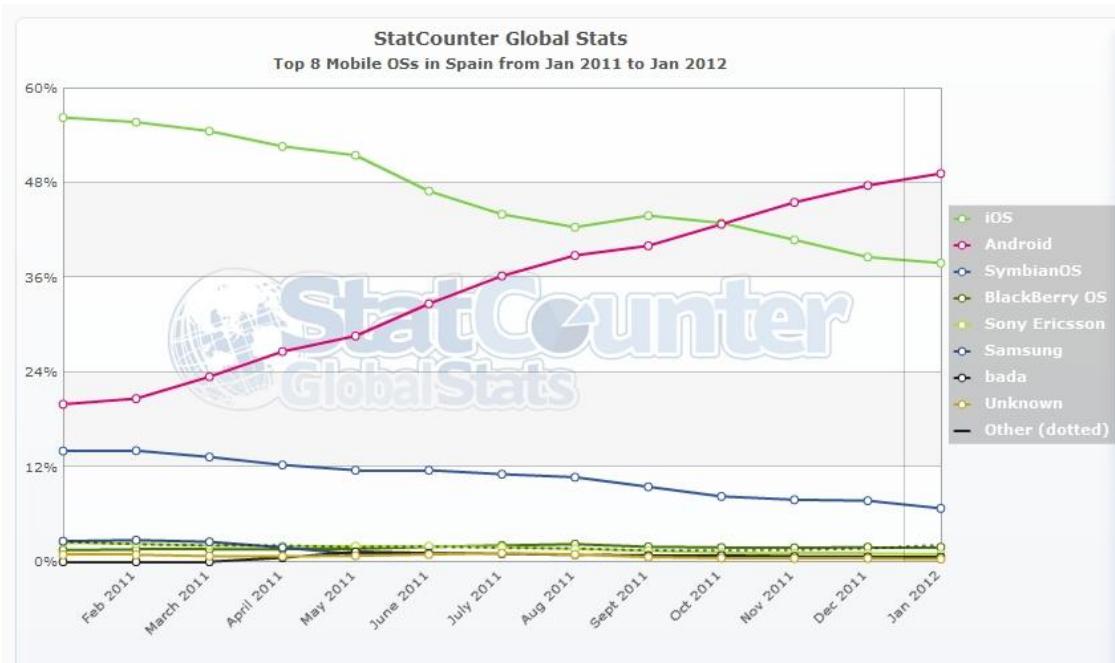


Figura 2.2 Cuota de mercado Android España

A pesar de que a nivel mundial aún no ha superado a SymbianOS y iOS en cuanto a número de móviles con dicho sistema operativo (*Figura 2.3*), basándonos en los datos ofrecidos por Nielsen en Septiembre 2011 para U.S. [21], podemos comprobar (*Figura 2.4*) que el sistema operativo Android es el que está experimentando un mayor crecimiento en el mercado, con un porcentaje superior al 50% en nuevas adquisiciones de móviles de última generación.

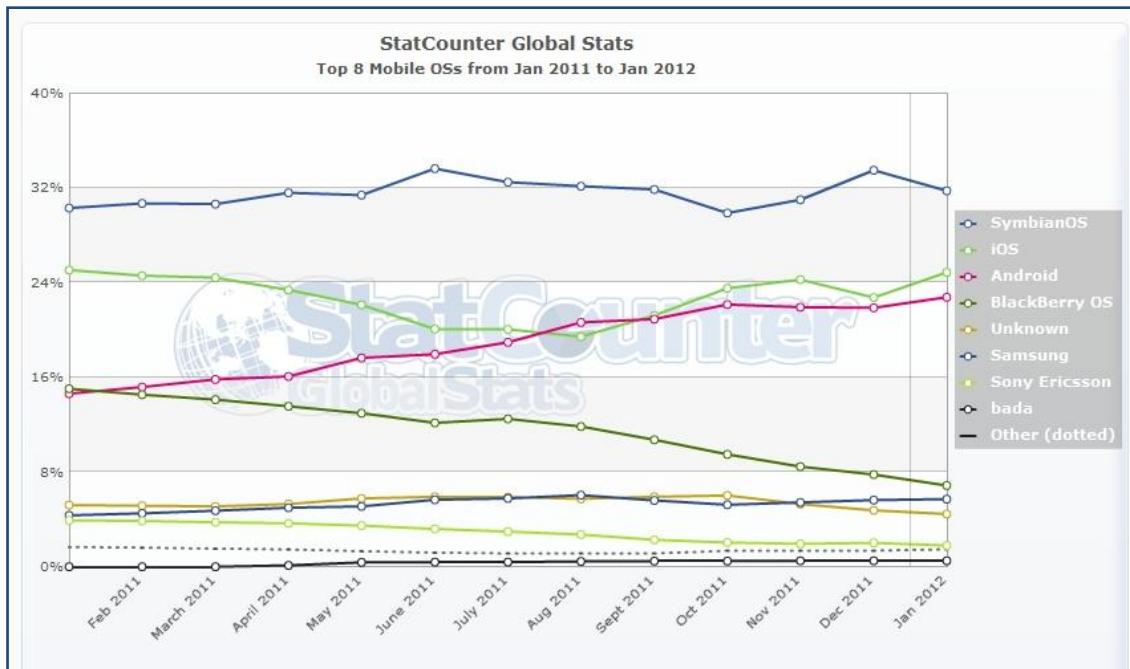
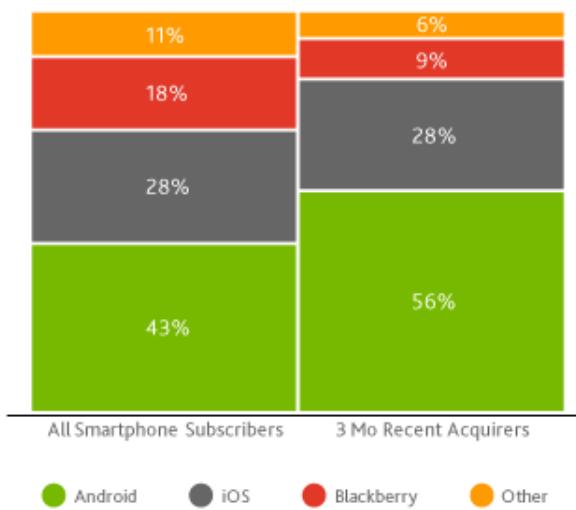


Figura 2.3 Cuota de mercado Smartphone a escala mundial

In August, 56% of recent acquirers chose an Android device

Operating System Share – All Subscribers and Recent Acquirers

Aug 2011, Nielsen Mobile Insights, National



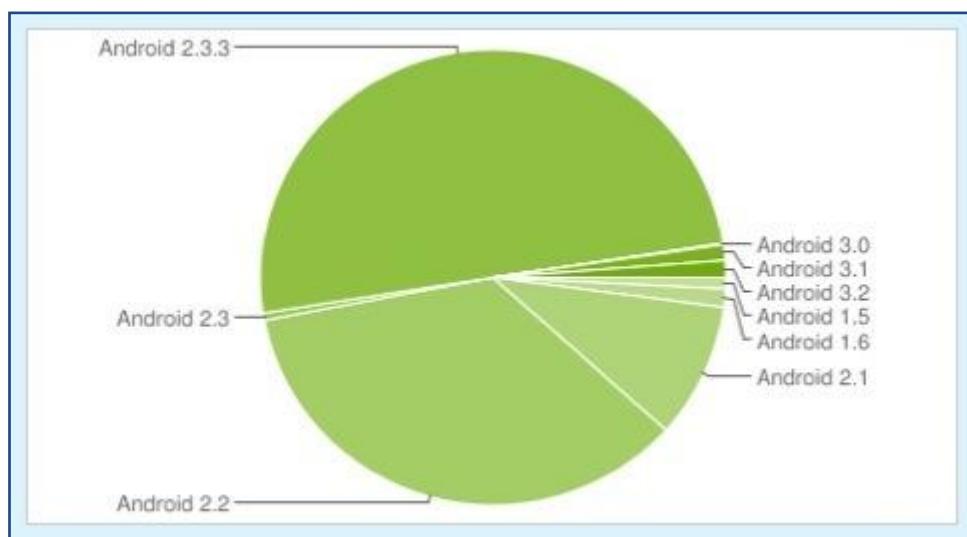
Source: Nielsen

nielsen

Figura 2.4 Cuota de mercado de las nuevas adquisiciones

En cuanto a las estadísticas de uso de versión de Android, basándonos en los datos mostrados por Nexus404 en Diciembre 2011 [22] sobre la actividad hasta Noviembre 2011, encontramos que la versión más usada es la Android 2.3.3+ ([Figura 2.5](#)), quitándole terreno a la versión 2.2 que hasta hacía pocos meses estaba en el primer puesto. La versión 3.x está poco extendida ya que es la versión para *tablets*, y en el mercado hay muchos más móviles que *tablets*. La nueva versión 4.0 aún no se considera ya que hoy por hoy son únicamente los móviles más nuevos los que la llevan.

- Android 1.5 – 0.8 %
- Android 1.6 – 1.3 %
- Android 2.1 – 9.6 %
- Android 2.2 – 35.3 %
- Android 2.3 to 2.3.2 – 0.5 %
- Android 2.3.3 to 2.3.4 – **50.1 %**
- Android 3.0 – 0.1 %
- Android 3.1 – 1.1 %
- Android 3.2 – 1.2 %



[Figura 2.5 Porcentaje de uso, versiones Android](#)

2.2 3D Logic

El juego elegido, 3D Logic, es una variante del famoso cubo de *Rubik*⁹. Se trata de un juego tipo puzzle que fue desarrollado por primera vez por Alexei Matveev (Minsk) en formato flash en el año 2006.

En esta variante del cubo de *Rubik*, el objetivo es ir uniendo las parejas de colores repartidas alrededor del cubo hasta conseguir completarlas todas. La premisa es muy simple, pero a medida que el tamaño del cubo y el número de parejas de colores crece, también lo hace la dificultad, convirtiéndose así en un rompecabezas desafiante, entretenido y adictivo a partes iguales.

A continuación se muestran las diferentes versiones del juego que han ido apareciendo hasta la fecha.

2.2.1 3D Logic Original

Primera aparición del juego. Desarrollado por Alexei Matveev (Minsk) y diseñada por Poll Harvey, en formato flash en el año 2006 (*Figura 2.6*). Se puede jugar de forma gratuita en diferentes páginas de juegos, por ejemplo *kongregate.com*¹⁰. Cuenta con 30 niveles que van apareciendo sucesivamente, música de fondo, y efectos sonoros al interactuar con el cubo. Tiene además la opción de continuar el juego por el nivel en el que lo dejamos la última vez.

⁹ http://es.wikipedia.org/wiki/Cubo_de_Rubik

¹⁰ 3D Logic en Kongregate.com: <http://www.kongregate.com/games/AlexMatveev/3d-logic>

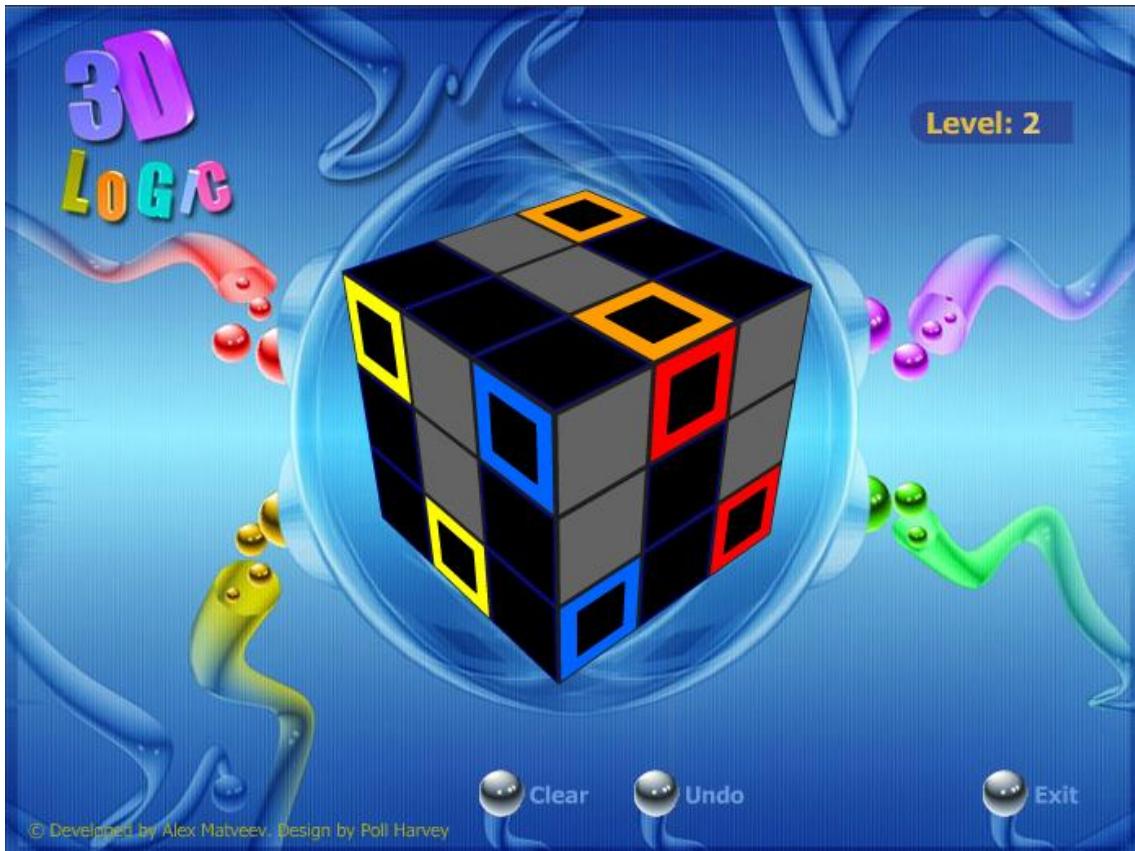


Figura 2.6 3D Logic original

A parte de aprovechar el concepto principal, algunos de los niveles originales también han sido reutilizados en el proyecto, con pequeñas modificaciones o alternando las caras del cubo. Además, también se ha extraído de esta versión el tipo de animación de transición entre niveles (aparece/desaparece el cubo).

2.2.2 3DLogic II - Stronghold of sage¹¹

En noviembre de 2007, el desarrollador de la primera parte presentó una segunda versión del título, 3D Logic II – Stronghold Of Sage. Esta cuenta de 30 nuevo niveles y introduce una pequeña historia que se va contando a medida que se superan los diferentes niveles ([Figura 2.7](#)). Añade además la opción de activar o desactivar la historia y la música de fondo, y podemos apreciar unas animaciones y efectos más cuidados que en la primera versión. También cuenta con un cronómetro que nos va indicando cuánto tardamos en superar los niveles.

¹¹ 3D Logic II en Kongregate.com: <http://www.kongregate.com/games/AlexMatveev/3d-logic-2-stronghold-of-sage>

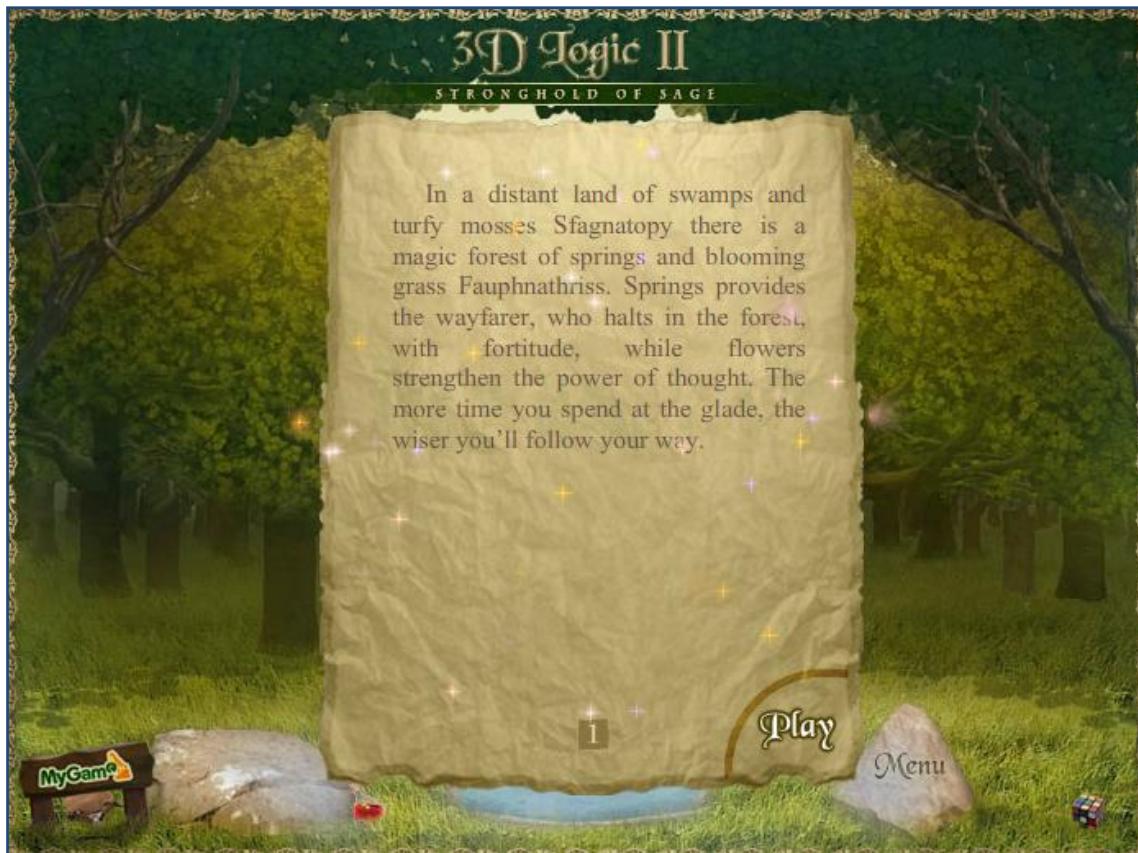


Figura 2.7 Historia 3D Logic II

En esta versión, el fondo de pantalla es visible a través de la transparencia de las casillas del cubo. Esta propiedad ha sido utilizada en el proyecto.

2.2.3 iPhone 3D Logic

En Julio de 2009, Spielwerk¹² puso a la venta una versión del 3D Logic en la AppStore¹³, para iPhone y iPod touch (*Figura 2.8*). Una de las cosas a destacar es que cuenta con un diseño muy cuidado, diferentes fondos de pantalla (idea utilizada en el proyecto) y cambios en la forma de mostrar el cubo y las casillas a medida que se superan los niveles. Dispone de 50 niveles, pero la gran mayoría son niveles extraídos de la primera o segunda versión del juego original.

De esta versión se ha extraído también la idea de añadir un selector de nivel, así como diferentes animaciones a la hora de seleccionar casillas, o completar un color.

En YouTube¹⁴ podemos encontrar un video que muestra su funcionamiento básico.

¹² <http://www.mobiledvhq.com/spielwerk-9808/developer>

¹³ 3D Logic - App Store <http://www.appstorehq.com/3dlogic-iphone-26822/app#details>

¹⁴ iPhone 3D Logic video: <http://www.youtube.com/watch?v=sN3CBb7X0xo>



Figura 2.8 iPhone 3D Logic

2.2.4 Cuby¹⁵

Es un proyecto de código libre desarrollado por Sebastian-Torsten Tillmann¹⁶ que pretende llevar el juego 3D Logic a diferentes sistemas operativos, GNU/Linux, Mac OS X, Windows y iPhone OS. Está desarrollado usando Raydium 3D Game Engine¹⁷.

Como podemos ver en la *Figura 2.9*, cuenta con un diseño limpio y minimalista. Los niveles de los que dispone son la unión de los de la primera y segunda versión del 3D Logic original. Y la principal novedad es la inclusión de un modo de edición en el que el usuario puede crear sus propios niveles.

El ser un proyecto de código libre, se han aprovechado un par de efectos sonoros de esta versión.

¹⁵ Página web oficial: <http://cuby.sourceforge.net/>

¹⁶ Página del autor en Sourceforge.net <http://sttillmann.users.sourceforge.net/>

¹⁷ <http://raydium.org/>

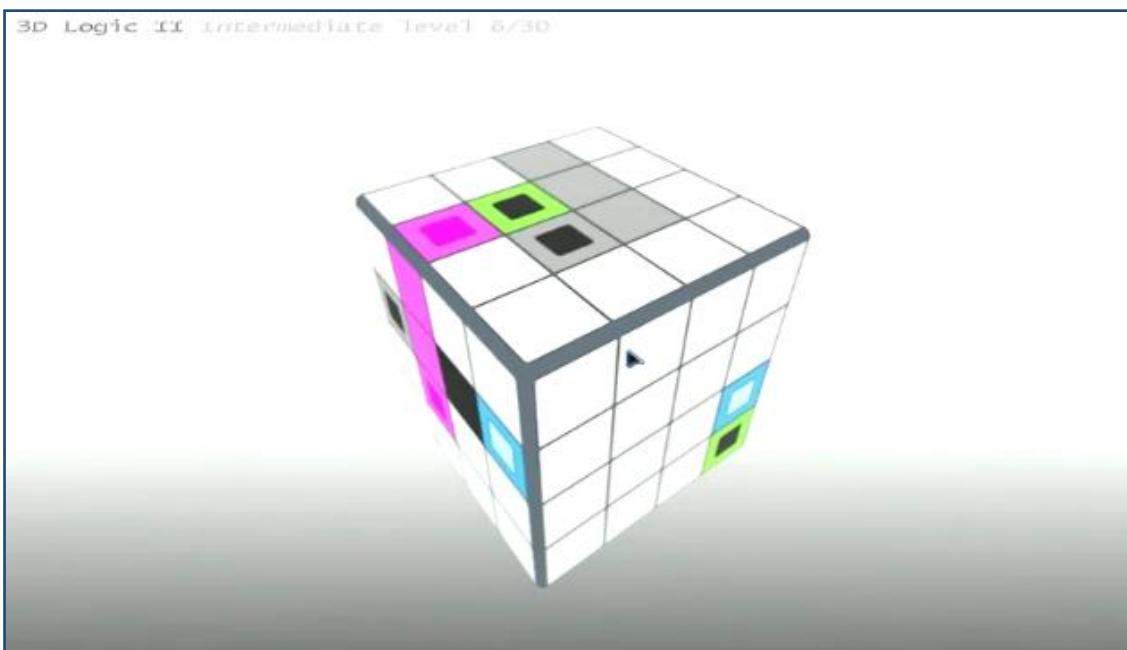


Figura 2.9 Cuby 3D Logic

3 Android para juegos

En este apartado podemos encontrar una lista de los elementos que es necesario conocer a la hora de iniciarse en el desarrollo de juegos 3D para Android.

Se empieza explicando los conceptos básicos comunes a todas las aplicaciones Android, y a continuación encontramos una lista de propiedades Android y OpenGL importantes, junto con un ejemplo de uso concreto en el proyecto.

3.1 Conceptos básicos

A la hora de programar en Android se deben tener en cuenta una serie de conceptos para realizar un buen desarrollo, los más importantes están descritos a continuación:

- Actividad (Activity): Componente de una aplicación que presenta una interfaz gráfica al usuario. (Toda GUI tiene una activity)
- Ciclo de vida de una actividad: En el sistema, las actividades están almacenadas en una pila. Cuando una nueva actividad es iniciada, se coloca al principio de la pila y empieza a ejecutarse. Las actividades anteriores no volverán a ejecutarse hasta que no se cierren las que están por encima.

Una actividad tiene esencialmente cuatro estados:

- Ejecución (*running*): Si la actividad está en pantalla (al principio de la pila).
- Pausada (*paused*): Si alguna otra actividad (transparente o que no sea de tamaño completo) se muestra por delante. Una actividad pausada está completamente viva, pero puede ser eliminada por el sistema en condiciones extremas de memoria.
- Parada (*stopped*): Si otra actividad la tapa completamente aún conserva todos los estados e información, sin embargo, no es visible para el usuario, por lo que se oculta la ventana y a menudo puede ser eliminada por el sistema cuando se requiere memoria en otra aplicación.
- Finalizada (*finishes*): Si una actividad está pausada o parada el sistema puede retirarla de la memoria, ya sea pidiéndole que finalice, o simplemente

eliminando el proceso. Cuando se vuelva a mostrar al usuario, debe ser arrancada de nuevo y se debe cargar su estado previo.

El siguiente diagrama (*Figura 3.1*) muestra los cambios de estado más importantes de una actividad. Los cuadrados rectangulares representan funciones que se pueden reprogramar para ampliar las operaciones realizadas cuando una actividad cambia de estado. Los cuadros coloreados son los estados principales en los que puede estar una actividad.

- *Services*: Son lo que comúnmente se conoce como procesos. Estos seguirán corriendo aunque no haya una interfaz gráfica para mostrar la aplicación. Por ejemplo cuando uno tiene un programa para reproducir música y “lo minimiza”, se continuará escuchando el sonido ya que se habrá creado un *Service* encargado de la reproducción de los sonidos.
- *Intents*: Es un mecanismo para comunicar a las distintas aplicaciones y actividades. Pueden notificarnos, por ejemplo, eventos del sistema, como que una target SD sea retirada del móvil, o se conecte el cable USB. También pueden ser programados para pedir a otras aplicaciones que ejecute alguna acción.
- *Content Providers*: Provee datos a otras aplicaciones. (proveedor de base de datos (SQLite))
- *Broadcast Recivers*: Componente para la ejecución de tareas pequeñas en segundo plano. Se utilizan para que una aplicación responda a un determinado evento del sistema. Por ejemplo, se puede utilizar un *Broadcast Receiver* en un programa para que cuando el teléfono se esté quedando sin batería se muestre un mensaje advirtiendo al usuario.

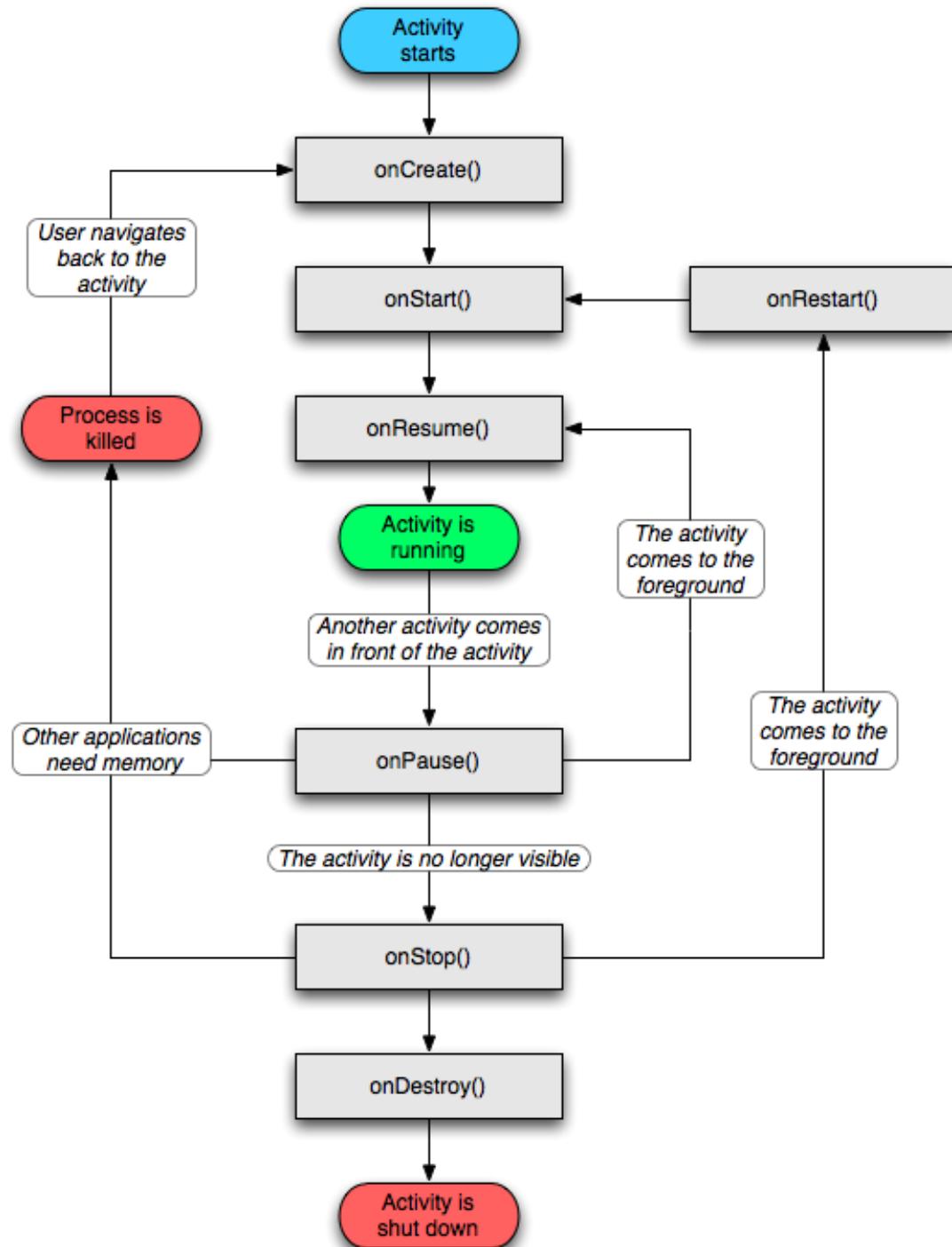


Figura 3.1 Ciclo de vida de una actividad

3.2 Estructura de un proyecto

Al crear un nuevo proyecto Android en Eclipse, se genera automáticamente la estructura de carpetas necesaria para poder desarrollar posteriormente la aplicación. Esta estructura es común a cualquier aplicación, independientemente de su tamaño y complejidad.

En la siguiente imagen (*Figura 3.2*) vemos los elementos creados inicialmente para un nuevo proyecto Android:

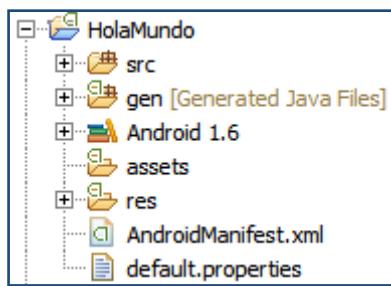
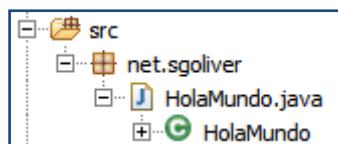


Figura 3.2 Estructura de un proyecto en Eclipse

A continuación se describen los elementos principales de un proyecto:

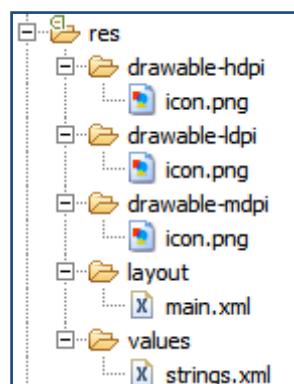
- Carpeta /src/: Contiene todos los archivos de código fuente de la aplicación, interfaz gráfica, clases auxiliares, etc. Eclipse creará por nosotros el código básico de la pantalla (actividad) principal de la aplicación, siempre bajo la estructura del paquete java definido.



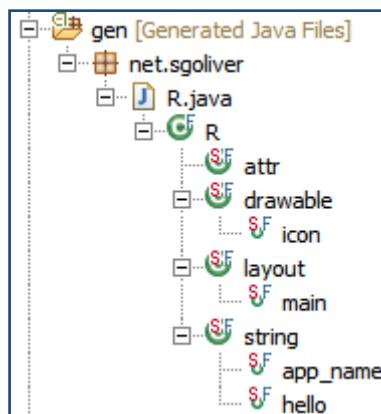
- Carpeta /res/: Aquí se guardan todos los ficheros de recursos necesarios para el proyecto: imágenes, archivos de sonido, vídeos, cadenas de texto, etc. Los diferentes tipos de recursos se distribuyen entre las siguientes carpetas:
 - /res/drawable/: Contienen las imágenes de la aplicación. Se pueden dividir entre /drawable-ldpi, /drawable-mdpi y /drawable-hdpi para utilizar imágenes de diferente calidad o tamaño dependiendo de la resolución del dispositivo.

- /res/layout/: Contienen los ficheros XML que definen los elementos de las diferentes pantallas de la interfaz gráfica. Podemos dividirla en /layout y /layout-land para separar distintos *layouts* dependiendo de la orientación del dispositivo.
- /res/anim/: Contiene la definición de las animaciones utilizadas por la aplicación (en caso de usar alguna).
- /res/menu/: Contiene la definición de los menús de la aplicación.
- /res/values/: Contiene otros recursos de la aplicación como por ejemplo cadenas de texto (strings.xml), estilos (styles.xml), colores (colors.xml), etc.
- /res/xml/: Contiene otros ficheros XML utilizados por la aplicación.
- /res/raw/: Contiene recursos adicionales, normalmente en formato distinto a XML, que no se incluyan en el resto de carpetas de recursos.

Por ejemplo, para un proyecto nuevo Android, se crean los siguientes recursos básicos para la aplicación:



- Carpeta /gen/: Contiene ficheros generados automáticamente al compilar el proyecto. Cada vez que generamos nuestro proyecto, el compilador de Android genera por nosotros una serie de ficheros fuente en Java, con el fin de facilitar la gestión de los recursos de la aplicación (aunque pueden llegar a ser un poco inestables).



El principal y más importante es el que observamos en la imagen anterior, el fichero R.java, y la clase R contenida en él.

Esta clase contiene en todo momento una serie de constantes con los identificadores de todos los recursos de la aplicación que estén incluidos en la carpeta /res/. Así, por ejemplo, la constante R.drawable.icon contendrá el ID de la imagen "icon.png" almacenada en la carpeta /res/drawable/ .

- [Carpeta /assets/](#): Contiene el resto de ficheros auxiliares necesarios para la aplicación (y que se incluirán en su propio paquete), como por ejemplo ficheros de configuración, de datos, etc.

La principal diferencia entre los recursos incluidos en la carpeta /res/raw/ y los incluidos en la carpeta /assets/ es que para los primeros se generará un ID en la clase R, y se deberá acceder a ellos mediante esas referencias. En cambio, para los segundos no se generará ID y se podrá acceder a ellos como a cualquier otro fichero del sistema, por su ruta. Se debe usar uno u otro dependiendo de las necesidades de la aplicación.

- [Fichero AndroidManifest.xml](#): Contiene la definición en XML de los aspectos principales de la aplicación, como por ejemplo su identificación (nombre, versión, ícono, etc), sus componentes (pantallas, mensajes, etc), o los permisos necesarios para su ejecución. Veremos más detalles de este fichero en el apartado que sigue a continuación.

3.3 Manifest

Una aplicación Android no solo tiene un hilo de ejecución, cuenta de diferentes componentes que interactúan con ella. Los componentes de una aplicación Android son iniciados, o llamados para realizar una acción, mediante un *Intent* específico.

Para definir de qué componentes está compuesta nuestra aplicación, y qué *Intents* pueden interactuar con la misma, se utiliza el archivo *manifest*. El sistema Android usa este archivo para conocer qué compone nuestra aplicación, así como cuál es la actividad principal a mostrar cuando la aplicación se inicia.

El archivo *manifest* no solo define los componentes de una aplicación. La siguiente lista resume las partes relevantes en cuanto al desarrollo de videojuegos.

- La versión de la aplicación (Ej. 1.0), para poder realizar actualizaciones en Android Market
- Las versiones de Android en las que puede utilizarse nuestra aplicación.
- Los requisitos hardware de la aplicación (Ej. *multitouch*, resoluciones de pantalla específicas, soporte para OpenGL ES 2.0)
- Permisos para usar componentes específicos, como por ejemplo escribir en la tarjeta SD o acceder a internet.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      android:versionCode="1"
4      android:versionName="1.0"
5      android:installLocation="auto"
6      package="com.A_3D_Logic">
7
8      <application android:icon="@drawable/icon"
9          android:label="@string/app_name"
10         android:debuggable="false">
11         <activity android:name="com.A_3D_Logic.A_3D_Logic"
12             android:screenOrientation="portrait"
13             android:configChanges="keyboardHidden|orientation"
14             android:label="@string/app_name">
15             <intent-filter>
16                 <action android:name="android.intent.action.MAIN" />
17                 <category android:name="android.intent.category.LAUNCHER" />
18             </intent-filter>
19         </activity>
20     </application>
21
22
23
24

```

```

25
26     <uses-permission android:name="android.permission.VIBRATE"/>
27     <uses-permission android:name="android.permission.WAKE_LOCK"/>
28
29     <uses-feature android:name="android.hardware.sensor.accelerometer" />
30
31     <uses-sdk android:minSdkVersion="4" android:targetSdkVersion="8"/>
32
33 </manifest>

```

Código 3.1 3D Logic Manifest

En el fragmento anterior ([Código 3.1](#)) podemos ver el archivo *manifest* del proyecto, los elementos XML más relevantes del mismo son los siguientes:

- Elemento <manifest> (línea 2) : Es el elemento principal del archivo, dentro de este definimos los componentes de la aplicación, permisos, requisitos hardware y versiones soportadas. Estos son los atributos básicos:
 - `versionCode` y `versionName` definen la versión de nuestra aplicación. `versionCode` ha de ser incrementado cada vez que publicamos una versión de la aplicación. Y `versionName` se mostrará a los usuarios que la busquen, puede ser cualquier texto.
 - En atributo `installLocation` en `auto` permite instalar la aplicación en la target SD.
- Elemento <application> (línea 8) : Para entender el contenido de este elemento hay que tener en cuenta que una aplicación cuenta con muchos archivos XML, no únicamente el *manifest*. Para referenciar a un atributo de otro archivo se usa “@” al principio del nombre, seguidamente el nombre de la carpeta dentro del directorio `res/` (mirar el apartado [3.2 - Estructura de un proyecto](#)), y finalmente el nombre del atributo.
 - El atributo `icon` hace referencia a la imagen en `res/drawable` que se usará como ícono de la aplicación en los menús del móvil.
 - En atributo `label` es el nombre que aparecerá bajo el ícono en los menús.
 - Finalmente, `debuggable` indica si la aplicación podrá ser monitorizada por consola o no. Durante el desarrollo debe estar activado para poder realizar pruebas e identificar errores.

- Elemento <activity> (línea 15) : Echemos un vistazo a los atributos de este elemento:
 - name define la ruta a la actividad principal del paquete especificado en el elemento <manifest>.
 - En atributo label será mostrado en la barra de título de la actividad.
 - screenOrientation especifica la orientación que usará la actividad. Es importante definirlo para forzar una orientación concreta, en lugar de que esta cambie automáticamente al girar el móvil, cosa que no nos interesa. Al asignarle portrait definimos una orientación vertical.
 - Al tratarse de un juego que no usará el teclado, en configChanges hemos de especificar que no se muestre el teclado en pantalla al cambiar la orientación del móvil.
- Elemento <intent-filter> (línea 11) : Cómo indiqué anteriormente, no hay un solo punto de entrada para una aplicación, sino múltiples que se indican con *intents* específicos. Es aquí donde se indica cuáles están enlazados a la aplicación. En nuestro caso especificamos dos tipos de *intent* diferentes:
 - El elemento <action> le dice a Android que esa actividad es la entrada principal de la aplicación
 - El elemento <category> especifica que nuestra actividad ha de ser añadida al lanzador de aplicaciones.
- Elemento <uses-permission> (línea 23) : Una vez acabados de definir los elementos de <application>, seguimos con otros elementos importantes de <manifest>. Uno de ellos es <uses-permission>. Android tiene un sistema de seguridad bien elaborado. Cada aplicación es ejecutada en su propio proceso y máquina virtual, con su propio usuario Linux y grupo, y no puede interferir con otras aplicaciones. Además, se restringe el uso de recursos del sistema, como acceso a internet, la tarjeta SD, el hardware de grabación de sonido, etc. Si nuestra aplicación quiere usar alguno de esos recursos, tenemos que pedir permiso. Y eso se hace con

este tipo de elemento. En este caso se pide permiso para poder hacer vibrar el móvil y prevenir que se desactive la pantalla si no se toca en cierto tiempo.

- Elemento <uses-feature> (línea 26) : Con este tipo de elemento definimos qué tipo de requisitos hardware ha de tener un móvil para poder ejecutar el juego. A aquellos usuarios cuyo móvil no cumpla estos requisitos no se les mostrará la aplicación en el Android Market. Por ejemplo, son de este tipo el soporte para OpenGL ES 2.0 o el acelerómetro (usado en el proyecto).
- Elemento <uses-sdk> (línea 28) : Cada versión de Android tiene un número asociado, también conocido como *SDK versión*. Este elemento especifica la mínima versión soportada por nuestra aplicación, así como la versión principal para la que va dirigida, mediante los atributos `minSdkVersion` y `targetSdkVersion`.

3.4 GLSurfaceView

Android ofrece principalmente dos APIs para pintar en pantalla. Una de ellas se usa para la programación de gráficos 2D, y la otra para gráficos 3D con aceleración hardware. En este proyecto nos centraremos en los gráficos 3D con OpenGL, y para tratarlos es necesario usar la clase `GLSurfaceView` que nos proporcionan las APIs.

Esta clase nos ofrece un hilo de ejecución independiente al de la interfaz de usuario, por lo que nosotros no nos tenemos que preocupar de crear un bucle que pinte de nuevo la escena a cada *frame*, se hará automáticamente. Lo único que necesitamos hacer es implementar la interfaz encargada del pintado de la escena, `GLSurfaceView.Renderer`, y registrarla en nuestra `GLSurfaceView`. La interfaz tiene tres métodos:

```
interface Renderer {
    public void onSurfaceCreated(GL10 gl, EGLConfig config);
    public void onSurfaceChanged(GL10 gl, int width, int height);
    public void onDrawFrame(GL10 gl);
}
```

El método `onSurfaceCreated()` es llamado cada vez que se crea la `GLSurfaceView`. Esto pasa la primera vez que iniciamos la actividad principal y cada vez que la actividad pasa a estar en primer plano después de estar en estado de pausa. La función recibe dos parámetros,

una instancia de GL10 y un EGLConfig. La instancia GL10 nos permite usar comandos OpenGL ES. EGLConfig nos proporciona ciertas propiedades de la vista, como por ejemplo la profundidad de color, y normalmente podemos ignorarlo. Lo ideal es iniciar las texturas y otras propiedades que no cambiarán durante la escena en esta función.

La función `onSurfaceChanged()` se llama cada vez que se redimensiona la vista, por ejemplo si permitimos cambio de orientación del móvil. Nos proporciona el nuevo ancho y alto en píxeles, además de la instancia GL10 para poder ajustar parámetros OpenGL.

Y la función `onDrawFrame()` podríamos considerarla el motor del juego. Es llamada tan a menudo como puede el hilo de ejecución de renderizado que configura la clase GLSurfaceView por nosotros. Y es en esta función donde se realiza todo el renderizado necesario en cada *frame*.

Además de implementar la interfaz Renderer, es necesario llamar a `GLSurfaceView.onPause()`/`onResume()` dentro del respectivo `onPause()`/`onResume()` de nuestra actividad principal. La razón es simple, el GLSurfaceView arrancará el hijo de ejecución de renderizado en `onResume()`, y lo desactivará en `onPause()`. Con esto conseguiremos que el render no haga trabajo extra cuando la actividad principal está pausada.

Hay que tener en cuenta que cada vez que la actividad principal es pausada, el GLSurfaceView será destruido, y al reanudar la actividad creará una nueva instancia OpenGL. Por lo tanto, habrá que volver a configurar todos los parámetros de nuevo.

A continuación ([Código 3.2](#)) podemos encontrar una estructura básica de la implementación de un Renderer. A partir de aquí se puede extender su funcionamiento para adaptarlo a nuestras necesidades. Se puede encontrar un extracto del código usado en el juego en el los apéndices ([Código C.2](#)).

