

Portada



## 0 AGRADECIMIENTOS

En la consecución de este proyecto han contribuido varias personas, corrigiendo, apoyando, aconsejando y animando cuando más se complicaba la cosa.

En primer lugar quiero dar las gracias a mi director de proyecto Germán Rigau Claramunt y al departamento de Lenguajes y Sistemas Informáticos por su apoyo y por haber estado siempre dispuesto a ayudar en cualquier problema que surgiese.

Gracias a mis compañeros, que siempre me apoyaron con ideas y nuevos puntos de vista, y sobre todo por haber convivido estos últimos 3 años. Asier, Fernando, David, Tania, Ion, Diego, Adrián y Borja, gracias.

A mis padres y a mi hermana por haber estado siempre ahí, en los malos y en los buenos momentos. A mi novia Miren por apoyarme en todos los pasos que daba y sigo dando. A sus padres que siempre me animaron a comenzar esta aventura.

Gracias.



## INDICE DE CONTENIDO

0. AGRADECIMIENTOS.....	3
1. INTRODUCCIÓN.....	9
2. DOCUMENTO DE OBJETIVOS DEL PROYECTO (DOP).....	11
2.1. OBJETIVOS DEL PROYECTO.....	11
2.2. MÉTODO DE TRABAJO.....	11
2.3. ALCANCE.....	12
2.4. PLANIFICACIÓN.....	14
2.5. PLAN DE CONTINGENCIA.....	17
2.6. RECURSOS.....	18
2.7. ANÁLISIS DE FACTIBILIDAD.....	18
3. ELECCIÓN TECNOLÓGICA.....	19
3.1. ANDROID.....	19
3.2. JAVA.....	20
3.3. ECLIPSE.....	20
3.4. OPENGL.....	21
3.5. SOURCEFORGE.....	21
3.6. LIBREOFFICE.....	21
3.7. GLIFFY.....	21
4. ARQUITECTURA DEL SISTEMA.....	23
5. CAPTURA DE REQUISITOS.....	25
5.1. CASOS DE USO DEL USUARIO.....	25
5.2. MODELO DE DOMINIO.....	26
6. ANÁLISIS.....	29
6.1. CASO DE USO GIRAR FILA.....	29
6.2. CASO DE USO VOLVER ESTADO INICIAL.....	31
7. DISEÑO.....	33
7.1. DIAGRAMA DE SECUENCIA.....	33
8. IMPLEMENTACIÓN.....	39
8.1. MANUAL DE IMPLEMENTACIÓN.....	39
8.2. ALTERNATIVAS DE IMPLEMENTACIÓN.....	42
8.3. DIAGRAMA DE CLASES.....	44
9. PRUEBAS.....	45
9.1. PRUEBAS DE CAJA NEGRA.....	45
9.2. PRUEBAS DE CAJA BLANCA.....	46
9.3. OTRAS PRUEBAS.....	46
10. IMPLANTACIÓN.....	49
10.1. ESPECIFICACIONES.....	49
10.2. COMPILACIÓN.....	49
10.3. INSTALACIÓN EN DISPOSITIVOS.....	49
11. GESTIÓN.....	51
12. CONCLUSIONES.....	57
12.1. OBJETIVOS LOGRADOS.....	57
12.2. VALORACIÓN PERSONAL.....	57
12.3. TRABAJOS FUTUROS.....	57

13. APÉNDICES.....	59
13.1. MANUAL DEL USUARIO.....	59
13.2. GUÍA DE INSTALACIÓN PARA EL USUARIO.....	62
13.3. GUÍA DE INSTALACIÓN PARA EL DESARROLLADOR.....	63
13.4. OPENGL.....	70
14. BIBLIOGRAFIA.....	71

## Índice de ilustraciones

Ilustración 1: Cubo de rubik estándar girando.....	10
Ilustración 2: Diagrama EDT.....	14
Ilustración 3: Duración de las tareas.....	15
Ilustración 4: Diagrama de Gantt.....	16
Ilustración 5: Distribución de capas del sistema.....	23
Ilustración 6: Casos de uso del usuario.....	25
Ilustración 7: Modelo de dominio.....	27
Ilustración 8: Diagrama de secuencia del caso de uso girar fila.....	29
Ilustración 9: Diagrama de secuencia del caso de uso salir.....	31
Ilustración 10: Diagrama de secuencia de la función Pulsar(x,y).....	34
Ilustración 11: Diagrama de secuencia de la función Mover(x,y).....	36
Ilustración 12: Diagrama de secuencia de la función Soltar(x,y).....	37
Ilustración 13: Diagrama de secuencia de la función Estado Inicial().....	38
Ilustración 14: Ciclo de vida de las aplicaciones de Android.....	40
Ilustración 15: Diagrama de clases.....	44
Ilustración 16: Gráfico Horas Previsión vs Horas Reales.....	52
Ilustración 17: Diagrama de Gantt previsión vs realidad.....	53
Ilustración 18: Duración de las tareas previstas vs reales.....	54
Ilustración 19: Gráfico de horas de las tareas previstas vs reales.....	55
Ilustración 20: Inicio de la aplicación.....	59
Ilustración 21: Primer movimiento en la aplicación.....	60
Ilustración 22: Soltar el primer movimiento de la aplicación.....	60
Ilustración 23: La aplicación tras varios movimientos.....	61
Ilustración 24: Habilitar orígenes desconocidos en el dispositivo.....	62
Ilustración 25: Buscar Synaptic en el equipo.....	63
Ilustración 26: Buscar eclipse en Synaptic.....	64
Ilustración 27: Añadir el repositorio de Android.....	65
Ilustración 28: Determinar la ubicación del SDK.....	66
Ilustración 29: Instalar las APIs deseadas.....	67
Ilustración 30: Crear un nuevo dispositivo virtual de Android.....	67
Ilustración 31: Ejecutando el emulador.....	68
Ilustración 32: Comando lsusb mostrando el idVendor y el idProduct.....	69





## 1 INTRODUCCIÓN

Esta memoria corresponde al proyecto de final de carrera de la Ingeniera Técnica de Informática de Sistemas (ITIS) del alumno Gorka Revilla Fernández. Cursado en la Facultad de Informática de la Universidad del País Vasco (UPV/EHU) y dirigida por el profesor Germán Rigau Claramunt del departamento de Lenguajes y Sistemas Informáticos.