```

1  class MyRenderer implements GLSurfaceView.Renderer {
2      private Scene mScene;
3      private float mAngleX=45, mAngleY=45;
4
5      public MyRenderer() {
6          mScene = new Scene();
7      }
8
9      public void onDrawFrame(GL10 gl) {
10         // Lo primero que se suele hacer es limpiar la escena.
11         // La manera más eficiente de hacerlo es usando glClear()
12         gl.glClear(GL10.GL_COLOR_BUFFER_BIT |
```

```

13     GL10.GL_DEPTH_BUFFER_BIT);
14
15     // Configuramos la cámara de la escena
16     gl.glMatrixMode(GL10.GL_MODELVIEW);
17     gl.glLoadIdentity();
18     gl.glTranslatef(0, 0, -3.0f);
19     gl.glRotatef(mAngleY, 0, 1, 0);
20     gl.glRotatef(mAngleX, 1, 0, 0);
21
22     // Habilitamos propiedades OpenGL necesarias
23     gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
24     gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
25
26     // Y dibujamos la escena 3D
27     mScene.draw(gl);
28 }
29
30 public void onSurfaceChanged(GL10 gl, int width, int height) {
31     gl.glViewport(0, 0, width, height);
32
33     // Normalmente no necesitamos cambiar la proyección a
34     // cada nuevo frame, pero al menos hay que configurarlo
35     // cuando la pantalla cambia de tamaño.
36     float ratio = (float) width / height;
37     gl.glMatrixMode(GL10.GL_PROJECTION);
38     gl.glLoadIdentity();
39     gl.glFrustumf(-ratio, ratio, -1, 1, 1, 10);
40 }
41
42 public void onSurfaceCreated(GL10 gl, EGLConfig config) {
43     // Las inicializaciones OpenGL que no vaya a cambiar se
44     // suelen incluir en esta función.
45     gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT,
46               GL10.GL_FASTEST);
47     gl.glEnable(GL10.GL_CULL_FACE);
48     gl.glShadeModel(GL10.GL_SMOOTH);
49     gl.glEnable(GL10.GL_DEPTH_TEST);
50 }
51 }
```

Código 3.2 Render OpenGL básico

3.5 Pantalla táctil

La pantalla táctil es probablemente la forma principal que tiene el usuario de interactuar con el móvil. Hasta la versión Android 2.0, la API sólo soportaba eventos de la pantalla táctil en un solo punto al mismo tiempo (un solo dedo). El *multitouch* se introdujo en la versión 5 del SDK (Android 2.0). Empezaremos explicando los eventos simples, disponibles en todas las versiones de Android, y a continuación se mostrará una lista de los cambios necesarios para gestionar múltiples puntos de interacción con la pantalla táctil.

- **Un solo punto (Single-Touch):**

Android nos reporta eventos mediante *listeners*. Los eventos de la pantalla táctil no son diferentes, por lo que se pasan a la implementación de la siguiente interfaz:

```
@Override public boolean onTouchEvent(MotionEvent e) {
```

Que podemos sobrescribir en cualquier actividad, y puede ser enlazada a cualquier vista personalizada mediante la función `View.setOnTouchListener()`.

El `MotionEvent` recibido cuenta de tres métodos relevantes para nosotros:

- `MotionEvent.getX()` y `MotionEvent.getY()`: Reporta la coordenada x e y del evento táctil. El sistema de coordenadas del sistema está definido con el origen arriba a la izquierda de la vista, las x apuntan hacia la derecha y las y hacia abajo. Las coordenadas vienen en *pixels*.
- `MotionEvent.getAction()`: Esta función devuelve el tipo del evento. Es un entero que puede contener uno de los siguientes valores: `MotionEvent.ACTION_DOWN`, `MotionEvent.ACTION_MOVE`, o `MotionEvent.ACTION_UP`.

Es sencillo, `MotionEvent.ACTION_DOWN` nos llega cuando un dedo toca la pantalla en un punto. Cuando el dedo se desliza a través de la pantalla, sin levantarla, se reciben `MotionEvent.ACTION_MOVE`. Cuando finalmente se levanta el dedo, el evento `MotionEvent.ACTION_UP` es reportado.

En el [Código 3.3](#) se muestra la lógica básica utilizada en el juego para gestionar los eventos táctiles simples.

```

1  switch (e.getAction()) {
2      case MotionEvent.ACTION_MOVE:
3          tipe = CURSOR_MOVE;
4          // Obtener desplazamiento dx y dy
5          [...]
6          mRender.setAngleX(antX + dx);
7          mRender.setAngleY(antY + dy);
8          [...]
9      case MotionEvent.ACTION_DOWN:
10         tipe = CURSOR_DOWN;
11         [...]
12     case MotionEvent.ACTION_UP:
13         tipe = CURSOR_UP;
14         // Obtener posición X e Y
15         [...]
16         mRender.picking(X, Y, tipe);
17         [...]
18     }

```

Código 3.3 Gestión táctil simple

- **Más de un punto (Multitouch):**

Estos eventos se reciben mediante el mismo MotionEvent que se usa para la gestión táctil simple, llegando de igual forma a la interfaz `onTouchEvent()`. Esto puede crear un poco de confusión.

Las diferencias empiezan cuando queremos acceder a las coordenadas de un evento táctil. `MouseEvent.getX()` y `MouseEvent.getY()` devuelven las coordenadas de la interacción del primer punto en pantalla. Cuando procesamos múltiples puntos simultáneos, tenemos que indicar el índice al que queremos acceder.

Por ejemplo: `event.getX(index);`

El problema es que no están ordenados (que el índice 0 fuera el primer dedo en tocar la pantalla, el 1 el segundo etc), lo que complica el seguimiento de cada punto.

El identificador real de cada punto se obtiene mediante la función:

```
int ident = event.getPointerIdentifier(index);
```

El identificador que nos devuelva esta función sí que será el mismo hasta que el dedo implicado se levante de la pantalla.

Para obtener el índice de un evento usamos:

```
int index=(event.getAction()&MotionEvent.ACTION_POINTER_ID_MASK)
>> MotionEvent.ACTION_POINTER_ID_SHIFT;
```

Resumiendo, en los bits del 8 al 15 de `getAction()` se guarda el índice que queremos, por lo que ponemos el resto de bits a 0 con la función `and`, y desplazamos los bits que nos interesan a la posición 0 a 7.

Teniendo esto en cuenta, es fácil entender que el tipo de la acción se obtiene de la siguiente forma:

```
int action = event.getAction() & MotionEvent.ACTION_MASK;
```

Y en cuanto a las acciones que podemos recibir, estos son los cambios relevantes con respecto al evento táctil simple:

`MotionEvent.ACTION_DOWN` sigue produciéndose cuando el primer dedo toda la pantalla, pero los siguientes generarán un evento de tipo `MotionEvent.ACTION_POINTER_DOWN` en su lugar.

El evento `MotionEvent.ACTION_UP` se recibirá cuando se levante el último dedo que esté tocando la pantalla, que no tiene por qué ser necesariamente el primero que la tocó. El resto generarán un `MotionEvent.ACTION_POINTER_UP`.

Lo que falta ahora es conocer la forma de detectar cuándo se está tocando la pantalla con más de un dedo (y saber con cuántos) para poder tratar los eventos adecuadamente. Para esto se usa la función `event.getPointerCount()`, que nos devuelve el número de puntos de interacción en pantalla.

A continuación ([Código 3.4](#)) se muestra el extracto de código del juego encargado de gestionar el zoom del cubo mediante la gestión multi-táctil.

```
1 public static boolean multitouch(MotionEvent event,
2                                     GameRenderer mRenderer) {
3     boolean singleTouch = true;
4     if (mode == ZOOM) {
5         singleTouch = false;
6         if (!(event.getPointerCount() > 1)) {
7             // Si solo hay un puntero en pantalla,
8             // desactivamos el modo zoom
9             mode = NONE;
10        }
11    }
12    switch (event.getAction() & MotionEvent.ACTION_MASK) {
13        case MotionEvent.ACTION_POINTER_DOWN:
14            // Registramos la distancia inicial entre los
15            // dos primeros dedos.
16            oldDist = spacing(event);
17            // La pantalla es muy sensible, hay que
18            // asegurarse de que los punteros están
19            // ligeramente separados.
20            if (oldDist > 10f) {
```

```
21             mode = ZOOM;
22         }
23     break;
24 case MotionEvent.ACTION_MOVE:
25     if (mode == ZOOM) {
26         float newDist = spacing(event);
27         // Zoom in/out dependiendo de la diferencia
28         // de distancia entre los dos primeros dedos.
29         if (newDist > 10f) {
30             mRenderer.zoom((newDist -
31                             oldDist)*TOUCH_ZOOM_FACTOR);
32         }
33         oldDist = newDist;
34     }
35     break;
36 }
37 // Notificamos si se ha de tartar el singleTouch en
38 // lugar del multiTouch
39 return singleTouch;
40 }
```

Código 3.4 Gestión multi-táctil

Como último consejo, hay que tener cuidado al usar al mismo tiempo *singleTouch* y *multiTouch*. Pongamos por ejemplo que gestionamos el movimiento de un objeto con un dedo, y el zoom con dos. Pulsamos con el primer dedo, se activa la opción de movimiento, que dependiendo de cuanto nos movemos, desplaza más o menos el objeto. A continuación tocamos con otro dedo y se activa la opción de zoom, pero acto seguido soltamos el primer dedo y volvemos a la gestión de movimiento. Si no tenemos cuidado, la aplicación considerará que nos hemos movido desde la posición del primer dedo hasta la del segundo, desplazando el objeto de golpe cuando no es lo que deseamos.

3.6 Texturas

Aunque es posible hacer un juego con formas básicas y usar colores simples para pintarlas, el resultado dejaría mucho que desear. Lo ideal es que un diseñador cree los modelos y fondos de pantalla, y nosotros los carguemos a partir de archivos de imagen PNG o JPEG. Para poder cargar dichos archivos, Android proporciona la clase `Bitmap`.

Una vez cargadas las imágenes que contienen el estilo de los diferentes elementos del juego, hemos de identificarlas como texturas, y mediante OpenGL aplicarlas a los triángulos que se renderizan en escena.

A continuación ([Código 3.5](#)) se muestra cómo cargar un `Bitmap`, partiendo de las referencias del archivo R autogenerado, y la forma de almacenar los identificadores de cada textura usada.

```

1  public Textures(GL10 gl) {
2      mTextureID = new int[N_TEXTURES];
3      // Generamos identificadores
4      glGenTextures(N_TEXTURES, mTextureID, 0);
5
6      // Cargamos las texturas que queremos
7      glBindTexture(GL_TEXTURE_2D, mTextureID[BACKGROUND_ID]);
8      setTextureAttributes();
9      LoadTexture(gl, R.drawable.background);
10
11     [...] // Lo mismo para el resto de texturas
12
13     // En este caso, activamos blending para las transparencias.
14     glEnable(GL_BLEND);
15     glBlendFunc(GL_ONE, GL_ONE_MINUS_SRC_ALPHA);
16
17 }
18
19 public static void setTextureAttributes() {
20     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
21                     GL_NEAREST);
22     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
23                     GL_LINEAR);
24     [...] // El resto de parámetros que deseemos configurar.
25 }
26
27 public static void LoadTexture(GL10 gl, int R_RAW_ID) {
28     // Obtenemos la imagen fuente mediante el archivo R
29     InputStream is =
30         mContext.getResources().openRawResource(R_RAW_ID);
31     Bitmap bitmap;
32     try {
33         BitmapFactory.Options options =
34             new BitmapFactory.Options();
35         // Nos aseguramos de que se cargue con buena calidad

```

```

36     options.inPreferredConfig = Bitmap.Config.ARGB_8888;
37     // Creamos el bitmap.
38     bitmap = BitmapFactory.decodeStream(is, null, options);
39 } finally {
40     try {
41         is.close();
42     } catch (IOException e) { //Ignore}
43 }
44 // Asociamos el bitmap al identificador de textura activo.
45 GLUtils.texImage2D(GL_TEXTURE_2D, 0, bitmap, 0);
46 bitmap.recycle(); // Liberamos el bitmap
47 }

```

Código 3.5 Cargar texturas en OpenGL

Una vez cargadas, el proceso normal para utilizar las texturas es activar la que deseemos mediante su identificador, con `glBindTexture(GL_TEXTURE_2D, ID)`; y relacionar cada vértice de la escena (aquellos que pertenezcan a figuras que deban ser texturizadas) con una posición x e y de la textura.

Pero ¿qué pasa en el caso de los fondos de pantalla, en los menús del juego y otros elementos que han de mantenerse estáticos con respecto a los movimientos de la cámara en escena? ¿Hemos de definir una nueva figura OpenGL mediante cuatro vértices, uno a cada esquina de la pantalla, y moverlos a medida que se mueve la escena? Afortunadamente no es así.

Una alternativa es pintar esos elementos en una perspectiva ortogonal, y luego cambiar a una perspectiva isométrica para pintar la escena. Pero afortunadamente OpenGL ES 1.1 nos proporciona una extensión que facilita esta tarea, *OES_draw_texture*. Las funciones de esta extensión están optimizadas para obtener el mejor rendimiento, y nos permiten pintar fondos de pantalla y otros elementos simplemente indicando el tamaño y posición que ocupara la textura (o porción de esta) que le indiquemos.

A continuación ([Código 3.6](#)) se muestra el código usado en el juego para renderizar el fondo de pantalla.

```

1 public static void renderBackground(GL10 gl, int level) {
2     // Cargamos la textura
3     glBindTexture(GL_TEXTURE_2D, mTextureID[BACKGROUND_ID]);
4
5     // Definimos los parámetros: 2D, OES, fragmento de textura
6     ((GL11) gl).glTexParameteriv(GL10.GL_TEXTURE_2D,
7                                  GL11Ext.GL_TEXTURE_CROP_RECT_OES,
8                                  // Podemos elegir la textura entera o sólo una parte.
9                                  // Como el eje Y en escena está invertido, indicamos la
10                                 // altura con signo negativo. (iniX, iniY, width, height)

```

```

11         new int[] {0, BACKGROUND_HEIGHT,
12                         BACKGROUND_WIDTH,
13                         -BACKGROUND_HEIGHT
14                         }, 0);
15 // Finalmente usamos la función de la extensión OES para
16 // mostrar la textura. (x, y, z, width, height)
17 ((GL11Ext) gl).glDrawTexiOES(0, 0, 1, SCREEN_WIDTH,
18                               SCREEN_HEIGHT);
19 // El fondo de pantalla, el menú, y otros elementos estarán
20 // a la misma profundidad, y no queremos que se estorben
21 // entre ellos, ni que interactúen con la escena 3D. Por lo
22 // que los pintaremos antes de pintar la escena y limpiaremos
23 // el depth buffer para que no interfieran con esta.
24 gl.glClear(GL_DEPTH_BUFFER_BIT);
25 }

```

Código 3.6 Fondo de pantalla y otras texturas estáticas en escena

Como apunte final en cuanto a las texturas, hay que tener en cuenta que la mayoría de dispositivos únicamente acepta archivos de textura con tamaños en potencias de 2. Por lo que un textura con tamaño 30x30, por ejemplo, no se mostraría en pantalla.

3.7 Sonido

Android proporciona un par de APIs para reproducir efectos sonoros, y archivos de música, que cumplen perfectamente con las necesidades a la hora de programar juegos.

Si tienes un móvil Android, habrás notado que cuando pulsas los botones de control de volumen, controlas diferentes configuraciones dependiendo de en qué aplicación te encuentres en ese momento. En una llamada telefónica, por ejemplo, controlas el volumen de la voz del interlocutor. En cambio en un video de YouTube controlas el volumen del video. O en el menú principal del teléfono controlarás el volumen del “ring” de llamada. Y en todos estos casos se trata de un control de volumen diferente. Por lo tanto, lo primero que debemos hacer, en cuanto a la gestión de sonido en nuestra aplicación, es indicar qué tipo de sonido se gestiona en esta.

En juegos y aplicaciones se usar la configuración llamada *music stream*. Y para activarla, partiendo del *context* de la actividad principal, se usa un *AudioManager* de la siguiente forma:

```
context.setVolumeControlStream(AudioManager.STREAM_MUSIC);
```

Hay principalmente dos tipos de sonido que podemos reproducir: efectos sonoros cortos que se almacenan en memoria, y canciones largas en *streaming* que normalmente procederán de almacenamiento externo.

En este proyecto se han usado únicamente efectos sonoros cortos, por lo que no entrará en detalles en cuanto a cómo reproducir música en *streaming*, para más detalles es conveniente leer la documentación de la clase `MediaPlayer` que proporcionan las APIs. Android nos proporciona la clase `SoundPool` para reproducir efectos sonoros fácilmente. Para empezar a usar una instancia de la clase procedemos de la siguiente forma:

```
soundPool = new SoundPool(10, AudioManager.STREAM_MUSIC, 0);
```

Donde el primer parámetro define el número de efectos sonoros que se pueden reproducir simultáneamente. El segundo la configuración de audio por la que se reproducirá el sonido. Y el tercero parámetro no se usa actualmente, por lo que lo dejaremos por defecto a 0.

Para cargar en memoria los efectos de sonido que queremos, usamos la función:

```
soundID[0] = soundPool.load(context, R.raw.bip, 1);
```

Donde indicamos el `context` de la actividad principal, el archivo a cargar (que en este caso obtenemos mediante las referencias del archive R autogenerado) y finalmente la prioridad de ese efecto (un parámetro que actualmente no tiene ningún efecto). Y nos devuelve el identificador del efecto, que hemos de registrar para más adelante poder reproducirlo.

Hasta ahora hemos explicado cómo cargar los efectos sonoros, un proceso que normalmente sólo es necesario realizar una vez al iniciar la aplicación. A continuación ([Código 3.7](#)) se muestra, a modo de ejemplo, la función del proyecto encargada de reproducir un efecto sonoro previamente cargado:

```
1 public static void play(int ID) {
2     // Obtenemos el nivel de volume actual.
3     AudioManager audioManager = (AudioManager)
4             mContext.getSystemService(Context.AUDIO_SERVICE);
5     float actualVolume = (float) audioManager
6             .getStreamVolume(AudioManager.STREAM_MUSIC);
7     float maxVolume = (float) audioManager
8             .getStreamMaxVolume(AudioManager.STREAM_MUSIC);
9     // El volume es un atributo entre 0 y 1.
10    float volume = actualVolume / maxVolume;
11
12    // Y si el sonido está activado y el efecto está
13    // cargado, lo reproducimos.
14    if (loaded[ID] && sound_ON) {
```

```

15         soundPool.play(soundID[ID], volume, volume, 1, 0, 1f);
16     }
17 }
```

Código 3.7 Reproducción de efectos sonoros

El primer parámetro de la función `play` (línea 15) es el identificador del efecto. El segundo y tercer parámetro son el nivel de volumen del auricular izquierdo y derecho respectivamente. El cuarto es la prioridad (que no se usa). El quinto indica el número de veces que se ha de repetir el efecto (-1 significa infinitamente, 0 significa reproducirlo una única vez, sin repetición, y más de 0 el número de repeticiones). Y finalmente el último atributo permite cambiar la velocidad del efecto. Indicando un valor inferior a 1 el efecto se reproducirá más despacio de lo normal, y con un valor superior a 1 más deprisa.

Por último, si deseamos liberar de memoria alguno de los efectos cargados, hemos de usar la función `soundPool.unload()`. Y cuando ya no vayamos a reproducir ningún efecto sonoro más, debemos llamar a `soundPool.release()`, lo que borrará de memoria todos los efectos cargados.

3.8 Acelerómetro

Una entrada de datos interesante para ciertos juegos es la procedente del acelerómetro. Todos los móviles Android cuentan con un acelerómetro que detecta movimiento en los tres ejes espaciales.

Para obtener esa información, como con el resto de notificaciones, necesitamos registrar un *listener*. En concreto, la interfaz que debemos implementar se llama `SensorEventListener`, y cuenta de dos métodos:

```

Public void onSensorChanged(SensorEvent event) :
Public void onAccuracyChanged(Sensor sensor, int accuracy);
```

El que nos interesa es el primero, que nos notifica de los nuevos eventos recibidos. El segundo nos informa de cambios de precisión, y podemos ignorarlo para un uso estándar del acelerómetro.

Para activar o desactivar la recepción de notificaciones se usa un `SensorManager`. Con este podemos comprobar que el acelerómetro esté disponible:

```
mSensorManager =
        (SensorManager) getSystemService(Context.SENSOR_SERVICE);
hasAccel =
    mSensorManager.getSensorList(Sensor.TYPE_ACCELEROMETER).size() > 0;
```

Si hay un acelerómetro instalado, podemos activar (registrar) el *listener* de la siguiente forma:

```
mSensorManager.registerListener(mSensorListener,
    mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
    SensorManager.SENSOR_DELAY_NORMAL);
```

Donde `mSensorListener` es nuestra implementación de la interfaz. Que en este proyecto se usa para detectar si el usuario ha agitado el móvil. A continuación se muestra el extracto (*Código 3.8*) comentado y simplificado encargado de esta funcionalidad:

```
1 private final SensorEventListener mSensorListener =
2     new SensorEventListener() {
3
4     public void onSensorChanged(SensorEvent se) {
5         if(listenShake){
6             float x = se.values[0];
7             float y = se.values[1];
8             float z = se.values[2];
9             mAccelLast = mAccelCurrent;
10            mAccelCurrent = (float) Math.sqrt((double)
11                (x*x + y*y + z*z));
12            float delta = mAccelCurrent - mAccelLast;
13            mAccel = mAccel * 0.9f + delta;
14            // Si consideramos el movimiento lo
15            // suficientemente grande, realizamos
16            // la acción deseada.
17            if(mAccel >= SHAKED) {
18                mRenderer.resetLevel();
19                listenShake = false;
20            }
21        }
22    else {
23        // Esperamos un tiempo prudencial entre
24        // acciones de este tipo para poder gestionar
25        // adecuadamente cada una de ellas.
26        ++nShakesWaiting;
27        if (nShakesWaiting >= 10) {
28            listenShake = true;
29            nShakesWaiting = 0;
30        }
31    }
32}
33};
34};
```

Código 3.8 Gestión de acelerómetro (agitar el móvil)

Por último, hay que tener en cuenta que lo ideal es desactivar el control del acelerómetro (`mSensorManager.unregisterListener(mSensorListener)`) cuando la aplicación está en segundo plano (`onStop()`), y volver a activarlo cuando se reanuda la actividad (`onResume()`). De esta forma evitamos comportamientos no deseados y que se gaste batería sin ser necesario.

3.9 Pantalla completa y actividad constante

Por defecto, todas las actividades muestran una barra de título de la aplicación y la barra de estado del móvil, en la que se indica el nivel de batería, notificaciones recientes, la hora, etc. Si creemos importante que nuestra actividad o juego ocupe toda la pantalla, podemos eliminar estas barras de la siguiente forma:

```
requestWindowFeature(Window.FEATURE_NO_TITLE); // 1
getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN, // 2
 WindowManager.LayoutParams.FLAG_FULLSCREEN);
```

Bajo mi punto de vista, la barra de estado del móvil es importante mantenerla. De lo contrario el usuario puede despistarse y consumir toda la batería jugando, o para consultar la hora necesita salir de la aplicación, entre otros. Por lo tanto en el juego desarrollado en este proyecto únicamente se desactiva la barra de título.

Por otro lado, si dejamos de interactuar con el móvil durante unos segundos, la pantalla se oscurecerá, y finalmente el móvil se bloqueará. Sólo si pulsas la pantalla (o algún botón) la pantalla recupera el brillo normal (o pedirá ser desbloqueada). Este comportamiento por lo general es muy útil para reducir el gasto de batería del móvil. Pero en el caso de un juego en el que tengas que pensar sin interactuar con la pantalla, o que por ejemplo sólo haga uso del acelerómetro, es un inconveniente, una molestia para el usuario. Pero Android nos proporciona una utilidad para controlar este comportamiento, los *wake lock*.

Lo primero que hemos de hacer es añadir un `<uses-permission>` en el *manifest* de nuestra aplicación (`android.permission.WAKE_LOCK`). De lo contrario no podremos usar la clase `WakeLock`.

Podemos obtener una instancia `WakeLock` mediante la clase `PowerManager`. A continuación ([Código 3.9](#)) se muestra el código básico para crear y gestionar un `WakeLock`:

```
1  @Override
2  protected void onCreate(...) {
3      [...] // Quitar el título
4      requestWindowFeature(Window.FEATURE_NO_TITLE)
5
6      // Al crear la actividad principal, creamos el wakeLock
7      PowerManager powerManager = (PowerManager)
8          this.getSystemService(Context.POWER_SERVICE);
9      wakeLock = powerManager
10         .newWakeLock(PowerManager.FULL_WAKE_LOCK, "My Lock");
11    }
12    [...]
13  @Override
14  protected void onResume() {
15      [...]
16      // Cuando la actividad pasa a estar en primer plano
17      // habilitamos el wakeLock.
18      wakeLock.acquire();
19    }
20    [...]
21  @Override
22  protected void onPause() {
23      [...]
24      // Cuando la actividad deja de estar en primer plano
25      // desactivamos el wakeLock.
26      wakeLock.release();
27  }
```

Código 3.9 WakeLock y quitar título

4 Análisis

El principal objetivo del proyecto es crear un juego funcional, que aproveche las posibilidades de los móviles de última generación, y que sea compatible con el mayor número de móviles Android del mercado.

Con ese fin, en este apartado se expone una lista de los requisitos necesarios para que el juego cumpla las expectativas. Y a continuación, los principales casos de uso disponibles para un usuario.

4.1 Análisis de requisitos

A continuación se especifican los diferentes requisitos del juego. Además de las restricciones surgidas a causa de las decisiones de diseño tomadas.

Requisitos funcionales:

- El juego ha de contar con diferentes niveles que vayan apareciendo consecutivamente.
- La aplicación ha de detectar la interacción mediante la pantalla táctil y ejecutar algoritmos de selección y movimiento de objetos en escena.
- Se ha de controlar qué color está activo en cada momento.
- Se ha de comprobar a cada nueva casilla pintada, si ya se ha completado ese camino.
- Al completar cada camino, se ha de comprobar si ya se ha completado el nivel actual.
- Al completar cada nivel, se ha de comprobar si ya se ha superado el juego entero.
- Se usarán diferentes sonidos para identificar las acciones del usuario y el estado del juego.
- El móvil indicará el final de un nivel con una ligera vibración.
- El juego contará con un menú interactivo para navegar entre las diferentes opciones.
- El usuario ha de poder activar o desactivar los efectos sonoros y vibración si así lo desea.
- El tamaño del cubo debe poderse ajustar manualmente.
- Se ha de guardar el estado del juego: ultimo nivel completado y configuración.
- Se ha de poder elegir qué nivel de juego se desea jugar.
- El juego ha de indicar el número del nivel actual.

- Una vez en la pantalla de juego, se ha de poder volver al menú con facilidad.
- Se ha de poder volver a empezar un nivel agitando el móvil.
- Una vez superado el juego, se activará un modo de juego especial, en el que los niveles alternen las caras de cubo de forma aleatoria, para que sea más difícil recordarlos y así la vida del juego aumente.

Requisitos no funcionales:

- Rendimiento: Ha de notarse una ejecución fluida para una interacción satisfactoria para el usuario.
- Usabilidad: Ha de ser sencillo de utilizar y contar con un menú intuitivo para el usuario.
- Compatibilidad: Al tratarse de un juego no demasiado complejo ni exigente gráficamente, uno de los objetivos es que sea compatible con el mayor número de versiones Android disponibles.

Restricciones de diseño:

- Se utilizará OpenGL 1.0. La versión 2.0 no es compatible con el emulador, y al inicio del proyecto no disponía de móvil en el que realizar pruebas, por lo tanto era inviable empezar a desarrollar en la versión 2.0 ya que no podría haber probado ningún ejemplo de código desarrollado.
Además, al tratarse de un juego poco exigente gráficamente, usar la versión 1.0 no implica una desventaja. Y por otro lado, se garantiza una mayor compatibilidad con móviles antiguos.
- El lenguaje de programación utilizado será Java/C++, ya que desarrollar aplicaciones para Android así lo requiere.

4.2 Casos de uso

En este apartado se explican los principales casos de uso con tal de entender con facilidad las posibilidades que ofrece el juego.

4.2.1 Diagrama de casos de uso

A continuación (*Figura 4.1*) se encuentra el diagrama que muestra los casos de uso que puede iniciar el usuario.

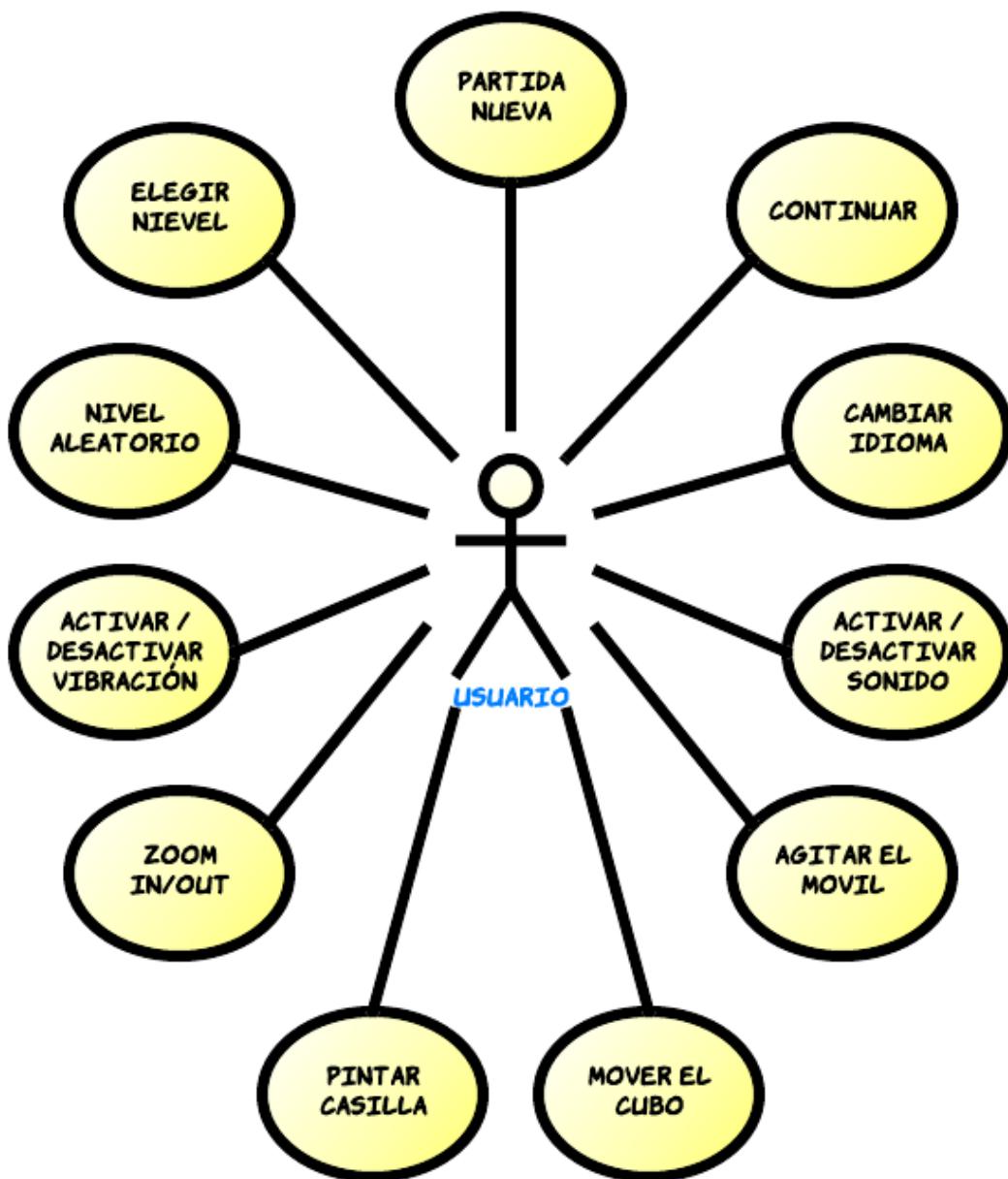


Figura 4.1 Diagrama de casos de uso

4.2.2 Partida Nueva

Breve descripción: El usuario selecciona la opción *Partida Nueva* del menú *Jugar*.

Actor principal: Usuario

Flujo básico: Partida Nueva

1. El usuario selecciona la opción *Jugar* en el menú principal.
2. La aplicación reproduce el sonido de opción seleccionada.
3. La aplicación muestra al usuario el menú *Jugar*.
4. El usuario selecciona la opción *Partida Nueva* en el menú *Jugar*.
5. La aplicación carga el primer nivel del juego en el cubo y prepara los *buffers OpenGL* necesarios para renderizar la escena.
6. La aplicación muestra el fondo de pantalla, el indicador de nivel (01), y el botón para volver al menú del juego.
7. La aplicación reproduce el sonido de inicio de juego.
8. La aplicación muestra una animación en la que aparece el cubo de menor a mayor opacidad hasta estabilizarse.

Flujo alternativo: Sonido desactivado

Los pasos 2 y 7 no se ejecutan y se sigue en el paso 3, 8 respectivamente.

Flujo alternativo: Cancelar

El usuario no sigue el paso 4.

1. El usuario selecciona la opción *back*.
2. La aplicación muestra de nuevo el menú principal.

4.2.3 Elegir Nivel

Breve descripción: El usuario selecciona la opción *Elegir Nivel* del menú *Jugar*.

Actor principal: Usuario

Flujo básico: Elegir Nivel

1. El usuario selecciona la opción *Jugar* en el menú principal.
2. La aplicación reproduce el sonido de opción seleccionada.
3. La aplicación muestra al usuario el menú *Jugar*.
4. El usuario selecciona la opción *Elegir Nivel* en el menú *Jugar*.
5. La aplicación muestra la primera página del apartado *Elegir Nivel*. Un botón para volver al menú anterior, y otro para pasar a la página siguiente (dependiendo de si estamos en la primera, intermedia, o última página, se muestra o no el botón de página anterior/siguiente). Los niveles que aún no ha sido superados por el usuario se muestran en rojo, y los ya superados, en verde.
6. El usuario selecciona uno de los niveles disponibles.
7. La aplicación carga en el cubo el nivel seleccionado (en modo espejo si el usuario ya se ha pasado el juego) y prepara los *buffers OpenGL* necesarios para renderizar la escena.
8. La aplicación muestra el fondo de pantalla, el indicador de nivel, y el botón para volver al menú del juego.
9. La aplicación reproduce el sonido de inicio de juego.
10. La aplicación muestra una animación en la que aparece el cubo de menor a mayor opacidad hasta estabilizarse.

Flujo alternativo: Sonido desactivado

Los pasos 2 y 9 no se ejecutan y se sigue en el paso 3, 10 respectivamente.

Flujo alternativo: Cancelar

El usuario no sigue el paso 4 o 6.

1. El usuario selecciona la opción *back*.
2. La aplicación muestra de nuevo el menú anterior, *principal* o *Jugar*.

Flujo alternativo: Nivel no disponible

El usuario selecciona un nivel no disponible (en rojo) en el paso 6.

1. No ocurre nada, el juego no cambia de estado.

Flujo alternativo: Cambiar de página

El usuario no ejecuta el paso 6, en su lugar pulsa el botón de página anterior/siguiente.

1. La aplicación muestra la anterior/siguiente página del apartado *Elegir Nivel*. Como en el paso 5 del flujo principal.

4.2.4 Continuar

Breve descripción: El usuario selecciona la opción *Continuar* del menú principal.

Actor principal: Usuario

Flujo básico: Continuar

1. El usuario selecciona la opción *Continuar* en el menú principal.
2. La aplicación carga en el cubo el último nivel jugado por el usuario, y prepara los *buffers OpenGL* necesarios para renderizar la escena.
3. La aplicación muestra el fondo de pantalla, el indicador de nivel, y el botón para volver al menú del juego.
4. La aplicación reproduce el sonido de inicio de juego.
5. La aplicación muestra una animación en la que aparece el cubo de menor a mayor opacidad hasta estabilizarse.

Flujo alternativo: Sonido desactivado

El paso 4 no se ejecuta y se sigue en el paso 5.

4.2.5 Nivel Aleatorio

Breve descripción: El usuario selecciona la opción *Nivel Aleatorio* del menú *Jugar*.

Actor principal: Usuario

Flujo básico: Nivel Aleatorio

1. El usuario selecciona la opción *Jugar* en el menú principal.
2. La aplicación reproduce el sonido de opción seleccionada.
3. La aplicación muestra al usuario el menú *Jugar*.
4. El usuario selecciona la opción *Nivel Aleatorio* en el menú *Jugar*.
5. La aplicación carga un nivel al azar, de entre los ya superados por el usuario, en el cubo, y prepara los *buffers OpenGL* necesarios para renderizar la escena.
6. La aplicación muestra el fondo de pantalla, el indicador de nivel, y el botón para volver al menú del juego.
7. La aplicación reproduce el sonido de inicio de juego.
8. La aplicación muestra una animación en la que aparece el cubo de menor a mayor opacidad hasta estabilizarse.

Flujo alternativo: Sonido desactivado

Los pasos 2 y 7 no se ejecutan y se sigue en el paso 3, 8 respectivamente.

Flujo alternativo: Cancelar

El usuario no sigue el paso 4.

1. El usuario selecciona la opción *back*.
2. La aplicación muestra de nuevo el menú principal.

4.2.6 Zoom In/Out

Breve descripción: El usuario aplica zoom in/out usando dos de sus dedos.

Actor principal: Usuario

Prerrequisito: Estamos en la pantalla de juego, con la escena cargada y el cubo visible.

Flujo básico: Zoom

1. El usuario pulsa la pantalla con dos dedos.
2. La aplicación registra la distancia inicial entre los dos dedos del usuario y entra en modo *zoom*, en este modo el cubo no se mueve y no se pintan las casillas al deslizar los dedos por pantalla.
3. El usuario amplia (los dos dedos se alejan) o reduce (los dos dedos se acercan) la distancia entre sus dedos, sin levantarlos de la pantalla.
4. La aplicación calcula la diferencia entre la distancia inicial y la nueva distancia, y en función de esta diferencia, muestra el cubo más grande o más pequeño. La distancia inicial pasa a ser la nueva distancia, para cálculos posteriores.

Flujo alternativo: Límites de tamaño

En el paso 3, el usuario insiste en ampliar/reducir el tamaño una vez llegado al límite.

1. El cubo conserva su tamaño en el límite máximo/mínimo de zoom.

Flujo alternativo: Cancelar

El usuario no sigue el paso 3.

1. El cubo conserva el tamaño previo.

4.2.7 Agitar el móvil

Breve descripción: El usuario agita el móvil para volver a empezar el nivel.

Actor principal: Usuario

Prerrequisito: Estamos en la pantalla de juego, con la escena cargada y el cubo visible.

Flujo básico: Reiniciar nivel

1. El usuario agita el móvil durante un corto periodo de tiempo.
2. La aplicación considera que el móvil ha sido agitado lo suficiente como para interpretar que el usuario desea reiniciar el nivel.
3. La aplicación reinicia el nivel actual del cubo, y prepara de nuevo los *buffers OpenGL* para renderizar la escena.
4. La aplicación reproduce el sonido de inicio de juego.
5. La aplicación muestra una animación en la que aparece el cubo de menor a mayor opacidad hasta estabilizarse.

Flujo alternativo: Sonido desactivado

El paso 4 no se ejecuta y se sigue en el paso 5.

Flujo alternativo: Poco agitado

En el paso 2 la aplicación no considera suficiente el movimiento como para interpretar que el usuario desea reiniciar el nivel.

1. No ocurre nada, el estado del juego no cambia.

4.2.8 Pintar casilla

Breve descripción: El usuario selecciona una de las casillas del cubo para pintarla de un color previamente elegido.

Actor principal: Usuario

Prerrequisito: Estamos en la pantalla de juego, con la escena cargada y el cubo visible.

Flujo básico: Pintar casilla

1. El usuario pulsa la pantalla en un punto que corresponde a una casilla del cubo.
2. La aplicación ejecuta un algoritmo de selección para averiguar a qué casilla del cubo 3D corresponde el punto pulsado en pantalla.
3. La aplicación comprueba el tipo de la casilla seleccionada (casilla normal).
4. La aplicación comprueba si se ha roto algún camino ya completado (no es el caso)
5. La aplicación comprueba si se ha completado el camino del color actual (no se ha completado).
6. La aplicación reproduce el sonido de selección de casilla.
7. La aplicación muestra una animación en la que la casilla seleccionada pasa del negro al color actualmente activo.

Flujo alternativo: Sonido desactivado

El paso 6 no se ejecuta y se sigue en el paso siguiente, 7. Sucede lo mismo en los pasos de reproducción de sonido de los flujos alternativos.

Flujo alternativo: Generador de color

En el paso 3 se verifica que la casilla seleccionada es un generador de color. El resto de pasos se substituyen por los siguientes:

1. Se desactivan las casillas *generadoras* del color anterior, y se activan las del nuevo color.
2. La aplicación reproduce el sonido de selección de generador de color.
3. La aplicación muestra una animación en la que el generador de color seleccionado, y su compañero, pasan del negro a su color original.

Flujo alternativo: Romper camino

En el paso 4 se verifica que se ha roto un camino previamente completado. El resto de pasos se substituyen por los siguientes:

1. Se desactivan las casillas *generadoras* del color del camino que se ha roto.
2. La aplicación reproduce el sonido de camino roto.
3. Se sigue en el paso 5 del flujo principal.

Flujo alternativo: Camino completado

En el paso 5 se verifica que el camino del color actual ha sido completado. El resto de pasos se substituyen por los siguientes:

1. La aplicación comprueba si se ha completado el nivel actual (no se ha completado).

2. La aplicación cambia el estado de las casillas *generadoras* del color actual a *completado*.
3. La aplicación reproduce el sonido de camino completado.
4. La aplicación muestra una animación en la que, todas las casillas que componen el camino del color actual, pasan del negro al color que toca (incluidos los generadores, que ahora se muestran *completados*). Cada una de ellas con un retraso relativo a la posición en el camino.

Flujo alternativo: Nivel completado

En el paso 1 del flujo alternativo “Camino completado” se verifica que se ha completado el nivel actual. El resto de pasos se substituyen por los siguientes:

1. La aplicación reproduce el sonido de nivel completado.
2. El teléfono móvil vibra durante aproximadamente un segundo.
3. La aplicación muestra una animación en la que el cubo desaparece de escena.
4. La aplicación comprueba si el nivel completado es el último del juego (no es el caso).
5. La aplicación carga el siguiente nivel en el cubo, y prepara los *buffers* OpenGL necesarios para renderizar la escena.
6. La aplicación muestra una animación en la que el cubo reaparece en escena con el siguiente nivel cargado.

Flujo alternativo: Juego completado

En el paso 4 del flujo alternativo “Nivel completado” se verifica que se ha completado el último nivel del juego. El resto de pasos se substituyen por los siguientes:

1. La aplicación comprueba si es la primera vez que el usuario se pasa el juego (así es).
2. La aplicación muestra una pantalla de final de juego. Y le notifica que se ha activado el modo espejo.
3. La aplicación registra en la base de datos que el usuario se ha pasado el juego.

Flujo alternativo: Juego completado de nuevo

En el paso 1 del flujo alternativo “Juego completado” se verifica que el usuario se ha pasado el juego más de una vez. El resto de pasos se substituyen por los siguientes:

1. La aplicación muestra una pantalla de final de juego.

Flujo alternativo: Vibración desactivada

El paso 2 del flujo alternativo “Nivel completado” no se ejecuta, y se sigue en el paso siguiente, 3.

4.2.9 Mover el cubo

Breve descripción: El usuario mueve el cubo deslizando un dedo por pantalla.

Actor principal: Usuario

Prerrequisito: Estamos en la pantalla de juego, con la escena cargada y el cubo visible.

Flujo básico: Mover el cubo

1. El usuario pulsa la pantalla en una zona diferente de la del cubo.
2. La aplicación registra las coordenadas del punto pulsado.
3. El usuario, lo desliza por la superficie de la pantalla, sin levantararlo.
4. La aplicación calcula la dirección y longitud del movimiento con respecto al punto pulsado (que se va actualizando).
5. La aplicación muestra de nuevo el cubo, desplazado en la dirección seleccionada.
6. El usuario finalmente levanta el dedo de la pantalla.

Flujo alternativo: Límites de movimiento

En el paso 3, el usuario llega al límite de alguna de las caras del cubo, y sigue deslizando el dedo en esa dirección.

1. El cubo permanece en el límite de la cara.

Flujo alternativo: Pintado simultaneo

En paso 1, el usuario pulsa una casilla del cubo en lugar de una zona vacía.

1. Se ejecuta simultáneamente el caso de uso [4.2.8 - Pintar casilla](#) cada vez que deslicemos el dedo por una casilla nueva. Con la excepción de que no puede producirse el flujo alternativo “Generador de color”

4.2.10 Cambiar idioma

Breve descripción: El usuario selecciona uno de los idiomas disponibles en el menú de Configuración.

Actor principal: Usuario

Flujo básico: Cambiar idioma

1. El usuario selecciona la opción *Configuración* en el menú principal.
2. La aplicación reproduce el sonido de opción seleccionada.
3. La aplicación muestra al usuario el menú de *Configuración*.
4. El usuario pulsa la bandera de otro idioma disponible.
5. La aplicación cambia el idioma de los menús.
6. La aplicación registra el cambio en la base de datos, para recordar la configuración del usuario la próxima vez que abra el juego.
7. La aplicación reproduce el sonido de opción seleccionada.
8. La aplicación muestra de nuevo el menú de *Configuración*, esta vez con los títulos en el idioma seleccionado por el usuario.

Flujo alternativo: Sonido desactivado

Los pasos 2 y 7 no se ejecutan y se sigue en el paso 3, 8 respectivamente.

Flujo alternativo: Cancelar

El usuario no sigue el paso 4.

1. El usuario selecciona la opción *back*.
- La aplicación muestra de nuevo el menú principal.

Flujo alternativo: Mismo idioma

En el paso 4 el usuario vuelve a seleccionar la bandera del idioma actual.

1. No sucede nada, el estado de la aplicación no cambia.

4.2.11 Activar/desactivar vibración

Breve descripción: El usuario pulsa el botón de activar/desactivar vibración en el menú de configuración.

Actor principal: Usuario

Flujo básico: Activar/desactivar vibración

1. El usuario selecciona la opción *Configuración* en el menú principal.
2. La aplicación reproduce el sonido de opción seleccionada.
3. La aplicación muestra al usuario el menú de *Configuración*.
4. El usuario pulsa si/no dependiendo de si quiere activar/desactivar la vibración.
5. La aplicación registra el cambio en la base de datos, para recordar la configuración del usuario la próxima vez que abra el juego.
6. La aplicación reproduce el sonido de opción seleccionada.
7. La aplicación muestra de nuevo el menú de *Configuración*, esta vez con el botón de la opción seleccionada pulsado.

Flujo alternativo: Sonido desactivado

Los pasos 2 y 6 no se ejecutan y se sigue en el paso 3, 7 respectivamente.

Flujo alternativo: Cancelar

El usuario no sigue el paso 4.

1. El usuario selecciona la opción *back*.
2. La aplicación muestra de nuevo el menú principal.

Flujo alternativo: La misma opción

En el paso 4 el usuario pulsa la opción que ya estaba seleccionada.

1. Se sigue en el paso 6.

4.2.12 Activar/desactivar sonido

Breve descripción: El usuario pulsa el botón de activar/desactivar sonido en el menú de configuración.

Actor principal: Usuario

Flujo básico: Activar/desactivar sonido

1. El usuario selecciona la opción *Configuración* en el menú principal.
2. La aplicación reproduce el sonido de opción seleccionada.
3. La aplicación muestra al usuario el menú de *Configuración*.
4. El usuario pulsa sí/no dependiendo de si quiere activar/desactivar el sonido.
5. La aplicación registra el cambio en la base de datos, para recordar la configuración del usuario la próxima vez que abra el juego.
6. La aplicación reproduce el sonido de opción seleccionada.
7. La aplicación muestra de nuevo el menú de *Configuración*, esta vez con el botón de la opción seleccionada pulsado.

Flujo alternativo: Sonido desactivado

Los pasos 2 y 6 no se ejecutan y se sigue en el paso 3, 7 respectivamente.

Flujo alternativo: Cancelar

El usuario no sigue el paso 4.

1. El usuario selecciona la opción *back*.
2. La aplicación muestra de nuevo el menú principal.

Flujo alternativo: La misma opción

En el paso 4 el usuario pulsa la opción que ya estaba seleccionada.

1. Se sigue en el paso 6.

5 Diseño e implementación del juego

En este apartado se explica en detalle cómo funciona internamente la aplicación. Se muestra un diagrama de clases simplificado para entender la estructura general de la aplicación, y a continuación los diagramas de secuencia que representan a los casos de uso principales del apartado anterior ([4.2 - Casos de uso](#)).

Por último se explica la metodología de desarrollo, junto con las herramientas utilizadas tanto para el diseño de los diferentes diagramas, como las que componen el entorno de desarrollo Android.

Para obtener más detalles sobre las clases principales de la aplicación, en el apartado [C - Clases del juego](#) en los apéndices se incluye una explicación detallada de estas, en el orden aproximado en que fueron programadas. Cada una de ellas con su respectivo diagrama; mostrando en este los atributos y relaciones principales de la clase correspondiente, y de cada clase relacionada, destacando aquellas funciones usadas desde la clase que se explica en ese momento.

5.1 Diagrama de clases general

En la *Figura 5.1* podemos ver el diagrama de clases general de la aplicación. Es una versión simplificada para entender la estructura básica. En los apéndices (*C - Clases del juego*) se explicarán con más detalle aquellas clases que pueden plantear alguna duda, o de las cuales se considera importante destacar alguna decisión de diseño o consejos a tener en cuenta.

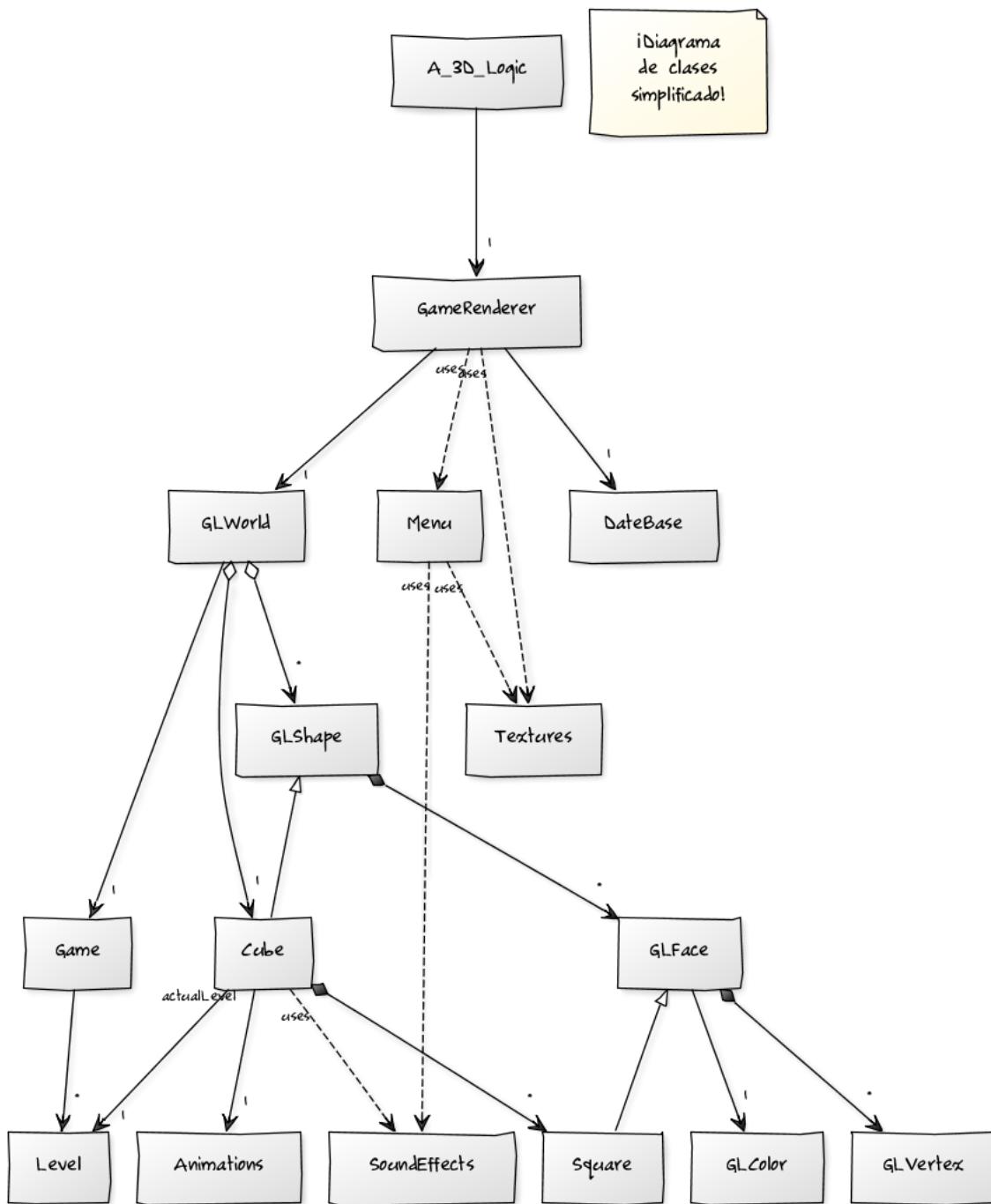


Figura 5.1 Diagrama de clases

5.2 Diagramas de secuencia

En este apartado podemos encontrar los diagramas de secuencia más relevantes de la aplicación. En ellos se muestra cómo fluye la información entre las clases dependiendo de la funcionalidad que se esté ejecutando.

Hay que tener en cuenta que el juego cuenta principalmente de dos hilos de ejecución diferentes: el que recibe las notificaciones de la interfaz de usuario, y el render gráfico automático. Y cada uno va por su cuenta. Por lo que los casos de uso que inicia el usuario se completan asíncronamente. Teniendo que esperar a que el render gráfico pinte el siguiente *frame*, en el que se le mostrarán los resultados al usuario.

Teniendo esto en cuenta, en los diagramas se ha añadido al actor *clock*, que represente al render gráfico. De esta forma se puede mostrar la secuencia desde que el usuario realiza una acción hasta que la pantalla se actualiza.

5.2.1 Interacción con el menú

A continuación (*Figura 5.2*) se muestra la secuencia de ejecución al pulsar el botón de un apartado del menú que nos lleva a otro apartado del menú. El ejemplo mostrado es el paso del menú principal al menú jugar mediante el botón *Jugar*. El diagrama sería equivalente para el resto de botones que únicamente cambian el apartado del menú: Configuración, Ayuda, Elegir Nivel, los botones de volver al menú anterior, o las diferentes páginas del apartado Elegir Nivel.

Como todos los elementos de los menús no son más que texturas, es la clase *Textures* la encargada de mostrarlos en pantalla. Los elementos a mostrar dependen del apartado en el que estemos, y sus proporciones del tamaño de la pantalla del móvil que se esté usando. Pero la secuencia de ejecución es la misma en todos los casos: el menú nos indica si se ha cambiado de apartado, y si es así reproducimos un sonido “bip” y mostramos el apartado seleccionado.

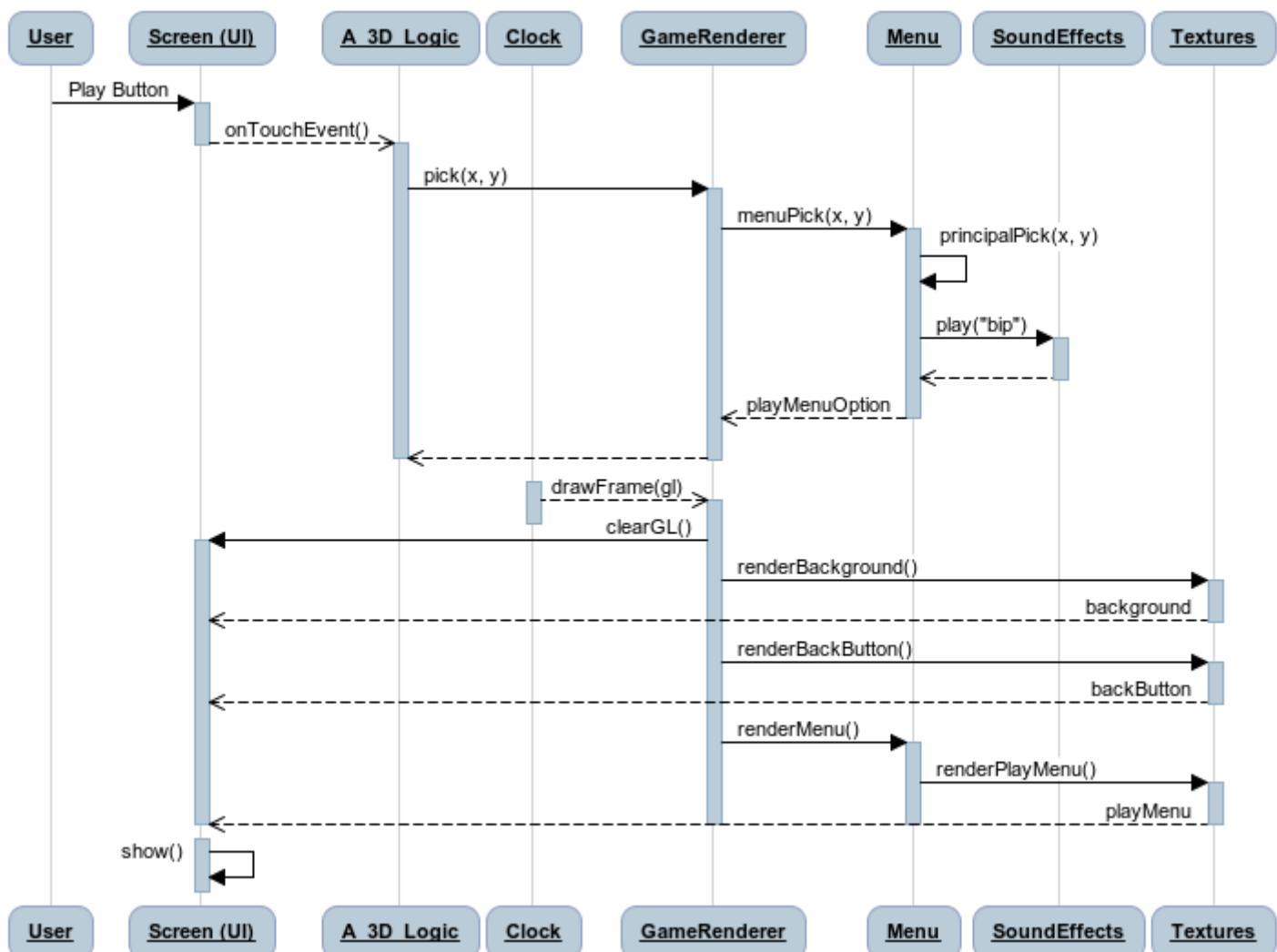


Figura 5.2 Interacción con el menú (Diagrama de secuencia)

5.2.2 Iniciar nivel

A continuación (*Figura 5.3*) se muestra la secuencia de ejecución al pulsar un botón que provoca el inicio de un nivel del juego. Esto puede suceder al pulsar el botón Continuar, Partida Nueva, o eligiendo alguno de los niveles disponibles (en verde) del apartado *Elegir Nivel*.

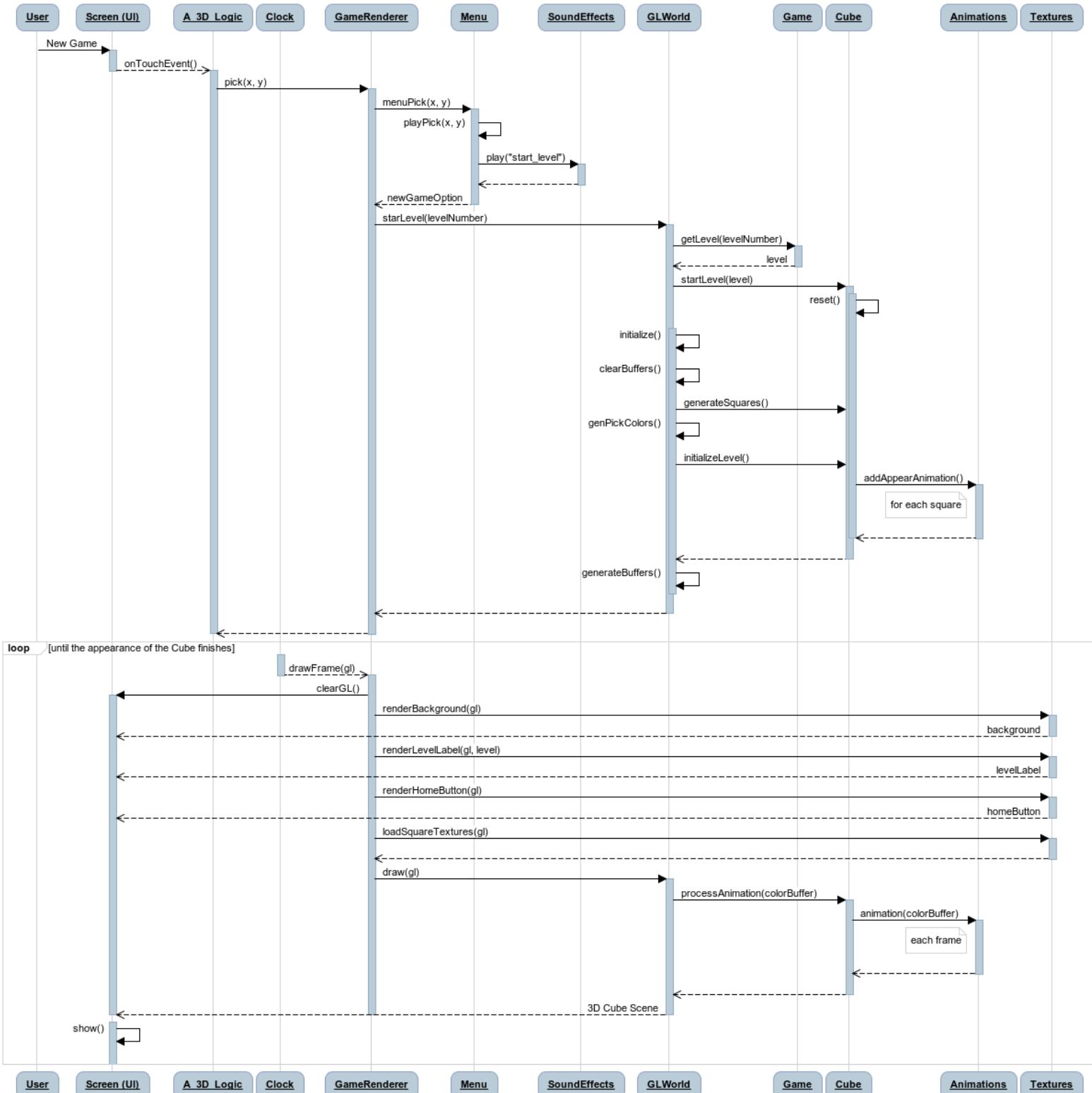


Figura 5.3 Iniciar nivel (Diagrama de secuencia)

Si el usuario elige iniciar un nivel, se ha de volver a generar la escena, ya que cada nivel es de un tamaño concreto y cada casilla puede ser de un tipo diferente en cada nivel. Para ello lo primero que se hace es obtener la configuración del nivel elegido, mediante la clase Game. Si nos fijamos en el diagrama, el proceso de inicialización cuenta de varios pasos. Tras limpiar los *buffers* de escena, llamamos a la función `generateSquares()` de la clase Cube. Esta función crea las casillas del cubo y por defecto asigna un color diferente a cada una de ellas, ignorando la configuración del nivel. Esta información nos sirve para generar el *buffer* de color especial (`generatePickColors()`) que usaremos en el algoritmo de selección de casillas. Y una vez generado, ya podemos inicializar el nivel en el cubo y finalmente generar el resto de *buffers*: vértices, texturas y colores del nivel.

Una vez preparada la escena, al siguiente *frame* procedemos a mostrarla en pantalla. Siendo el inicio de un nivel, se aplicará la animación de aparición del cubo, por lo que pasarán unos cuantos *frames* hasta que finalice la animación y podamos considerar que se ha cargado del todo el nivel elegido.

5.2.3 Pintar casilla

Cuando el usuario pinta una casilla, hay muchas alternativas. Puede simplemente pintar la casilla de un color (con su respectiva animación), cambiar de color, completar un camino, romper un camino ya completado, o superar el nivel y pasar al siguiente.

Esta es la acción principal del juego. Relacionada con la lógica de resolución de los niveles. Por lo que se dedicará especial atención a cada una de las alternativas.

En el siguiente diagrama ([Figura 5.4](#)) se muestra la secuencia básica, pintar una casilla de un color. En caso de tratarse de alguna de las alternativas, los cambios en la secuencia únicamente afectan a la función `processPick(square)` en la clase Cube. Los siguientes diagramas muestran los cambios relevantes de cada alternativa:

- Cambiar de color: [Figura 5.5](#)
- Completar un camino: [Figura 5.6](#)
- Romper un camino ya completado: [Figura 5.7](#)
- Superar el nivel: [Figura 5.8](#)

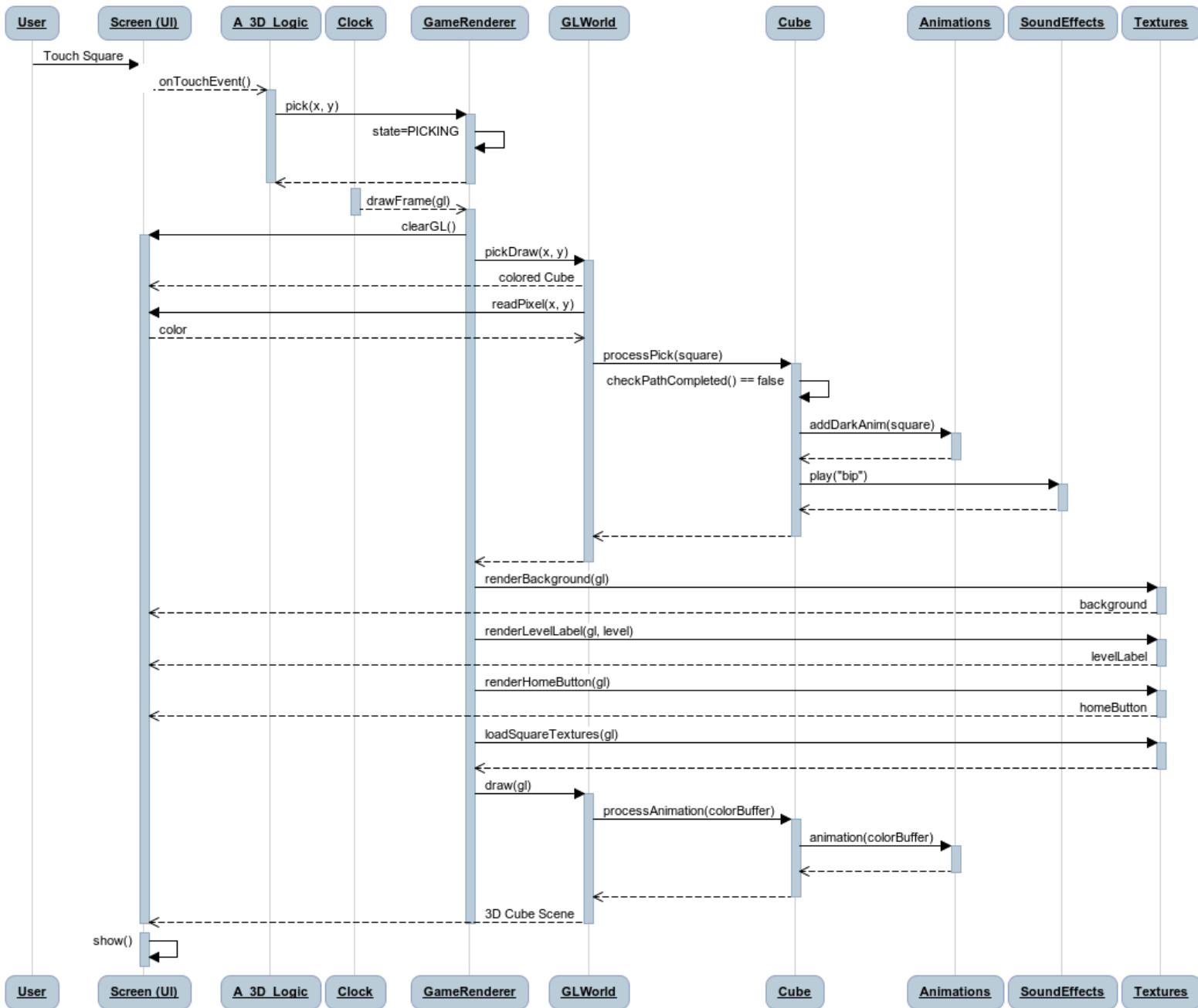


Figura 5.4 Pintar casilla (Diagrama de secuencia)

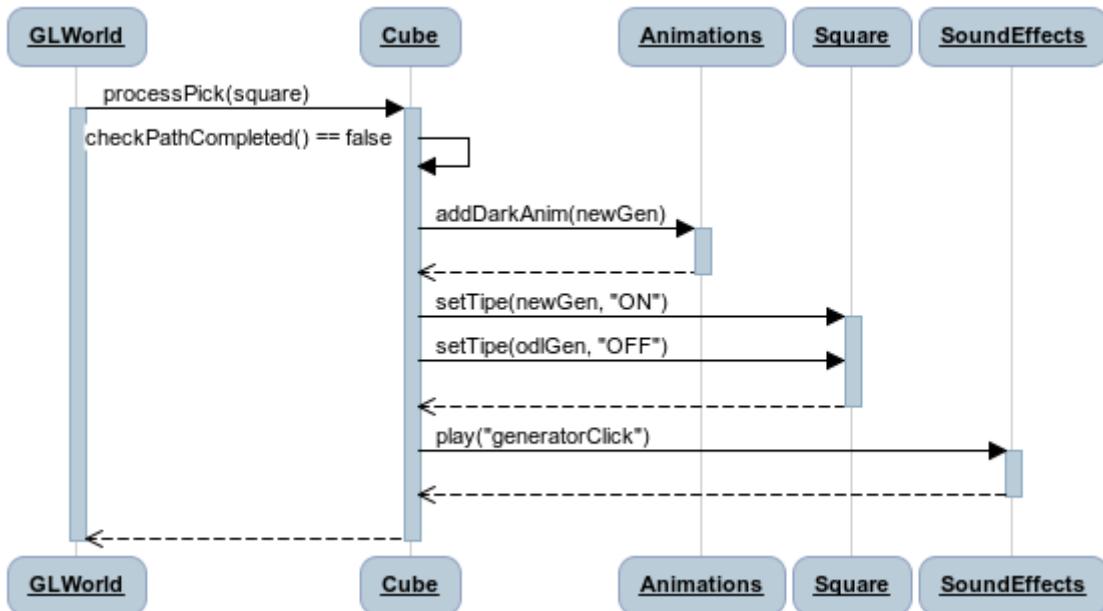


Figura 5.5 Pintar casilla (Cambiar de color)

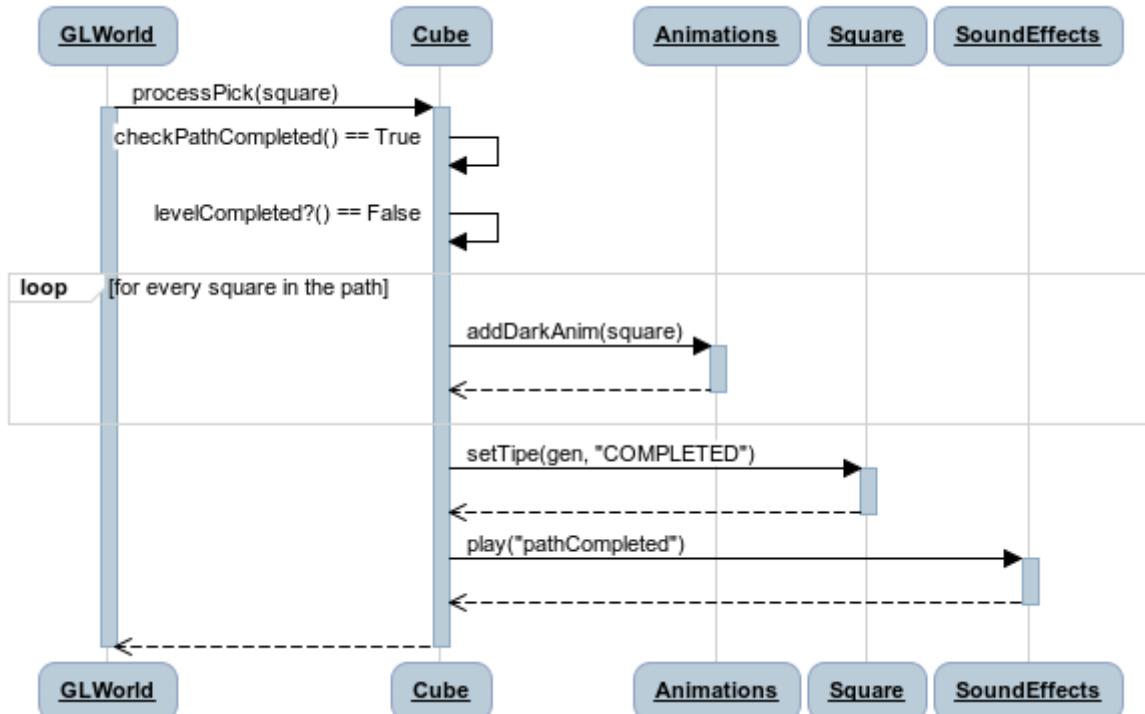


Figura 5.6 Pintar casilla (Completar camino)

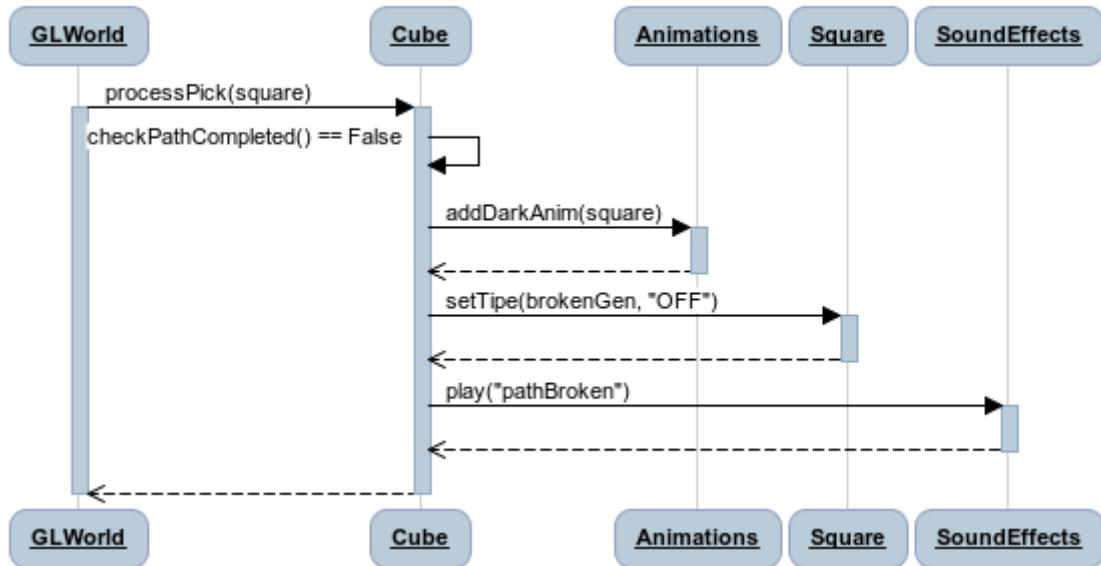


Figura 5.7 Pintar casilla (Romper camino)

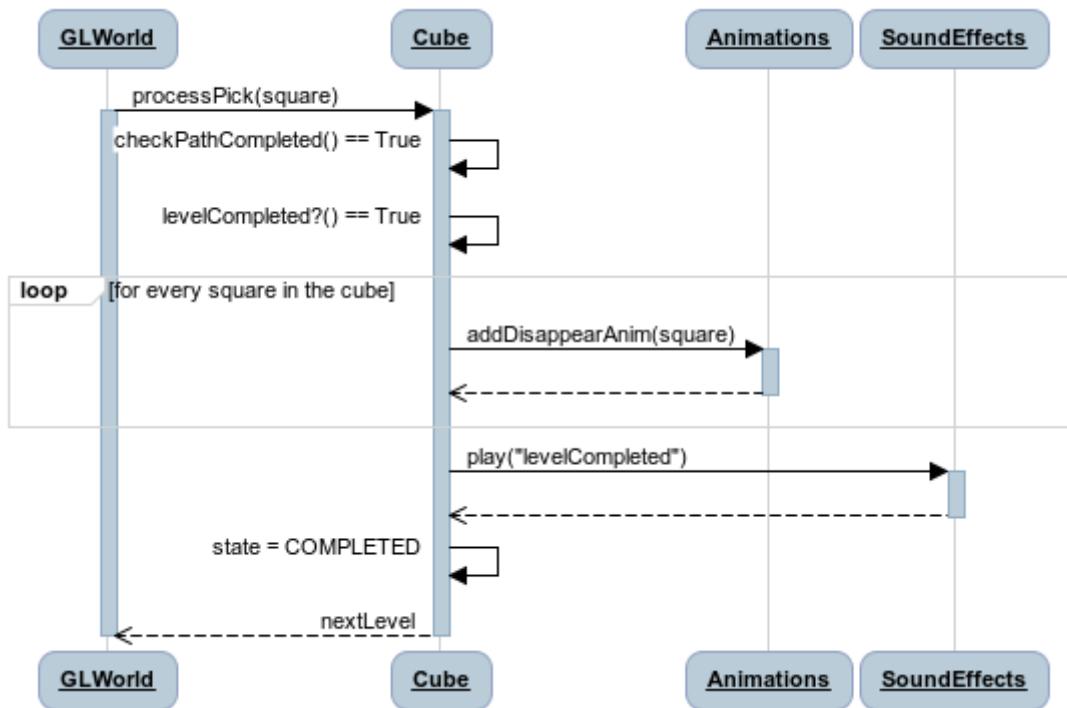


Figura 5.8 Pintar casilla (Completar nivel)

5.2.4 Siguiente nivel

En el apartado anterior hemos visto qué sucede cuando pintamos la última casilla del cubo y superamos el nivel. Pero es sólo la primera parte. En el siguiente diagrama (*Figura 5.9*) se muestra el proceso que llevan a cabo el resto de clases para cambiar de nivel.

El diagrama simplifica aquellas partes que se han visto en detalle en apartados anteriores. Y la secuencia empieza cuando recibimos la notificación de pintar un nuevo *frame*.

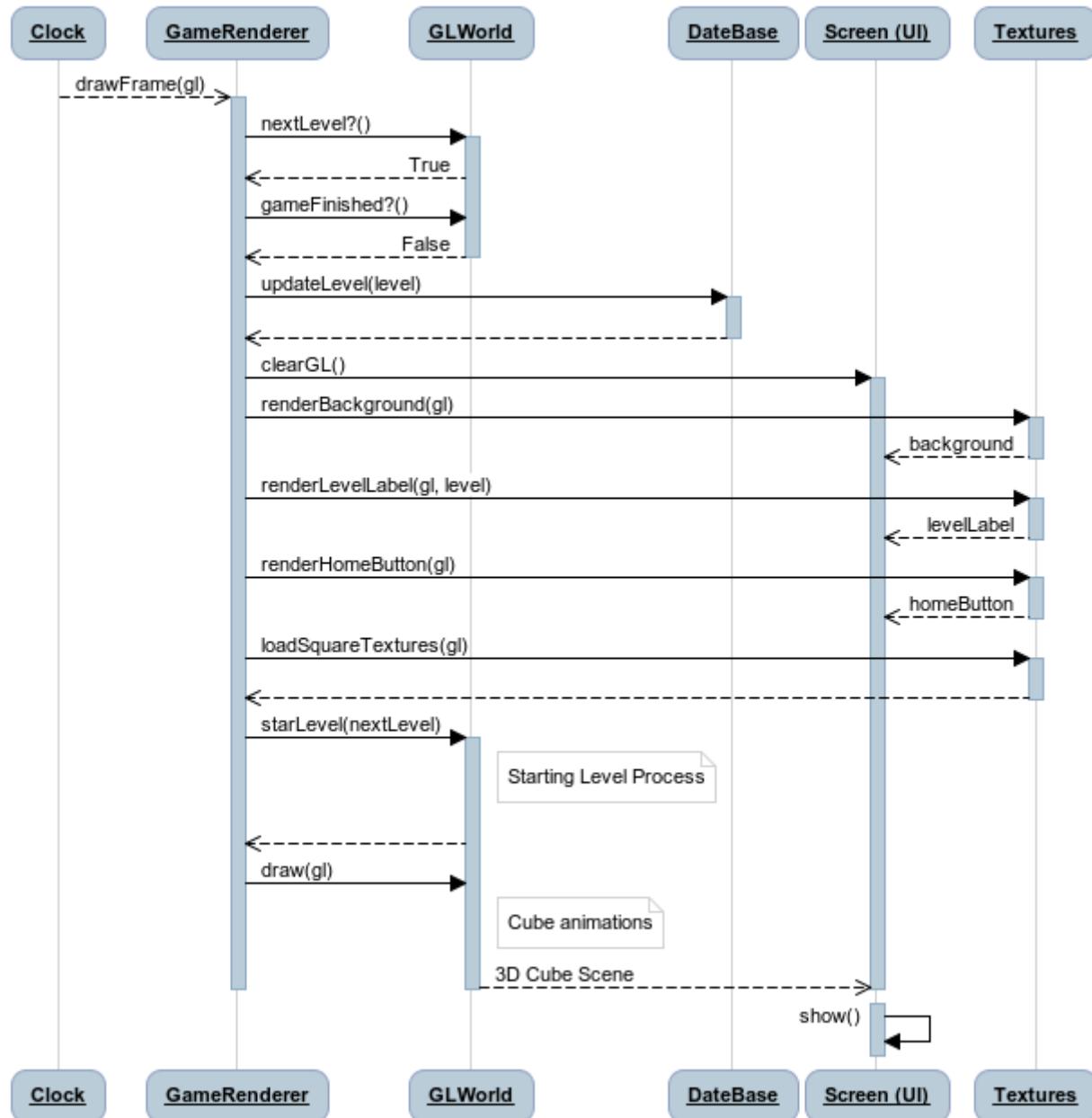


Figura 5.9 Siguiente nivel (Diagrama de secuencia)

5.2.5 Mover cubo

Si el usuario desliza el dedo por la pantalla, el cubo se mueve. Dependiendo de si pulsó dentro o fuera del cubo, también le permitirá ir pintando casillas a medida que lo nuede. El siguiente diagrama (*Figura 5.10*) muestra los cambios relevantes en cuanto al movimiento del cubo. En caso de ir pintando casillas al mismo tiempo, se ejecutaría también la secuencia descrita en el apartado 5.2.3 - *Pintar casilla*, con la excepción de que no se permitiría cambiar de color (*Figura 5.5*).

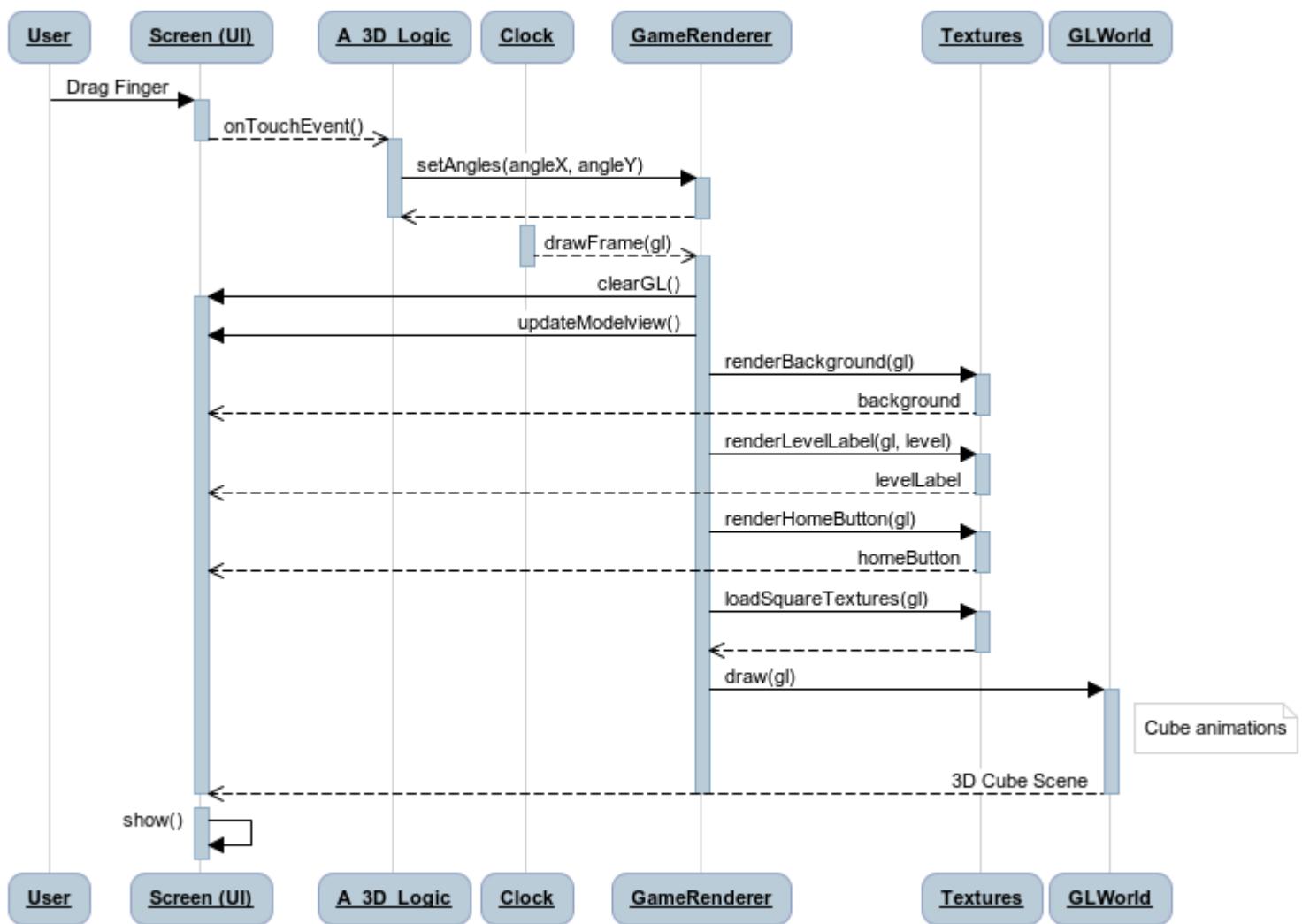


Figura 5.10 Mover cubo (Diagrama de secuencia)

5.2.6 Zoom

Si el usuario desliza dos dedos a la vez en pantalla, puede agrandar el cubo (alejar un dedo del otro) o encogerlo (acercar un dedo al otro). En este modo se bloquea el movimiento y pintado de casillas del cubo, por lo que sólo el efecto de zoom es visible.

El siguiente diagrama de secuencia (*Figura 5.11*) muestra el proceso a seguir para realizar este efecto.

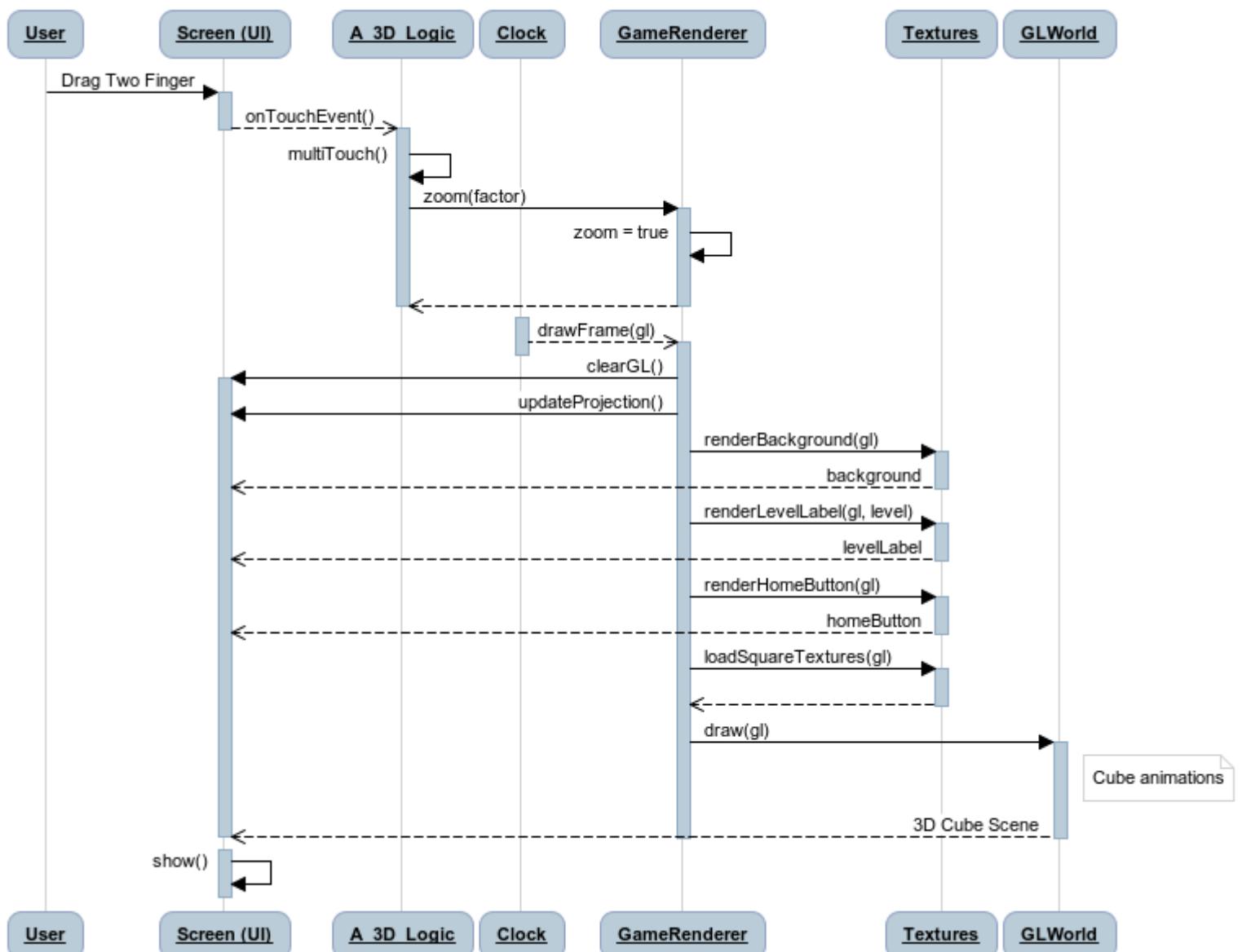


Figura 5.11 Zoom (Diagrama de secuencia)

5.2.7 Agitar móvil

A continuación (Figura 5.12) se muestra la secuencia de ejecución que sigue el juego cuando el usuario agita el móvil para volver a empezar el nivel que estaba jugando. Cabe destacar, que al tratarse del mismo nivel, no es necesario cambiar las casillas ni los vértices en escena. Únicamente es necesario resetear la posición de la cámara, los colores y animaciones, y el grado de completitud del nivel (todos los caminos incompletos).

Al finalizar la secuencia, el cubo empezará a aparecer en pantalla (animación).

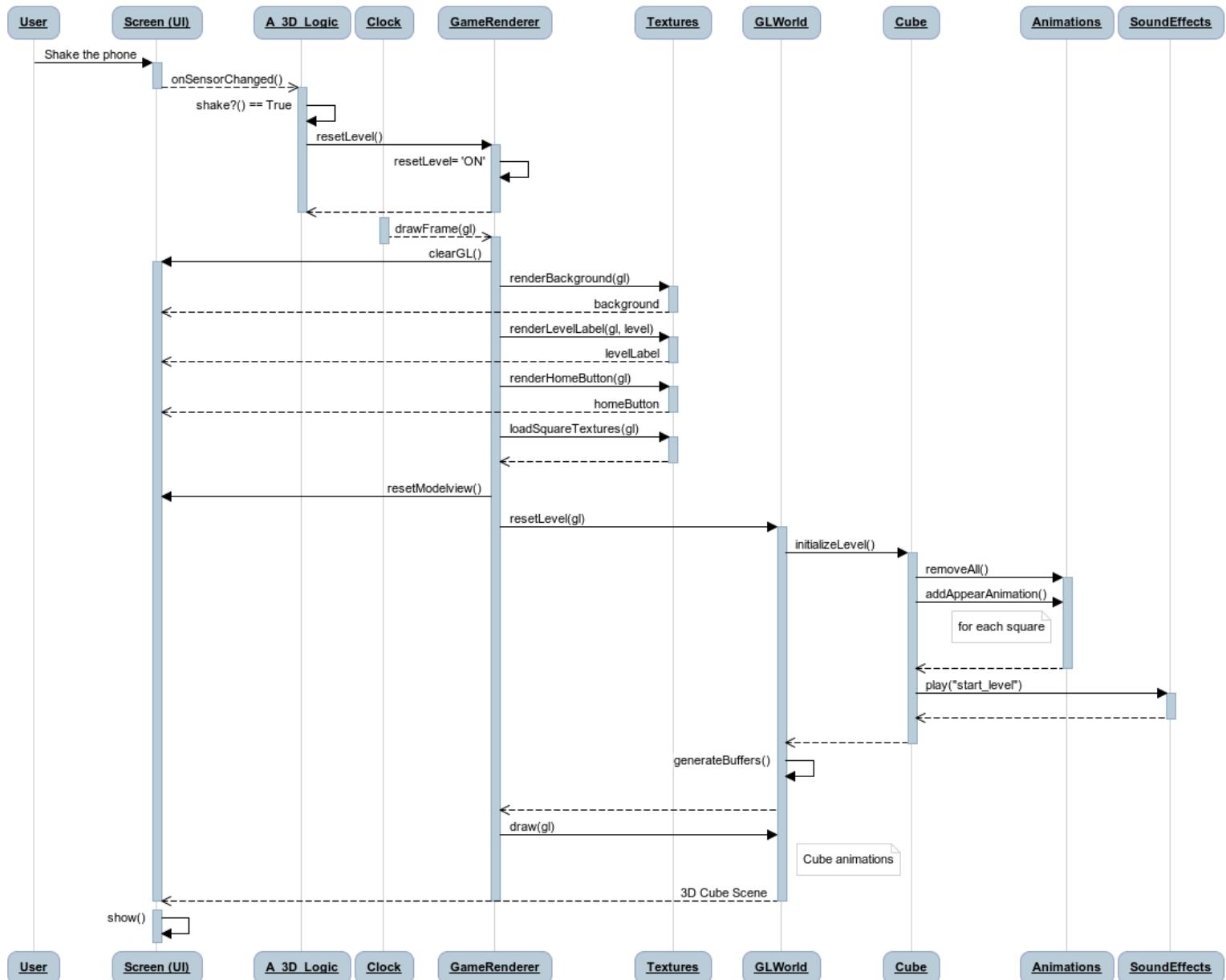


Figura 5.12 Agitar móvil (Diagrama de secuencia)

5.2.8 Cambiar configuración

Tanto al cambiar de idioma como al activar/desactivar la vibración o el sonido, la secuencia de ejecución es la misma. Y al mostrar de nuevo el menú de configuración, el único cambio, en cuanto a la ejecución, es la elección de una textura u otra para representar el idioma y estado de la vibración y sonido.

El siguiente diagrama (*Figura 5.13*) muestra la secuencia de ejecución al cambiar el idioma del juego.

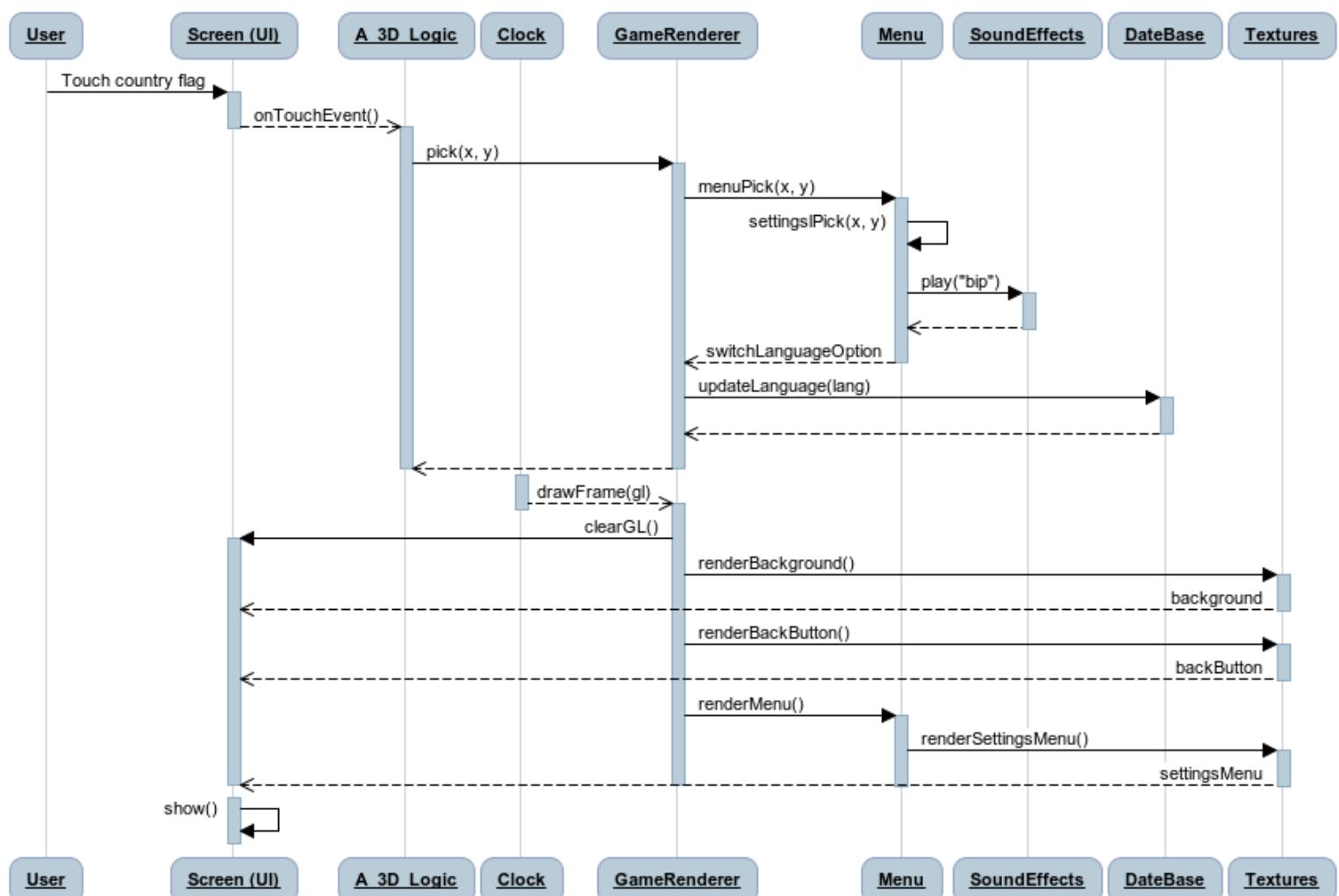


Figura 5.13 Cambiar configuración (Diagrama de secuencia)

5.2.9 Fin del juego

Este es un caso especial que se activa al superar el último nivel del juego. Es una variante de la secuencia descrita en el apartado 5.2.4 - *Siguiente nivel*, pero en este caso (*Figura 5.14*) dejamos de mostrar la escena 3D y en su lugar volvemos al menú. Dependiendo de si es la primera vez o no que superamos el juego, la información a mostrar en la pantalla final es diferente, avisando de la disponibilidad del nuevo modo espejo, o no.

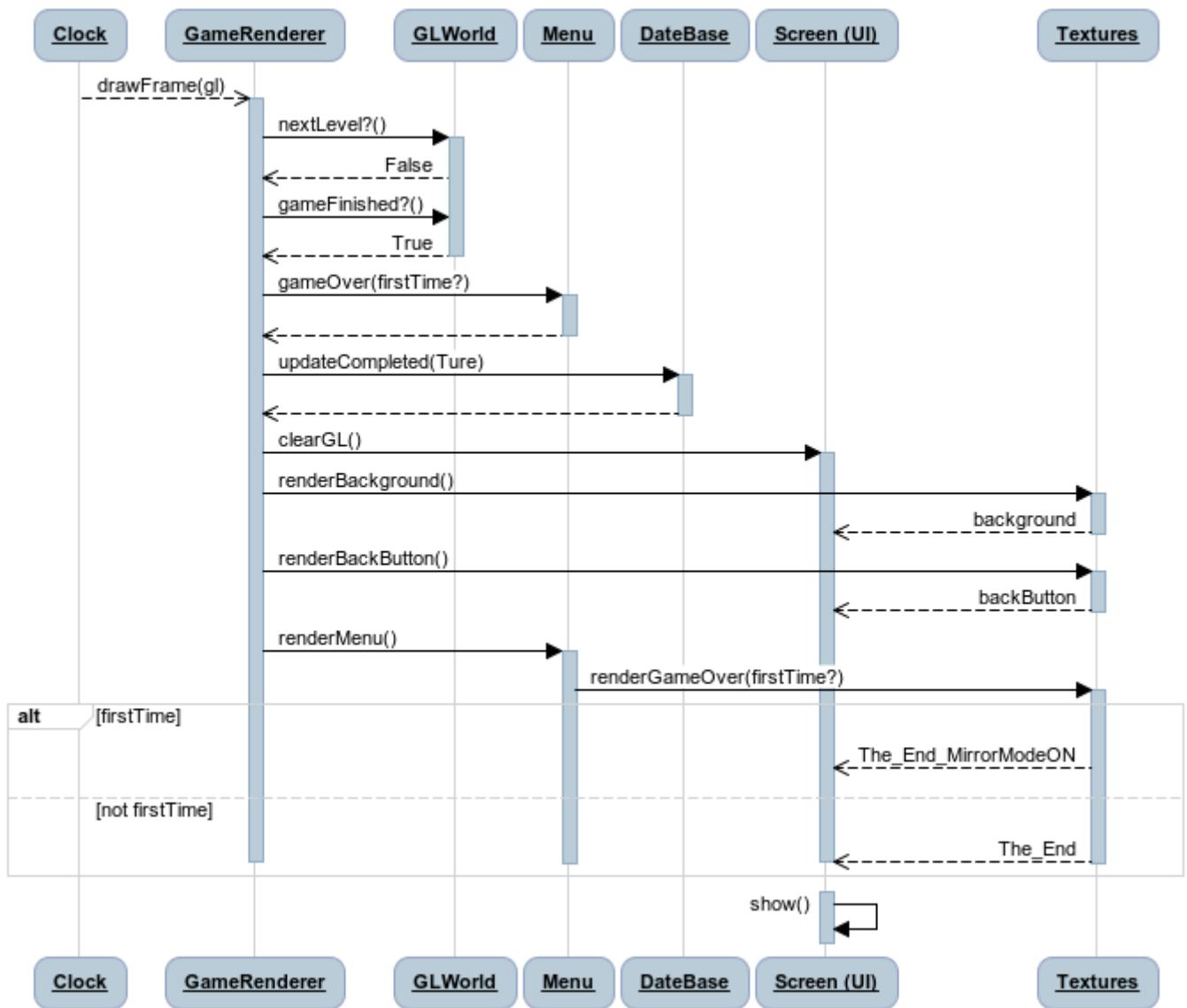


Figura 5.14 Fin del juego (Diagrama de secuencia)

5.3 Metodología de desarrollo

El primer paso que seguí para empezar a desarrollar el juego fue probar los códigos de ejemplo que proporciona Android (las API Demos [24]). Los ejemplos abarcan prácticamente cualquier funcionalidad que un desarrollador quiera usar, por lo que son totalmente recomendables y por lo tanto los he consultado en repetidas ocasiones a lo largo del desarrollo. Para ver por ejemplo cómo cargar y mostrar texturas ([3.6 - Texturas](#)), permitir objetos translúcidos en escena, gestión simple de la pantalla táctil ([3.5 - Pantalla táctil](#)), etc.

Entre el apartado dedicado a los gráficos y OpenGL, encontré el ejemplo *Kube* [25], que muestra la estructura básica de un proyecto OpenGL en Android; la actividad principal, que crea un GLSurfaceView ([3.4 - GLSurfaceView](#)) para gestionar OpenGL; y las diferentes clases básicas para mostrar una escena 3D: GLWorld (escena 3D, [C.3](#)), GLShape, GLFace, GLVertex y GLColor. Así que basándome en estas clases empecé a desarrollar el juego.

Como se muestra en el diagrama de clases completo ([Figura 5.1](#)), he conservado el nombre de las clases originales, a excepción de la actividad principal A_3D_Logic ([C.1](#)) y el render GameRenderer ([C.2](#)), que también creé inicialmente basándome en el código de los ejemplos. Aunque al final todas ellas han sido ampliadas para cumplir con las necesidades del juego.

Una vez disponible la estructura básica, la primera etapa consistió en ampliar el código para conseguir mostrar mi propio cubo personalizado en pantalla, ya que el del ejemplo estaba construido de una forma diferente a lo que yo necesitaba. Así que creé las clases Cube ([C.4](#)) y Square ([C.5](#)), que son la extensión de GLShape y GLFace respectivamente, adaptadas a las necesidades del juego. Lo que sí aprovechar es la forma de renderizar la escena, usando *VertexArray*¹⁸ para vértices, texturas (añadidas) y colores.

De esta primera etapa, la primera dificultad a destacar fue la decisión de representar las casillas con texturas en lugar de simples colores (y hacerlo además con cierta transparencia), ya que tuve que probar diferentes configuraciones y códigos hasta comprobar qué se podía y qué no se podía hacer mezclando texturas y colores.

El segundo reto fue conseguir interactuar con el cubo. En concreto el algoritmo de selección de casillas, ya que OpenGL ES 1.0 no proporciona un método sencillo para realizar la selección de objetos.

¹⁸ Ejemplo de vertex array: http://www.songho.ca/opengl/gl_vertexarray.html

Finalmente decidí usar el algoritmo de *selección por color*¹⁹, que se muestra más detalladamente en el apartado *Código C.4 Algoritmo de selección (por color)* de los apéndices. En resumen consiste en pintar cada objeto de un color diferente cuando el usuario pulsa la pantalla, y dependiendo del color del pixel pulsado, conoceremos el objeto seleccionado. La teoría es muy sencilla, pero a la práctica la obtención del color del píxel pulsado es aproximada, y para colmo, cuando conseguí que funcionara en el emulador, al probarlo en un móvil real comprobé que cada dispositivo tiene una aproximación y rango de error diferente, por lo que acabar de ajustar correctamente los parámetros del algoritmo llevó un tiempo considerable.

Y hasta aquí se puede considerar que duró la primera etapa del desarrollo. Conseguir mostrar un cubo en pantalla, y poder pulsar las diferentes casillas.

La siguiente etapa consistió en añadir la lógica del juego, por lo que creé las clases Game(*C.6*) y Level(*C.7*); Game básicamente gestiona los diferentes niveles (*Level*) del juego. Y añadí a la clase Cube (*C.4*) la información necesaria para poder resolver los niveles. Incluido el algoritmo de resolución; un simple DFS²⁰ que busca una conexión entre los generadores del color seleccionado (para más detalle consultar el *Código C.6* de los apéndices).

En esta etapa me topé con los problemas que puede ocasionar no tener en cuenta que el hilo de ejecución de la interfaz de usuario no ha de interferir en las acciones del hilo de ejecución del render gráfico automático. La simple modificación de un atributo compartido puede ocasionar problemas muy difíciles de encontrar, por lo que hay que ir con mucho cuidado.

También en esta etapa empecé a tratar con cuidado la interfaz de usuario, por lo que añadí efectos sonoros al interactuar con el cubo mediante la clase SoundEffects(*C.10*); animaciones de tratamiento de color al pintar casillas, completar caminos, niveles, etc, con la clase Animations (*C.9*); y cree la clase Textures (*C.8*) para empezar a mostrar fondos de pantalla, indicador de nivel y otros elementos que iba diseñando.

La tercera etapa consistió en crear los diferentes menús del juego, tanto de opciones como de configuración, por lo que creé la clase Menu (*C.11*), que se encarga principalmente de gestionar la selección de opciones de los menús; y DataBase, para recordar el estado del juego en la memoria del teléfono.

¹⁹ Ejemplo colorPick: <http://www.lighthouse3d.com/opengl/picking/index.php3?color1>

²⁰ Depth-first search: http://en.wikipedia.org/wiki/Depth-first_search

La dificultad en esta etapa fue más una cuestión de diseño de la interfaz, que de desarrollo en sí. Sin embargo, llegar a una configuración de color compatible con la mayoría de dispositivos móviles (configuración final: [Código C.1](#)), se puede considerar uno de los problemas más difíciles de solucionar del proyecto (ver el apartado [7 - Pruebas](#) para más detalles).

Finalmente, la última etapa antes de las pruebas finales consistió en revisar el diseño inicial y añadir algunas funcionalidades. Como por ejemplo la posibilidad de agrandar el cubo, facilitando así el pintado de las casillas a las personas con manos grandes o móviles pequeños; la posibilidad de reiniciar un nivel agitando el móvil, por si el usuario se ha liado tanto pintado el cubo que prefiere volver a empezar; la extensión de la vida del juego mediante el modo espejo, cuyo algoritmo resumido se encuentra en el apartado [Cálculo de nivel espejo](#) de los apéndices; y por último el diseño de los últimos niveles del juego, mediante modificaciones de los niveles de los juegos originales ([2.2 - 3D Logic](#)).

5.4 Herramientas de diseño y desarrollo

A continuación describiré brevemente las herramientas y el entorno utilizado para llevar a cabo el diseño y desarrollo del proyecto.

En el apartado [A - Instalación del entorno de desarrollo](#) se puede encontrar una guía de instalación de los principales programas y *plugins* necesarios para empezar a desarrollar aplicaciones para Android. Ya que hay bastantes pasos a seguir y puede resultar complicado la primera vez.

5.4.1 Eclipse



Eclipse²¹ es, junto a NetBeans, uno de los entornos de desarrollo de software multiplataforma más conocidos y utilizados actualmente.

Eclipse está compuesto por un entorno de desarrollo integrado (IDE) y un sistema de plugins extensible, por lo que es posible desarrollar aplicaciones en prácticamente cualquier lenguaje de programación con un gran número de herramientas a nuestro alcance.

El motivo de escoger Eclipse frente a NetBeans fue básicamente por ser el que ofrece soporte oficial al kit de desarrollo de software de Android (Android SDK).

5.4.2 Android Development Tools Plugin (ADT)



Android Development Tools (ADT)²² es un *plugin* para Eclipse diseñado para proporcionarnos un poderoso entorno de desarrollo integrado para desarrollar aplicaciones de Android.

ADT extiende las capacidades de Eclipse para permitirnos crear nuevos proyectos de Android rápidamente, crear la interfaz gráfica, añadir componentes basados en la API de Android, depurar nuestras aplicaciones usando las herramientas de Android SDK e incluso exportar ficheros para distribuir nuestra aplicación.

²¹ Web Eclipse: <http://www.eclipse.org/>

²² Web Android Development Tools: <http://developer.android.com/sdk/eclipse-adt.html>

5.4.3 Android SDK

El kit de desarrollo de software de Android (Android SDK)²³ incluye un grupo de herramientas de desarrollo indispensable para la implementación de aplicaciones para dispositivos móviles Android.

El kit incluye un depurador, librerías, un emulador, documentación, código de ejemplo y tutoriales.



Figura 5.15 Emulador Android

5.4.4 Control de versiones

Apache Subversion²⁴ (habitualmente abreviado SVN) es un sistema de control de versiones de software, utilizado por los desarrolladores para mantener versiones actuales e históricas de archivos como código fuente, páginas web y documentación.

He utilizado esta herramienta de control de revisión, frente a otras, por ser la que conozco mejor. En concreto he usado los repositorios gratuitos que ofrece ProjectLocker²⁵.



²³ Web Android SDK: <http://developer.android.com/sdk/index.html>

²⁴ Web SVN: <http://subversion.apache.org/>

²⁵ Web de ProjectLocker: <http://projectlocker.com/>

5.4.5 Diagramas

Para crear los diferentes diagramas mostrados en el proyecto he usado las siguientes herramientas online:

- **yUML²⁶**: Permite crear diagramas UML²⁷ no demasiado complejos. La aplicación interpreta órdenes en texto plano para generar el diagrama. Se ha usado para crear el diagrama de clases general (*Figura 5.1*) y los individuales (*C - Clases del juego*). A modo de ejemplo, se han adjuntado las ordenes textuales de los diferentes diagramas en el apartado *E.1 - yUML*.



- **Cacoo²⁸**: Es una aplicación que permite crear una gran variedad de diagramas, como mapas, UML, interfaces web, etc. Tiene una versión gratuita y otra de pago con más opciones. Se ha usado para diseñar los diagramas de la interfaz de usuario (*6 - Diseño de la interfaz de usuario*).



- **LucidChar²⁹**: Al igual que Cacoo, nos permite crear una gran variedad de diagramas diferentes. Se ha usado para crear el diagrama de casos de uso (*Figura 4.1*) y de navegabilidad (*Figura 6.9*).



- **WebSequenceDiagrams³⁰**: De igual forma que yUML, nos permite crear diagramas mediante órdenes en texto plano. En este caso permite crear únicamente diagramas de secuencia (un tipo de diagrama que no ofrece yUML). Personalmente prefiero este tipo de aplicaciones, que generan los diagramas automáticamente, ya que cualquier

²⁶ Aplicación yUML: <http://yuml.me/>

²⁷ Unified Modeling Language: http://en.wikipedia.org/wiki/Unified_Modeling_Language

²⁸ Web de cacoo: <https://cacoo.com/>

²⁹ Web de LucidChart: <http://www.lucidchart.com/>

³⁰ Aplicación WebSequenceDiagrams: <http://www.websequencediagrams.com/>

modificación en un diagrama creado manualmente con Cacoo o Lucidchart, por ejemplo, implica retocar y mover elementos que han quedado descuadrados. Los diagramas autogenerados son menos configurables, pero son más fáciles y rápidos de usar.

Todos los diagramas de secuencia del proyecto ([5.2 - Diagramas de secuencia](#)) se han creado con esta aplicación. Y por si algún diagrama no se visualizase correctamente debido a la compresión de la imagen en el documento, se adjuntan las órdenes de creación en el apartado [E.2 - Web Sequence Diagrams de los apéndices](#).



5.4.6 Google Docs

Google Docs³¹ es un paquete de programas ofimáticos gratuitos de Google. Está basado en Web y permite crear documentos en línea. Incluye un procesador de textos, hojas de cálculo, presentaciones y formularios entre muchas otras funcionalidades. Mediante los formularios he podido hacer encuestas a amigos para elegir los diferentes elementos de la interfaz de usuario.



³¹ Web Google Docs: <http://docs.google.com/>

6 Diseño de la interfaz de usuario

Al tratarse de un juego, la interfaz de usuario es uno de los puntos más importantes. Si no se diseña adecuadamente, el juego no será usable, entorpeciendo la interacción con el usuario y causando que no genere interés.

Este apartado muestra todas las pantallas accesibles en el juego a modo de manual de usuario de la interfaz. En las imágenes se numeran los elementos importantes en pantalla, y por cada uno de ellos se ofrece una breve explicación. Además, para cada una de las pantallas se muestran las navegabilidades parciales a otras pantallas, y al final podemos encontrar el diagrama de navegabilidad completo.

En el último apartado se explica el proceso que se ha llevado a cabo para poner a prueba la interfaz de usuario, así como los elementos que la componen.

Una parte importante del tiempo dedicado al proyecto ha sido diseñar los diferentes elementos gráficos del juego. Por lo que se ha dedicado un apartado especial en los apéndices (*D - Creación de los elementos de la interfaz*) en el que se explican las técnicas básicas utilizadas para la creación de los mismos, junto con algunos diseños que finalmente se descartaron por votación popular.

6.1 Menú Principal

Esta es la primera pantalla que muestra el juego (*Figura 6.1*). Es el punto de origen desde el que podemos navegar a los diferentes submenús del juego. Además, a modo de acceso rápido, se añade el botón *Continuar*, que permite al usuario empezar a jugar desde donde lo dejó la última vez (sin necesidad de navegar entre los menús).

Al tratarse de un juego cuyo propósito es completar caminos de diferentes colores, se ha optado por usar la misma filosofía en el resto de elementos del juego. Por lo tanto, el logo y las diferentes opciones de los menús cuentan con colores llamativos acordes a los que aparecen en el cubo. Exceptuando los fondos de pantalla, que muestran colores suaves para que no interfieran demasiado con los colores del cubo, que es donde los usuarios han de centrar su atención, y un fondo llamativo podría llegar a ser molesto a la vista.

Los elementos numerados en pantalla son los siguientes:

- 1) Botón de acceso al *Menú Jugar*
- 2) Botón de acceso directo al último nivel jugado por el usuario.
(*Pantalla de juego*)
- 3) Botón de acceso al *Menú Configuración*
- 4) Botón de acceso al *Menú Ayuda*



Figura 6.1 Menú Principal

6.2 Menú Jugar

Esta pantalla (*Figura 6.2*) muestra los diferentes modos de juego disponibles. Exceptuando *Continuar*, que al tratarse de la opción que probablemente más se use, se incluye en el menú principal a modo de acceso rápido.

A esta pantalla se accede desde el *Menú Principal*. Junto con este, son los dos apartados principales del menú, y recogen todas las opciones del juego. Como vemos, se usa el mismo diseño que en el menú principal.

Los elementos numerados en pantalla son los siguientes:

1. Botón de acceso al primer nivel del juego. Este modo de juego ignora si está activado o no el modo espejo. Permitiendo jugar los niveles en su forma original. (*Pantalla de juego*)
2. Botón de acceso al *Menú Elegir Nivel*
3. Botón de acceso a un nivel al azar de entre los ya superados. (*Pantalla de juego*)
4. Botón de retorno al menú anterior. (*Menú Principal*)



Figura 6.2 Menú Jugar

6.3 Menú Configuración

Desde este menú (*Figura 6.3*) el usuario puede cambiar ciertos parámetros del juego. En concreto, el juego permite cambiar el idioma de los diferentes menús, además de dar la opción de activar o desactivar la vibración y efectos de sonido con tal de ahorrar la batería del teléfono móvil.