Desde que Android fuera comprado por Google, el desarrollo de este sistema operativo ha sido imparable. Los celulares pasaron de ser teléfonos móviles, que solo realizan llamadas y envían SMS, a Smartphones, dispositivos inteligentes capaces de realizar todo lo que el usuario desee.

A todo esto hay que añadir la potencia de estos aparatos. Muchos de ellos con memorias RAM de casi un 1GB, un procesador de mas de 1GHz e incluso ahora muchos incorporan los nuevos microprocesadores de doble núcleo.

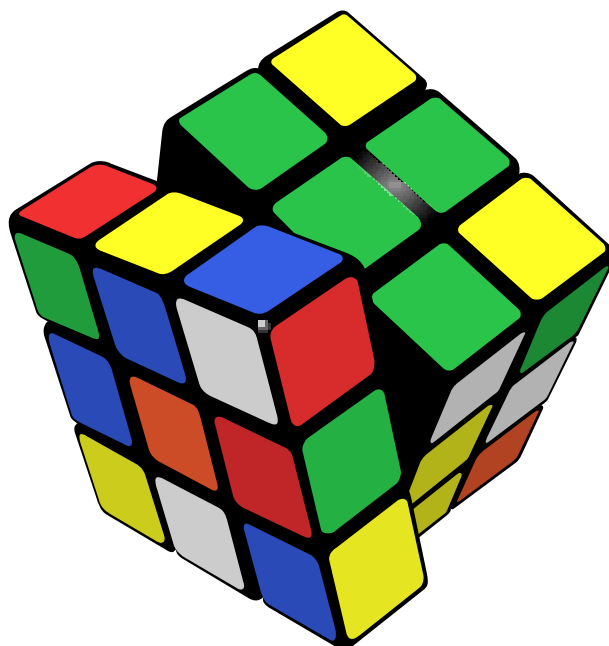
Con estos ingredientes ya se había preparado el caldo de cultivo para el desarrollo de nuevas aplicaciones para este tipo de dispositivos, lo cual llamaba mucho la atención en un nuevo mercado que se estaba abriendo. Qué mejor forma de acabar la carrera que realizando una aplicación basada en algo que seguramente marque un antes y un después en el desarrollo de las aplicaciones.

Una vez decidida la plataforma solo faltaba encontrar algo que desarrollar. Una aplicación que realmente supusiese un reto para mi. Debía de ser algo que me motivase y a la vez que pudiera lograr en el tiempo de duración del proyecto. Mi idea era desde un principio, realizar un juego, ya que hasta ahora me había llamado mucho la atención y ya tenía algo de experiencia sobre ellos puesto que hacia 2 años había realizado un juego en 2 dimensiones pero para plataforma PC. Se me ocurrió dar un paso más en el desarrollo de videojuegos y realizarlo esta vez en 3 dimensiones.

Con todo esto ya solo me tocaba decidir qué realizar. Tenía que ser simple, pero completo. Pensando se me ocurrió la idea de hacer un cubo de rubik en 3 dimensiones.

El cubo de rubik es un rompecabezas mecánico, cuyas caras están divididas en cuadrados de un mismo color que se pueden cambiar de posición. El objetivo del rompecabezas consiste en colocar todos los cuadrados de cada cara del cubo del mismo color.

En la ilustración 1 podemos observar un cubo de dimensión 3x3x3, esto es, cada cara esta constituida por 9 cuadrados y su nombre es “Rubik estándar”. También existen otras versiones del cubo: la 2x2x2 “cubo de bolsillo”, el 4x4x4 “La venganza de Rubik”, el 5x5x5 “El cubo del Profesor” y desde septiembre de 2008 el 6x6x6 “V-Cube 6” y el 7x7x7 “V-Cube 7”.



*Ilustración 1: Cubo de rubik estándar girando*

Si nos centramos en el cubo de rubik estándar tendríamos mas de 43 trillones de permutaciones posibles. Y el récord mundial en su resolución lo tiene Feliks Zemdegis con un tiempo de 5,66 segundos. Aunque es posible que haya tiempos mejores pero que no están registrados por no ser en competiciones reguladas (Ref. [11]).

Por tanto, ya estaba decidido qué era lo que iba a realizar. La aplicación consistiría en un juego que simulase los movimientos de un cubo de rubik, todo ello para un dispositivo Android y mediante gráficos en 3 dimensiones.

## **2 DOCUMENTO DE OBJETIVOS DEL PROYECTO (DOP)**

### **2.1 OBJETIVOS DEL PROYECTO**

Este proyecto consiste en la creación de una aplicación, a modo de juego, que represente un cubo de rubik. El juego se realizará en 3 dimensiones y debe de permitir el movimiento de cada una de las filas de las caras que se muestran en la pantalla, realizando movimientos coherentes a como los realiza un cubo de rubik físico. Es importante el hecho de que esto es una demo, simplemente simulará la situación de partir con un cubo de rubik inicial y poder realizar los giros deseados, pudiendo desordenarlo como se desee e intentar volver al estado inicial.

Otra dificultad añadida es que tiene que desarrollarse para dispositivos Android con versión superior a la 2.1. Esto conlleva que tiene que ser una aplicación basada en la API del dispositivo. Que se ejecute en cualquier dispositivo de cualquier marca sin necesidad de instalaciones previas de ningún otro software. Así que una restricción a tener en cuenta es que en ningún momento se podrá utilizar ningún software de terceras partes, como podría ser algún framework o software parecido.

### **2.2 MÉTODO DE TRABAJO**

Se definirán 3 iteraciones para la consecución del proyecto, a las cuales se les definirán unos objetivos y un plazo para poder ir controlando el avance del proyecto. Se tiene en cuenta que la memoria se irá realizando a medida que se va avanzado con el proyecto. Es importante destacar que para la realización de este proyecto no se cuenta con una dedicación completa, ya que el alumno tendrá que compaginarlo con los estudios del ultimo año de carrera. Se prevé la entrega del proyecto para la convocatoria extraordinaria de Septiembre procurando acabar el proyecto durante el curso (hasta junio) y dejando el verano libre de carga de trabajo o para poder realizar últimos detalles en el mismo.

Al finalizar cada iteración se acordará una reunión con el director del proyecto en la que se revisarán todos los objetivos conseguidos y los plazos. También se utilizarán para resolver posibles dudas que hayan surgido.

Se contabilizarán todas las horas invertidas en el proyecto desde el principio hasta el final para así poder realizar un buen análisis de la gestión llevada a cabo. Para todos los temas relacionados con la gestión se utilizará la aplicación de Microsoft llamada Microsoft Project 2010.

Se sabe a priori que en este proyecto tendrá especial importancia la formación. Ya que el alumno no sabe nada sobre los dispositivos en los que se va a trabajar. Se tendrá que profundizar mucho en el tema de investigación en los comienzos del proyecto.

Toda la información recogida, así como los ejemplos, o el propio código del programa serán almacenados en un servidor externo para en caso de fallo del equipo en el que se trabaje haya una copia de respaldo en el servidor.

## **2.3 ALCANCE**

Se recogerá mediante el diagrama de Estructura de Descomposición del Trabajo (EDT) todos los aspectos que se realizarán durante la duración del proyecto de esta forma conseguiremos una buena planificación y organización del mismo. También acotaremos nuestro trabajo para no excedernos en nuestras tareas. Todas ellas se dividirán en 3 partes claras: Tácticas, Operativas y Formativas.

### **2.3.1 Procesos Tácticos**

Para realizar un correcto seguimiento del proyecto se crean varios pasos a seguir. Estos nos ayudarán a poder llevar una correcta gestión y planificación del mismo.

El primero de ellos será la realización de unas reuniones de control periódicas entre el director y el alumno. En estas reuniones se controlarán los objetivos alcanzados y los tiempos. También pueden servir de ayuda para la resolución de algún problema puntual.

Se redactará un Documento de Objetivos del proyecto (DOP), en el que se recogerán todas las acciones a realizar en el mismo.

### **2.3.2 Procesos Operativos**

Durante el desarrollo del proyecto se deberán de indicar también varios pasos a seguir. De esta forma nos ahorramos tiempo y aseguramos una mejor implementación de la aplicación.

Se comenzará realizando una captura de requisitos, identificando cuales son los casos de uso necesarios para cumplir con los objetivos. Tras esta fase, se realizará un análisis que profundizará en los casos de uso, describiendo de forma mucho mas detallada cada caso. Una vez finalizado el análisis, se realizará una fase de diseño de la aplicación, en ésta se deberán indicar las clases de la aplicación, así como su interacción de unas con otras.

Tras estas fases, se comenzará con la implementación del proyecto usando las tecnologías elegidas. Es bastante probable que surjan problemas debido a un mal diseño y por lo tanto se deberá de volver a la fase de diseño para rehacerlo correctamente. Una vez finalizada se entrará en el periodo de pruebas donde también puede que se tenga que volver a la fase anterior de implementación para corregir errores.

### **2.3.3 Procesos Formativos**

Partiendo de la base de que no se conoce nada en absoluto sobre Android y gráficos en 3D, esta parte va a tomar gran importancia en la consecución del proyecto. Primeramente se deberá de realizar un exhaustivo análisis del mercado para ver que herramientas utilizar, buscar algún ejemplo, realizar alguna prueba, etc.

Tras esto y una vez tomada la decisión tecnológica, se entrará en un periodo de formación sobre las herramientas a utilizar. Esta fase se deberá de tener muy en cuenta antes de comenzar con los diseños y la implementación de la aplicación ya que sera esencial para estas dos fases.

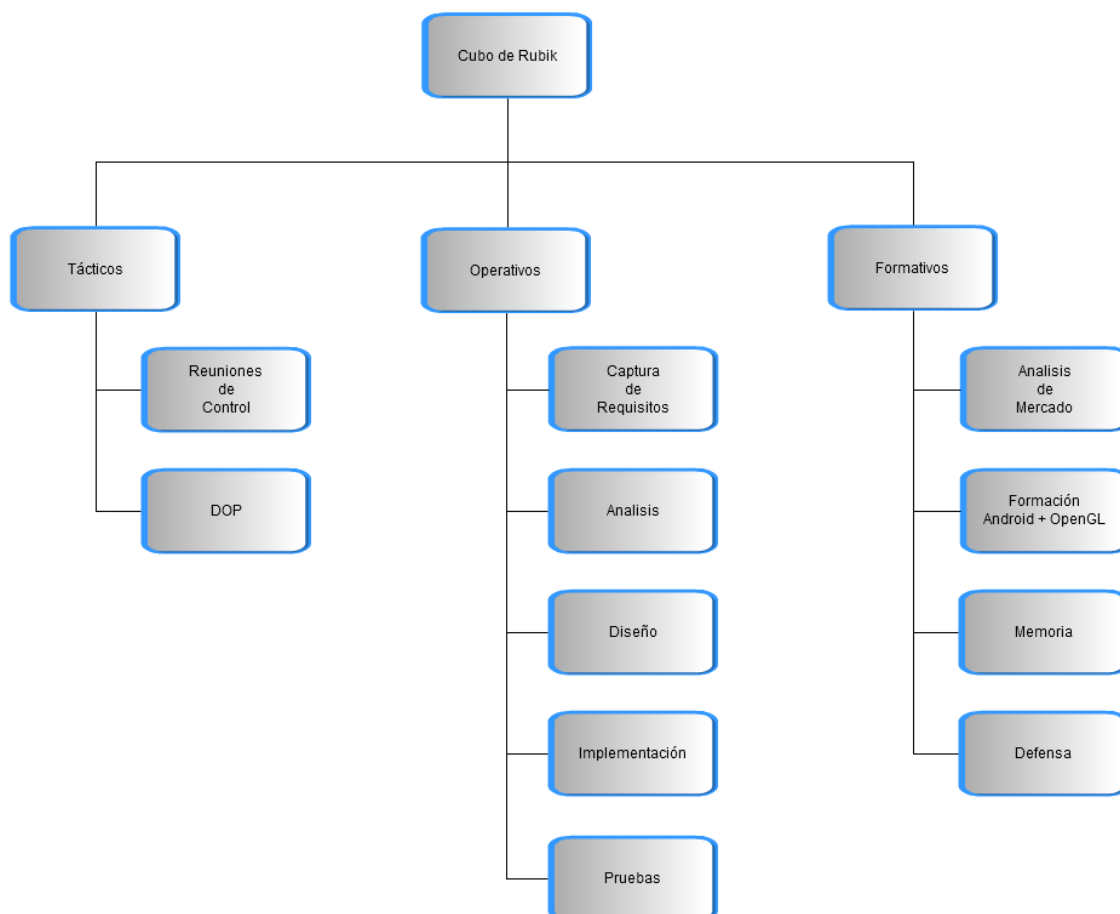
A la vez que se desarrolla el proyecto se deberá de ir redactando una memoria donde se recogerán todos los avances del mismo por escrito. Esta memoria será un entregable al finalizar todo el desarrollo. Al ser esto un PFC se considerará que esta parte es un proceso