La configuración elegida por el usuario se registra en la memoria del teléfono, por lo tanto, cuando el usuario vuelva a abrir el juego otro día, se mantendrá la última configuración elegida.

Los elementos numerados en pantalla son los siguientes:

- 1) Configuración de idioma: Actualmente el juego permite cambiar entre los idiomas Español e Inglés.
- 2) Activar / desactivar vibración: Al finalizar cada nivel del juego, el teléfono móvil vibra durante aproximadamente 1 segundo. Esta opción se puede desactivar si no se desea gastar batería con esa funcionalidad.
- 3) Activar / desactivar sonido: A pesar de poder anular el sonido bajando el volumen de la aplicación a 0 (con los botones del teléfono), de esa forma se verían afectadas el resto de aplicaciones, por lo tanto, el juego permite desactivar los efectos sonoros si el usuario así lo desea.
- 4) Botón de retorno al menú anterior. (*Menú Principal*)



Figura 6.3 Menú Configuración

6.4 Menú Ayuda

Esta pantalla (*Figura 6.4*) es una guía rápida de uso del juego. En ella se explica la lógica básica del juego para completar un nivel, además de las funcionalidades difíciles de descubrir sin explicación previa, que en concreto son la posibilidad de reiniciar un nivel al agitar el móvil durante un corto periodo de tiempo; y la posibilidad de modificar el tamaño del cubo para facilitar el pintado de las casillas.

En resumen, la lógica de un nivel del juego es la siguiente:

Para superar un nivel hay que completar todos los caminos de colores. Y por cada camino, se ha de pulsar primero uno de los generadores del color deseado (las casillas con un androide en el centro), y a continuación pintar (ya sea de una en una o deslizando el dedo por encima de unas cuantas) un camino que une los dos generadores de ese color.

Los elementos numerados en pantalla son los siguientes:

- 1) Botón de retorno al menú anterior. (*Menú Principal*)



Figura 6.4 Menú Ayuda

6.5 Menú Elegir Nivel

Este menú (*Figura 6.5*), como su propio nombre indica, permite al usuario elegir el nivel que desea jugar. Hay que tener en cuenta que sólo se pueden seleccionar aquellos niveles que ya se han superado en el modo de juego normal.

El juego cuenta actualmente con 70 niveles diferentes. Del 1 al 15 de nivel 3; del 16 al 30 de nivel 4; del 31 al 50 de nivel 5; y del 51 al 70 de nivel 6.

En caso de haberse superado el juego, estará activo el modo espejo. Esto significa que cada nivel puede mostrarse de 5 formas diferentes, al girar o alternar las caras que componen el cubo. También cambiará de forma aleatoria la posición de cada color, para dificultar así que el usuario recuerde con exactitud la forma de resolver dicho nivel, y le siga suponiendo un reto el seguir jugando a los mismos niveles.

Los elementos numerados en pantalla son los siguientes:

- 1) Niveles disponibles (previamente superados). Se muestran de color verde y son los únicos que se pueden seleccionar.
- 2) Niveles bloqueados (aún no superados). Se muestran de color rojo y no pueden ser seleccionados.
- 3) Botón de acceso a la página anterior: Nos permite ver los niveles anteriores a los mostrados en la página actual. En caso de estar en la primera página, este botón no se muestra.
- 4) Botón de acceso a la página siguiente: Nos permite ver los niveles siguientes a los mostrados en la página actual. En caso de estar en la última página, este botón no se muestra.
- 5) Botón de retorno al menú anterior. (*Menú Jugar*)



Figura 6.5 Menú Elegir Nivel

6.6 Pantalla de juego

Esta es la pantalla más importante. Aquí se lleva a cabo la lógica del juego. Es la pantalla del juego en sí (*Figura 6.6*), con la que interactúa el usuario para superar los diferentes niveles.

Los elementos numerados en pantalla son los siguientes:

- 1) Casilla vacía: Se muestran de color blanco y puede ser pintada.
- 2) Generador inactivo: Así es como se representan los generadores de color inactivos en ese momento (androide translúcido en el centro, con borde de color sólido).
- 3) Camino completado: Los generadores de color de aquellos caminos ya completado se muestran de esta forma (androide de color sólido con borde translúcido).
- 4) Generador activo: El generador activo (androide con contorno translúcido) representa el color con el que estamos pintando actualmente
- 5) Casilla bloqueada: Este tipo de casilla no puede ser pintada.
- 6) Casilla pintada: Al pulsar una casilla (o deslizar el dedo sobre esta) se pinta del mismo color que el generador activo. Al pintar la última casilla del último camino, pasamos de nivel. Y si es el último nivel del juego, aparece la pantalla de *Fin del juego*.
- 7) Indicador del nivel que estamos jugando actualmente.
- 8) Fondo de pantalla: Diferente según el nivel de dificultad (3-6). Si pulsamos inicialmente aquí para mover el cubo, las casillas no se pintarán a medida que deslizamos el dedo sobre ellas.
- 9) Botón de retorno al *Menú Principal*.
- 10) Si el móvil tiene *trackball*, se puede usar para mover el cubo.

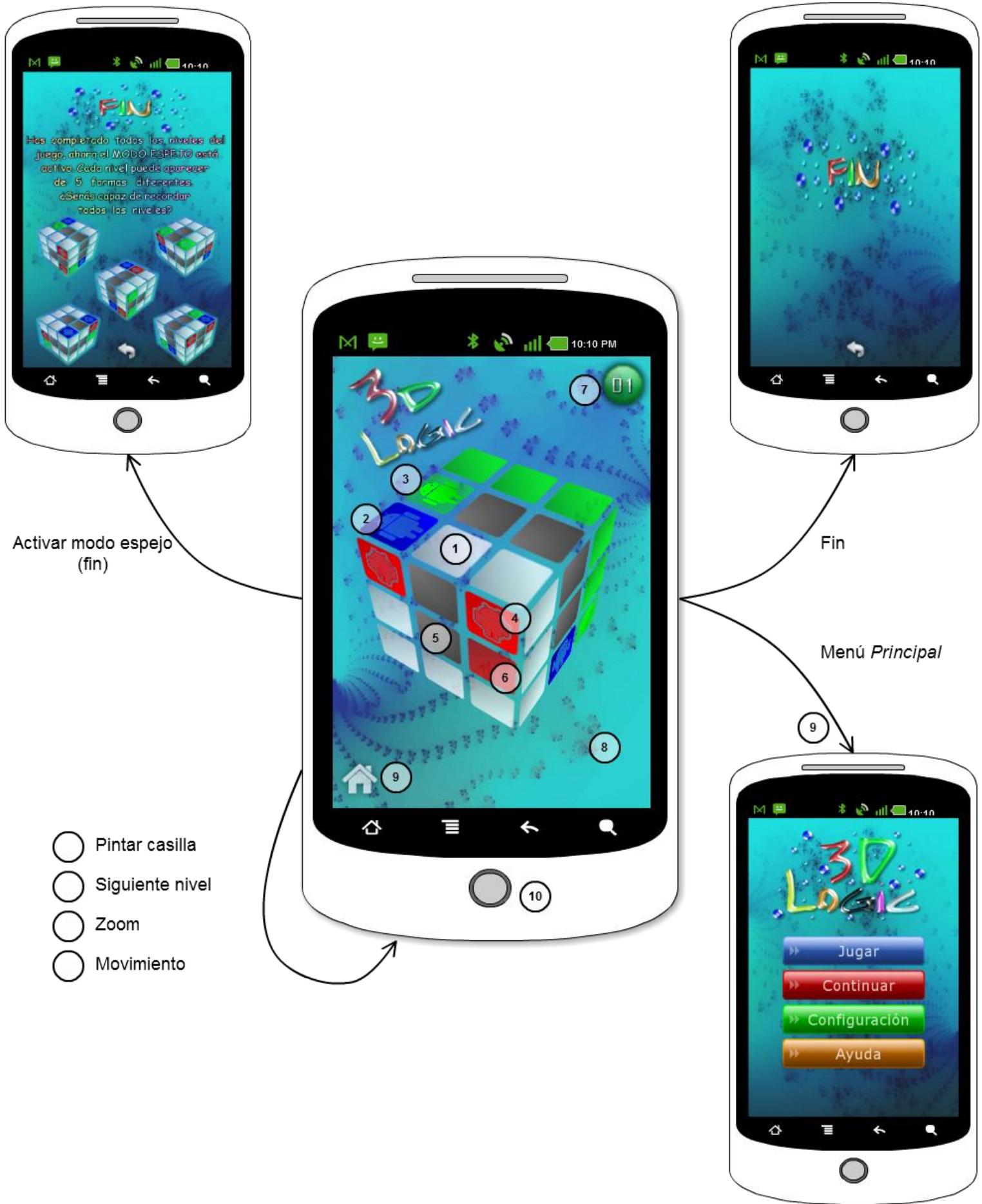


Figura 6.6 Pantalla de juego

6.7 Fin del juego

Al finalizar el juego nos pueden aparecer dos pantallas diferentes dependiendo de si es o no la primera vez que lo finalizamos.

En caso de ser la primera vez, la pantalla de fin de juego nos notifica que se ha activado el modo espejo (*Figura 6.7*).

Los elementos numerados en pantalla son los siguientes:

- 1) Notificación y ejemplo del modo espejo: La pantalla muestra las 5 posibilidades con el primer nivel del juego.
- 2) Botón de retorno al *Menú Principal*.

Si en cambio ya habíamos superado el último nivel del juego previamente, la pantalla final sólo indica el final del juego (*Figura 6.8*).

Los elementos numerados en pantalla son los siguientes:

- 1) Botón de retorno al *Menú Principal*.



Figura 6.7 Fin del juego (modo espejo)



Figura 6.8 Fin del juego

6.8 Diagrama de navegabilidad completo

A continuación se muestra el diagrama de navegabilidad completo (*Figura 6.9*). Es la unión de todas las navegabilidades parciales mostradas previamente en las pantallas del juego. De esta forma es más fácil entender las diferentes formas de acceder a cada apartado del juego.

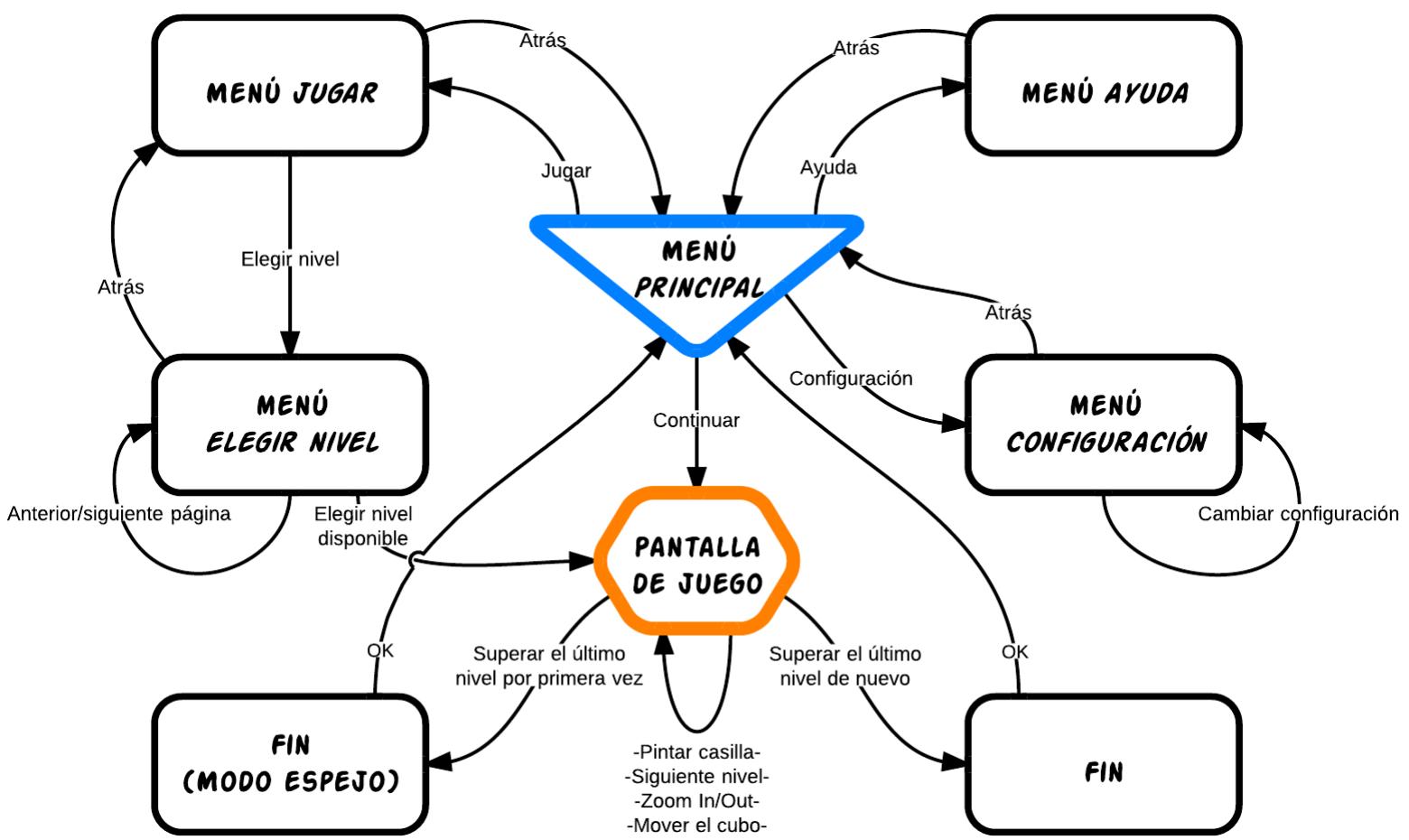


Figura 6.9 Diagrama de navegabilidad

6.9 Testeando la interfaz de usuario

Haciendo uso de los formularios de Google Docs ([5.4.6 - Google Docs](#)) realicé encuestas³² a diferentes compañeros de universidad, con el fin de que me ayudaran a elegir ciertos elementos de la interfaz gráfica.

Además, conté con la colaboración de dos compañeros con teléfonos móviles Android, que me iban reportando su opinión a medida que añadía los diferentes elementos definitivos.

Por último, una vez dispuse del Nexus One que me prestó la universidad, empecé a mostrar a familiares y amigos la aplicación para comprobar si tenían algún problema con la interfaz de usuario. De esta forma llegué a la conclusión de que se debía añadir la opción de agrandar el cubo, ya que a algunas personas les costaba pulsar una casilla concreta con el pulgar cuando el cubo era de tamaño 5x5x5 o 6x6x6.

³² Ejemplo de encuesta, elección del logo:
<https://docs.google.com/spreadsheets/embeddedform?formkey=dHh4RkRIZUxiOGlyX0RoeGdHandmQVE6MQ>

7 Pruebas

En este apartado se encuentra la experiencia personal a la hora de probar la aplicación en diferentes dispositivos. Exponiendo las ventajas e inconvenientes encontrados al usar el emulador oficial, y los problemas encontrados al probar por primera vez el juego en diferentes dispositivos móviles reales.

7.1 Emulador

En emulador viene por defecto con el entorno de desarrollo Android. A continuación se muestra una lista de ventajas e inconvenientes:

Ventajas:

- Permite probar aplicaciones Android sin disponer de un móvil real. Sin emulador no habría podido llevar a cabo este proyecto.
- Nos permite probar todas las versiones diferentes de Android creando diferentes máquinas virtuales.
- Es posible conectarse a internet y reproducir sonidos.

Inconvenientes:

- A medida que el proyecto crece, cada vez es mucho más lento cargarlo en el emulador.
- Al emular una arquitectura totalmente diferente, es muy lento, por lo que probar proyectos de gráficos 3D es tedioso y no representa para nada el rendimiento final.
- No es posible probar funciones de vibración, pantalla multi-táctil o gestión del acelerómetro.
- Y sobre todo, no representa lo que finalmente se verá en un móvil real. Como demuestran los errores encontrados en los dispositivos reales (leer los sub-apartado siguientes).

7.2 Android x86

Cansado de la lentitud del emulador normal, investigué otras formas de probar aplicaciones Android mediante máquinas virtual, y encontré el proyecto Android x86³³, que es una adaptación del sistema operativo Android para máquinas con arquitectura x86.

Prometía ofrecer mejores resultados que el emulador normal, por lo que probé a instalarlo mediante el programa VirtualBox³⁴. Pero el resultado no fue satisfactorio, ya que a pesar de arrancar más rápido y tener mejor rendimiento en los menús del sistema operativo, no tiene soporte gráfico de calidad, ni fui capaz de activar el sonido, por lo que el juego iba igual de lento que en emulador normal, y además sin sonido.

7.3 Móviles reales

De forma personal, y con opción de *debug*, sólo he podido probar el juego en el Samsung Galaxy S1 y el Nexus One que me prestó la universidad, por lo que en este apartado explico los problemas encontrados en estos dos. Y como añadido de última hora, explico el problema encontrado con Samsung Galaxy S2 una vez el juego ya estaba publicado en el Android Market.

Para conectar en modo *debug* un móvil en linux, se necesita un archivo de configuración específico según el modelo (archivo Nexus One [39]). Si en cambio sólo queremos probar un archivo .apk concreto, hay tutoriales por internet (ejemplo [38]) que explican el proceso.

A destacar en cuanto a la comparación con el emulador oficial, usar un móvil real durante el desarrollo es una opción mucho mejor si se dispone de esta posibilidad, ya que la velocidad de carga es mucho mejor, y garantizas que al menos funciona en un dispositivo real, cosa que no pasa al usar el emulador, como veremos a continuación.

³³ Web Android x86: <http://www.android-x86.org/>

³⁴ Herramienta para crear máquinas virtuales gratuita. Web: <https://www.virtualbox.org/>

7.3.1 Galaxy S1

Fue el primer móvil real en el que probé el juego. Una vez el cubo ya se mostraba correctamente en el emulador (con texturas en las casillas y en el fondo, transparencias, algoritmo de selección, etc.) decidí probarlo en un móvil real.

El resultado fue desastroso. No se mostraba ninguna textura, el algoritmo de selección fallaba en según qué casillas, y no había efecto de transparencia. Fue cuanto menos chocante que aquello que aparecía tan bien en el emulador oficial, se mostrase tan mal en un móvil real.

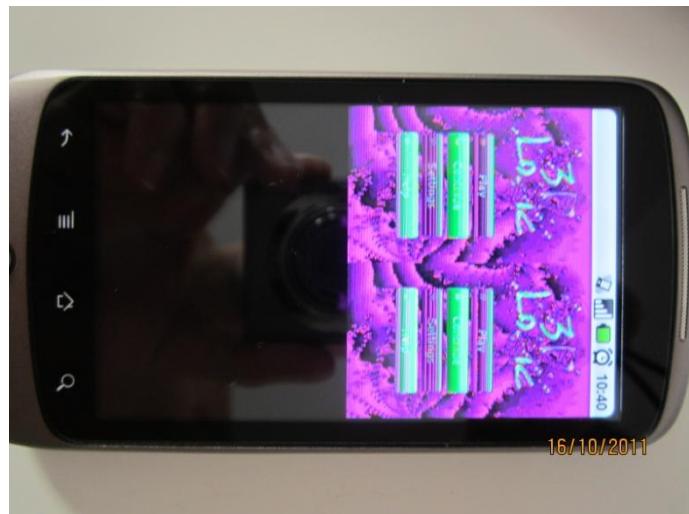
Tras buscar información por internet, descubrí que las texturas han de tener dimensiones de potencias de 2 para que todos los móviles reales las acepten (pero el emulador las acepta todas). Una vez conseguí que se mostraran las texturas, comprobé que no se adaptaban correctamente a la resolución del móvil, y el problema resultó ser que en el Android *manifest* ([3.3 - Manifest](#)) había olvidado especificar la versión *target*, y esto provocaba que la vista OpenGL no se configurase adecuadamente, puesto que las primeras versiones de Android no soportaban pantallas de alta resolución.

El algoritmo de selección de casillas tuve que ajustarlo para que finalmente funcionase correctamente ([Código C.4](#)).

Y finalmente, para activar la transparencia de las texturas tuve que añadir diferentes parámetros OpenGL, que en el emulador están activos por defecto, pero sucede lo mismo en los móviles reales.

7.3.2 Nexus

Un compañero que disponía de un Nexus One probó el juego una vez ya funcionaba correctamente en el Samsung Galaxy S1, y la siguiente imagen ([Figura 7.1](#)) fue lo que obtuve por respuesta.



[Figura 7.1 Primera prueba en Nexus One](#)

Por lo que tuve que revisar la configuración de color de la vista, y tras varias pruebas llegué a la configuración definitiva que no ha vuelto a cambiar ([Código C.1](#))

Un tiempo después, cuando pude probar por mi cuenta el juego en el Nexus One prestado por la universidad, descubrí que no se aplicaba correctamente el *CULL_FACE* (en el emulador y galaxy sí que se mostraba correctamente), por lo que al tener una cara del cubo de frente, las otras eran visibles por detrás, creando un efecto extraño. Algo interfería en esa propiedad entre que se activaba y finalmente se mostraba la escena, por lo que tuve que activar la propiedad de nuevo justo antes de pintar la escena con tal de evitar el problema.

Por otro lado, al no haber probado ningún teléfono móvil con *trackball*, descubrí que mediante este se podía mover el cubo más allá de los límites permitidos, por lo que también tuve que ajustar este aspecto.

7.3.3 Otros

Después de perder la esperanza de que el juego pudiese ser compartido sin miedo a que en cada nuevo móvil surgieran problemas diferentes, empecé a pedir a conocidos con otros modelos de móvil Android que probaran el juego.

Por suerte, una vez configurado correctamente para los dos móviles anteriormente mencionados, el juego empezó a mostrarse correctamente en otros modelos de móviles Android. Por lo tanto, al final pude cumplir el objetivo de publicar el juego en el Android Market, y hasta la fecha los usuarios no han reportado ningún error tras más de 1000 instalaciones.

Sin embargo (añadido de última hora), hace un par de días descubrí que el juego no funcionaba correctamente en los móviles Samsung Galaxy S2. El problema consistía en que nada más arrancar el juego lo único que aparecía era una pantalla de color gris. Pero tanteando con los dedos se podía llegar a empezar un nivel, y el cubo se mostraba correctamente (no así el fondo de pantalla, del cual sólo se mostraba la mitad, en forma de triángulo, y parpadeante), por lo que el problema estaba en los menús.

Inicialmente pensé que volvía a ser un problema de configuración de color, pero lo descarté tras pedir a un compañero que probase varias versiones diferentes con configuraciones distintas, y que todas se mostrasen de la misma forma.

Finalmente descubrí que el problema procedía de pintar el cubo y los fondos y menús de formas diferentes. La activación de los `VertexArray` creaba conflictos con la extensión `OES_draw_texture` de OpenGL 1.1 usada para pintar los fondos y elementos del menú. Por lo tanto, la solución consistió en activar `VertexArray` justo antes de pintar la escena 3D, y seguidamente desactivarlo para que no crease conflictos con `OES_draw_texture`.

Para concluir el capítulo, tengo que decir que tras el último incidente mencionado, he vuelvo a coger un poco de miedo a que el juego no esté funcionando correctamente en según qué móviles, ya que parece que cada uno gestiona OpenGL de una forma diferente.

8 Publicar el juego en el Android Market

Una vez acabada y testeada la aplicación, lo ideal es publicarla en el Android Market para que cualquier persona pueda tener acceso a ella.

En este apartado se muestran todos los pasos a seguir para publicar satisfactoriamente una aplicación Android.

8.1 Cuenta de desarrollador

Lo primero que hay que hacer es crear una cuenta de desarrollador, cuesta \$US25 y es para toda la vida.

Dependiendo del país es posible publicar aplicaciones de pago o no. En España es posible, y para ello es necesario también tener una cuenta Google Checkout³⁵ para recibir los pagos. En este caso el juego se publicará de forma gratuita, por lo que no puedo proporcionar datos detallados, pero se ha de tener en cuenta que Google se queda con el 30% de los beneficios.

Para realizar el registro de una cuenta de desarrollador, es tan sencillo como entrar en la página de publicación del market³⁶ e introducir la información solicitada, entre otros: nombre de usuario, número de móvil para verificar la cuenta, datos personales e información de la tarjeta de crédito con la que se pagarán los \$US25 dólares.

³⁵ <https://checkout.google.com/?hl=es>

³⁶ <https://market.android.com/publish/signup>

8.2 Preparar la aplicación para ser publicada

Para poder publicar una aplicación, se ha de firmar previamente el archivo APK³⁷, pero antes es conveniente asegurarse de que el archivo Android Manifest tiene la configuración adecuada.

A continuación se muestra una lista de cosas a tener en cuenta:

- Si el atributo debug (`android:debuggable`) de la etiqueta `<application>` está activo, se ha de quitar.
- Asegurate de haber indicado las versiones mínima (`android:minSdkVersion`) y objetivo (`android:targetSdkVersion`) para las que está indicada tu aplicación. Es muy importante, ya que AndroidMarket sólo mostrará la aplicación a aquellos usuarios que cumplan los requisitos. Prueba una última vez en el emulador la aplicación con todas las versiones de Android especificadas para garantizar que los usuarios no encontrarán errores imprevistos de incompatibilidad de versiones.
- No olvides indicar los atributos de versión de código (`android:versionCode`) y nombre de la versión (`android:versionName`) de la etiqueta `<manifest>`. Y en caso de ya haber publicado previamente una versión, se ha de incrementar el número de versión de código y opcionalmente cambiar el nombre de la versión.
- Es recomendable definir el atributo `android:installLocation` a `auto` para permitir así que la aplicación pueda ser almacenada en dispositivos de almacenamiento externo, como memorias SD³⁸.
- Revisa la etiqueta `<uses-permission>` y elimina todos aquellos permisos de hardware que finalmente no sean necesarios en tu aplicación. Los permisos necesarios se muestran a los usuarios que vayan a descargar tu aplicación, y estos pueden desconfiar si encuentran elementos cuyo uso no está justificado.

Una vez revisados los ajustes de la lista, se puede proceder a firmar el archivo APK para poder publicarlo sin problemas en el market. En el apéndice *B - Firmar aplicación Android en Eclipse*, se detallan los pasos a seguir para firmar una aplicación en Eclipse.

³⁷ Android application package file

³⁸ Secure Digital (SD) http://es.wikipedia.org/wiki/Secure_Digital

8.3 Publicación

Ya tenemos todo lo necesario para publicar la aplicación. Nos dirigimos pues a la página de publicación³⁹ y nos autenticamos. Nos aparecerá nuestra lista de aplicaciones (*Figura 8.1*) con estadísticas de cada una (vacía la primera vez que entramos), y la opción de subir una nueva aplicación.



Figura 8.1 Publicación de la aplicación, paso 1

Para subir una aplicación hay que rellenar un formulario que consta de cuatro partes: Archivo APK, recursos, detalles y opciones de publicación:

Archivo APK: Es lo primero que se ha de proporcionar. Se muestra un formulario de subida de archivo (*Figura 8.2*) en el que tenemos proporcionar el archivo APK firmado previamente. Mientras se sube podemos llenar el resto de información necesaria.

This is a screenshot of a modal dialog titled 'Subir nuevo APK'. It contains a file input field with a placeholder 'Examinar...' (Browse...) button, a large empty text area for app details, and a 'Publicar' (Publish) button. At the bottom left is a 'Cerrar' (Close) button.

Figura 8.2 Publicación de la aplicación, paso 2

³⁹ <http://market.android.com/publish>

Recursos: Como se muestra en la *Figura 8.3*, podemos proporcionar imágenes y videos que acompañen la aplicación. Es obligatorio proporcionar como mínimo 2 capturas de pantalla y un “Icono de aplicación” que se mostrará a los usuarios que quieran descargar la aplicación. Opcionalmente, es recomendable proporcionar un video (URL de YouTube⁴⁰) que muestre el funcionamiento básico de la aplicación. Los campos “Grafico...” están más indicados para aplicaciones de pago, permitiendo al Android Market publicitar la aplicación con ellos.

Subir recursos

<p>Capturas de pantalla al menos 2 añadir otra captura de pantalla</p>  <p>Sustituir esta imagen suprimir</p> <p>Icono de aplicación de alta resolución [Más información]</p>  <p>Sustituir esta imagen suprimir</p> <p>Gráfico promocional opcional</p> <p>Añade un gráfico promocional: <input type="button" value="Examinar..."/> <input type="button" value="Publicar"/></p> <p>Gráfico de funciones opcional [Más información]</p> <p>Añade un gráfico de funciones: <input type="button" value="Examinar..."/> <input type="button" value="Publicar"/></p> <p>Vídeo promocional opcional</p> <p>Añade un enlace de vídeo promocional: <input type="text" value="http://www.youtube.com/watch?v=YYkmpvLylwM"/> <input type="button" value="Publicar"/></p> <p>Excluir marketing</p> <p><input checked="" type="checkbox"/> No promocionar mi aplicación salvo en Android Market y en los sitios web o para móviles propiedad de Google. Asimismo, soy consciente de que cualquier cambio relacionado con esta preferencia puede tardar sesenta días en aplicarse.</p>	<p>Capturas de pantalla: archivo PNG o JPEG (no alpha) de 24 bits de 320 x 480, 480 x 800, 480 x 854, de 1280 x 720, 1280 x 800. Sangrado completo, sin bordes. Puedes subir capturas de pantalla en orientación horizontal. Parecerá que las miniaturas están giradas, pero se mantendrán la orientación y las imágenes reales.</p> <p>Icono de aplicación de alta resolución: imagen de 32 bits PNG o JPEG y 512 x 512, máximo: 1024 KB</p> <p>Gráfico promocional: archivo de 24 bits PNG o JPEG (no alpha) y 180 an x 120 al Sin bordes</p> <p>Gráfico de funciones: archivo de 24 bits PNG o JPEG (no alpha) y 1024 x 500; el tamaño se reducirá a mini o micro.</p> <p>Vídeo promocional: introducir URL de YouTube</p>
--	---

Figura 8.3 Publicación de la aplicación, paso 3

Detalles: En este apartado (*Figura 8.4*) se especifican el nombre, descripción y tipo de aplicación. Pudiendo definir la información en diferentes idiomas.

⁴⁰ Video de 3D Logic proporcionado: <http://www.youtube.com/watch?v=YYkmpvLYlwM>

Especificación de detalles

Idioma añadir idioma	*English (en) español (es) El signo de estrella (*) indica el idioma predeterminado.
<input checked="" type="checkbox"/> Eliminar entrada en Inglés	
Title (Inglés)	<input type="text" value="Android 3D Logic"/> 16 caracteres (máximo 30)
Description (Inglés)	<pre>3D Logic by Alex Matveev now in Android A simply amazing logic puzzle! Link colored squares to complete the cube!</pre> <p>115 caracteres (máximo 4000)</p>
Recent Changes (Inglés) Nombre de la versión: 1.0 [Más información]	
<p>0 caracteres (máximo 500)</p>	
Promo Text (Inglés)	<input type="text"/> <p>0 caracteres (máximo 80)</p>
Tipo de aplicación	
<input type="button" value="Juegos"/>	
Categoría	
<input type="button" value="Puzzles y juegos para ejercitarse la mente"/>	

Figura 8.4 Publicación de la aplicación, paso 4

Opciones de publicación: En este apartado (*Figura 8.5*) definimos los permisos, si es o no para todos los públicos, o si tiene algún tipo de restricción por países.

Opciones de publicación	
Protección contra copias	<input checked="" type="radio"/> Desactivado (la aplicación se puede copiar desde el dispositivo) <input type="radio"/> Activado (evita la copia de esta aplicación desde el dispositivo. Aumenta la cantidad de memoria del teléfono necesaria para instalar la aplicación).
Clasificación de contenido [Más información]	<input type="radio"/> Nivel de madurez alto <input type="radio"/> Nivel de madurez medio <input type="radio"/> Nivel de madurez bajo <input checked="" type="radio"/> Para todos
Precios Dispositivos admitidos [Más información]	Gratis ¿Quieres vender aplicaciones? Configura una cuenta de comerciante en Google Checkout. <input checked="" type="checkbox"/> Todos los países

Figura 8.5 Publicación de la aplicación, paso 5

Finalmente hemos de aceptar los términos y directrices de contenidos Android y opcionalmente proporcionar información de contacto como sitio web, e-mail o teléfono.

Una vez cumplimentada toda la información, solo falta pulsar el botón *Publicar*, y en unos minutos la aplicación será visible en el Android Market por millones de usuarios de todo el mundo.

8.4 Reporte de errores

Una vez publicada la aplicación, los usuarios la podrán descargar y probar libremente. En caso de encontrar errores, pueden reportarlos en la página de la aplicación. Como se muestra en la *Figura 8.6*, cada aplicación tiene un apartado de estadísticas de informes de errores, que pueden ayudar a los desarrolladores a corregirlos.

Informes de errores de la aplicación

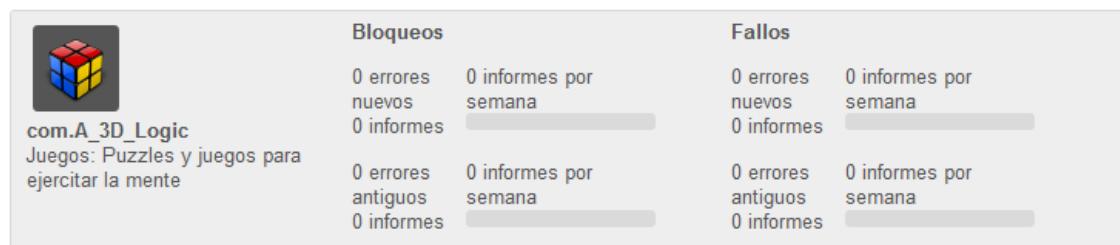


Figura 8.6 Informes de errores

9 Planificación y costes

Este capítulo presenta los diferentes aspectos de gestión del proyecto. Empezando por cómo se han planificado las diferentes partes de proyecto, y mostrando a continuación un análisis económico del mismo.

9.1 Planificación

El proyecto fue inscrito el 28 de Junio de 2011, y unos meses antes ya había tenido una reunión con el director del proyecto. Sin embargo, hasta que no matriculé el proyecto no empecé a dedicarle un mínimo de horas semanales, ya que hasta entonces seguía cursando las últimas asignaturas de la carrera. Por esto, contaré como que el proyecto se empezó el día de la inscripción, representando la dedicación previa en el margen de tiempo comprendido entre la inscripción y la matrícula del proyecto.

A continuación (*Figura 9.2*) se muestra un diagrama Gantt con la planificación final del proyecto, mostrando la cantidad de horas que se han dedicada a cada una de las tareas que lo componen.

Se ha de tener en cuenta que ha sido simplificado para mostrar las partes más importantes. Además, la líneas representan las tareas que se estaban realizando en esa etapa, pero no quiere decir que me estuviese dedicando a jornada completa a esa tarea. Es por eso que podemos encontrar diferentes tareas solapadas en una determinada etapa. Finalmente, se muestran las fechas clave a lo largo del proyecto.

Los primeros meses, en la época vacacional hasta septiembre, el objetivo fue familiarizarme con Android, documentarme en el desarrollo de juegos para esta plataforma, e intentar definir los requisitos y opciones de las que contaría el juego, así como un primer diseño inicial. La dedicación fue de aproximadamente 25 a 30 horas semanales.

En septiembre y octubre me dediqué a jornada completa a desarrollar la estructura básica del juego, añadiendo los diferentes elementos funcionales y de interfaz de usuario. Preveía poder acabar el proyecto para diciembre, pero en noviembre empecé a trabajar en una empresa a media jornada, lo que prolongó el tiempo previsto inicialmente para finalizar la implementación, corregir errores surgidos en las pruebas, y redactar la memoria del proyecto.

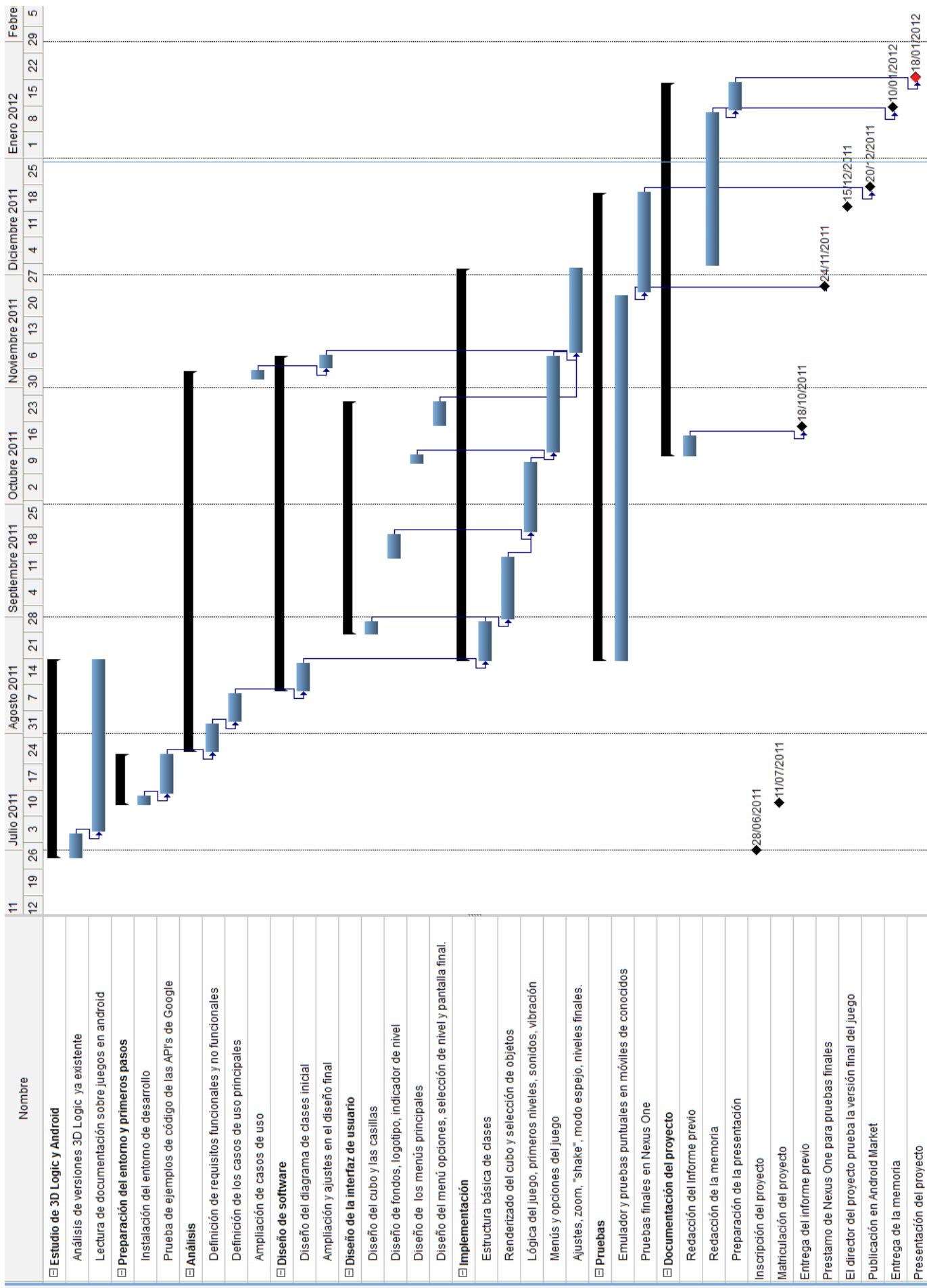


Figura 9.1 Planificación (Gantt)

A partir del diagrama anterior, y teniendo en cuenta la dedicación en las diferentes etapas del proyecto, podemos obtener una aproximación del número de horas que ha llevado cada grupo de tareas.

Tarea	Dedicación (Horas)
Estudio inicial	60
Preparación del entorno y primeros pasos	50
Análisis	70
Diseño de software	60
Diseño de la interfaz de usuario	80
Implementación	300
Pruebas	90
Documentación	100
Total:	810

Tabla 9-1 Número de horas dedicadas a cada grupo de tareas

9.2 Coste del proyecto

En este apartado se efectúa una estimación del coste económico del proyecto, basado en la cantidad de horas dedicadas a cada grupo de tareas, según lo descrito en la planificación del apartado anterior.

Durante el desarrollo del proyecto se han realizado diferentes tipos de tareas. Y si nos fijamos, los diferentes grupos de tareas pertenecen a perfiles profesionales distintos, y estos cuentan con diferentes sueldos. Por lo tanto, para dar una estimación más rigurosa, primero distribuiremos las horas entre los siguientes perfiles profesionales:

- **Analista:** Tiene como cometido describir una solución, definir requisitos funcionales y no funcionales de la aplicación, casos de uso, etc. En resumen, define *qué* ha de hacer la aplicación, y es también el principal responsable de escribir la documentación técnica.
- **Diseñador de software:** Basándose en la descripción del analista, diseña una solución utilizando una tecnología concreta (en este caso impuesta de antemano por la plataforma Android). En resumen, define *cómo* funcionara internamente la aplicación.
- **Diseñador gráfico y de interfaces:** Encargado de crear todos los elementos gráficos y diseñar una interfaz de usuario usable.
- **Programador:** Encargado de implementar la aplicación partiendo de la especificación y diseño facilitados. Además será el encargado de realizar las pruebas de la aplicación.

A continuación (*Tabla 9-2*), se muestra cómo queda la distribución de horas, según el perfil, partiendo de las estimaciones por grupo de tareas del apartado anterior, *Tabla 9-1*.

Tarea	Total (Horas)	Analista (Horas)	Diseñador Software (Horas)	Diseñador gráfico (Horas)	Programador (Horas)
Estudio inicial	60	25	25	0	10
Preparación del entorno y primeros pasos	50	10	10	0	30
Análisis	70	50	20	0	0
Diseño de software	60	15	45	0	0
Diseño de la interfaz de usuario	100	0	0	100	0
Implementación	290	10	10	0	270
Pruebas	80	0	0	0	80
Documentación	100	50	20	20	10
Total:	810	160 (20%)	130 (16%)	120 (14%)	400 (50%)

Tabla 9-2 Dedicación correspondiente a cada perfil

Una vez calculado el número de horas correspondiente a cada uno de los diferentes perfiles profesionales, el siguiente paso consiste en multiplicar esas horas por el sueldo correspondiente a una hora de trabajo de cada uno de los perfiles. Así obtendremos el coste de cada perfil, y sumándolos tendremos el coste total de personal.

Basándonos en la información de *Tusalario*⁴¹, los sueldos de los diferentes perfiles en España son los siguientes:

Perfil profesional	Sueldo medio/hora
Analista de sistemas	11.32€
Diseñador de software	10.83€
Diseñador gráfico	8.97€
Programador	9.46€

Tabla 9-3 Sueldo medio/hora por perfil profesional

Los datos ofrecidos para obtener cada uno de los sueldos (*Tabla 9-1*) están basados en el perfil de trabajador joven en una *start-up*⁴² de aplicaciones informáticas, con poca experiencia tras haber acabado sus estudios universitarios.

⁴¹ <http://www.tusalario.es/main>

Selecciona tu nivel de educación	*
Licenciatura	<input type="button" value="▼"/>
Indica cuantos años de experiencia profesional tienes	*
1	<input type="button" value="▼"/>
¿Tienes contrato indefinido?	*
<input checked="" type="radio"/> No <input type="radio"/> Sí	
¿Desarrollas una función de supervisión de otros trabajadores?	*
<input checked="" type="radio"/> No <input type="radio"/> Sí	
¿Tu empresa es de propiedad extranjera?	*
<input checked="" type="radio"/> No <input type="radio"/> Sí	
¿La mayoría de tus compañeros de trabajo son hombres?	*
<input type="radio"/> No <input checked="" type="radio"/> Sí	
Selecciona el tamaño de tu empresa	*
<input type="radio"/> Menos de 10 <input checked="" type="radio"/> Entre 10 y 50 <input type="radio"/> Más de 50	
Promedio de horas de trabajo (jornada completa). ¿Cuántas horas de trabajo semanales suponen la jornada completa en tu empresa?	*
40	<input type="button" value="▼"/>

Tabla 9-4 Perfil de los trabajadores

Llegados a este punto, ya disponemos de todos los datos necesarios para calcular el coste del proyecto asociado a los trabajadores. El resultado se muestra en la *Tabla 9-5*.

⁴² http://es.wikipedia.org/wiki/Compa%C3%B1a%C3%A1da_startup

Perfil profesional	Número de horas	Salario medio/hora	Coste
Analista de sistemas	160	11.32€	1811.2€
Diseñador de software	130	10.83€	1407.9€
Diseñador gráfico	120	8.97€	1076.4€
Programador	400	9.46€	3784€
Total:			8079.5€

Tabla 9-5 Coste del proyecto asociado a los trabajadores

Una vez calculado el coste asociado a los trabajadores, se ha de tener en cuenta el resto de gastos relacionados con el proyecto.

El software utilizado es gratuito, por lo que el coste no se ve incrementado en este aspecto. En cuanto al hardware, hay que tener en cuenta el coste de un ordenador de gama media/alta para mover con soltura el emulador de Android, y aunque en la realización de este proyecto ha sido prestado, un Smartphone Android es también uno de los requisitos. En la siguiente tabla se muestra el cálculo de los costes:

	Precio	Vida útil (meses)	Meses de uso	Coste
Ordenador	700€	48	7	700/48 * 7 = 102€
Smartphone	300€	24 ⁴³	4	300/24 * 4 = 50€
Total:				152€

Tabla 9-6 Otros costes

Ya tenemos todos los datos necesarios para calcular el coste total del proyecto. Teniendo en cuenta los costes de los trabajadores y de los recursos utilizados, obtenemos:

Tipo de coste	Coste
Trabajadores	8079.5€
Recursos	152€
Total:	8231.5€

Tabla 9-7 Coste total del proyecto

Así, concluimos que el coste total aproximado del proyecto sería de unos **8231€**.

⁴³ Vida útil de un teléfono móvil

http://www.bbc.co.uk/mundo/noticias/2011/02/110218_1021_guias_respuestas_telefonos_celulares_1_tlquqssynsaap_dc.shtml#danio

10 Conclusiones

Llegados a este punto, sólo queda evaluar el trabajo realizado en los últimos meses y exponer las conclusiones obtenidas en relación a los diferentes aspectos que engloban el proyecto.

En este capítulo, en primer lugar se evalúan los objetivos completados a lo largo del proyecto, seguidas de las principales dificultades encontradas. A continuación se expone una lista de posibles ampliaciones con tal de seguir mejorando el juego. Y finalmente se encuentran las conclusiones personales con respecto al proyecto.

10.1 Evaluación de objetivos

Ya finalizado el proyecto, podemos listar los objetivos superados en la realización del proyecto:

- El entorno de desarrollo Android, así como las posibilidades que ofrece, ha sido estudiado, comprendido, y finalmente utilizado para desarrollar el juego.
- He logrado programar satisfactoriamente una versión del juego 3D Logic para teléfonos móviles Android.
- He sacado partido de las principales funciones de los móviles de última generación. Como por ejemplo los efectos sonoros, la vibración, gestión multi-táctil, almacenamiento en la memoria del teléfono, uso del acelerómetro, etc.
- Se ha tratado con especial cuidado la compatibilidad del juego con diferentes versiones del sistema operativo Android. Por lo que el resultado final es un juego que funciona en prácticamente todas las versiones del sistema operativo disponibles en el mercado, incluida la última versión 4.0, para la que se tuvieron que ajustaron ciertos parámetros.
- El juego ha sido desarrollado pensando también en las diferentes resoluciones de pantalla de los dispositivos que circulan actualmente por el mercado, y se da soporte tanto a pantallas de baja resolución de dispositivos económicos, como a los móviles de último modelo con altas resoluciones, incluidas las *tablets*.
- Se han identificado y superado los principales inconvenientes a la hora de desarrollar juegos 3D para Android.
- Se ha conseguido diseñar una interfaz de usuario fácil de usar. Puesta a prueba por los propios usuarios.

- El juego ha sido probado en diferentes dispositivos para garantizar su correcto funcionamiento.
- Finalmente, el objetivo final se ha cumplido, y desde el 20 de diciembre de 2011 está disponible en el Android Market, a disposición de cualquier usuario que desee probarlo.

10.2 Dificultades

A continuación podemos encontrar la lista de las principales dificultades encontradas a lo largo del desarrollo del proyecto.

- El no disponer inicialmente (en verano de 2011) de un terminal Android en el que hacer pruebas, tuve que descartar la utilización de OpenGL 2.0 (más nuevo que el OpenGL 1.0 utilizado) ya que el emulador no es compatible, y finalmente no he podido dedicarle el tiempo que me habría gustado. En su lugar, me he centrado en garantizar la compatibilidad que ofrece OpenGL 1.0 si se programa adecuadamente.
- El entorno de desarrollo es aún inestable a veces. Por ejemplo, el archivo autogenerado *R*, de referencias a recursos del proyecto, se corrompe con facilidad al modificar los recursos. Esto acaba provocando errores que realmente no existen, y es necesario recargar los recursos con cuidado para solucionar los problemas.
- La dificultad más importante a la que me he enfrentado ha sido la incompatibilidad entre diferentes dispositivos. El hecho de que en el emulador todo se vea correctamente, pero luego cada dispositivo presente problemas diferentes, es el problema que más ha costado superar a lo largo del proyecto.
- El hecho de no tener nociones de diseño gráfico también resultó una dificultad. Con tal de diseñar una interfaz atractiva, tuve que dedicar una gran cantidad de horas a probar y descartar alternativas poco satisfactorias. Lo cual no era uno de los objetivos principales del juego, pero sí muy importante.
- Finalmente, en noviembre de 2011 me ofrecieron trabajar en una empresa, y puesto que consideraba que el proyecto llevaba un buen ritmo, acepté. Y al final he ido un poco justo de tiempo en la redacción de la presente memoria, con tal de explicar en detalle todos los aspectos del proyecto, e incluir el contenido que fui reuniendo, mediante anotaciones, a lo largo del desarrollo del proyecto.

10.3 Posibles ampliaciones

Es difícil decidir cuándo dejar de añadir funcionalidades a una aplicación, pero se ha de hacer con tal de limitar un proyecto a un tiempo concreto. Sin embargo, siempre quedan en el tintero ideas de nuevas funcionalidades que podrían añadirse.

A continuación muestro la lista de las posibles ampliaciones que se podrían incluir en futuras versiones del juego:

- La opción de permitir al usuario crear sus propios niveles personalizados.
- Calcular los tiempos del usuario al superar los niveles, con tal de motivarlo a superar sus propias marcas.
- Añadir opciones online. Las dos propuestas anteriores se verían potenciadas al poder compartir datos con otros usuarios. Se podrían descargar niveles personalizados de otros usuarios, y consultar los rankings generales, que representen la velocidad de los usuarios al superar todos los niveles del juego.
- Añadir nuevos modos de juego que modifique ligeramente las reglas originales. Como por ejemplo, la dificultad añadida de conocer que dos generadores de color tienen sus posiciones intercambiadas, y tener que descubrir cuáles son tratando de resolver un nivel de entrada imposible si no se soluciona ese problema.
- Mejorar las animaciones del juego y los menús al interactuar con ellos. Efectos al pulsar los botones, pasar páginas mediante la pantalla táctil, etc.

10.4 Conclusiones personales

La realización de este proyecto me ha iniciado en el desarrollo de videojuegos para móviles Android, un objetivo que me llamaba la atención, y que me abre las puertas a seguir desarrollando futuras aplicaciones y juegos para esta plataforma, pudiendo obtener remuneración de los trabajos futuros. Además, todo indica que la tecnología utilizada tomará una relevancia notable en el futuro, y estar familiarizado con esta no hace otra cosa que abrirme puertas profesionales.

Considero haber cumplido los objetivos que me marqué inicialmente, y me satisface haber podido publicar finalmente el juego de forma oficial, permitiendo así que usuarios de todo el mundo puedan sacar partido al trabajo realizado (y ya van más de 1000).

En cuanto a los aspectos negativos, creo que la plataforma Android aún necesita ser depurada. Deberían crearse estándares más estrictos que todo fabricante de teléfonos móviles con sistema operativo Android cumpliese. Ya que, bajo mi punto de vista, no es aceptable que exista el nivel de incompatibilidades que existe actualmente entre diferentes dispositivos móviles con la misma versión del sistema operativo.

Finalmente, evaluando globalmente el trabajo realizado en el proyecto, puedo concluir que estoy satisfecho con los resultados obtenidos.

Bibliografía

- [1] Beginning Android Games [Libro] / autor: Mario Zechner. 2011. Apress.
- [2] OpenGL ES Game Development [Libro] /autores: Dave Astle y Dave Durnil. 2010. Premier Press
- [3] Android Developer [Online] – Application Fundamentals -
<http://developer.android.com/guide/topics/fundamentals.html>
- [4] Sgoliver.net [Online] – Estructura de un proyecto Android -
<http://www.sgoliver.net/blog/?p=1278>
- [5] Anddev [Online] – OpenGL ES 1.1 vs 2.0 - <http://www.anddev.org/android-2d-3d-graphics-opengl-problems-f55/opengl-es-1-1-vs-2-0-t50168.html>
- [6] Twodee [Online] – Interactive 3D graphics: OpenGL ES 2.0 and NDK -
<http://www.twodee.org/blog/?p=742>
- [7] Anddev [Online] – Problem installin ADT plugin for Eclipse -
<http://www.anddev.org/sdk-adt-emulator-problems-f16/problem-installing-adt-plugin-for-eclipse-t56377.html>
- [8] iLectronx [Online] – Installing the Android SDK on Ubuntu 11.04 - <http://ilektron-x.blogspot.com/2011/08/installing-android-sdk-on-ubuntu-1104.html>
- [9] Pro Android Games [Libro] / autor: Vladimir Silvia. 2010. Apress
- [10] Nuxeo [Online] – How to speed up de Android Emulator by up to 400% -
<http://blogs.nuxeo.com/dev/2011/10/speeding-up-the-android-emulator.html>
- [11] Mauricioaedo [Online] – Solución a invalid command-line parameter de ADV –
<http://mauricioaedo.com/sistemas-operativos/android/solucion-a-invalid-command-line-parameter-de-avd/>
- [12] StackOverflow [Online] – Prevent onPause from trashing OpenGL -
<http://stackoverflow.com/questions/2112768/prevent-onpause-from-trashing-opengl-context>
- [13] Anddev [Online] – Face selection, picking - <http://www.anddev.org/android-2d-3d-graphics-opengl-tutorials-f2/object-polygon-face-selection-aka-picking-t11572.html>
- [14] StackOverflow [Online] – Transparent a color on a texture -
<http://stackoverflow.com/questions/4983959/android-opengl-es-transparent-a-color-on-a-texture>
- [15] StackOverflow [Online] – Transparent texture in OpenGL ES -
<http://stackoverflow.com/questions/2361602/transparent-texture-in-opengl-es-for-android>

- [16] Quakeboy'z Dev Area [Online] – How to load a texture in Android OpenGL ES -
<http://qdevarena.blogspot.com/2009/02/how-to-load-texture-in-android-opengl.html>
- [17] StackOverflow [Online] – I want to shake it -
<http://stackoverflow.com/questions/2317428/android-i-want-to-shake-it>
- [18] Android Developer [Online] – Activity -
<http://developer.android.com/reference/android/app/Activity.html>
- [19] Androideity [Online] - Arquitectura de Android -
<http://androideity.com/2011/07/04/arquitectura-de-android/>
- [20] StatCounter [Online] – Movile OS, Spain - http://gs.statcounter.com/#mobile_os-ES-monthly-201101-201201
- [21] Nielsen[Online] – New Smartphone Buyers Increasingly Embracing Android -
http://blog.nielsen.com/nielsenwire/online_mobile/in-u-s-market-new-smartphone-buyers-increasingly-embracing-android/
- [22] Nexus404 [Online] / auth Robert Nelson – Google Releases Android Platform Version Statistics For November 2011 - <http://nexus404.com/Blog/2011/12/02/google-releases-android-platform-version-statistics-for-november-2011-latest-android-usage-numbers-showing-a-big-increase-in-gingerbread-with-big-decrease-in-froyo-a-minor-increase-in-honeycomb/>
- [23] Blogdecomputacion [Online] – Instalación de java en Ubuntu 11.04 -
<http://blogdecomputacion.com/blog/2011/04/12/como-instalar-java-en-ubuntu-11-04/#ixzz1fzE7U0Db>.
- [24] Android Developer [Online] – ApiDemos -
<http://developer.android.com/resources/samples/ApiDemos/index.html>
- [25] Android Developer [Online] – Kube, ApiDemos -
<http://developer.android.com/resources/samples/ApiDemos/src/com/example/android/apis/graphics/kube/index.html>
- [26] Youtube [Online] – Water text effect gimp -
<http://www.youtube.com/watch?v=7V9E7wZGnBE>
- [27] Gimpusers [Online] – Stylish buttons - <http://www.gimpusers.com/tutorials/create-stylish-buttons>
- [28] PacDV [Online] – Free sound effects - <http://www.pacdvc.com/sounds/index.html>
- [29] StoneWashed [Online] – Sound Effects - <http://www.stonewashed.net/sfx.html>
- [30] Vogella [Online] – Android Sounds, Tutorials -
<http://www.vogella.de/articles/AndroidMedia/article.html>

- [31] Konreu [Online] – Vibration examples for Android Phone Development -
<http://android.konreu.com/developer-how-to/vibration-examples-for-android-phone-development/>
- [32] Humbug [Online] – Problema en la resolución y densidad de un surfaceview -
<http://www.humbug.in/stackoverflow/es/emuladores-de-androide-no-simulando-resolucion-o-densidad-correctas-posiblemente-un-surfaceview-asunto-de-implementacion--4042753.html>
- [33] StackOverflow [Online] – Android pinch Zoom -
<http://stackoverflow.com/questions/5375817/android-pinch-zoom>
- [34] PoderPDA [Online] – Publicar una aplicación en Android Market -
<http://www.poderpda.com/plataformas/tutorial-publicar-una-aplicacion-en-el-android-market/>
- [35] Tusalario.es [Online] – Salarios de diferentes perfiles profesionales en España -
<http://www.tusalario.es/main>
- [36] GoFree [Online] – Como crear fondos abstractos en GIMP -
<http://gofree.com/Espanol/Descarga/Gimp/Tutoriales/AbstractoUsandoGIMP.php>
- [37] Android developers [Online] – Backward Compatibility for Applications -
<http://developer.android.com/resources/articles/backward-compatibility.html>
- [38] TheTechJournal [Online] – Install APK Files on your Android -
<http://thetechjournal.com/electronics/android/install-apk-files-on-your-android-how-to-guide.xhtml>
- [39] JavaDude [Online] – Debug Android Nexus One with Eclipse -
<http://javadude.wordpress.com/2010/10/25/debug-android-nexus-one-with-eclipse/>

A. Instalación del entorno de desarrollo

En este apartado se muestran los pasos a seguir para tener a punto el entorno de desarrollo utilizado en la realización del proyecto.

Instalar Java JDK⁴⁴.

```
$ sudo apt-get install openjdk-6-jdk
```

Hay gente que comenta que tiene problemas con el openJDK trabajando con Android. En tal caso se debería instalar el Sun JDK⁴⁴[23] .

```
sudo add-apt-repository ppa:ferramroberto/java  
sudo apt-get update  
sudo apt-get -y install sun-java6-jdk sun-java6-plugin sun-java6-fonts
```

Instalar eclipse:

```
$ sudo apt-get install eclipse
```

Actualmente se instala la versión 3.7 (Indigo).

El único problema que tuve fue que en el apartado Help > Add new software no estaba por defecto el repositorio de Indigo, y esto provocaba problemas al instalar el plugin ADT. No tuve más que añadirlo manualmente.

```
Add -> "http://download.eclipse.org/releases/indigo"
```

⁴⁴ Java Development Kit (Equipo de desarrollo de Java)

Instalar android SDK.

Podemos seguir las instrucciones de la página oficial ⁴⁵

En resumen:

Descargamos el SDK de la página de descarga de Android ⁴⁶ y lo descomprimimos:

```
$ tar -xvzf android-sdk_r15-linux_x86.tgz
```

Una vez descargado lo movemos a una carpeta conveniente, por ejemplo:

```
$ sudo mv android-sdk_r15-linux_x86 /opt/android-sdk
```

Es recomendable crear una nueva variable de entorno que apunte al directorio raíz del SDK de Android, para poder ejecutar comandos desde consola sin necesidad de estar en la carpeta concreta.