formativo de la misma.

Por último y una vez finalizado el desarrollo y escrita la memoria se preparará una defensa del proyecto por parte del alumno, que incluirá unas diapositivas y alguna demo del proyecto, así como unos ensayos de la presentación. Al igual que con la memoria, ésta parte se considerará un proceso formativo.

### 2.3.4 Diagrama de Estructura de Descomposición del Trabajo (EDT)

Todos los procesos necesarios para este proyecto se recogen clasificados mediante el diagrama EDT de la ilustración 2.



*Ilustración 2: Diagrama EDT*

## 2.4 PLANIFICACIÓN

Se realizará una planificación de la duración de cada tarea a realizar. Se mostrará todo en un diagrama de Gantt. Todo esto es una estimación y es probable que estos tiempo realmente no se cumplan, aunque se intentará ajustarse a ellos. Se decide que sera un proyecto a largo plazo ya que el alumno esta cursando el 3º curso de la ITIS. Por ello se decide que se presentará a la convocatoria extraordinaria de septiembre de 2011.

La primera de las iteraciones consistirá en el análisis del mercado, realización del DOP y algunas pequeñas pruebas con las elecciones tecnológicas tomadas en el análisis del mercado.

Ya que el proyecto comienza a principios de noviembre, esta fase se prevee su finalización para finales de noviembre o principios de diciembre, justo antes de la preparación de los

exámenes del primer cuatrimestre. Como se puede ver en la Ilustración 4 que recoge el Gantt, entre el DOP y la Captura de requisitos habrá un pequeño parón de dos meses aproximadamente.

Tras ello y una vez alcanzados los objetivos de la primera iteración, se comenzará con la segunda, la cual incluirá la captura de requisitos, el análisis y diseño, así como la implementación de una gran parte del código, dejando para la siguiente fase los detalles y alguna parte que haya dado problemas. También se realizará gran parte de la memoria en esta fase. Al finalizar esta iteración la aplicación debería de ser más o menos funcional. Esta será la fase más complicada de todo el desarrollo del proyecto, por esto mismo se le asigna la mayor parte del tiempo, la cual irá desde la finalización de los exámenes del primer cuatrimestre (finales de enero), hasta el comienzo de la preparación de los exámenes de la convocatoria ordinaria de junio (principios de mayo), lo que vendría siendo casi todo el segundo cuatrimestre.

Una vez finalizadas las dos iteraciones anteriores se comenzará con la tercera, la cual debería de ser la más ligera debido a que se realizará en época de exámenes. Esta iteración consistirá en finalizar lo que quede de la parte de implementación de la fase 2 y la realización de las pruebas y su corrección. Se terminará de redactar la memoria y se preparará la defensa. Comenzará tras finalizar la convocatoria ordinaria de junio y si todo ha ido bien, esto debería de finalizar para antes de septiembre.

En la Ilustración 3 se indica la duración de las tareas en forma de tabla, donde se ven claramente las fechas para realizar cada una de las tareas.

	Nombre de tarea ▼	Duración ▼	Comienzo ▼	Fin ▼	Predecesoras ▼
1	Análisis de Mercado	10 días	mar 02/11/10	lun 15/11/10	
2	DOP	10 días	lun 08/11/10	vie 19/11/10	
3	Captura de Requisitos	10 días	mar 25/01/11	lun 07/02/11	
4	Análisis	10 días	mar 08/02/11	lun 21/02/11	3
5	Diseño	15 días	mar 22/02/11	lun 14/03/11	4
6	Implementación	40 días	mar 15/03/11	lun 09/05/11	5
7	Pruebas	2 días	vie 10/06/11	lun 13/06/11	6
8	Memoria	100 días	mar 25/01/11	lun 13/06/11	
9	Formación	160 días	mar 02/11/10	lun 13/06/11	

*Ilustración 3: Duración de las tareas*

En la Ilustración 4 podemos ver el diagrama de Gantt indicando todas las tareas a realizar y la planificación del tiempo para cada una de ellas. También se observa cada iteración de un color distinto. Destaca la formación con respecto a otras tareas ya que será algo que se realice durante todo el proyecto puesto que según se avance sobre él, se tendrán que seguir investigando nuevas cosas. Por lo tanto, es algo que estará presente durante toda la duración del proyecto. La memoria comenzará junto con la captura de requisitos y finalizará tras realizar las pruebas. Durante el periodo entre la finalización de la implementación y las pruebas se redactarán todos los demás puntos de la memoria. Se espera que finalice para el 13 de Junio de 2011.