```
$ sudo gedit ~/.bashrc
```

Añade al final del archivo las líneas:

```
export ANDROID_HOME=/opt/android-sdk/
export PATH=$PATH:$ANDROID_HOME/tools:$ANDROID_HOME/platform-tools/
```

Cerrar y volver a abrir la consola o ejecutar el siguiente comando:

```
$ source ~/.bashrc
```

Y si todo ha ido bien seremos capaces de ejecutar las aplicaciones del Android SDK Manager:

```
$ android
```

Una vez abierto el Android SDK Manager se puede aprovechar para instalar Android 2.3.3 o Android 4 junto con los ejemplos de prueba.

⁴⁵ <http://developer.android.com/sdk/installing.html>.

⁴⁶ <http://developer.android.com/sdk/index.html>

Instalar el plugin ADT para eclipse:

Podemos seguir las instrucciones de la página oficial ⁴⁷

*En resumen:**Instalación:*

Abrir Eclipse, seleccionar Help > Install New Software

```
Add > "https://dl-ssl.google.com/android/eclipse/"  
Ok > Next > ... > Finish
```

Finalmente reiniciamos Eclipse para aplicar los cambios.

Configurar:

```
Window > Preferences > Android
```

En SDK buscar Location seleccionar Browse y localizar el directorio del Android SDK, en mi caso "/opt/android-sdk/"

```
Apply > OK
```

Crear una maquina virtual:

En Eclipse,

```
Window > AVB Manager > New
```

Elegimos un nombre, un Target de los que hayamos descargado previamente con el Android SDK Manager, ej. "Android 2.3.3" y en Skin > Build-in elegimos un tipo de pantalla. Yo recomiendo HVGA de resolución media, ya que el que viene por defecto tienen una resolución demasiado grande y el rendimiento es muy pobre.

```
Create AVD > Start
```

Si todo ha ido bien, se ejecutará una máquina virtual con el sistema operativo Android.

⁴⁷ <http://developer.android.com/sdk/eclipse-adt.html#installing>

B. Firmar aplicación Android en Eclipse.

Es necesario firmar el archivo APK para poder publicar una aplicación sin problemas en el market. Desde Eclipse, estos son los pasos a seguir:

1. Botón derecho en el nombre del proyecto > Android Tools > Export Signed Application Package. Se mostrará una pantalla similar a la *Figura B.1*.

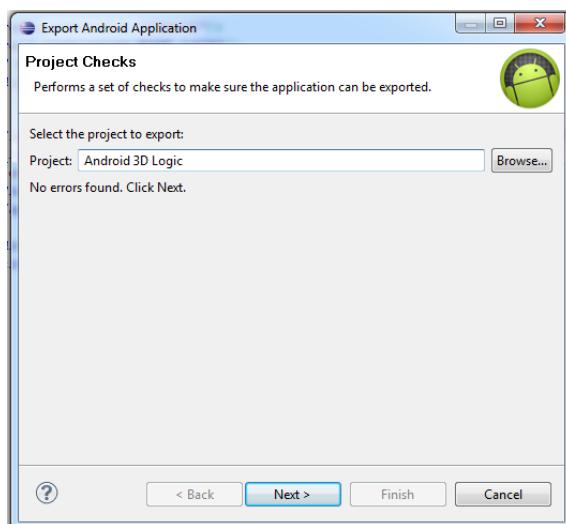


Figura B.1 Firmar aplicación, paso 1

2. Pulsar *Siguiente* para mostrar la Figura B.2 , donde especificaremos el almacén de certificados (*keystore*⁴⁸) con el que realizar la firma, o crear uno nuevo si fuera necesario.

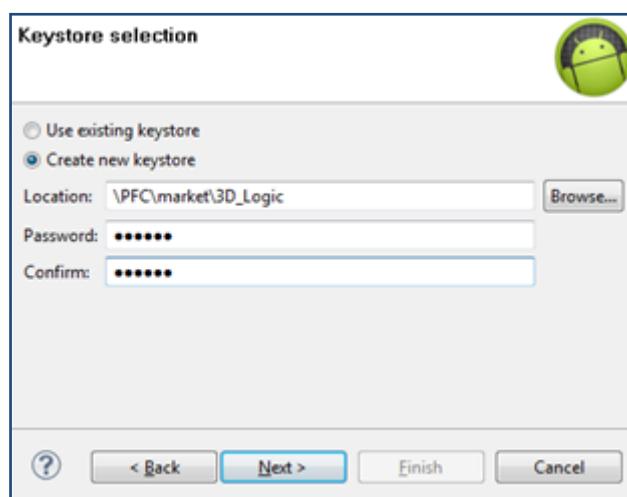


Figura B.2 Firmar aplicación, paso 2-1

⁴⁸ Fichero protegido por contraseña que almacena las llaves con las que se firman los archivos, en este caso el archivo APK.

3. En caso de tener que crear uno nuevo, marca la casilla *Create new keystore*, especifica una carpeta en la que guardar el archivo y asigna una contraseña nueva. Tras pulsar *Siguiente*, se han de proporcionar cierta información, como se muestra en la Figura B.3, los campos *alias*, *password*, *validity* y *first and last name* son obligatorios.

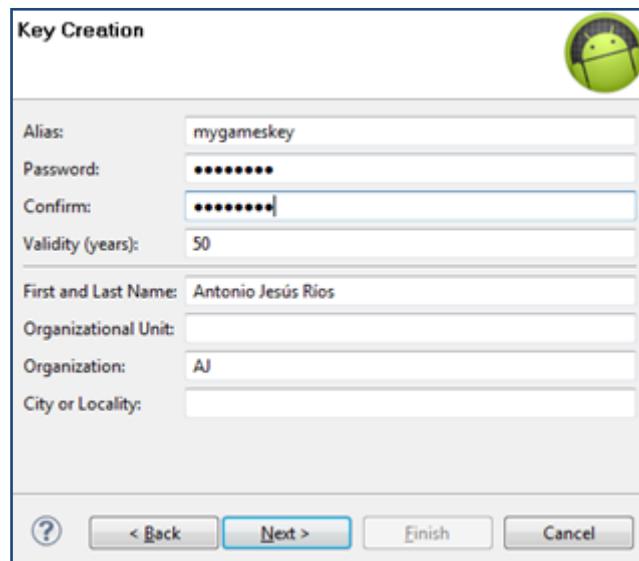


Figura B.3 Firmar la aplicación, paso 2-2

4. Pulsa *Siguiente* para mostrar la última ventana, Figura B.4, sólo nos falta especificar la dirección en la que guardar el archivo APK firmado.

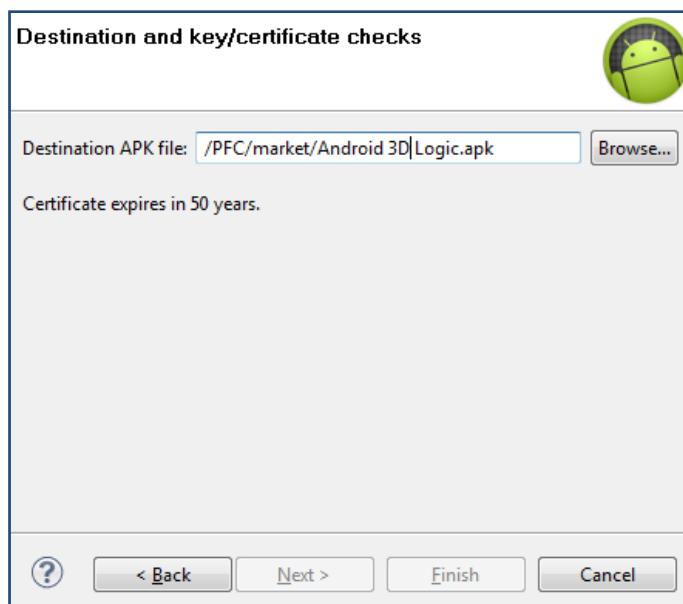


Figura B.4 Firmar aplicación, paso 3

Y eso es todo. El archivo APK generado está preparado para ser publicado en el Android Market.

C. Clases del juego

En este apartado se explica detalladamente cómo funciona la aplicación internamente. Cada clase fundamental cuenta con una explicación, pudiendo ser una breve explicación en caso de ser una clase sin demasiada complejidad, o entrando en detalle sobre algoritmos utilizados o principios básicos a tener en cuenta, junto con ejemplos concretos en caso de tratarse de los elementos más importantes para el funcionamiento del juego.

Para cada una de ellas se muestra un diagrama de clases mostrando los atributos y relaciones principales. Y por cada una de esas relaciones, las funciones a las que accede la clase citada.

C.1 A_3D_Logic

Es la clase principal, la actividad que se ejecuta nada más iniciar el juego, tal y como se indica en el *manifest* ([3.3 - Manifest](#)).

Es la encargada de crear el *GLSurfaceView* ([3.4 - GLSurfaceView](#)), configurar los parámetros básicos y ejecutar la vista OpenGL. Además, al ser la actividad principal, es la que recibe notificaciones de los sensores del móvil, como puede ser el de la pantalla táctil/multitáctil, acelerómetro o el *trackball*.

Está basada en el código de ejemplo de las APIDemos[24], que proporciona una actividad simple enlazada con una vista *GLSurfaceView* para gestionar OpenGL. Se ha añadido la gestión de los diferentes eventos de la aplicación y la capacidad de soportar transparencias.

A continuación se puede encontrar información detallada del funcionamiento de la clase. Sus atributos y relaciones principales (*Figura C.1*), así como una breve explicación de los elementos más importantes.

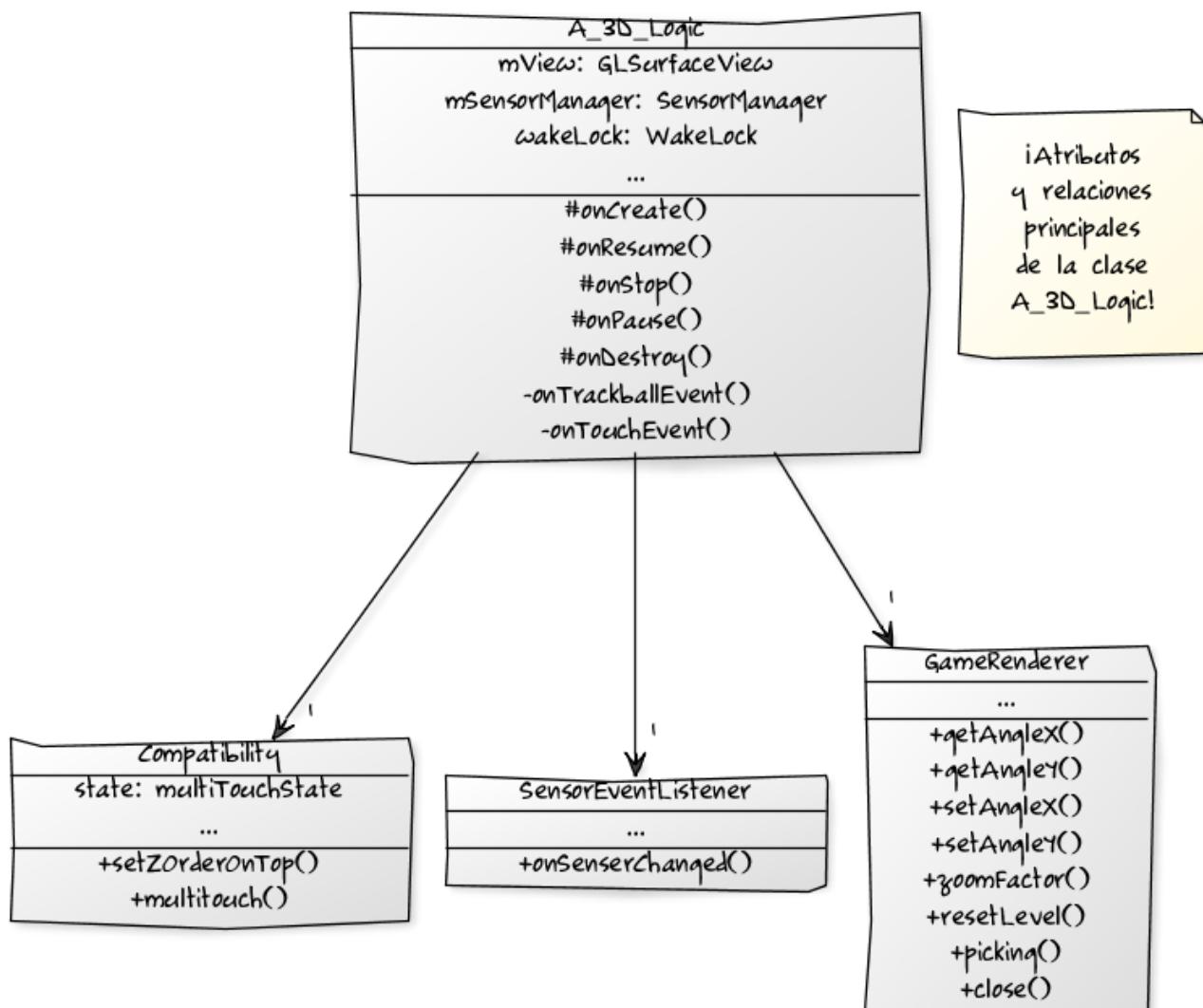


Figura C.1 Clase A_3D_Logic

Relaciones y atributos básicos

A continuación se muestra la lista de atributos relevantes de la clase. Se omiten los atributos específicos para facilitar el entendimiento.

- **mView:** Vista que nos permite mostrar una escena OpenGL en una actividad Android.
- **mSensorManager:** Encargado de gestionar los sensores de la actividad. En este caso se usa para manejar el sensor de movimiento (acelerómetro) que en la [Figura C.1](#) tiene el nombre *SensorEventListener*.
- **wakeLock:** Como se explica en apartado [3.9](#), este atributo nos permite mantener siempre activa una actividad. Al ser esta la actividad principal, es aquí donde debemos activar esta propiedad.
- **SensorEventListener:** Maneja las notificaciones de movimiento del acelerómetro, que en este caso se usa para detectar si el usuario ha agitado el móvil.
- **Compatibility:** El juego es compatible con las versiones de la API 4 en adelante. Pero algunas características usadas aún no estaban disponibles en la versión 4 de la API. En concreto la función `setZOrderOnTop()` y el *multitouch* sólo están disponibles a partir de la versión 5 de la API. Por lo tanto, se requiere implementar esas funcionalidades en esta clase para evitar problemas de compatibilidad.
- **GameRenderer:** Es el render OpenGL que se enlaza con la actividad. Se puede obtener más información sobre esta clase en el apartado [C.2](#).

Funciones y algoritmos principales

En este apartado se puede encontrar un resumen de las acciones que realizan las funciones principales de la clase.

- **onCreate:** Aquí definimos las propiedades de la actividad principal. En el fragmento [Código C.1](#) se muestran las configuraciones más importantes. En las líneas 4 y 5 se habilita la profundidad de color, los parámetros son: red, green, blue, alpha, depth, stencil. Es importante destacar que se activa el canal alpha y depth ya que en caso contrario no sería posible visualizar las transparencias de las que cuenta el juego. A continuación, en las líneas 8 y 9, se le asigna a la actividad principal el contenido, que no es otro que el *surfaceView* encargado de gestionar OpenGL, con un render personalizado *GameRenderer*.

```

1 if (android.os.Build.VERSION.SDK_INT >= 5)
2     compatibility.setZOrderOnTop(true);
3
4 mView.setEGLConfigChooser(8, 8, 8, 8, 16, 0);
5 mView.getHolder().setFormat(PixelFormat.RGBA_8888);
6
7 mRenderer = new GameRenderer(this);
8 mView.setRenderer(mRenderer);
9 setContentView(mView);

```

Código C.1 Asignar propiedades de la actividad principal

Además, es también en esta función donde se establece el *wakeLock* de la actividad, así como el sensor de movimiento y otros pequeños ajustes como el hecho de que la actividad no muestre un título estándar.

- **onResume/Stop/Pause/Destroy:** En una actividad Android, estas funciones no harían más que llamar a la función genérica de la clase Activity principal. Pero en este caso es necesario un poco más de configuración.

Para gestionar OpenGL se crea una vista especial (*GLSurfaceView*), la cual también hay que pausar, parar, destruir, etc... al mismo tiempo que la actividad principal.

Lo mismo pasa con el *wakeLock* y el sensor de movimiento. De no gestionarse permanecerían activos una vez cerrada la aplicación, y afectarían al resto de aplicaciones o simplemente consumirían batería innecesariamente.

- **onTrackballEvent:** Para aquellos terminales que disponen de *trackball*, como por ejemplo el Nexus One, esta se aprovecha para girar el cubo en los niveles del juego. Inicialmente programé una versión de prueba de esta función, pero como no podía probarla en el emulador por falta de *trackball* en este, la olvidé. Cuando finalmente dispuse de un móvil Nexus One descubrí que, al no haber prestado atención a esta función, era posible girar el cubo más allá de los límites permitidos, y tuve que ajustarla para que funcionase correctamente.

- **onTouchEvent:** En esta función se gestiona la interacción con la pantalla táctil. Tanto simple como multi-táctil en caso que el terminal sea compatible, llamando a la función *Compatibility.multitouch*.

Para la función táctil simple se diferencian tres eventos diferentes como se muestra en el [Código 3.3](#).

El movimiento del cubo hace uso de las funciones *get* y *set* para gestionar el ángulo de cámara de la clase *GameRenderer* (*Figura C.1*).

Los eventos no están divididos, ya sea al pulsar, mover o soltar siempre se llama a la función de selección (*picking*) de GameRenderer ([Figura C.1](#)), y una vez allí se decide el comportamiento dependiendo de si hemos pulsado una casilla, un botón, o nada. Esto es lo que nos permite pintar las casillas del cubo a medida que lo movemos. Dejando a esta clase únicamente la tarea de identificar en qué lugar se ha pulsado, o cuánto nos hemos desplazado.

- **Compatibility.multitouch:** En caso de que el móvil sea compatible, onTouchEvent() llamará a esta función. Aquí se comprueba que se esté interactuando con la pantalla desde dos o más puntos ([Código 3.4](#)). Y en caso de ser así, mediante la diferencia de distancia entre el primer y segundo puntero en pantalla, se calcula el tamaño del cubo y se notifica a GameRenderer mediante la función mRenderer.zoom() .
- **SensorEventListener.onSensorChanged:** Es aquí donde se gestiona el detector de movimiento tal y como se explica en [3.8 - Acelerómetro](#). Y en caso de considerar que se ha agitado el móvil, se le comunica a GameRenderer que se ha de reiniciar el nivel actual, mediante la función mRenderer.resetLevel() .

C.2 GameRenderer

Esta clase es la encargada de gestionar el render de todos los elementos del juego ([Figura C.2](#)). Controla si el juego está en el menú o un nivel del juego, y muestra unos elementos u otros dependiendo del caso.

Las decisiones se toman en esta clase ya que es la que recibe la notificación de pintar un nuevo *frame*, llamando a la función `onDrawFrame()` tan frecuentemente como lo permite el hilo de ejecución del `GLSurfaceView` que gestiona la vista OpenGL.

Es muy importante tener en cuenta que hay diferentes hilos de ejecución en la aplicación. Por ejemplo, no es el mismo hilo el encargado de mostrar un nuevo *frame* de pantalla, con todo el proceso de *render* necesario, que el encargado de recibir los eventos de la pantalla táctil. Por lo tanto, si intentas alterar lo que se muestra en pantalla cuando alguien pulsa en esta, estarás interfiriendo en la ejecución de otros hilos, pudiendo provocar multitud de errores difíciles de detectar.

Para evitar esto, las notificaciones procedentes de los sensores del móvil que gestiona la clase *A_3D_Logic*, descritas en el apartado anterior ([C.1 - A_3D_Logic](#)), se anotan y quedan pendientes de ser tratadas en el siguiente *frame* a mostrar.

A continuación se explica con más detalle cómo se encarga esta clase de gestionar los diferentes elementos del juego y hacer de puente de información para el resto de clases.

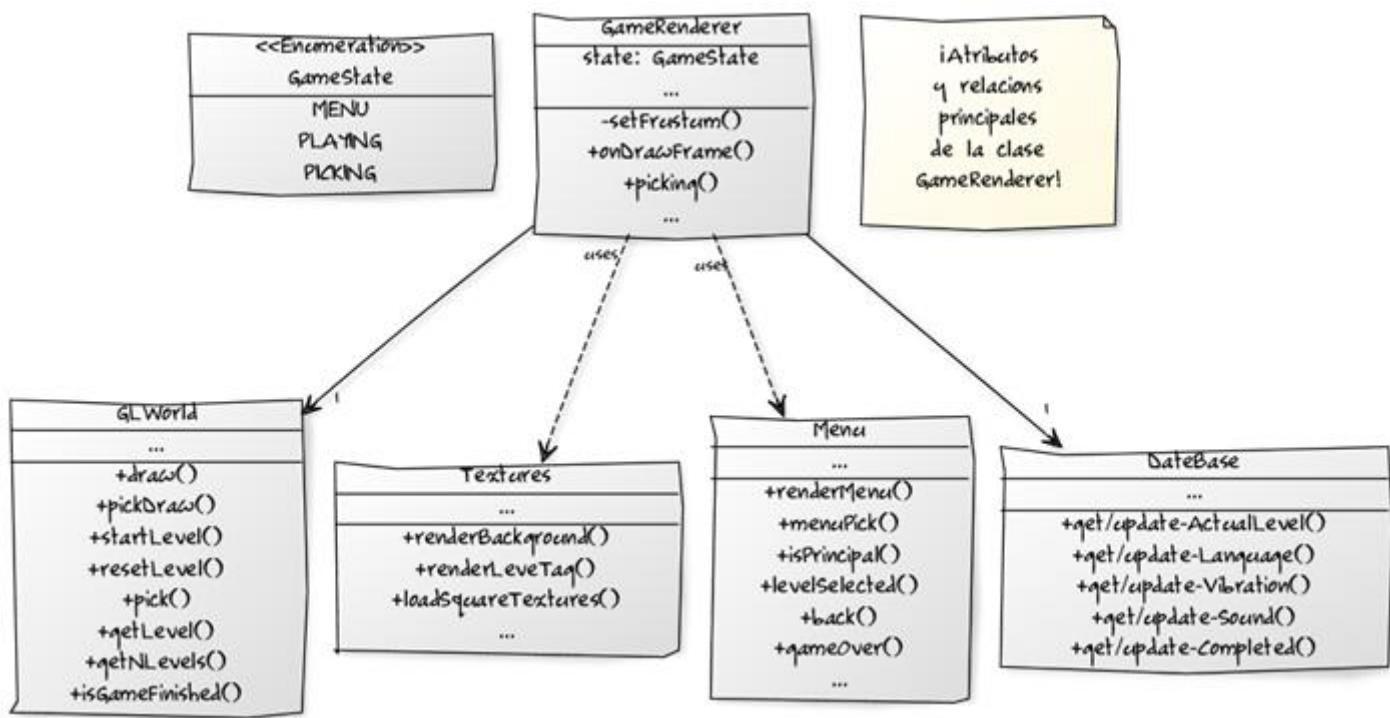


Figura C.2 Clase GameRenderer

Relaciones y atributos básicos

En esta clase, los atributos a destacar son aquellos que definen el estado del juego. Tanto instantáneo de la ejecución del juego, como el permanente registrado en la base de datos. Además de las relaciones con el resto de clases encargadas de mostrar y gestionar los elementos en pantalla, para las que GameRenderer hace de puente de información.

- **State:** En cuanto al *renderizado* en sí se refiere, hay tres estados básicos a tener en cuenta: MENU del juego, PLAYING si se está en un nivel del juego, y PICKING, un estado en el que se pinta el cubo de una forma diferente ([Código C.4](#)), para gestionar la selección de las casillas del cubo.
- **DataBase:** Al pintar cada nuevo *frame* del juego, se comprueba si ha cambiado algo que requiera ser registrado. Si se ha pasado a un nivel superior, el juego ha sido terminado, etc... es esta clase la encargada de registrarlos en la base de datos.
- **Menu:** En caso de que estemos en alguno de los apartados del menú del juego, se le cede la responsabilidad de mostrar los elementos específicos de ese apartado a la clase [Menu](#) ([C.11 - Menu](#)). Y no solo para mostrarlos. Para gestionar la selección de algún elemento de estos, también se encarga dicha clase.

- **Textures:** En caso de estar en un nivel del juego, antes de pasar a la gestión de los elementos 3D (*C.3 - GLWorld*), es aquí donde se renderiza el fondo de pantalla (*Textures.renderBackground()*), el indicador de nivel (*Textures.renderLevelTag()*), y se cargan las texturas de las casillas del cubo (*Textures.loadSquareTextures()*).
- **GLWorld:** Tanto para obtener información del nivel actual del juego, como para mostrar en pantalla los elementos 3D (el cubo en este caso) y gestionar la interacción con estos, se hace a través de la clase *GLWorld*.

Funciones y algoritmos principales

Esta clase gestiona la escena OpenGL a mostrar, y hace de puente de información en la interacción con los elementos en pantalla. Por tanto, las funciones a destacar son las siguientes:

- **setFrustum:** Aquí se definen las propiedades de la cámara. Y se ajusta el zoom del cubo en pantalla cuando el usuario lo modifica. Se utiliza una vista perspectiva.
- **onDrawFrame:** Esta función es el motor del juego. Aquí se decide qué se ha de mostrar en pantalla. En el fragmento *Código C.2* se muestra la ejecución básica. Primero se limpia la pantalla (línea 3), luego se gestionan los cambios pendientes (línea 10), y a continuación se decide qué se ha de mostrar en el *frame* actual (línea 33).

```

1  public void onDrawFrame(GL10 gl) {
2
3      // Limpiamos la pantalla
4      glTexEnvx(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
5                  GL_MODULATE);
6      gl.glClearColor(0.5f,0.5f,0.5f,1);
7      gl.glClear(GL10.GL_COLOR_BUFFER_BIT |
8                  GL10.GL_DEPTH_BUFFER_BIT);
9
10     // Gestionamos los cambios
11
12     if (zoom) [...] setFrustum(gl); // Zoom
13
14     // Si un Nuevo nivel ha sido superado, se guarda en
15     // la base de datos
16     int level = mWorld.getLevel();
17     if (level > maxLevel) {
18         db.updateActualLevel(level);
19         [...]
20     }
21     // Comprobamos si el juego se ha acabado
22     if (mWorld.gameFinished()) {
23         state = MENU;
24         Menu.gameOver(completed);
25         db.updateCompleted(true); // Lo registramos en la

```

```

26           [...]                                // base de datos.
27       }
28   // Control de profundidad para menús y escena.
29   gl.glEnable(GL10.GL_DEPTH_TEST);
30
31   // Mostramos el fondo de pantalla
32   Textures.renderBackground(gl, [...]);
33   if (state == PLAYING || state == PICKING) {
34       // Posicionamos la cámara
35       gl.glMatrixMode(GL10.GL_MODELVIEW);
36       [...]
37       gl.glRotatef(mAngleX, 1, 0, 0);
38       gl.glRotatef(mAngleY, 0, 1, 0);
39
40       // Activamos propiedades openGL necesarias para
41       // utilizar vertex array con color y texturas.
42       gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
43       glEnableClientState(GL_TEXTURE_COORD_ARRAY);
44       gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
45       [...]
46
47   if(state == PICKING) { // Casilla seleccionada
48       mWorld.pickDraw(gl, pickType);
49   }
50   Textures.renderBackToHome(gl);
51   Textures.renderLevelTag(gl, level);
52   Textures.loadSquareTextures(gl);
53   **** mWorld.draw(gl); **** // Mostrar escena 3D
54   }
55   else if (state == MENU) {
56       Menu.renderMenu(gl, [...]);
57       [...]
58   }
59 }
```

Código C.2 Pintar nuevo frame de pantalla

picking: Esta función recibe la posición (x, y) pulsada en pantalla por el usuario y la gestiona dependiendo del estado del juego.

Primero de todo se comprueba si se ha pulsado el botón de vuelta al menú/menú anterior, en tal caso esa es la única acción a realizar.

Si estamos en un apartado del menú, se pide a la clase `Menu` que comunique si se ha seleccionado alguna opción, y en tal caso se ejecuta la acción requerida: Iniciar nivel X del juego, activar/desactivar sonido/vibración, etc...

Si estamos en un nivel del juego, se anota una posible selección de casilla del cubo, y en el siguiente pintado de la escena se ejecuta el algoritmo de selección de casillas (*Código C.4*).

Esta función trajo algunos problemas a causa de los diferentes hilos de ejecución. Para un funcionamiento correcto, antes de decidir qué hacer cuando se ha pulsado la pantalla, deben haberse tratado las selecciones anteriores. Parece lógico, el problema es que para tratar la selección de objetos se ha de esperar al siguiente *frame*, y antes de eso pueden llegar varios eventos de la pantalla táctil al mismo tiempo. La solución fue hacer que el hilo de ejecución de pintado de cada *frame* notificase cuándo había acabado la selección de objetos, y hasta entonces se ignoran las peticiones.

C.3 GLWorld

Esta clase es la encargada de gestionar la información de la escena 3D (*Figura C.3*). Almacena los *buffers* de vértices, texturas y colores para renderizar la escena; se encarga de ejecutar el algoritmo de selección por color utilizando un buffer especial con un color diferente para cada casilla; y hace de puente de información entre GameRenderer y Game hacia Cube .

La gestión de los *buffers* y *vertexArray* se aprovechó parcialmente de las APIDemos [24]. Por lo que lo que más trabajo llevó fue configurar adecuadamente el algoritmo de selección *pickDraw()* .

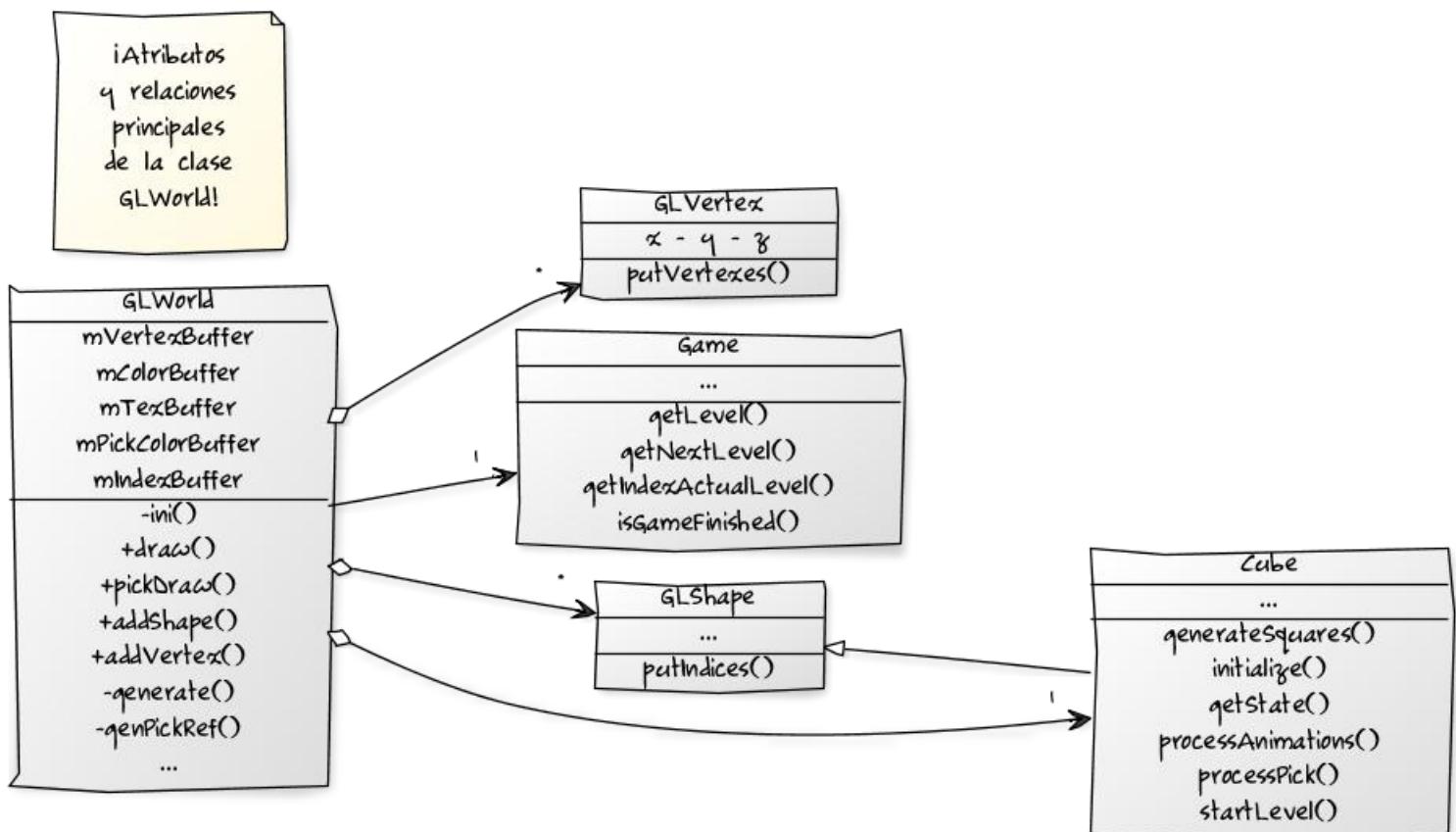


Figura C.3 Clase GLWorld

Relaciones y atributos básicos

En esta clase, al tratarse de la encargada de mostrar la escena 3D, los atributos que cabe mencionar son las figuras en escena y *buffers* OpenGL para mostrar.

- **mVertexBuffer:** Conjunto de vértices de la escena
- **mColorBuffer:** Configuración de color de las casillas.
- **mTexBuffer:** Coordenadas de texturas de cada casilla.
- **mPickColorBuffer:** *Buffer* especial con un color diferente para cada casilla para poder aplicar el algoritmo de selección de objeto por color.
- **mIndexBuffer:** Índices de las caras del cubo
- **Game:** Desde esta clase se gestiona la obtención de niveles que finalmente se le pasan a la clase Cube.
- **Cube:** Es el único objeto en escena. Esta clase hace de puente de información. Le indica cuándo gestionar la selección de una casilla, cargar un nivel nuevo, etc.
- **GLVertex:** Es aquí donde se almacenan todos los vértices individuales de la escena para poder generar los *buffers* OpenGL.

Funciones y algoritmos principales

El pintado normal y el algoritmo de selección son las funciones principales de la clase. Aunque también cabe destacar las funciones necesarias para crear los elementos de la escena.

- **ini:** En esta función se prepara la escena para un nuevo nivel del juego. Se ordena a la clase Cube que inicie el nivel `myCube.initialize()`, y finalmente se generan los buffers OpenGL.
- **draw:** Esta es una de las funciones principales del juego. Se encarga de renderizar la escena 3D a partir de los buffers OpenGL. Además se encarga también de comprobar si hay que pasar al nivel siguiente del cubo, o si ya se ha superado el juego (`myGame.gameFinished()`), y le pasa el *buffer* de color a la clase Cube para aplicar las animaciones de color. A continuación se muestra el código ([Código C.3](#)) encargado del renderizado de escena.

```

1  public void draw(GL10 gl) {
2      // Reiniciar posición de buffers
3      mColorBuffer.position(0);
4      mVertexBuffer.position(0);
5      mIndexBuffer.position(0);
6      mTexBuffer.position(0);
7
8      // Obtener cubo y comprobar si se ha pasado de nivel

```

```

9      // o completado el juego
10     Cube myCube = ((Cube) mShapeList.get(CUBE_INDEX));
11     if (myCube.getState() == Cube.COMPLETED) {
12         if (myGame.gameFinished()) {
13             gameFinished = true;
14             return;
15         }
16         myCube.startLevel(myGame.getNextLevel(
17                             mirrorAllowed));
17         ini(myCube);
18     }
19     // Si se ha agitado el móvil y hay que reiniciar.
20     if (reset) {
21         reset = false;
22         myCube.initialize();
23         generate();
24     }
25     // Aplicar animaciones de color al cubo
26     myCube.processAnimations(mColorBuffer);
27
28
29
30     gl.glEnable(GL10.GL_CULL_FACE);
31     gl.glCullFace(GL10.GL_BACK);
32     gl.glFrontFace(GL10.GL_CW);
33     gl.glShadeModel(GL10.GL_FLAT);
34     // Preparar buffers
35     gl.glVertexPointer(3, GL10.GL_FIXED, 0, mVertexBuffer);
36     gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, mTexBuffer);
37     gl glColorPointer(4, GL10.GL_FIXED, 0, mColorBuffer);
38     // Pintar escena
39     gl.glDrawElements(GL10.GL_TRIANGLES, mIndexCount,
40                         GL10.GL_UNSIGNED_SHORT, mIndexBuffer);
41 }
```

Código C.3 Draw (Pintado de escena 3D)

- **pickDraw:** Aquí se implemente el algoritmo de selección ([Código C.4](#)). Concretamente la primera parte, encontrar qué casilla ha sido pulsada. Para ello se utiliza el buffer especial con un color diferente para cada casilla. Se pinta el cubo en escena y mediante la función `glReadPixels()` se obtiene el color del pixel pulsado en pantalla para descifrar qué casilla se ha pulsado. El problema es que es un valor aproximado. Inicialmente se usaba un color primario (rojo, verde, azul) con diferentes tonalidades para cada cara. Se empezaba con la máxima saturación (255) y se iba restando poco a poco. El problema es que el error se va acumulando, por lo que las tonalidades más claras (últimas casillas de la cara) tenían muy poca precisión y se generaban errores de aproximación, y al pulsar una cara se seleccionaba una de las de los lados.

Finalmente en lugar de usar un componente de color por cara pasé a usar dos y me limité a llegar a tonalidad media para evitar fallos de acarreo de error. Y esta última configuración finalmente funcionó de forma adecuada.

Si el algoritmo concluye que efectivamente se ha pulsado una casilla, se le cede la responsabilidad de tratar el evento a la clase Cube mediante la función myCube.precessPick() .

```

1  public int pickDraw(GL10 gl, int type)
2  {
3      gl.glDisable(GL10.GL_TEXTURE_2D);
4      // Limpiar escena para que nada interfiera en el
5      // algoritmo de selección
6      gl.glClear(GL10.GL_COLOR_BUFFER_BIT |
7                  GL10.GL_DEPTH_BUFFER_BIT);
8      [...] // Preparar buffers, especial atención al de color
9      gl.glVertexPointer(3, GL10.GL_FIXED, 0, mVertexBuffer);
10     gl glColorPointer(4, GL10.GL_FIXED, 0, mPickRef);
11     gl.glDrawElements(GL10.GL_TRIANGLES, mIndexCount,
12                       GL10.GL_UNSIGNED_SHORT, mIndexBuffer);
13
14     // Algoritmo de selección
15     ByteBuffer PixelBuffer = ByteBuffer.allocateDirect(4);
16     PixelBuffer.order(ByteOrder.nativeOrder());
17     gl.glPixelStorei(GL10.GL_UNPACK_ALIGNMENT, 1);
18
19     // Leer color del pixel pulsado.
20     gl.glReadPixels(pickingX, pickingY, 1, 1, GL10.GL_RGBA,
21                     GL10.GL_UNSIGNED_BYTE, PixelBuffer);
22     // Borrar el cubo especial de selección
23     gl.glClear(GL10.GL_COLOR_BUFFER_BIT |
24                 GL10.GL_DEPTH_BUFFER_BIT);
25
26     // Descifrar casilla a partir de color aproximado
27     byte b[] = new byte[4];
28     PixelBuffer.get(b);
29
30     int levelSize = ((Cube)
31     mShapeList.get(0)).getLevelSize();
32     int tamSide = levelSize*levelSize;
33     int cara = -1, faceCube = -1, square = -1;
34     // Cara 1 ?
35     if (b[0] == 0 && b[1] != 0 && b[2] != 0) {
36         cara = Cube.kTop;
37         faceCube = (256-(b[1] < 0? b[1]+256: b[1]))/8*16;
38         faceCube += (256-(b[2] < 0? b[2]+256: b[2]))/8;
39         square = faceCube;
40     // Cara 2?
41     } else if (b[0] != 0 && b[1] == 0 && b[2] != 0) {
42         cara = Cube.kLeft;
43         faceCube = (256-(b[0] < 0? b[0]+256: b[0]))/8*16;
44         faceCube += (256-(b[2] < 0? b[2]+256: b[2]))/8;
45         square = faceCube + tamSide;
46     // Cara 3?
47     } else if (b[0] != 0 && b[1] != 0 && b[2] == 0) {
48         cara = Cube.kFront;
```

```

49         faceCube = (256-(b[0] < 0? b[0]+256: b[0]))/8*16;
50         faceCube += (256-(b[1] < 0? b[1]+256: b[1]))/8;
51         square = faceCube + 2*tamSide;
52     }
53
54     // Si se ha seleccionado una casilla, se gestiona el
55     // evento
56     if ( cara != -1) ((Cube)
57             mShapeList.get(CUBE_INDEX).processPick(square,
58                                         mTexBuffer, tipe);
59
60     gl.glEnable(GL10.GL_TEXTURE_2D);
61     return cara;
}

```

Código C.4 Algoritmo de selección (por color)

- **addShape:** Mediante esta función se añaden elementos 3D a la escena. En este caso únicamente el cubo.
- **addVertex:** Mediante esta función se añaden todos los vértices de la escena. Al crear cada elemento 3D ya se encarga de enviar los vértices a esta clase.
- **generate:** Partiendo de la lista de vértices y elementos 3D, esta función genera los *buffers* OpenGL necesarios para renderizar la escena.
- **genpickRef:** Antes de generar los *buffers* normales, antes incluso de que la clase Cube cargue la configuración de casillas del nivel, esta función genera el *buffer* especial con un color diferente para cada casilla (cosa que realiza la clase Cube por defecto al crear la casillas de un cubo de tamaño concreto) que se usa en el algoritmo de selección por color.

C.4 Cube

Esta clase es la encargada de gestionar todo lo relacionado con el cubo 3D ([Figura C.4 Clase Cube](#)). Tanto la configuración del nivel actual en las diferentes casillas, como la resolución de niveles, o las diferentes animaciones de color y efectos sonoros y de vibración.

Cabe destacar el algoritmo de resolución mediante un DFS (Depth-first search) y la gestión de selección de casillas, que se encarga de gestionar la lógica del juego.

Relaciones y atributos básicos

Al tratarse del cubo, todos los atributos que definen el estado de las casillas y completitud del nivel actual son importantes

- **state:** El cubo puede estar en diferentes estados. Modo normal de juego, apareciendo al iniciar un nivel, desapareciendo al finalizarlo, o completo a la espera de que se cargue un nuevo nivel. Puesto que al inicial y finalizar el cubo se aplican animaciones de aparición y desaparición, se ha de esperar a que la clase Animation confirme que se han acabado las animaciones para cambiar de estado mediante `isRdyToChangeLevel()` y `idRdyToPlay()`.
- **pathCompleted:** Un *array* de *booleans* que indica qué caminos están completos y cuáles no.
- **vib:** Instancia de la clase `Vibrator` de Android que nos permite hacer vibrar el móvil de diferentes formas.
- **Square:** El cubo está formado de casillas (`Square`) que lo componen. Igual que una figura 3D está compuesta por caras formadas por triángulos. Cada casilla almacena la información del tipo de casilla que es y el identificador de las casillas cercanas a ella de cara al cálculo de caminos conectados.
- **Level:** Instancia del nivel que se está jugando actualmente. Se usa para configurar el cubo: tamaño, posición de generadores, casillas bloqueadas, etc.
- **Animations:** Clase encargada de aplicar animaciones de color a las casillas del cubo.
- **SoundEffect:** Al procesar la selección de una casilla, dependiendo de la lógica de juego que se haya de aplicar se reproducen sonidos mediante esta clase. Como pueden ser el de pintar casilla, camino completo, romper camino, activar generador de color o superar el nivel.

- **GLColor:** Puesto que para los niveles siempre se usa el mismo conjunto de colores, `GLColor` tiene pre-creados los colores básicos, y todas las casillas referencian a las mismas instancias de un mismo color, evitando duplicar información innecesaria. Además la clase `GLColor` ofrece funciones para tratar los colores, oscurecer, transparentar, etc, que se usan para las animación y otras funciones.

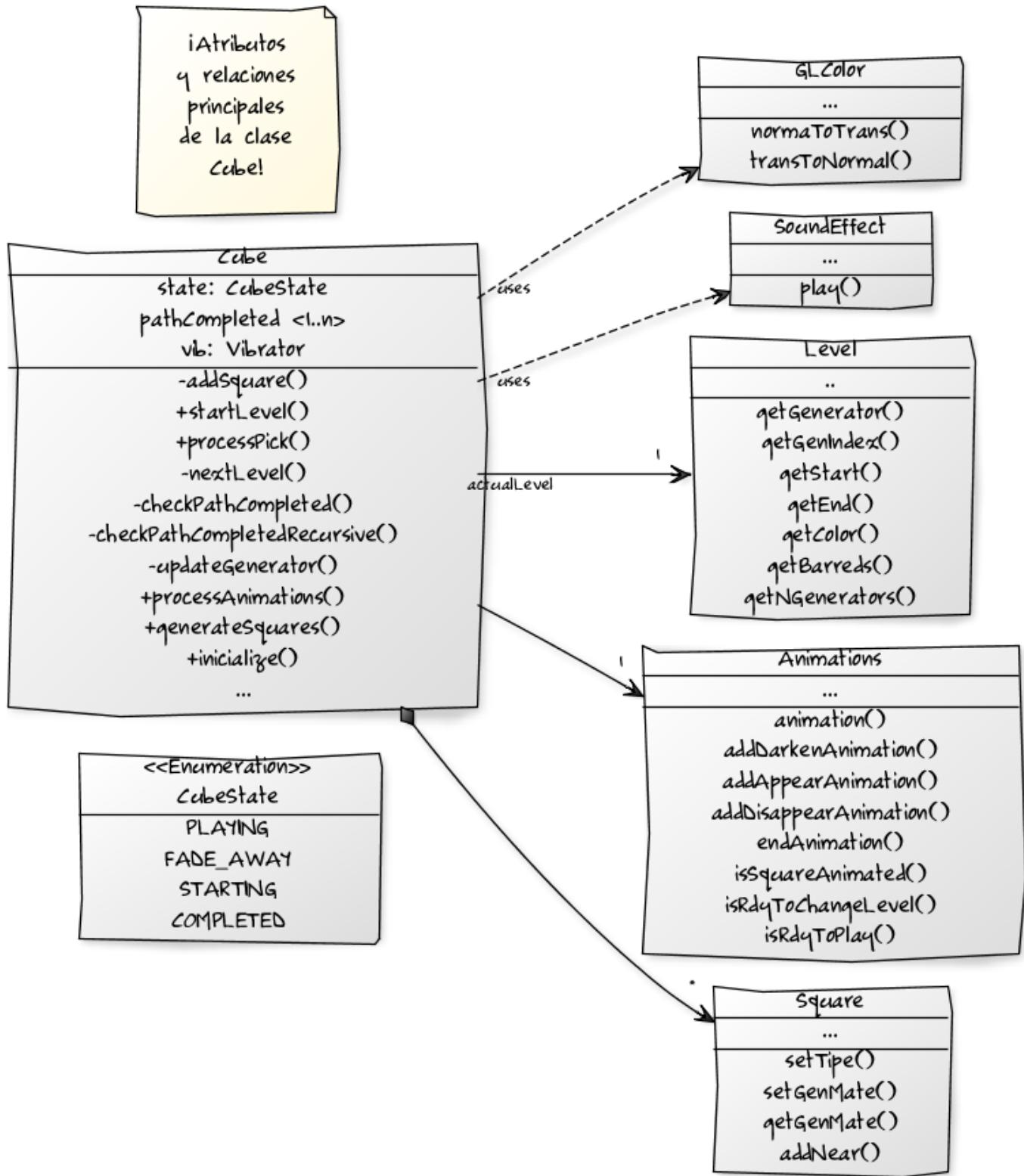


Figura C.4 Clase Cube

Funciones y algoritmos principales

Esta clase cuenta con un par de las funciones más importantes del juego. El algoritmo de resolución de caminos y la función de gestión de la lógica del juego al pintar una nueva casilla.

- **startLevel:** Asigna un nuevo nivel al cubo y limpia la lista de casillas antigua.
- **processPick:** Una de las funciones principales de la clase y del juego. Se encarga de gestionar la lógica del juego y la interacción con las casillas. A continuación ([Código C.5](#)) se muestra de forma resumida la lógica seguida para gestionar el pintado de una casilla.