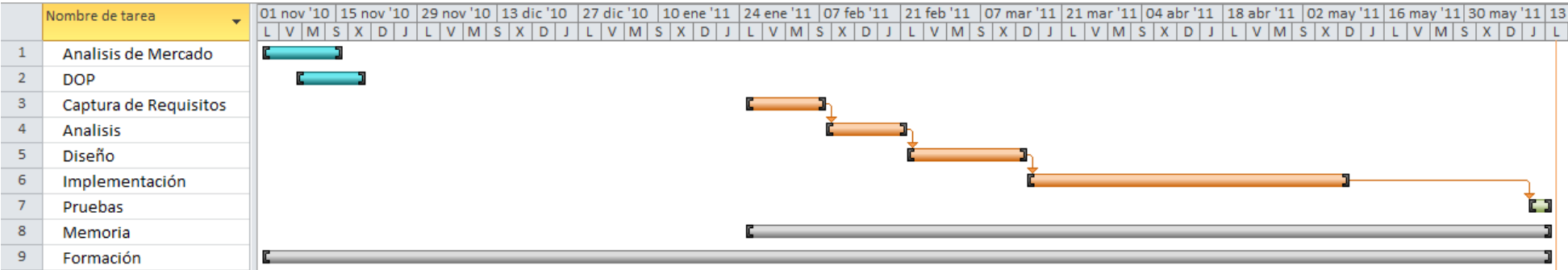


Ilustración 4: Diagrama de Gantt



### **2.4.1 Estimación de costes**

El tiempo de dedicación será muy variable de unas semanas a otras, dependiendo de la situación del alumno. Se calcula que se le dedicarán de promedio unas 15 horas a la semana al proyecto, siendo unas 20 semanas de desarrollo, ya que habrá que descontar el tiempo que se tendrá que dedicar a exámenes. Lo que equivale a unas 310 horas. A continuación se indica el tiempo total que se estima que durará cada tarea durante el transcurso de todo el proyecto.

DOP: 10 horas

Análisis de Mercado: 15 horas

Captura de Requisitos: 10 horas

Análisis: 10 horas

Diseño: 15 horas

Implementación: 120 horas

Pruebas: 5 horas

Memoria: 20 horas

Defensa: 15 horas

Formación: 90 horas

## **2.5 PLAN DE CONTINGENCIA**

Será necesario un buen plan de contingencia para poder solucionar los problemas que se generen durante el proyecto.

### **2.5.1 Retrasos en la consecución de los objetivos del proyecto**

Gravedad: Media

Probabilidad: Alta

Solución: Se intentará adaptarse lo máximo posible a los tiempos estimados. En caso de que sea necesario se volverán a replanificar.

### **2.5.2 No disponibilidad del servicio de almacenamiento (SourceForge)**

Gravedad: Media

Probabilidad: Muy Baja

Solución: En caso de que se pueda, se continuará con la copia en el ordenador del alumno y buscando otro sitio donde almacenar el proyecto mientras no esté disponible el servidor actual, para no perder datos en caso de una segunda caída. Es por lo cual que en todo momento se procurará tener una segunda copia almacenada en otro lugar distinto al que se esté trabajando.

### **2.5.3 Mala Elección de la Tecnológica**

Gravedad: Muy Alta

Probabilidad: Baja

Solución: Se intentará que afecte lo mínimo al desarrollo del proyecto procurando que sea lo mas modular posible y se buscara una elección tecnológica mas adecuada para el problema, intentando no cometer los mismos fallos.

### **2.5.4 Baja temporal de alguno de los miembros del proyecto**

Gravedad: Alta

Probabilidad: Baja

Solución: En caso de que la baja sea del director se intentará seguir adelante con el proyecto sin contar con su ayuda. En caso de que sea del alumno se esperará hasta la vuelta en activo del alumno. Se ajustarán los tiempos del proyecto según sea necesario.

## **2.6 RECURSOS**

El alumno cuenta con un portátil de gama media con Windows 7 y Ubuntu 11.04 instalado. Se utilizará Ubuntu para la realización de todas las actividades tales como documentación, redacción de la memoria o implementación, aunque se tendrá que acceder a Windows 7 para llevar la gestión del proyecto mediante Microsoft Project 2010 ya que no permite su ejecución sobre sistemas operativos Linux. Para la implementación utilizará el emulador de Android disponible para Linux. También cuenta con un Motorola Defy con Android 2.2 en el cual podrá realizar las pruebas necesarias. Serán necesarias hojas en blanco y bolígrafo para realizar esquemas.

## **2.7 ANÁLISIS DE FACTIBILIDAD**

La dificultad del proyecto es más que evidente, entre otras cosas por la falta de experiencia por parte del alumno en programación sobre Android y aplicaciones 3D. Aunque la motivación por la realización del proyecto y la cantidad de documentación que existe en internet ayudará a resolver cualquier duda que surja.

En vista de lo expuesto hasta el momento, parece que el proyecto es factible.

## 3 ELECCIÓN TECNOLÓGICA

### 3.1 ANDROID

Una de las decisiones más importantes fue la de elegir para que dispositivos desarrollar la aplicación. En un principio se pensó en desarrollarla para dispositivos Symbian, aunque era ya un sistema operativo que solamente utilizaba Nokia y que cada vez estaba cayendo en desuso en favor de otros sistemas operativos como iOS o Android (Ref. [11]). Tras esto, se pasó a elegir entre estos dos últimos y se terminó decantando por Android ya que el alumno no contaba con ningún dispositivo Apple, ni siquiera un Mac sobre el que desarrollar (imprescindible para aplicaciones sobre iOS). Por lo cual la decisión era clara.

#### 3.1.1 ¿Que es Android?

Android es uno de los sistemas operativos más usados para smartphones hoy en día. Es propiedad de Google y actualmente compite en el mercado de estos dispositivos con el sistema operativo de Apple iOS y con Windows Phone. Es el sistema operativo por el que las mas marcas comerciales han apostado (Samsung, Motorola, Sony Ericsson, HTC...) y actualmente lo podemos encontrar en móviles, tablets, netbooks y numerosos sistemas empujados. Android es un sistema operativo basado en Linux (El kernel de Linus Torvalds). Cuenta con numerosas bibliotecas (Entre las que está OpenGL por ejemplo), tiene una plataforma Java y cuenta con numerosas aplicaciones.

Existe un gran mercado de aplicaciones sobre este sistema operativo, muchas de ellas se encuentran en el Android Market de Google que viene por defecto en todos los dispositivos que cuentan con Android y desde donde podremos descargar aplicaciones teniendo una cuenta de Google.

Para este sistema operativo necesitaremos desarrollar aplicaciones específicas para él. No nos valdrían aplicaciones desarrolladas con Java para otros sistemas operativos.

#### 3.1.2 Máquina Virtual Dalvik

Android carece de la máquina virtual de Java y en su lugar cuenta con la máquina virtual Dalvik (Ref. [1]). Todas las aplicaciones desarrolladas para Android se compilan de manera diferente a aplicaciones normales de Java, aunque sí que se generan los archivos de bytecode de Java (.class). La máquina virtual Dalvik ofrece la ventaja de que requiere de poca memoria para trabajar y permite ejecutar varias instancias de ella misma al mismo tiempo.

Se podría decir que esta máquina virtual ha sido desarrollada para un mejor funcionamiento sobre dispositivos que no tengan tanto potencial como un ordenador de sobremesa. Con la pega de que las aplicaciones hay que desarrollarlas exclusivamente para ella.

#### 3.1.3 API

Una API es una interfaz que nos proporciona comunicación entre dos componentes de software. En este caso entre una aplicación desarrollada por nosotros y las bibliotecas del sistema operativo. Google ofrece una API por cada versión de Android disponible. Cada API

permite desarrollar aplicaciones para esa versión y sus versiones posteriores. Son gratuitas y de código abierto. La API permite el desarrollo de aplicaciones para el lenguaje java y para el lenguaje C++.

### **3.1.4 Versiones de Android**

Cada cierto tiempo Google saca una versión de Android. Recientemente Google ha publicado la versión 4.0 Ice Cream Sandwich. Esta versión valdrá tanto para móviles como para tablets, al contrario que las anteriores (la versión 3.x era para tablets y la 2.x para móviles). El que Google saque una nueva versión del sistema operativo no tiene que ver con que un dispositivo que ya tengamos pueda ser directamente actualizable a esta versión, ya que la empresa fabricante del dispositivo es la encargada de decidir si publica una versión para ese dispositivo. Por lo tanto puede que contemos con un dispositivo que su última versión sea la 2.1 y no se pueda actualizar hasta la versión 2.2 o incluso 4.0.

De esta forma son los propios fabricantes los que publican los sistemas operativos para sus propios dispositivos. Aunque también existe la posibilidad de poder instalar sobre nuestros dispositivos un sistema operativo de terceros, ajeno a la empresa que fabrica el dispositivo. En muchas ocasiones estos sistemas operativos pueden ofrecernos nuevas versiones de Android sobre dispositivos que la empresa no ha actualizado con la pega de los posibles fallos que podrían ocurrir y la pérdida de la garantía de software.

### **3.1.5 Tipos de Dispositivos**

Los dispositivos con Android más comunes son los smartphones y las tablets. Como se ha comentado en el punto anterior la versión de la rama 3.x corresponde a las tablets y la rama 2.x corresponde a los smartphones. Aunque cada fabricante fabrique dispositivos diferentes, con diferente hardware y especificaciones, al llevar todos el mismo sistema operativo serán capaces de poder ejecutar las mismas aplicaciones, a no ser que la versión sea inferior a la que requiere el software.

## **3.2 JAVA**

Para el desarrollo de aplicaciones para Android existen dos lenguajes de programación: Java y C++. Se eligió java porque es el lenguaje mas conocido por el desarrollador y porque la mayoría de aplicaciones se desarrollan para este sistema. Aparte de que la mayoría de aplicaciones analizadas estaban realizadas en Java y sus autores aseguraban que era mucho más sencillo la realización de juegos para este tipo de dispositivos mediante este lenguaje de programación.

## **3.3 ECLIPSE**

El SDK de Google esta diseñado para este IDE. Y toda su documentación así como tutoriales y ejemplos están todos realizados bajo este software. No convenía complicarse más de lo necesario y además el alumno conocía la herramienta, por lo que no se dudó en elegir Eclipse ([www.eclipse.org](http://www.eclipse.org)) como herramienta de trabajo.

### **3.4 OPENGL**

A la hora de representar nuestra aplicación vamos a tener que necesitar alguna de las librerías de las que nos ofrece Android. Uno de los requisitos del proyecto es que sea en 3 dimensiones para lo cual Android ofrece la librería de OpenGL ES 2.0. No se duda en utilizar esta tecnología, ya que es la herramienta que usan casi todas las aplicaciones y juegos de Android que trabajan con gráficos.

### **3.5 SOURCEFORGE**

Como método de almacenamiento se elige SourceForge ya que se ha trabajado anteriormente con el con muy buenos resultados y cubre todas las necesidades. Se instala el plug-in SubEclipse para poder usar el SVN de SourceForge junto con Eclipse, sin necesidad de salirse del IDE para tener que subir archivos o volver a versiones anteriores. El plug-in de SubEclipse se encuentra en los repositorios que vienen incorporados con la instalación clásica de Eclipse, por lo cual desde el propio programa se puede instalar fácilmente.

### **3.6 LIBREOFFICE**

Como se usa un sistema operativo Linux surge la duda entre OpenOffice y LibreOffice. A sabiendas de que LibreOffice es un fork de OpenOffice y que por lo tanto sus características son idénticas se toma la decisión de trabajar con LibreOffice ya que Oracle ha abandonado el proyecto de OpenOffice y LibreOffice ahora mismo continua manteniéndose por la comunidad.