```

1  public void processPick(int square, FloatBuffer mTexBuffer,
2  int pickType) {
3      // Obtenemos casilla y su tipo
4      Square cas = (Square) mFaceList.get(square);
5      int tipeC = cas.getType();
6
7
8      1) // Si pulsamos una casilla normal, ignorando si dejamos el
9         // dedo pulsado en pantalla y tratando sólo la primera vez
10     if (tipeC == Square.Normal && /*Primera vez*/) {
11         GLColor oldColor = cas.getColor();
12         cas.setColor(actualColor);
13         // Si no la acabamos de pulsar hace un momento
14         // añadimos animación de color
15         if (anims.casillaAnimated(square) != -1)
16             anims.addDarkenAnimation(...);
17
18         if /* Pintamos casilla de un color diferente */ {
19             if /*Pintamos en un camino completado*/ {
20
21                 // Comprobamos no haber roto un color ya completado
22                 int pathState =
23                     checkPathCompleted(oldColor, mTexBuffer);
24
25             }
26
27             if /*El camino actual no está completo*/ {
28                 int pathState =
29                     checkPathCompleted(actualColor, mTexBuffer);
30
31                 // Si hemos completado el nivel, siguiente!
32                 if (levelCompleted()) nextLevel();
33
34             }
35         }
36         if /* Pintado sin sorpresas*/
37             SoundEffects.play(SoundEffects.NORMAL_CLICK_SOUND);
38     }
39     2) // La casilla es un generador desactivado, la ignoramos
40     // en caso de pintado por desplazamiento.
41     else if (tipeC == Square.GeneratorOFF && pickType !=
42             CURSOR_MOVE) {
43         if /* No acabamos de pulsar esta casilla*/
44             SoundEffects.play(SoundEffects.GENERATOR_CLICK_SOUND);

```

```

45
46     [...] // Actualizar generador actual
47     actualGenerator = square;
48     updateGenerators(...);
49     actualColor = cas.getColor();
50
51     [...] // Añadir animaciones a los dos generadores
52     anims.addDarkenAnimation(squares)
53 }
54 3) // La casilla es uno de los generadores activos
55 else if (tipeC == Square.GeneratorON && pickTipe != CURSOR_MOVE) {
56     // Unicamente sonido y animación
57     SoundEffects.play(SoundEffects.GENERATOR_CLICK_SOUND);
58     anims.addDarkenAnimation(squares);
59     [...]
60 }
61
62 4) // La casilla es un generador completado
63 else if (tipeC == Square.GeneratorOK && pickTipe != CURSOR_MOVE) {
64     SoundEffects.play(SoundEffects.GENERATOR_CLICK_SOUND);
65     [...]// Actualizar generador actual
66     updateGenerators(...);
67
68     actualGenerator = square;
69     actualColor = cas.getColor();
70     // Animación
71     anims.addDarkenAnimation(squares);
72     [...]
73 }
74
75 prevSquare = square;
76 }

```

Código C.5 Procesar pintado de una casilla

- **nextLevel:** Activa la animación de desaparición del cubo (aplicando la animación addDisappearAnimation() a todas las casillas individuales) y hace uso de la clase Vibrator para hacer vibrar el móvil durante aproximadamente un segundo.
- **checkPathCompletedRecursive:** Este es el algoritmo recursivo DFS encargado de comprobar si hay un camino entre los dos generadores de un color a través de las casillas pintadas con ese color ([Código C.5](#)). En caso de ser así, se guarda el camino encontrado para aplicar una animación a todas las casillas que lo componen.

```

1 private boolean checkPathCompletedRecursive(int gen,
2 boolean[] visited, GLColor color, ArrayList<Integer> path)
3 {
4     // Obtenemos la casilla
5     Square square = (Square) mFaceList.get(gen);
6
7     int index = path.size();
8     path.add(gen);
9     boolean result = false;

```

```

10     ArrayList<Integer> mates = square.getNear();
11     // Para todas las casillas cercanas mientras aún no se
12     // ha encontrado un camino.
13     for (int i = 0; i < mates.size() && !result; ++i) {
14         if (!visited[mates.get(i)]) {
15             int mateIndex = mates.get(i);
16             Square mate = (Square) mFaceList.get(mateIndex);
17             int mateType = mate.getType();
18             GLColor mateColor = mate.getColor();
19             if /* Hemos llegado al otro generador */ {
20
21                 // Camino encontrado!
22                 path.add(mateIndex);
23                 result = true;
24             }
25             else if /* Casilla del mismo color */ {
26
27                 // No los lo apuntamos y probamos con sus compañeros
28                 visited[mateIndex] = true;
29                 result =
30                 checkPathCompletedRecursive(mateIndex, ...);
31             }
32         }
33     }
34     if (!result) path.remove(index);
35     return result;
36 }
```

Código C.6 Buscar camino entre generadores (DFS)

- **updateGenerator:** Actualiza el estado de los generadores de color dependiendo de si están activos, inactivos, o el color ya está completo
- **processAnimations:** Pasa el buffer de color a la clase Animations para que se apliquen las animaciones de color pertinentes.
- **generateSquares:** Crea todas las casillas del cubo de tamaño definido por el nivel actual. Por defecto con un color diferente para cada casilla para poder aplicar el algoritmo de selección mediante addAppearAnimation() por cada casilla
- **initialize:** Configura el cubo para representar el nivel actual: posición de generadores, casillas bloqueadas, compañeros de cada casilla, etc. Y activa la animación de aparición del cubo.

C.5 Square

Esta clase es una extensión de la clase `GLFace`, como podemos ver en la *Figura C.5*. Es aquí donde se guarda el tipo de casilla e identificadores de las casillas cercanas en el cubo, sean o no de la misma cara.

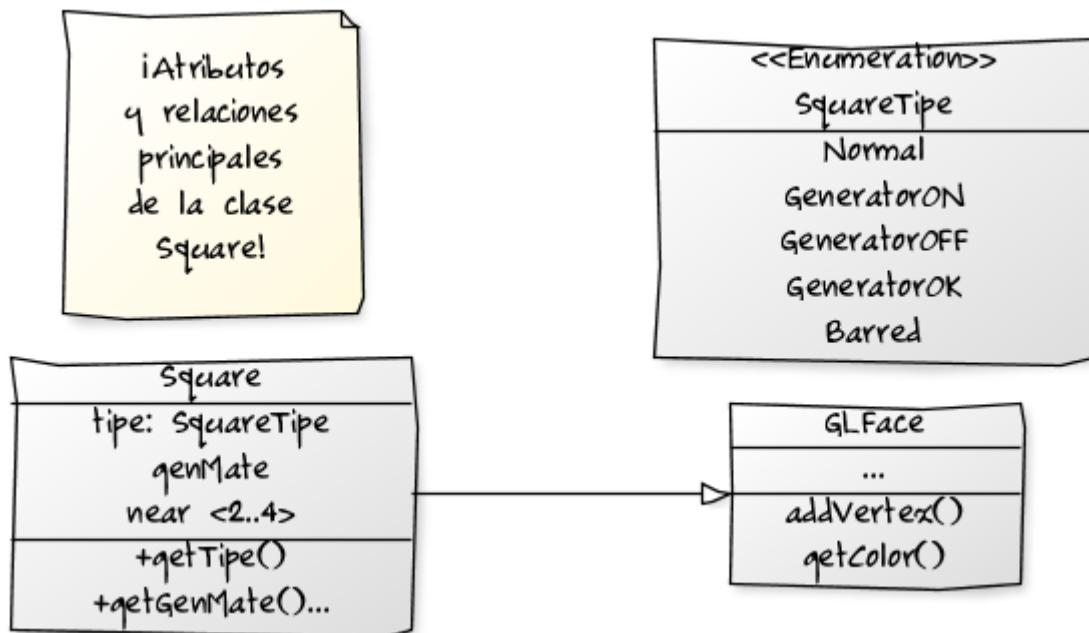


Figura C.5 Clase Square

Atributos básicos

Sólo cabe destacar aquellos atributos que definen el estado de la casilla.

- **tipe:** El tipo de la casilla. Puede ser una casilla normal que se pueda pintar, un generador de color activo/desactivo/completo, o una casilla bloqueada.
- **genMate:** En caso de ser un generador, este atributo es el identificador de casilla del otro generador del mismo color.
- **near:** Lista de identificadores de casillas cercanas a esta

C.6 Game

Esta clase es la encargada de gestionar los diferentes niveles del juego (*Figura C.6*). Controlar por qué nivel va el usuario y si ya se ha pasado el juego generar niveles espejo a partir de los originales.

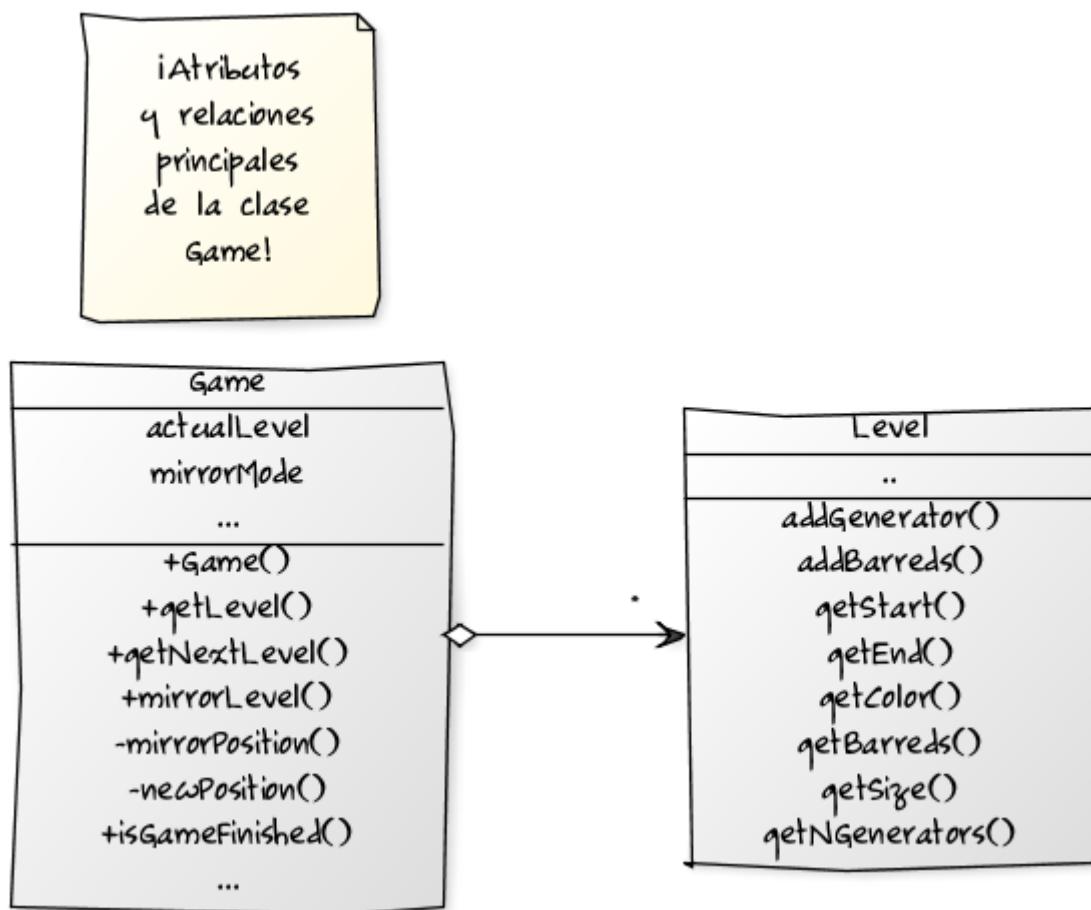


Figura C.6 Clase Game

Relaciones y atributos básicos

Aquí se define el estado del juego en cuanto a los niveles.

- **actualLevel:** Se va actualizando con el número del nivel que se juega en cada momento.
- **mirrorMode:** Booleano que nos indica si el modo espejo está activo.
- **Levels:** Lista de niveles (clase Level, *C.7 - Level*) de los que cuenta el juego.

Funciones y algoritmos principales

En esta clase encontramos las funciones que gestionan, crean o modifican los niveles.

- **Game:** En la creadora de la clase se crean todos los niveles del juego. Por cada uno de ellos se sigue la estructura que podemos ver en el [Código C.7](#). Cabe destacar que los identificadores de cada casilla son consecutivos. Por cara, de la primera casilla de arriba a la izquierda a la última casilla de abajo a la derecha. Empezando por la cara de arriba del cubo, luego la izquierda, y por último la derecha.

```

1 // Crear nivel
2 Level level1 = new Level(3);
3     // Añadir generadores del nivel
4     level1.addGenerator(GLColor.BLUE, 0, 25);
5     level1.addGenerator(GLColor.GREEN, 1, 26);
6     level1.addGenerator(GLColor.RED, 9, 11);
7     // Indicar casillas bloqueadas
8     level1.setBarreds(new int[]{4, 7, 10, 13, 19, 22});
9 levels.add(level1);

```

Código C.7 Añadir nivel nuevo

- **getLevel:** Devuelve la instancia del nivel solicitado.
- **getNextLevel:** Devuelve la instancia del nivel siguiente al jugado actualmente
- **mirrorLevel:** Este es el algoritmo que transforma un nivel normal en una de sus 5 variantes alternando caras y colores. Como podemos ver en el [Código C.8](#), se abstrae el problema en sí a comprobar en qué posición de una cara está originalmente una casilla, y dependiendo de la transformación aleatoria seleccionada, asignarle una nueva fila, columna y cara.

```

1 public Level mirrorLevel(int level) {
2     Level normal = levels.get(level);
3     // Elegir transformación a aplicar
4     int type = rand.nextInt(999)%50;
5     if (type == NORMAL) return normal;
6     type = type/10 + 1;
7
8     [...] // Obtener datos de la casilla
9
10    mirrorLevel = new Level(size);
11    ArrayList<GLColor> colors = new ArrayList<GLColor>();
12
13    // Para cada generador de color original
14    for (int i = 0; i < nGen; ++i) {
15        // Le cambiamos el color
16        GLColor color =
17            colors.remove(rand.nextInt(999)%colors.size());
18        mirrorLevel.addGenerator(color,
19        // Asignamos nuevas posiciones espejo

```

```

20         mirrorPosition(type, size, normal.getStart(i)),
21         mirrorPosition(type, size, normal.getEnd(i)));
22     }
23     // Y también por cada casilla bloqueada
24     for (int i = 0; i < normal_barreeds.size(); ++i) {
25         int oldPos = normal_barreeds.get(i);
26         // Asignamos su posición espejo correspondiente
27         barreeds[i] = mirrorPosition(type, size, oldPos);
28     }
29     mirrorLevel.setBarreeds(barreeds);
30 }
31
32 }
33
34 private int mirrorPosition(int type, int size, int old){
35     int faceSize = size*size;          // Tamaño de la cara
36     int face = old/faceSize;          // Numero de cara
37     int faceSquare = old%faceSize;    // Numero de casilla en
38                                         // esa cara
39     int row = faceSquare / size + 1;
40     int column = faceSquare % size + 1;
41     // Dependiendo del tipo de transformación elegida
42     if (type == MIRROR_LEFT) {
43         // Alternar las caras en sentido antihorario
44         // Dependiendo de la cara en la que estemos
45         if (face == 0)
46             // Asignamos una nueva fila, columna y cara
47             return newPos(-row, -column, size, faceSize);
48         else if (face == 1)
49             return newPos(-column, row, size, faceSize*2);
50         else if (face == 2)
51             return newPos(-column, row, size, 0);
52     }
53     else if (type == MIRROR_RIGHT) {
54         // Etc... // Alternar las caras en sentido horario
55     } else if (type == MIRROR_UP_LEFT) {
56         // Etc... // Alternar la cara de arriba con la izquierda
57     } else if (type == MIRROR_UP_RIGHT) {
58         // Etc... // Alternar la cara de arriba con la derecha
59     } else if (type == MIRROR_LEFT_RIGHT) {
60         // Etc... // Alternar la cara izquierda con la derecha
61     }
62     return -1;
63 }
64
65 private int newPos(int row, int column, int size, int ini)
66 {
67     // Realizamos los calculos necesarios para obtener el
68     // identificador de la casilla en el cubo partiendo
69     // de la fila, columna y cara.
70     if(row > 0) --row;
71     int newRow = (size+row)%size;
72
73     if(column > 0) --column;
74     int newColumn = (size+column)%size;
75
76     return (ini + newRow*size + newColumn);
77 }

```

Código C.8 Cálculo de nivel espejo

- **isGameFinished:** Esta función nos indica si ya se ha superado el último nivel del juego, y por lo tanto se ha acabado el juego.

C.7 Level

Como vemos en la *Figura C.7*. Un nivel está compuesto por más de un generador de color y un conjunto de indicadores de casillas bloqueadas.

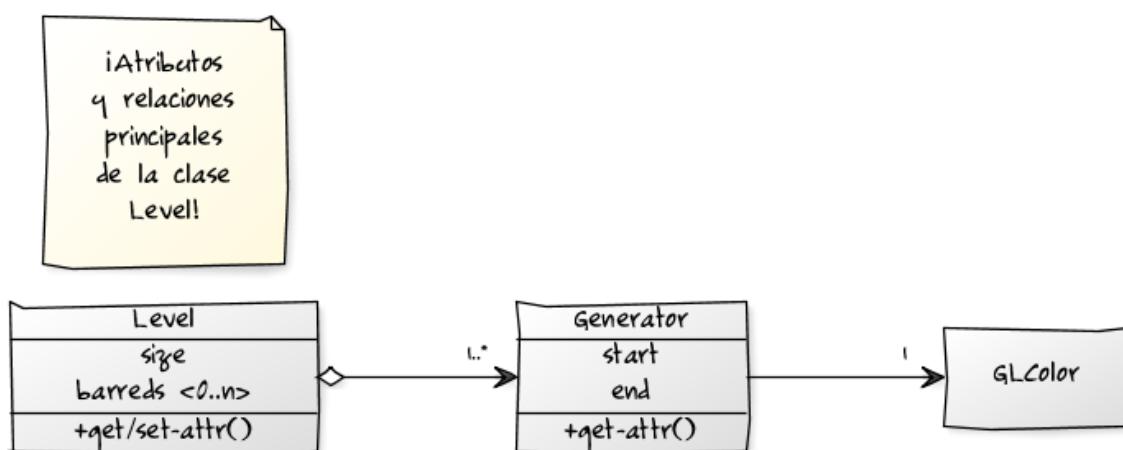


Figura C.7 Clase Level

Relaciones y atributos básicos

Aquí se define la composición de un nivel del juego

- **size:** El tamaño de cubo necesario, puede ser un valor entre 3 y 6.
- **Generator:** Cada generador de los que componen un nivel registra el identificador de casilla generadora de inicio y fin de un camino de color. Además de la referencia a un color primario de los que ofrece la clase GLColor.
- **barreds:** Lista de identificadores de casillas bloqueadas del nivel.

C.8 Textures

Como vemos en la *Figura C.8*, en esta clase se almacenan los identificadores de las texturas, y se encuentran las funciones de renderizado de cada textura. Cabe destacar que puesto que los diferentes móviles con sistema operativo Android pueden contar con pantalla de dimensiones totalmente dispares, el renderizado de las diferentes texturas no se hace a partir de las dimensiones originales de esta, si no basándose en porcentajes del tamaño de la pantalla del móvil pertinente.

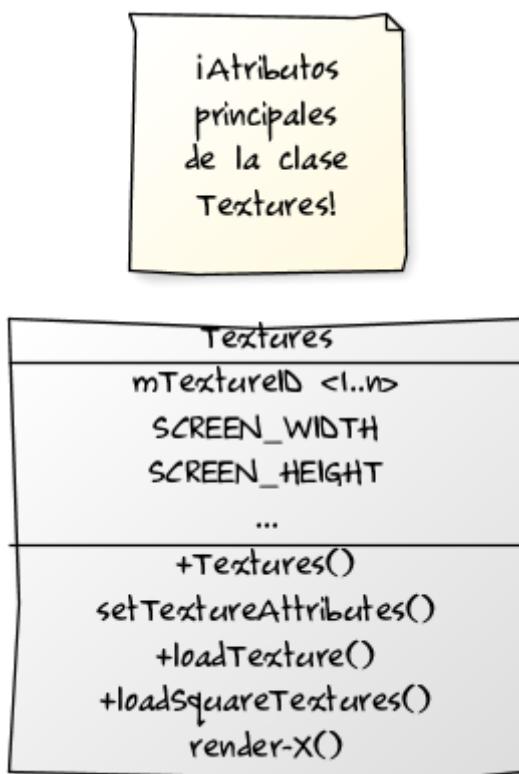


Figura C.8 Clase Textures

Relaciones y atributos básicos

- **mTextureID:** Lista de identificadores de las texturas de las que cuenta el juego.

Funciones y algoritmos principales

- **Textures:** En la creadora de esta clase se cargan todas las texturas y se asignan los parámetros OpenGL específicos.
- **setTextureAttributes:** Esta función es la encargada de asignar las propiedades de las texturas usadas, entre ellos, que no tengan repetición, que se puedan mezclar con un color de fondo, etc.

- **loadTexture:** Esta función carga la imagen ([Código 3.5](#)) y le asigna un identificador.
- **loadSquareTextures:** Prepara las texturas de las casillas para que estén activas cuando se renderice el cubo en escena.
- **render-“X”:** La función específica para cada textura a mostrar en pantalla. Todos los niveles tienen su propia función. Como comentaba en la introducción de la clase, el tamaño final depende de las proporciones de la pantalla del móvil, en porcentajes, en lugar del tamaño original de las texturas.

C.9 Animations

Como vemos en el diagrama (*Figura C.9*), esta clase se encarga de aplicar las animaciones de colores a las casillas del cubo. Pueden ser animación de aparición/desaparición al iniciar o finalizar un nivel, o pasar del negro al color de las casillas al interactuar con estas.

Cabe destacar que los cambios se aplican directamente al buffer de color de elementos de escena que nos proporciona GLWorld (*C.3*).

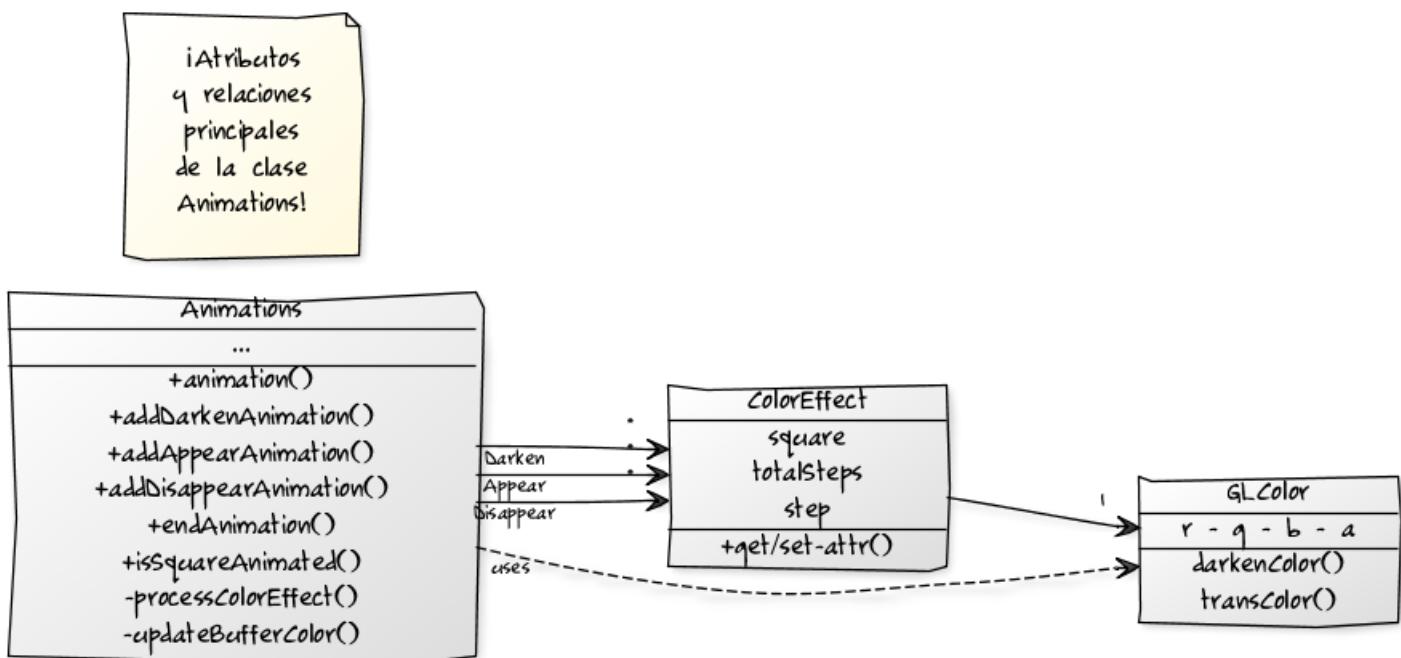


Figura C.9 Clase Animations

Relaciones y atributos básicos

Estos son los atributos y relaciones a destacar:

- **ColorEffect:** Esta clase representa el estado de un efecto de color. A una casilla concreta, una animación de un número de pasos, y el paso por el que va la animación.
- **GLColor:** La clase encargada de aplicar las transformaciones a los colores es finalmente GLColor, que ofrece funciones de oscurecer o transparentar en cierto grado un color dado.

Funciones y algoritmos principales

Las funciones de esta clase se encargan de gestionar el estado de las animaciones.

- **animation:** Aplica las animaciones pendientes a las casillas del cubo
- **endAnimation:** En caso de ser necesario, elimina todas las animaciones pendientes del cubo. Por ejemplo si finalizamos un nivel, se ha de pasar a la animación de desaparición y las animaciones pendientes sobran.
- **isSquareAnimated:** Nos devuelve si la casilla ya cuenta con una animación. No tiene sentido que una casilla tenga más de una animación al mismo tiempo, por lo que es importante comprobarlo.
- **processColorEffect:** Es aquí donde se ejecuta el cambio de color en sí de una casilla en un *frame* concreto. Llamando a las funciones que proporciona la clase `GLColor`, `darkenColor()` y `transColor()` con los parámetros necesarios. Y finalmente se llama a `updateBufferColor()`
- **updateBufferColor:** Aquí simplemente se cambia el color en la posición del *buffer* correspondiente a la casilla implicada.

C.10 SoundEffects

Esta clase se encarga de gestionar los efectos sonoros del juego. En el diagrama (*Figura C.10*), podemos ver, a modo de enumeración, los efectos sonoros disponibles en el juego. Los cuales fueron obtenidos de diferentes páginas online que ofrecen efectos gratuitos: [28] y [29].

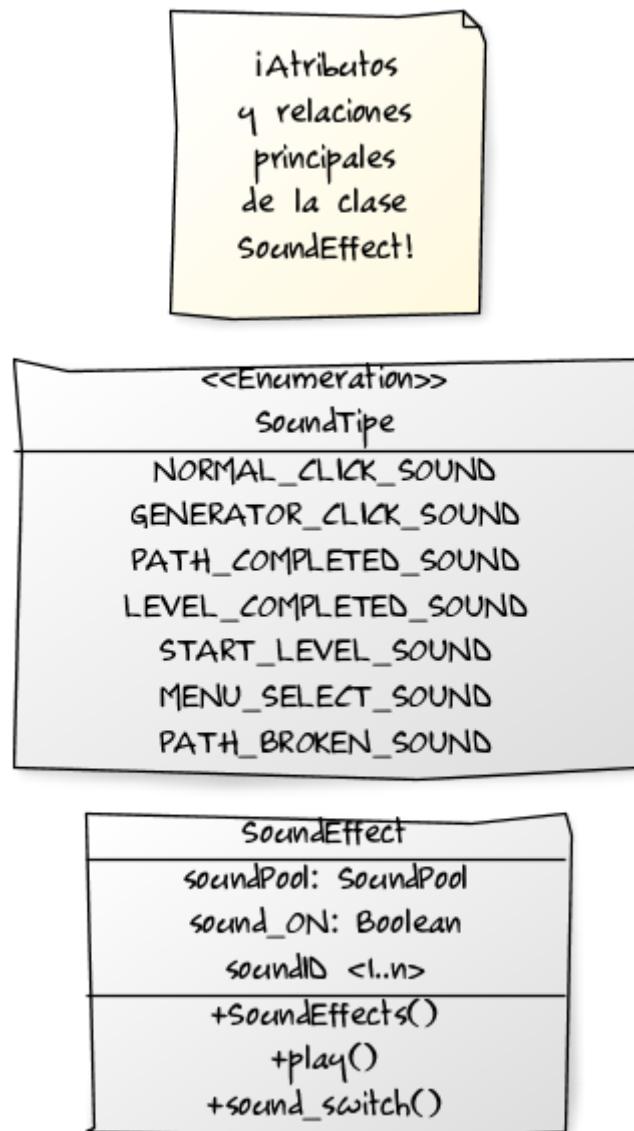


Figura C.10 Clase SoundEffect

Relaciones y atributos básicos

Los atributos principales son los siguientes:

- **soundPool:** Clase Android que nos permite reproducir sonidos ([3.7 - Sonido](#)).
- **sound_ON:** Nos indica si el sonido está activado o no. Dependiendo de esto se reproducirán o no las peticiones *play* que se reciban.
- **soundID:** Lista de identificadores de los diferentes archivos de sonido

Funciones y algoritmos principales

Las funciones principales para gestionar los efectos sonoros del juego son las siguientes.

- **SoundEffects:** En la creadora de esta clase se cargan en memoria los archivos de sonido como se explica en el apartado [3.7 - Sonido](#).
- **play:** Esta es la función principal de la clase, que nos permite reproducir un efecto sonoro concreto como muestra el [Código 3.7](#).
- **sound_switch:** Con esta función activamos o desactivamos los efectos sonoros en el juego.

C.11 Menu

La gestión de los diferentes apartados del menú del juego se realiza en esta clase. En el diagrama ([Figura C.11](#)) podemos ver una lista de los diferentes apartados del menú (MenuState) y una lista de las opciones (Option) que se pueden elegir. Como vemos, desde el menú (de configuración) se puede activar/desactivar el sonido, y reproducir diferentes efectos dependiendo de si cambiamos de menú o iniciamos un menú del juego.

Es también en esta clase donde se elige qué textura del menú hay que renderizar, y se aplica el algoritmo de selección de menús para comprobar qué opción se ha seleccionado. Para esto último, se comprueba si cierta zona de la pantalla ha sido pulsada, usando los mismos porcentajes de pantalla que se usan para renderizar las texturas.

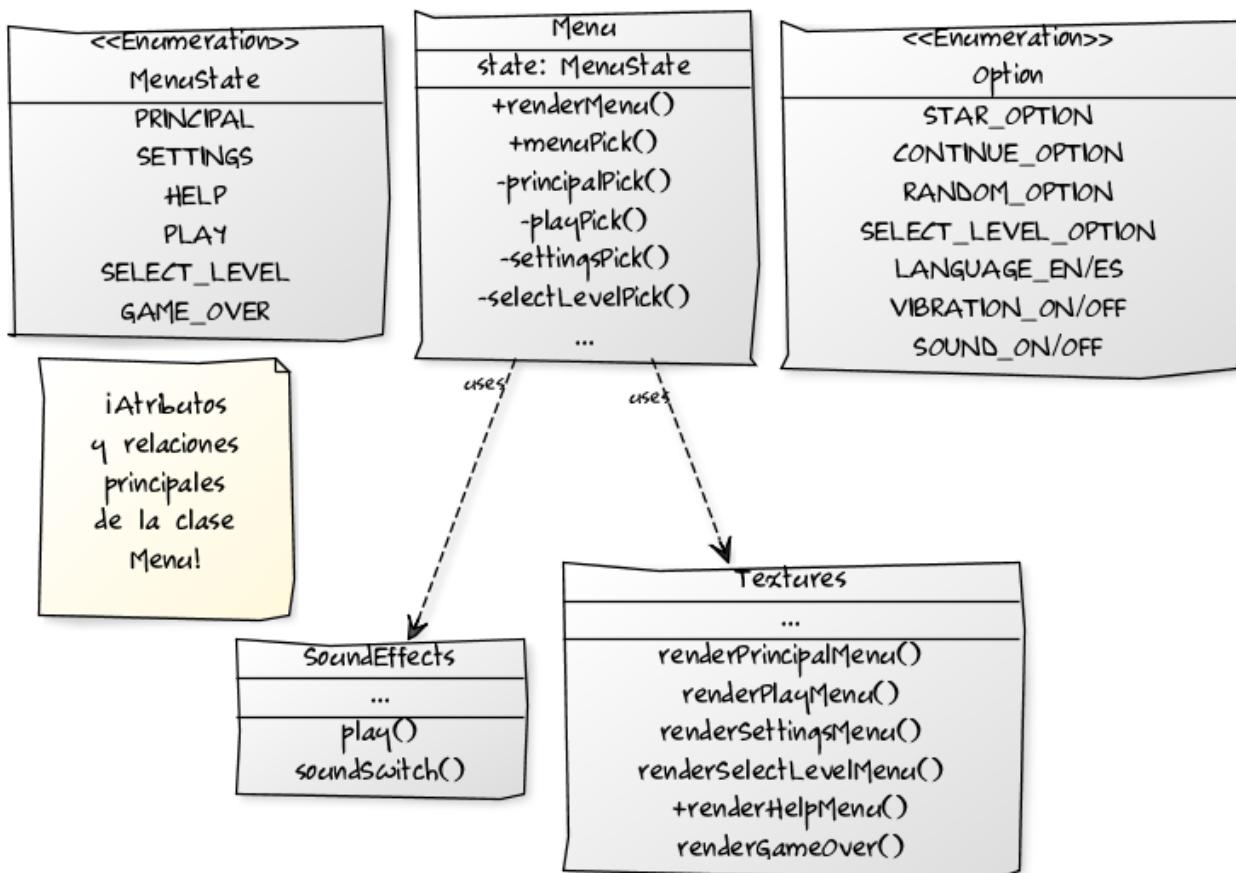


Figura C.11 Clase Menu

Relaciones y atributos básicos

Estos son los atributos y relaciones principales de esta clase.

- **state:** Indica el apartado del menú en el que nos encontramos
- **Textures:** Desde esta clase se decide qué textura de menú se ha de mostrar.
- **SoundEffects:** Con el menú de configuración se puede activar/desactivar el sonido. Y dependiendo de las opciones elegidas en el menú se reproducen unos sonidos u otros.

Funciones principales

En resumen, el renderizado y gestión de selección de opciones son las funciones principales.

- **renderMenu:** Es en esta función donde se decide qué menú se ha de mostrar dependiendo del estado en el que estemos.
- **X_Pick:** Funciones específicas de selección de opciones de los menús. Cada menú tiene una. En ellas se comprueba si se ha pulsado una zona de la pantalla que corresponde a un elemento del menú, basándose en las proporciones de la pantalla del móvil en cuestión y de los porcentajes y posiciones de pantalla en x e y asignados a cada elemento.

D. Creación de los elementos de la interfaz

El diseño gráfico de los diferentes elementos de la interfaz de usuario se ha realizado con el programa de retoque fotográfico GIMP⁴⁹. En este apartado se muestran brevemente las herramientas más utilizadas del programa. Y a continuación, las versiones finales y alternativas descartadas de los diferentes elementos de la interfaz. Junto con el tutorial seguido para su creación, o los pasos básicos a seguir en GIMP.

D.1 Técnicas básicas en GIMP

A continuación de muestra la lista de las principales herramientas utilizadas en el programa para crear los diferentes elementos de la interfaz gráfica:

- Selección: Nos permite indicar la forma/s que se verán afectadas por las modificaciones que realicemos con el resto de herramientas. Es bajo mi punto de vista la herramienta más importante. Nos permite, entre otras cosas:
 - Seleccionar todos los elementos de la capa activa
 - Deshacer la selección
 - Invertir la selección
 - Difuminar, para evitar bordes dentados
 - Encoger/agrandar
 - Pasar a seleccionar únicamente el borde de la forma seleccionada
 - Redondear los bordes del rectángulo de selección básica.
- Capas: Nos permiten tratar por separado diferentes partes de la imagen. Es una herramienta imprescindible para modificar los diferentes elementos, de forma independiente y sin alterar el resto.
 - Capa nueva
 - Borrar capa
 - Duplicar capa
 - Alfa a selección (pasan a estar seleccionados todos los elementos no transparentes de la capa)

⁴⁹ GNU Image Manipulation Program. Web GIMP: <http://www.gimp.org/>

- Alineación: Nos permite posicionar las diferentes partes. Centrarlas, colocarlas al principio o fin, basándonos en la imagen global o la posición de una capa concreta.
- Filtros:
 - Desenfoque>blur: Para suavizar los bordes de algunos elementos.
 - Renderizado>Naturaleza>Llama: Nos permite crear formas aleatorias modificando ciertos parámetros, útil para crear fondos de pantalla abstractos.
 - Renderizado>Explorador de fractales: A partir de imágenes, de por si abstractas, de una lista que ofrece el programa, nos permite aplicar transformaciones con el fin de generar otras imágenes abstractas personalizadas.
- Luces y sombra: Para crear efecto dimensional al arrojar una sombra de un objeto concreto.
- Herramientas de degradado: Tanto en blanco y negro (usado en las casillas), como de color (usados para realzar algunos fondos de pantalla).
- Otras herramientas básicas comunes a todos los programas de edición de imagen, como pueden ser borrar, capturar color, redimensionar, girar, pintar zona, añadir texto, etc.

D.2 Logo

El logo se creó partiendo de un video tutorial de internet [26], que ofrece un aspecto de letras desenfadado con efecto de agua. Y se le añadieron los colores primarios que representan el juego.

Inicialmente creé un gran número de versiones, cambiando el estilo, tonalidad, sombra, etc, como muestra la *Figura D.1*. Y tras las votaciones de mis compañeros (*6.9 - Testeando la interfaz de usuario*) me quedé finalmente con 5 de ellos (marcados en la imagen), que van cambiando a medida que aumenta la dificultad en el juego.

D.3 Indicador de nivel

Para el indicador de nivel seguí el mismo proceso que para el logo. Creé diferentes tipos de contenedores y números. Al principio basándome en el tipo de letra del logo, pero descartándolo tras las votaciones de mis compañeros (*6.9 - Testeando la interfaz de usuario*).

En la *Figura D.2 Elección del indicador de nivel* se muestran todas las alternativas propuestas, y aparece marcada la que finalmente usé. El único cambio posterior consistió en aplicarle diferentes colores, que cambian a cada nuevo nivel del juego en ciclos de 10.

Básicamente se trata de una circunferencia creada con el elemento de selección, un borde un poco más claro, un degradado simple, el número con sombra, y el suavizado de bordes.

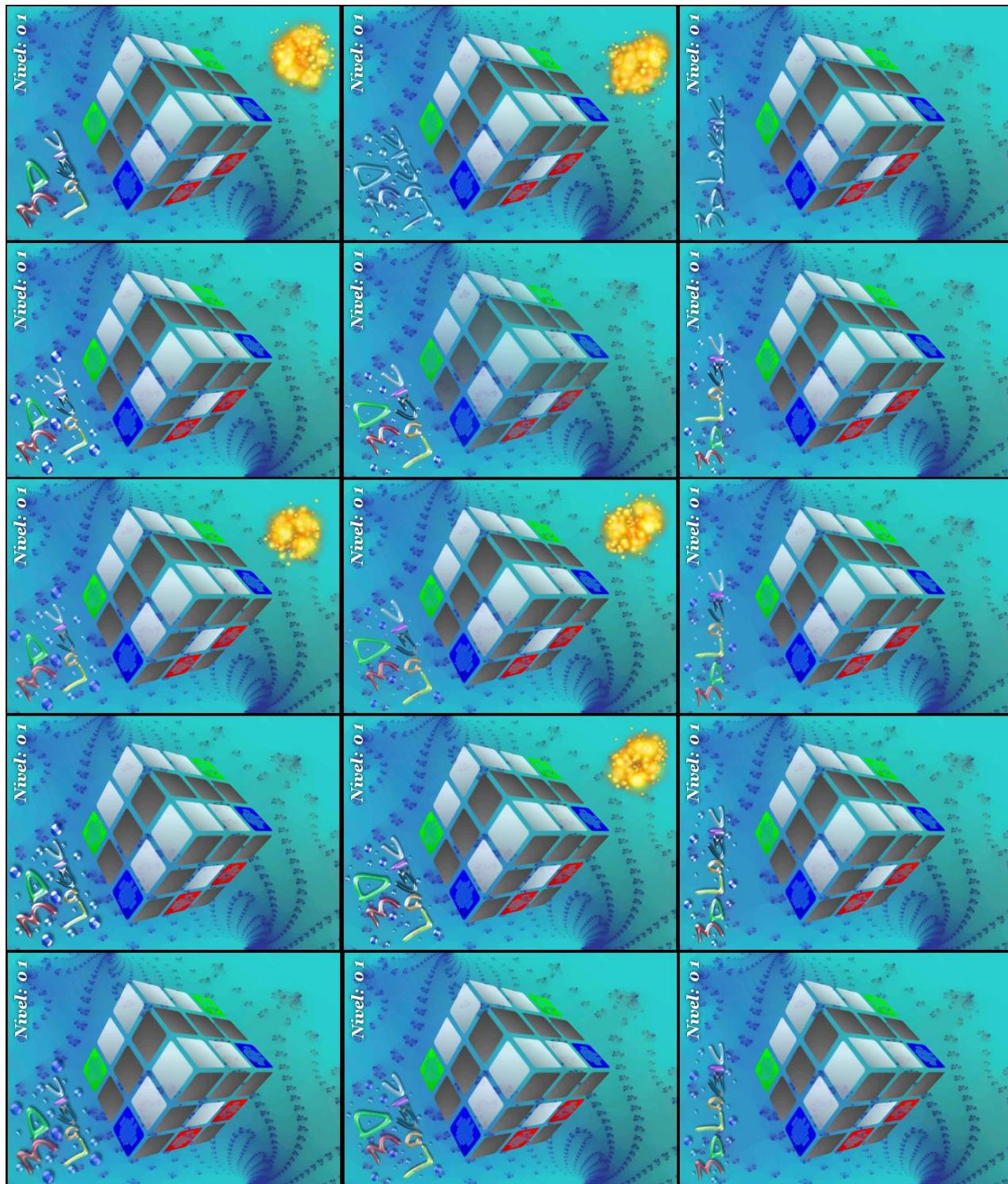


Figura D.1 Elección de logotipo

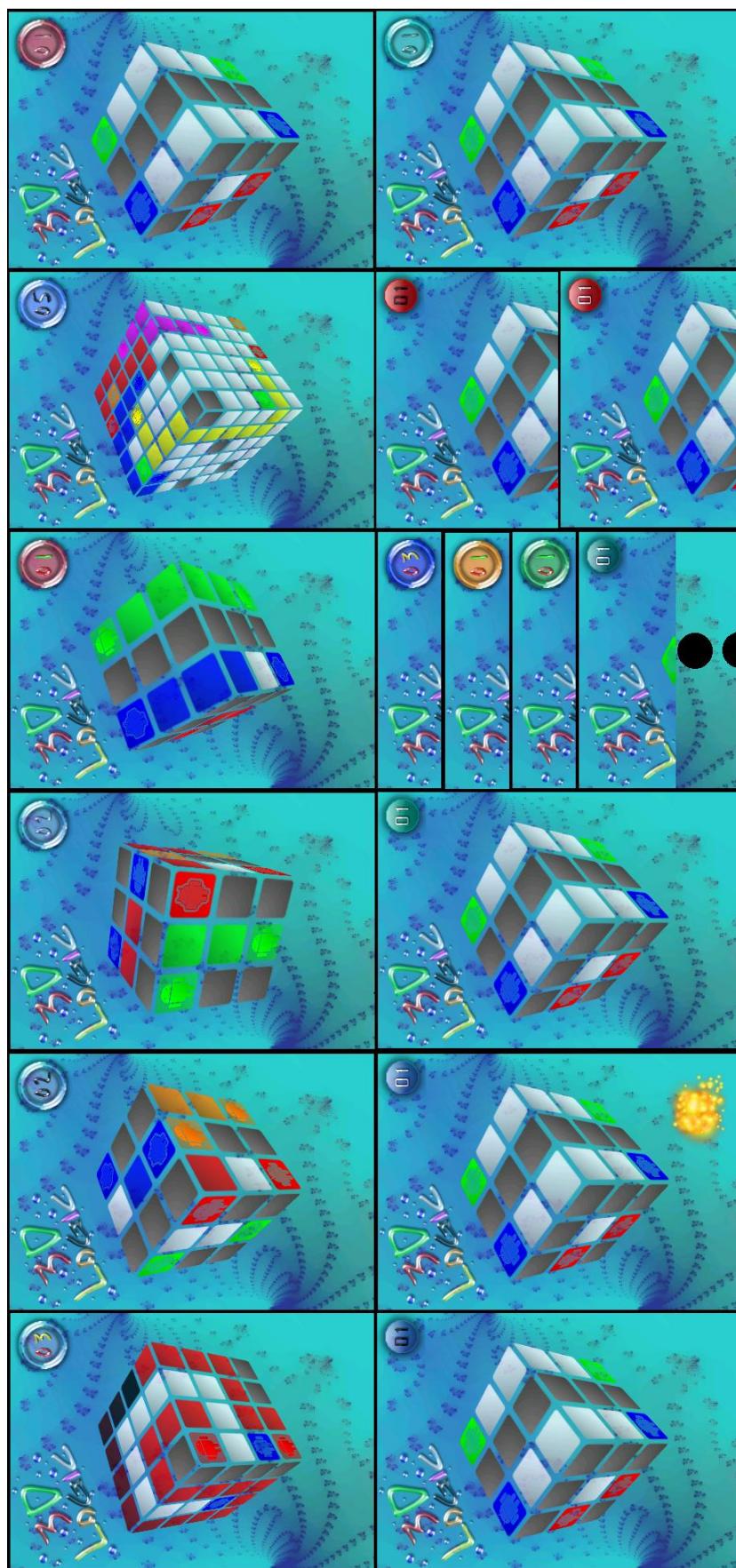


Figura D.2 Elección del indicador de nivel

D.4 Fondos

Los fondos se crearon mediante las herramientas de renderizado descritas en el apartado [D.1 - Técnicas básicas en GIMP](#) y retocando la intensidad de los colores y sombras. Inicialmente se crearon los fondos mostrados en la [Figura D.3](#). Y también por votación se eligieron los definitivos.

Siguiendo las recomendaciones, opté por fondos con colores suaves, para que el cubo, que es el elemento principal del juego, destaque por encima de cualquier fondo. Ya que un fondo demasiado llamativo interfiere con los colores del cubo y puede llegar a ser molesto a la vista y entorpecer el visionado del cubo en sí.

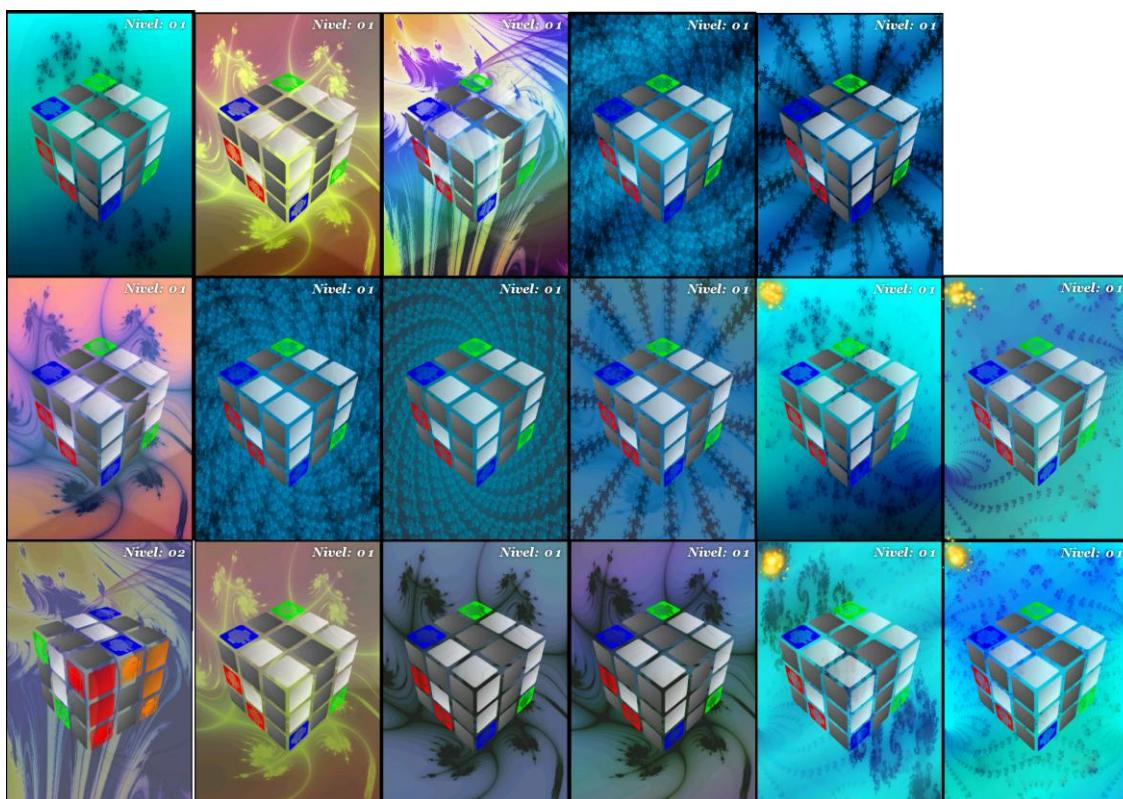


Figura D.3 Elección de fondos de pantalla

D.5 Casillas

Las casillas fueron lo primero que diseñé para poder mostrar el cubo adecuadamente.

El proceso fue el siguiente para cada casilla:

- Crear un cuadrado
- Reducir la selección para dejar un borde transparente
- Redondear y suavizar los bordes de la selección
- Añadir un androide en medio, de diferentes formas para representar un estado u otro del generador (sólo en caso de los generadores)
- Y finalmente aplicar un degradado hacia un lado u otro dependiendo de si es para una casilla normal, boqueada o generadora.

El resultado se muestra en la *Figura D.4* (con fondo azul para que se vea mejor). Como vemos, la textura está en tonos grises, ya que es mediante OpenGL que se mezclan con un color determinado.



Figura D.4 Diseño de las casillas

D.6 Menús

Para la creación de los diferentes botones de los menús seguí un tutorial online [25], aplicando ligeras modificaciones al resultado según el apartado del menú.

Básicamente son rectángulos, con un pequeño borde más claro, y dos degradados; uno blanco de arriba al medio; y otro negro de abajo al medio, que son los que le dan efecto de volumen.

Para ofrecer diferentes idiomas, simplemente si traduce el texto y se coloca el elemento repetido a su derecha, facilitando así la elección del idioma al renderizar la textura.

Para la pantalla final del juego se usó el mismo estilo que para el logo, y se añadió un texto con degradado de color para indicar la activación del modo espejo.

A continuación se muestran las texturas usadas para mostrar los diferentes apartados de los menús (algunas con fondo azul para destacar ciertos efectos):

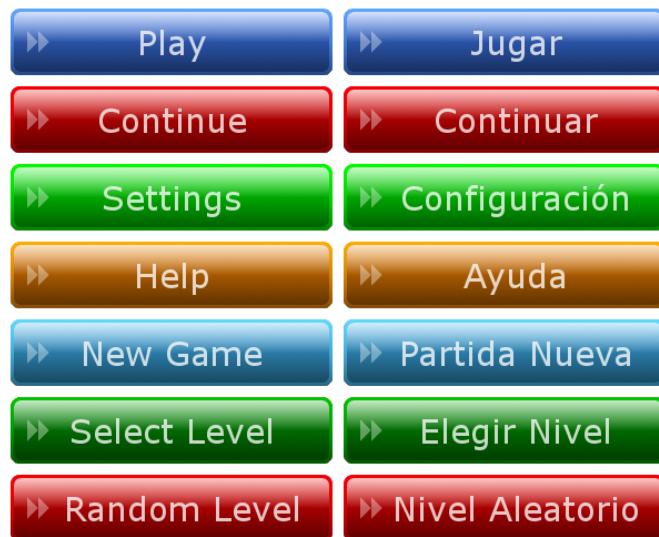


Figura D.5 Botones de los menús



Figura D.6 Títulos y elementos del menú de configuración



Figura D.7 Botones de navegabilidad por los menús e indicador de nivel

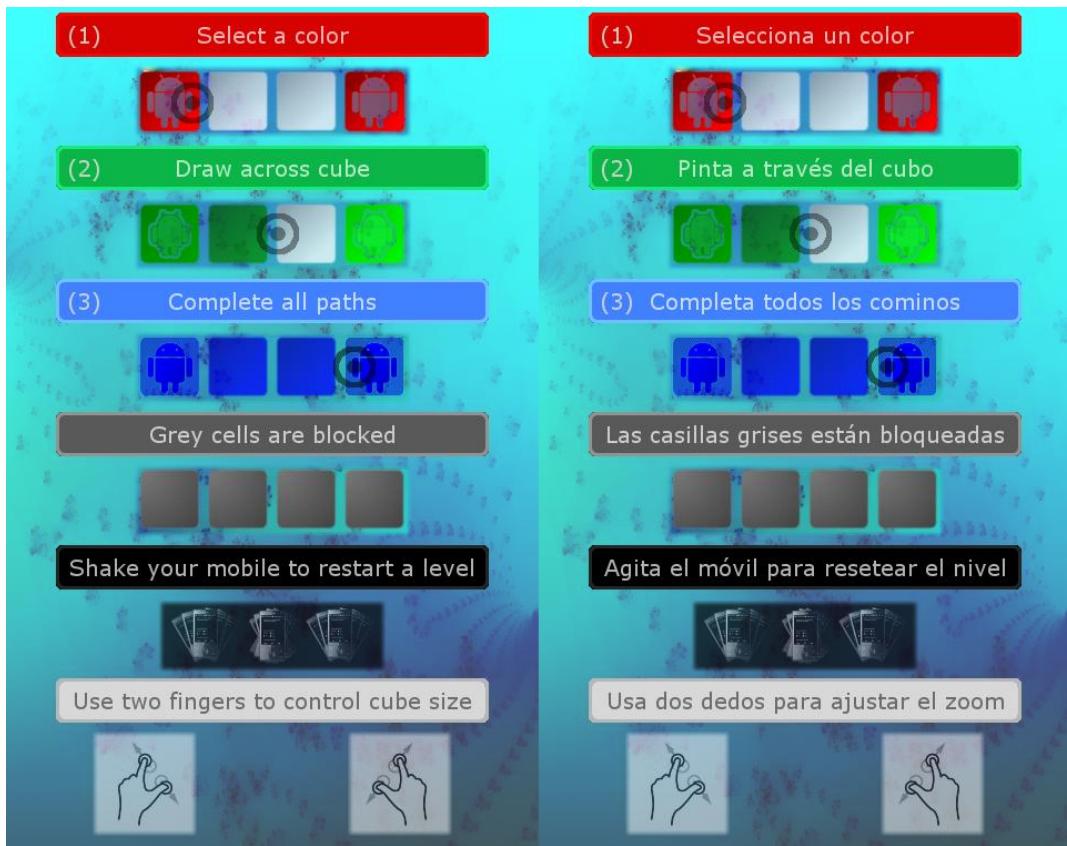


Figura D.8 Pantalla de ayuda en los dos idiomas

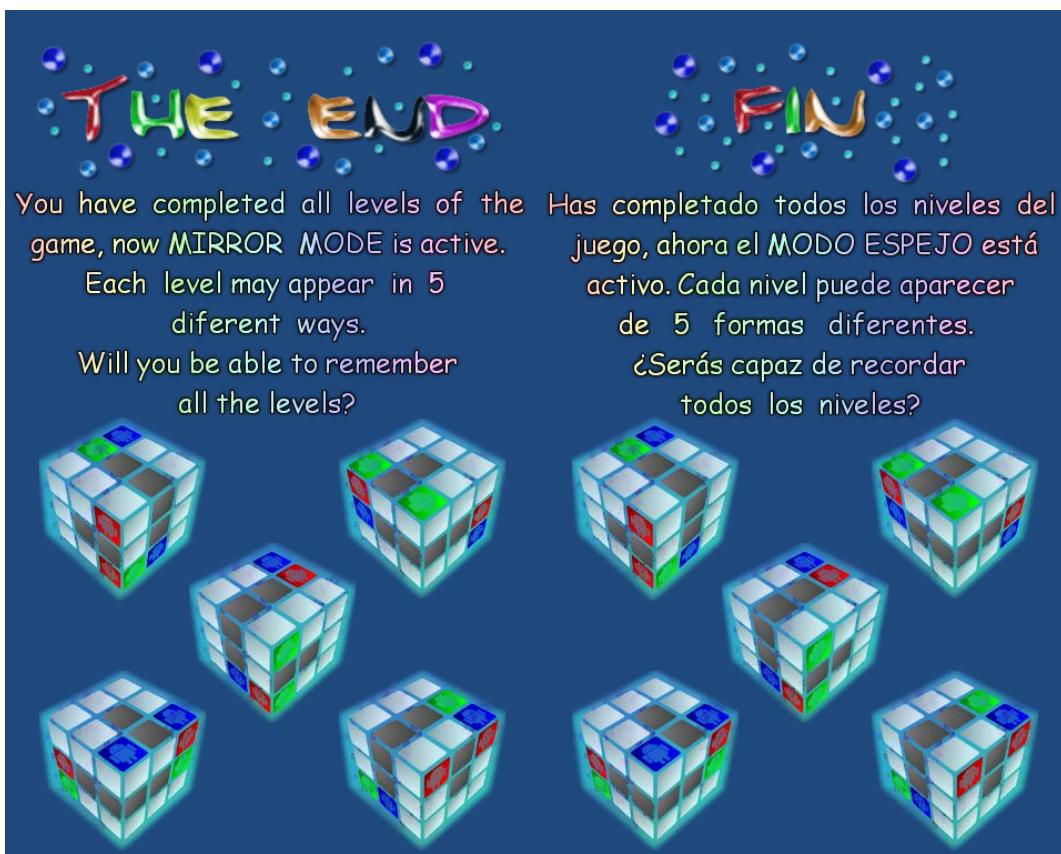


Figura D.9 Pantalla de fin de juego en los dos idiomas

E. Diagramas en formato texto

A modo de ejemplo y referencia, en este apartado se pueden encontrar las órdenes necesarias para generar los diagramas de clases y de secuencia mediante las aplicaciones online yUML y WebSequenceDiagrams, ambas descritas en el apartado [5.4.5 - Diagramas](#). Si copiamos y pegamos las órdenes en la aplicación correspondiente, obtendremos el diagrama original.

E.1 yUML⁵⁰

A continuación encontramos los diagramas de clases del proyecto. Usar la dirección a pie de página para probarlos.

Diagrama Completo yUML