### **3.7 GLIFFY**

Para la realización de diagramas se usa la aplicación web Gliffy ([www.gliffy.com](http://www.gliffy.com)) que ofrece 30 días de prueba gratuita. Se utiliza especialmente para el EDT y el diagrama de clases. En un principio no se cree que sea necesaria la compra del software, aunque en caso de ser necesario su precio tampoco es muy elevado y podría convenir comprarlo durante un mes.



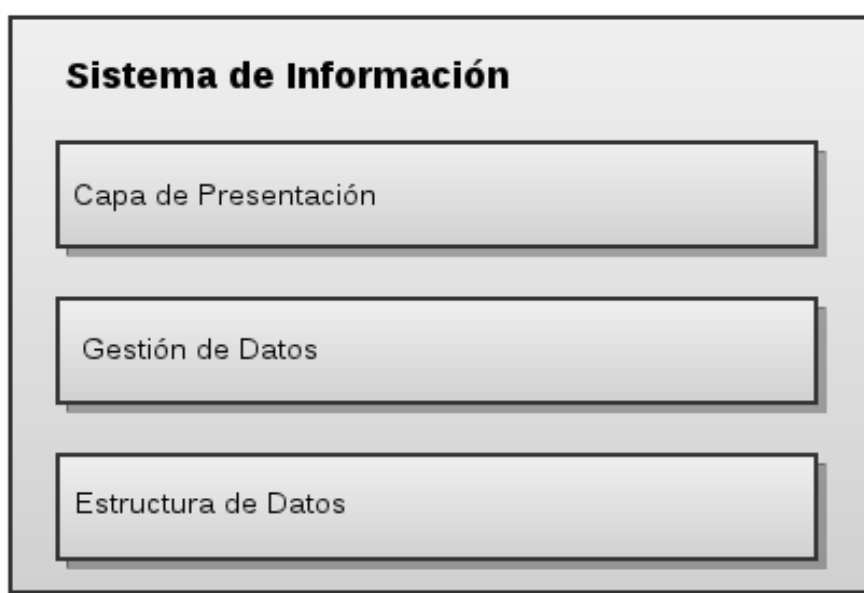
## 4 ARQUITECTURA DEL SISTEMA

Se utilizará un sistema de dos capas, ya que no será necesario almacenar los datos en disco mediante algún SGBD o parecido, por lo que toda la información será almacenado en objetos y residente en memoria. Mediante estas dos capas el sistema debería de ser capaz de poder gestionar toda la información de la aplicación.

La primera capa será la que deberá de encargarse de mostrar la información al usuario según las especificaciones del proyecto, de esta capa hay que destacar que una de las especificaciones del proyecto es que sea en 3 dimensiones, por lo cual para mostrar la aplicación se debe de usar una tecnología que incorpore Android y permita esto, es por lo cual por lo que se decidió el uso de OpenGL, el cual sera el encargado de representar toda la información por pantalla.

La segunda capa se encargará de gestionar todos los datos y su almacenamiento en las estructuras adecuadas. Esta capa en principio no debería de tener mayores problemas a la hora de implementarse ya que se usarán funciones en su mayoría de Android.

Por ello tendríamos una distribución de capas del sistema como como la mostrada en la ilustración 5.



*Ilustración 5: Distribución de capas del sistema*





## 5 CAPTURA DE REQUISITOS

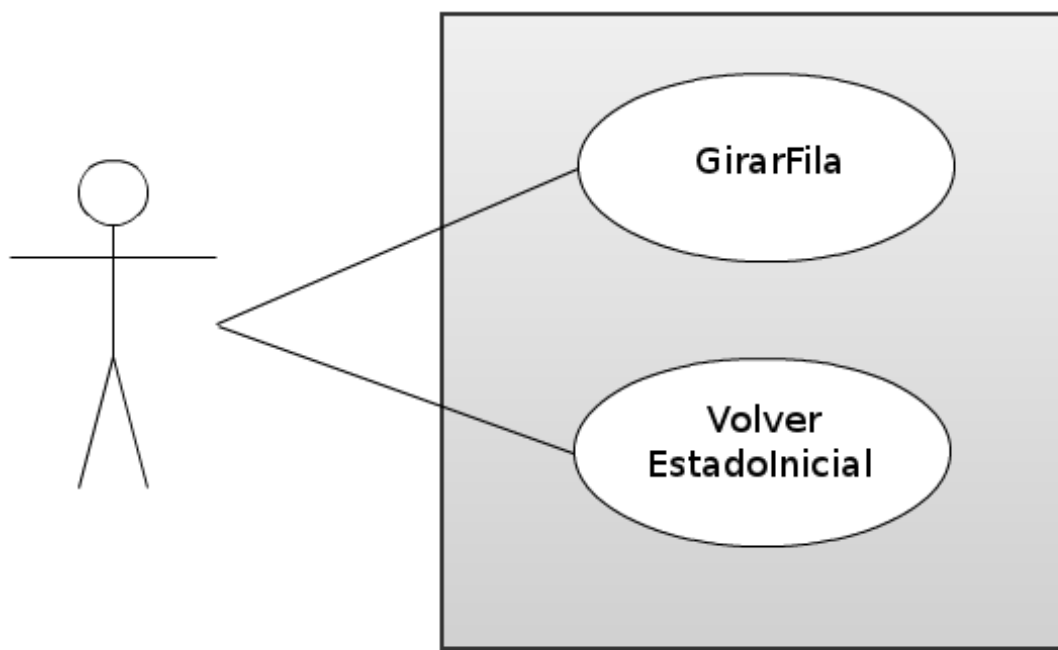
Se desarrollará una aplicación que reproduzca los movimientos de un cubo de rubik. Permitirá el giro de cualquier fila visible desde la perspectiva 3D desde la que se esta visionando el cubo. No contará con ningún tipo de menú, la aplicación directamente se ejecutará y mostrará el cubo permitiendo directamente realizar los giros.

Con la aplicación solo trabajará una persona, por lo tanto se identifica un solo actor al cual a partir de ahora se le denominará usuario.