```
[note: ¡Diagrama de clases simplificado!{bg:cornsilk}]
[A_3D_Logic]->1[GameRenderer]
[GameRenderer]->1[GLWorld]
[GameRenderer]->1[DataBase]
[GameRenderer]uses ..->[Textures]
[GameRenderer]uses ..->[Menu]
[Menu]uses-->[Textures]
[Menu]uses-->[SoundEffects]
[GLWorld]*<>.*>[GLShape]
[GLShape]*++-*>[GLFace]
[Cube]*++-*>[Square]
[GLFace]*++-*>[GLVertex]
[GLFace]^-[Square]
[GLFace]*->1[GLColor]
[GLWorld]<>-1>[Cube]
[GLShape]~-[Cube]
[GLWorld]->1[Game]
[Game]->*[Level]
[Cube]->1[Animations]
[Cube]uses ..->[SoundEffects]
[Cube]actualLevel->1[Level]
```

A_3D_Logic yUML

```
[note: ¡Atributos y relaciones principales de la clase A_3D_Logic!{bg:cornsilk}]
[A_3D_Logic|mView: GLSurfaceView;mSensorManager: SensorManager;wakeLock:
WakeLock;...|#onCreate(); #onResume();#onStop();#onPause();#onDestroy();-onTrackballEvent();-
onTouchEvent()](GameRenderer|...|+getAngleX();+getAngleY();+setAngleX();+setAngleY();+zoomFactor();+re
setLevel();+picking();+close())
[SensorEventListener| ... |+onSenserChanged()]
[Compatibility|state: multiTouchState; ... |+setZOrderOnTop();+multitouch()]
[A_3D_Logic]->1[GameRenderer]
[A_3D_Logic]->1[SensorEventListener]
[A_3D_Logic]->1[Compatibility]
```

⁵⁰ Página para probar los diagramas de clases: <http://yuml.me/diagram/scruffy/class/draw2>

Animations yUML

```
[note: ¡Atributos y relaciones principales de la clase Animations!{bg:cornsilk}]
[Animations|...|+animation(); +addDarkenAnimation();+addAppearAnimation(); +addDisappearAnimation();
+endAnimation(); +isSquareAnimated(); -processColorEffect(); -updateBufferColor()]

[ColorEffect|square; totalSteps; step| +get/set-attr()]
[Animations]Darken->*[ColorEffect]
[Animations]Appear->*[ColorEffect]
[Animations]Disappear->*[ColorEffect]
[GLColor|r - g - b - a|darkenColor(); transColor();]
[Animations]uses ->[GLColor]
[ColorEffect]->1[GLColor]
```

Cube yUML

```
[note: ¡Atributos y relaciones principales de la clase Cube!{bg:cornsilk}]
[<<<Enumeration>>>;CubeState | PLAYING; FADE_AWAY; STARTING; COMPLETED;]

[Cube|state: CubeState;pathCompleted <1..n>; vib: Vibrator; | -addSquare(); +startLevel();+processPick(); -
nextLevel(); -checkPathCompleted(); -checkPathCompletedRecursive(); -updateGenerator(); -
+processAnimations(); +generateSquares(); +inicialize(); ...]

[Square|...|setTipe(); setGenMate(); getGenMate(); addNear();]
[Cube]++-*>[Square]
[Animations|...|animation(); addDarkenAnimation(); addAppearAnimation(); addDisappearAnimation();
endAnimation(); isSquareAnimated(); isRdyToChangeLevel(); isRdyToPlay()]
[Cube]-1>[Animations]
[Level|..|getGenerator(); getGenIndex(); getStart(); getEnd();getColor(); getBarreeds(); getNGenerators()]
[Cube]actualLevel-1>[Level]
[Cube]uses-->[SoundEffect|...|play()]
[Cube]uses-->[GLColor|...|normaToTrans(); transToNormal()]
```

Game yUML

```
[note: ¡Atributos y relaciones principales de la clase Game!{bg:cornsilk}]
[Game|actualLevel; mirrorMode;...|+Game(); +getLevel(); +getNextLevel(); +mirrorLevel(); -mirrorPosition();
-newPosition(); +isGameFinished();...]
[Level|..|addGenerator(); addBarreeds(); getStart(); getEnd();getColor(); getBarreeds(); getSize();
getNGenerators()]
[Game]<-*>[Level]
```

GameRenderer yUML

```
[note: ¡Atributos y relaciones principales de la clase GameRenderer!{bg:cornsilk}]
[\"<\<Enumeration\>>;GameState | MENU; PLAYING; PICKING]
[GameRenderer|state: GameState;... | -setFrustum();+onDrawFrame();+picking();...]
[DataBase|... | +get/update-ActualLevel();+get/update-Language();+get/update-Vibration();+get/update-Sound();+get/update-Completed()]
[GameRenderer]-1>[DataBase]
[Menu|... | +renderMenu();+menuPick();+isPrincipal();+levelSelected();+back();+gameOver();...]
[GameRenderer]uses-->[Menu]
[Textures|... | +renderBackground();+renderLeveTag();+loadSquareTextures();...]
[GameRenderer]uses-->[Textures]
[GLWorld|... | +draw();+pickDraw();+startLevel();+resetLevel();+pick();+getLevel();+getNLevels();+isGameFinis
hed()]
[GameRenderer]-1>[GLWorld]
```

GLWorld yUML

```
[note: ¡Atributos y relaciones principales de la clase GLWorld!{bg:cornsilk}]
[GLWorld|mVertexBuffer; mColorBuffer; mTexBuffer; mPickColorBuffer;mIndexBuffer|-ini(); +draw();
+pickDraw(); +addShape(); +addVertex(); -generate(); -genPickRef(); ...]
[GLShape|... | putIndices()]
[GLWorld]<>-*>[GLShape]
[Cube|... | generateSquares(); initialize(); getState(); processAnimations(); processPick(); startLevel()]
[GLWorld]->-1>[Cube]
[GLShape]^-[Cube]
[Game|... | getLevel(); getNextLevel(); getIndexActualLevel(); isGameFinished()]
[GLWorld]-1>[Game]
[GLVertex|x - y - z;|putVertexes()]
[GLWorld]<>-*>[GLVertex]
```

Level yUML

```
[note: ¡Atributos y relaciones principales de la clase Level!{bg:cornsilk}]
[Level|size; barreds <0..n>|+get/set-attr()]
[Generator|start;end|+get-attr()]
[Level]<>->1..*[Generator]
[Generator]->1[GLColor]
```

Menu yUML

```
# Cool Class Diagram
[note: ¡Atributos y relaciones principales de la clase Menu!{bg:cornsilk}]
[\"<\<Enumeration\>>;Option | STAR_OPTION; CONTINUE_OPTION; RANDOM_OPTION;
SELECT_LEVEL_OPTION; LANGUAGE_EN/ES; VIBRATION_ON/OFF; SOUND_ON/OFF]
[\"<\<Enumeration\>>;MenuState | PRINCIPAL; SETTINGS; HELP; PLAY; SELECT_LEVEL; GAME_OVER]
[Menu|state: MenuState|+renderMenu();+menuPick();-principalPick();-playPick();-settingsPick();-
selectLevelPick();...]
[Textures|... | renderPrincipalMenu();renderPlayMenu();renderSettingsMenu();renderSelectLevelMenu();+re
nderHelpMenu();renderGameOver();...]
[Menu]uses-->[Textures]
[Menu]uses-->[SoundEffects| ... | play();soundSwitch()]
```

SoundEffect yUML

```
[note: ¡Atributos y relaciones principales de la clase SoundEffect!{bg:cornsilk}]
[\\<\\<Enumeration\\>\\>;SoundType | NORMAL_CLICK_SOUND; GENERATOR_CLICK_SOUND;
PATH_COMPLETED_SOUND; LEVEL_COMPLETED_SOUND; START_LEVEL_SOUND; MENU_SELECT_SOUND;
PATH_BROKEN_SOUND;]
[SoundEffect|soundPool: SoundPool;sound_ON: Boolean;soundID \\<1..n\\>;+SoundEffects(); +play();
+sound_switch()]
```

Square yUML

```
[note: ¡Atributos y relaciones principales de la clase Square!{bg:cornsilk}]
[\\<\\<Enumeration\\>\\>;SquareType | Normal; GeneratorON; GeneratorOFF; GeneratorOK; Barred;]
[Square|tipe: SquareType;genMate;near \\<2..4\\>;+getTipe();+getGenMate();...]
[GLFace|...|addVertex();getColor()]-[Square]
```

Textures yUML

```
[note: ¡Atributos principales de la clase Textures!{bg:cornsilk}]
[Textures|mTextureID \\<1..n\\>; SCREEN_WIDTH; SCREEN_HEIGHT;...|+Textures(); setTextureAttributes();
+loadTexture(); +loadSquareTextures(); +render-X()]
```

E.2 Web Sequence Diagrams⁵¹

A continuación encontramos los diagramas de secuencia del proyecto. Usar la dirección a pie de página para probarlos.

Diagrama de secuencia (Agitar el móvil)

```
User->Screen (UI): Shake the phone
activate Screen (UI)
Screen (UI)-->A_3D_Logic: onSensorChanged()
deactivate Screen (UI)
activate A_3D_Logic
A_3D_Logic->A_3D_Logic:shake?() == True
Clock->Clock:
A_3D_Logic->GameRenderer: resetLevel()
activate GameRenderer
GameRenderer->GameRenderer:resetLevel= 'ON'

GameRenderer-->A_3D_Logic:
deactivate GameRenderer
deactivate A_3D_Logic

Clock-->GameRenderer: drawFrame(gl)
activate GameRenderer
GameRenderer->Screen (UI): clearGL()
activate Screen (UI)
GameRenderer->Textures: renderBackground(gl)
activate Textures
Textures-->Screen (UI):background
deactivate Textures
GameRenderer->Textures: renderLevelLabel(gl, level)
activate Textures
Textures-->Screen (UI):levelLabel
deactivate Textures
GameRenderer->Textures: renderHomeButton(gl)
activate Textures
Textures-->Screen (UI):homeButton
deactivate Textures
GameRenderer->Textures: loadSquareTextures(gl)
activate Textures
Textures-->GameRenderer:
deactivate Textures
GameRenderer->Screen (UI): resetModelview()
GameRenderer->GLWorld:resetLevel(gl)
activate GLWorld

GLWorld->Cube: initializeLevel()
activate Cube
Cube->Animations: removeAll()
activate Animations
Cube->Animations: addAppearAnimation()
note left of Animations: for each square
Animations-->Cube:
deactivate Animations
Cube->SoundEffects: play("start_level")
activate SoundEffects
```

⁵¹ Pagina para probar los diagramas de secuencia: <http://www.websequencediagrams.com/>

```

SoundEffects-->Cube:
deactivate SoundEffects
Cube-->GLWorld:
deactivate Cube
GLWorld->GLWorld: generateBuffers()
GLWorld-->GameRenderer:
GameRenderer->GLWorld:draw(gl)
note right of GLWorld: Cube animations
GLWorld-->Screen (UI): 3D Cube Scene
deactivate GLWorld
deactivate GameRenderer

deactivate Screen (UI)
Screen (UI)->Screen (UI): show()
activate Screen (UI)

```

Diagrama de secuencia (Cambiar configuración)

```

User->Screen (UI): Touch country flag
activate Screen (UI)
Screen (UI)-->A_3D_Logic: onTouchEvent()
deactivate Screen (UI)
activate A_3D_Logic
Clock->Clock:
A_3D_Logic->GameRenderer: pick(x, y)
activate GameRenderer
GameRenderer->Menu: menuPick(x, y)
activate Menu
Menu->Menu:settings!pick(x, y)
Menu->SoundEffects: play("bip")
activate SoundEffects
SoundEffects-->Menu:
deactivate SoundEffects
Menu-->GameRenderer: switchLanguageOption
deactivate Menu
GameRenderer->DateBase:updateLanguage(lang)
activate DateBase
DateBase-->GameRenderer:
deactivate DateBase
GameRenderer-->A_3D_Logic:
deactivate GameRenderer
deactivate A_3D_Logic
Clock-->GameRenderer: drawFrame(gl)
activate GameRenderer
GameRenderer->Screen (UI): clearGL()
activate Screen (UI)
GameRenderer->Textures: renderBackground()
activate Textures
Textures-->Screen (UI):background
deactivate Textures
GameRenderer->Textures: renderBackButton()
activate Textures
Textures-->Screen (UI):backButton
deactivate Textures
GameRenderer->Menu: renderMenu()
activate Menu
Menu->Textures: renderSettingsMenu()activate Textures
Textures-->Screen (UI): settingsMenu

```

```

deactivate Textures
deactivate Menu
deactivate GameRenderer
deactivate Screen (UI)
Screen (UI)->Screen (UI): show()
activate Screen (UI)

```

Diagrama de secuencia (Cambiar de color)

```

GLWorld->Cube: processPick(square)
activate GLWorld
activate Cube
Cube->Cube: checkPathCompleted() == false
Cube->Animations: addDarkAnim(newGen)
activate Animations
Animations-->Cube:
deactivate Animations
Cube->Square: setTipe(newGen, "ON")
activate Square
Cube->Square: setTipe(odlGen, "OFF")
Square-->Cube:
deactivate Square
Cube->SoundEffects: play("generatorClick")
activate SoundEffects
SoundEffects-->Cube:
deactivate SoundEffects
Cube-->GLWorld:
deactivate Cube
deactivate GLWorld

```

Diagrama de secuencia (Camino completado)

```

GLWorld->Cube: processPick(square)
activate GLWorld
activate Cube
Cube->Cube: checkPathCompleted() == True
Cube->Cube: levelCompleted?() == False
loop for every square in the path
Cube->Animations: addDarkAnim(square)
activate Animations
Animations-->Cube:
end
deactivate Animations
Cube->Square: setTipe(gen, "COMPLETED")
activate Square
Square-->Cube:
deactivate Square
Cube->SoundEffects: play("pathCompleted")activate SoundEffects

SoundEffects-->Cube:
deactivate SoundEffects
Cube-->GLWorld:
deactivate Cube
deactivate GLWorld

```

Diagrama de secuencia (Completar nivel)

```

GLWorld->Cube: processPick(square)
activate GLWorld
activate Cube
Cube->Cube: checkPathCompleted() == True
Cube->Cube: levelCompleted?() == True
loop for every square in the cube
Cube->Animations: addDisappearAnim(square)
activate Animations
Animations-->Cube:
end
deactivate Animations

Cube->SoundEffects: play("levelCompleted")
activate SoundEffects
SoundEffects-->Cube:
deactivate SoundEffects
Cube->Cube: state = COMPLETED
Cube-->GLWorld: nextLevel
deactivate Cube
deactivate GLWorld

```

Diagrama de secuencia (Fin del juego)

```

Clock-->GameRenderer: drawFrame(gl)
activate GameRenderer
GameRenderer->GLWorld: nextLevel }()
activate GLWorld
GLWorld-->GameRenderer: False
GameRenderer->GLWorld: gameFinished(){}
GLWorld-->GameRenderer: True
deactivate GLWorld

GameRenderer->Menu: gameOver(firstTime?)
activate Menu
Menu-->GameRenderer:
deactivate Menu
GameRenderer->DataBase: updateCompleted(Ture)
activate DataBase
DataBase-->GameRenderer:
deactivate DataBase

GameRenderer->Screen (UI): clearGL()
activate Screen (UI)

GameRenderer->Textures: renderBackground()
activate Textures
Textures-->Screen (UI):background
deactivate Textures
GameRenderer->Textures: renderBackButton()
activate Textures
Textures-->Screen (UI):backButton
deactivate Textures
GameRenderer->Menu: renderMenu()
activate Menu
Menu->Textures: renderGameOver(firstTime?)

```

```

activate Textures
alt firstTime
Textures-->Screen (UI): The_End_MirrorModeON
else not firstTime
Textures-->Screen (UI): The_End
end
deactivate Textures
deactivate Menu
deactivate GameRenderer
deactivate Screen (UI)
Screen (UI)->Screen (UI): show()
activate Screen (UI)

```

Diagrama de secuencia (Mover Cubo)

```

User->Screen (UI): Drag Finger
activate Screen (UI)
Screen (UI)-->A_3D_Logic: onTouchEvent()
deactivate Screen (UI)
activate A_3D_Logic
Clock->Clock:
A_3D_Logic->GameRenderer: setAngles(angleX, angleY)
activate GameRenderer
GameRenderer-->A_3D_Logic:
deactivate GameRenderer
deactivate A_3D_Logic

Clock->GameRenderer: drawFrame(gl)
activate GameRenderer
GameRenderer->Screen (UI): clearGL()
activate Screen (UI)

GameRenderer->Screen (UI):updateModelview()

GameRenderer->Textures: renderBackground(gl)
activate Textures
Textures-->Screen (UI):background
deactivate Textures
GameRenderer->Textures: renderLevelLabel(gl, level)
activate Textures

deactivate Textures
GameRenderer->Textures: renderHomeButton(gl)
activate Textures
Textures-->Screen (UI):homeButton

deactivate Textures
GameRenderer->Textures: loadSquareTextures(gl)
activate Textures
Textures-->GameRenderer:
deactivate Textures

GameRenderer->GLWorld:draw(gl)
activate GLWorld
note right of GLWorld: Cube animations
GLWorld-->Screen (UI): 3D Cube Scene
deactivate GLWorld
deactivate GameRenderer

```

```

deactivate Screen (UI)
Screen (UI)->Screen (UI): show()
activate Screen (UI)

```

Diagrama de secuencia (Partida Nueva)

```

User->Screen (UI): New Game
activate Screen (UI)
Screen (UI)-->A_3D_Logic: onTouchEvent()
deactivate Screen (UI)
activate A_3D_Logic
Clock->Clock:
A_3D_Logic->GameRenderer: pick(x, y)
activate GameRenderer
GameRenderer->Menu: menuPick(x, y)
activate Menu
Menu->Menu: playPick(x, y)
Menu->SoundEffects: play("start_level")
activate SoundEffects
SoundEffects-->Menu:
deactivate SoundEffects
Menu-->GameRenderer: newGameOption
deactivate Menu
GameRenderer->GLWorld: starLevel(levelNumber)
activate GLWorld

GLWorld->Game: getLevel(levelNumber)
activate Game
Game-->GLWorld: level
deactivate Game
GLWorld->Cube: startLevel(level)
activate Cube
Cube->Cube: reset()
activate Cube
GLWorld->GLWorld: initialize()
activate GLWorld
GLWorld->GLWorld: clearBuffers()
GLWorld->Cube: generateSquares()
GLWorld->GLWorld: genPickColors()
GLWorld->Cube: initializeLevel()
Cube->Animations: addAppearAnimation()
activate Animations
note left of Animations: for each square
Animations-->Cube:
deactivate Cube
deactivate Animations
Cube-->GLWorld:
deactivate Cube
GLWorld->GLWorld: generateBuffers()
deactivate GLWorld
GLWorld-->GameRenderer:
deactivate GLWorld
GameRenderer-->A_3D_Logic:
deactivate GameRenderer
deactivate A_3D_Logic

```

loop until the appearance of the Cube finishes

```
Clock-->GameRenderer: drawFrame(gl)
activate GameRenderer
GameRenderer->Screen (UI): clearGL()
activate Screen (UI)
GameRenderer->Textures: renderBackground(gl)
activate Textures
Textures-->Screen (UI):background
deactivate Textures
GameRenderer->Textures: renderLevelLabel(gl, level)
activate Textures
Textures-->Screen (UI):levelLabel
deactivate Textures
GameRenderer->Textures: renderHomeButton(gl)
activate Textures
Textures-->Screen (UI):homeButton
deactivate Textures
GameRenderer->Textures: loadSquareTextures(gl)
activate Textures
Textures-->GameRenderer:
deactivate Textures

GameRenderer->GLWorld:draw(gl)
activate GLWorld
GLWorld->Cube: processAnimation(colorBuffer)
activate Cube
Cube->Animations: animation(colorBuffer)
note left of Animations: each frame
activate Animations
Animations-->Cube:
deactivate Animations
Cube-->GLWorld:
deactivate Cube
GLWorld-->Screen (UI): 3D Cube Scene
deactivate GLWorld
deactivate GameRenderer

deactivate Screen (UI)
Screen (UI)->Screen (UI): show()
activate Screen (UI)
```

Diagrama de secuencia (Pintar casilla)

```
User->Screen (UI): Touch Square
activate Screen (UI)
Screen (UI)-->A_3D_Logic: onTouchEvent()
deactivate Screen (UI)
activate A_3D_LogicClock->Clock:

A_3D_Logic->GameRenderer: pick(x, y)
activate GameRenderer
GameRenderer->GameRenderer: state=PICKING
GameRenderer-->A_3D_Logic:
deactivate GameRenderer
deactivate A_3D_Logic

Clock-->GameRenderer: drawFrame(gl)
activate GameRenderer
GameRenderer->Screen (UI): clearGL()
activate Screen (UI)

GameRenderer->GLWorld: pickDraw(x, y)
activate GLWorld
GLWorld-->Screen (UI): colored Cube
GLWorld->Screen (UI):readPixel(x, y)
Screen (UI)-->GLWorld: color
GLWorld->Cube: processPick(square)
activate Cube
Cube->Cube: checkPathCompleted() == false
Cube->Animations: addDarkAnim(square)
activate Animations
Animations-->Cube:
deactivate Animations
Cube->SoundEffects: play("bip")
activate SoundEffects
SoundEffects-->Cube:
deactivate SoundEffects
Cube-->GLWorld:
deactivate Cube
GLWorld-->GameRenderer:
deactivate GLWorld

GameRenderer->Textures: renderBackground(gl)
activate Textures
Textures-->Screen (UI):background
deactivate Textures
GameRenderer->Textures: renderLevelLabel(gl, level)
activate Textures
Textures-->Screen (UI):levelLabel
deactivate Textures
GameRenderer->Textures: renderHomeButton(gl)
activate Textures
Textures-->Screen (UI):homeButton
deactivate Textures
GameRenderer->Textures: loadSquareTextures(gl)
activate Textures
Textures-->GameRenderer:
deactivate Textures

GameRenderer->GLWorld:draw(gl)
activate GLWorld
GLWorld->Cube: processAnimation(colorBuffer)
```

```

activate Cube
Cube->Animations: animation(colorBuffer)
activate Animations
Animations-->Cube:
deactivate Animations
Cube-->GLWorld:
deactivate Cube
GLWorld-->Screen (UI): 3D Cube Scene

deactivate GLWorld
deactivate GameRenderer

deactivate Screen (UI)
Screen (UI)->Screen (UI): show()
activate Screen (UI)

```

Diagrama de secuencia (Play)

```

User->Screen (UI): Play Button
activate Screen (UI)
Screen (UI)-->A_3D_Logic: onTouchEvent()
deactivate Screen (UI)
activate A_3D_Logic
Clock->Clock:
A_3D_Logic->GameRenderer: pick(x, y)
activate GameRenderer
GameRenderer->Menu: menuPick(x, y)
activate Menu
Menu->Menu:principalPick(x, y)
Menu->SoundEffects: play("bip")
activate SoundEffects
SoundEffects-->Menu:
deactivate SoundEffects
Menu-->GameRenderer: playMenuOption
deactivate Menu
GameRenderer-->A_3D_Logic:
deactivate GameRenderer
deactivate A_3D_Logic
Clock-->GameRenderer: drawFrame(gl)
activate GameRenderer
GameRenderer->Screen (UI): clearGL()
activate Screen (UI)
GameRenderer->Textures: renderBackground()
activate Textures
Textures-->Screen (UI):background
deactivate Textures
GameRenderer->Textures: renderBackButton()
activate Textures
Textures-->Screen (UI):backButton
deactivate Textures
GameRenderer->Menu: renderMenu()
activate Menu
Menu->Textures: renderPlayMenu()
activate Textures
Textures-->Screen (UI): playMenu
deactivate Textures
deactivate Menu
deactivate GameRenderer

```

```

deactivate Screen (UI)
Screen (UI)->Screen (UI): show()
activate Screen (UI)

```

Diagrama de secuencia (Romper camino)

```

GLWorld->Cube: processPick(square)
activate GLWorld
activate Cube
Cube->Cube: checkPathCompleted() == False

Cube->Animations: addDarkAnim(square)
activate Animations
Animations-->Cube:

deactivate Animations
Cube->Square: setTipe(brokenGen, "OFF")
activate Square
Square-->Cube:
deactivate Square
Cube->SoundEffects: play("pathBroken")
activate SoundEffects
SoundEffects-->Cube:
deactivate SoundEffects
Cube->GLWorld:
deactivate Cube
deactivate GLWorld

```

Diagrama de secuencia (Siguiente nivel)

```

Clock-->GameRenderer: drawFrame(gl)
activate GameRenderer
GameRenderer->GLWorld: nextLevel }()
activate GLWorld
GLWorld-->GameRenderer: True
GameRenderer->GLWorld: gameFinished(){}
GLWorld-->GameRenderer: False
deactivate GLWorld
GameRenderer->DataBase: updateLevel(level)
activate DataBase
DataBase-->GameRenderer:
deactivate DataBase

GameRenderer->Screen (UI): clearGL()
activate Screen (UI)

GameRenderer->Textures: renderBackground(gl)
activate Textures
Textures-->Screen (UI):background
deactivate Textures
GameRenderer->Textures: renderLevelLabel(gl, level)
activate Textures
Textures-->Screen (UI):levelLabel
deactivate Textures

```

```
GameRenderer->Textures: renderHomeButton(gl)
activate Textures
Textures-->Screen (UI):homeButton
deactivate Textures
GameRenderer->Textures: loadSquareTextures(gl)
activate Textures

Textures-->GameRenderer:
deactivate Textures

GameRenderer->GLWorld:starLevel(nextLevel)
activate GLWorld
note right of GLWorld: Starting Level Process
GLWorld-->GameRenderer:
GameRenderer->GLWorld:draw(gl)
note right of GLWorld: Cube animations
GLWorld-->Screen (UI): 3D Cube Scene
deactivate GLWorld
deactivate GameRenderer

deactivate Screen (UI)
Screen (UI)->Screen (UI): show()
activate Screen (UI)
```

Diagrama de secuencia (Zoom)

```
User->Screen (UI): Drag Two Finger
activate Screen (UI)
Screen (UI)-->A_3D_Logic: onTouchEvent()
deactivate Screen (UI)
activate A_3D_Logic
A_3D_Logic->A_3D_Logic: multiTouch()
Clock->Clock:
A_3D_Logic->GameRenderer: zoom(factor)
activate GameRenderer
GameRenderer->GameRenderer: zoom = true
GameRenderer-->A_3D_Logic:
deactivate GameRenderer
deactivate A_3D_Logic

Clock-->GameRenderer: drawFrame(gl)
activate GameRenderer
GameRenderer->Screen (UI): clearGL()
activate Screen (UI)
GameRenderer->Screen (UI):updateProjection()

GameRenderer->Textures: renderBackground(gl)
activate Textures
Textures-->Screen (UI):background
deactivate Textures
GameRenderer->Textures: renderLevelLabel(gl, level)
activate Textures
Textures-->Screen (UI):levelLabel
deactivate Textures
GameRenderer->Textures: renderHomeButton(gl)
activate Textures
Textures-->Screen (UI):homeButton
deactivate Textures
```

```
GameRenderer->Textures: loadSquareTextures(gl)
activate Textures
Textures-->GameRenderer:
deactivate Textures

GameRenderer->GLWorld:draw(gl)

activate GLWorld
note right of GLWorld: Cube animations
GLWorld-->Screen (UI): 3D Cube Scene
deactivate GLWorld
deactivate GameRenderer

deactivate Screen (UI)
Screen (UI)->Screen (UI): show()
activate Screen (UI)
```