### 5.1 CASOS DE USO DEL USUARIO

El usuario iniciará la aplicación y solo podrá realizar giros en cualquiera de las filas visibles del cubo. Al iniciar la aplicación se mostrará un cubo ordenado, con cada cara de un color y según se vayan girando las filas irá cambiando el color de los cubos representados. De esta forma se identifica el primer caso de uso, que se le denominará girar fila. De otra manera el usuario podrá en cualquier momento salir de la aplicación lo cual hará que el cubo de rubik vuelva a su estado inicial y cierre el programa. Así mismo éste será el segundo caso de uso del usuario, volver a estado inicial.

En la Ilustración 6 se indican gráficamente los casos de uso del usuario.



*Ilustración 6: Casos de uso del usuario*

### **5.1.1 Caso de uso Girar Fila**

El usuario podrá realizar cualquier tipo de giro en la dirección que él desee de cualquiera de las filas de las 3 caras que se mostrarán en pantalla. Los giros se podrán realizar en el sentido que desee el usuario y podrá realizar giros de los grados que desee en función de cuanto desplace el dedo por la pantalla. Por ejemplo, podrá hacer un giro de 180° de una de las filas con un solo desplazamiento del dedo.

Tras un giro, el cubo de rubik tiene que seguir manteniendo su forma de cuadrado por lo cual si un giro no queda ajustado a la forma del cubo deberá de ajustarse al levantar el dedo.

En todo momento el cubo tiene que mantener su concordancia con los colores de cada uno de los cubitos que tiene el cubo.

Este caso de uso se podría repetir tantas veces como el usuario desee en cada una de sus filas.

### **5.1.2 Caso de uso Volver a Estado Inicial**

El usuario podrá reiniciar el estado de la aplicación en cualquier momento mediante la pulsación de la tecla Back de la que disponen todos los dispositivo. Mediante esta tecla la aplicación se cerrará y al volver a iniciarla el cubo volverá a su estado inicial, con cada una de las caras de un mismo color.

## **5.2 *MODELO DE DOMINIO***

La estructura que se modelará se descompondrá en 3 partes muy claras.

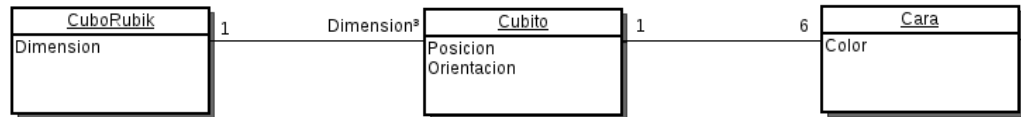
La primera será el cubo de rubik en sí mismo, el cual será único en la aplicación. Éste a su vez puede ser de una determinada dimensión, desde 2x2x2 hasta 7x7x7. La dimensión indicará en cada eje cuantos cubitos tendrá el cubo de rubik y por lo tanto la cantidad de cubitos que tiene cada cara. Estos cubitos son los que se girarán a la hora de realizar los giros de filas. A su vez estos cubitos tendrán 6 caras en total, de las cuales solo 1, 2 o 3 serán del color de toda la cara del cubo de rubik siendo las otras de color negro.

Los cubitos de 3 caras de color son los que se encuentran en los vértices del cubo de rubik y sus 3 caras son las que quedarían hacia el exterior, una para cada una de las 3 caras que unen.

Los cubitos de 2 caras son los que se encuentran en las aristas del cubo de rubik y unen los cubitos de los vértices. Estos cubitos se encuentran entre dos de los lados del cubo de rubik por lo tanto estas caras serán las que son del color de todo ese lado del cubo de rubik.

Por último, los cubitos de 1 cara de color, son los que se encuentran en la parte central de cada uno de los lados del cubo de rubik. Solamente tendrán esa cara del color del lado siendo las otras de color negro.

En la Ilustración 7 podemos apreciar la unión de los 3 objetos principales que tendremos que representar en este proyecto.



*Ilustración 7: Modelo de dominio*



## 6 ANÁLISIS

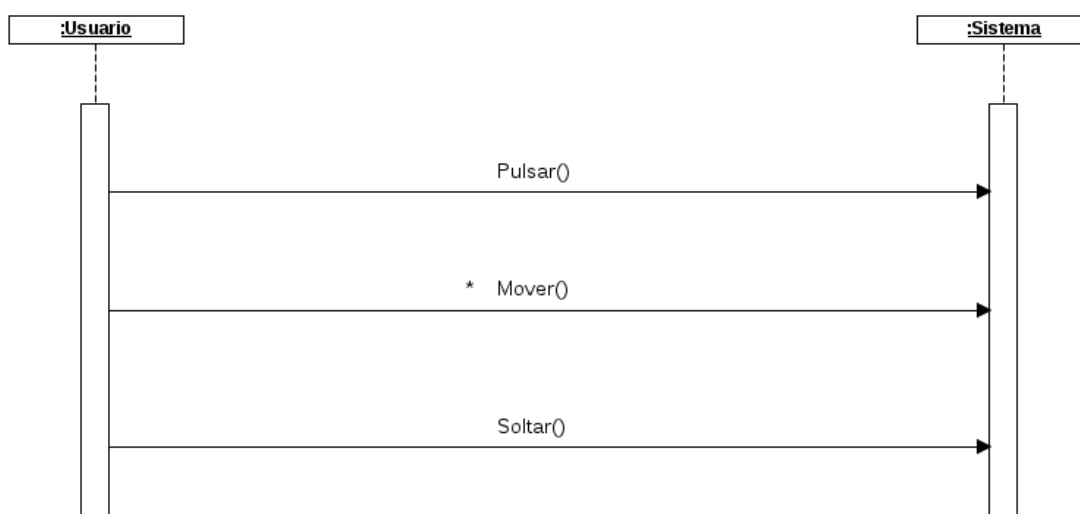
En este apartado se realizará un análisis mas detallado de los casos de uso que hemos obtenido del Capítulo 5.

Mediante el diagrama de secuencia mostraremos los eventos generados por el actor, su orden y los eventos que el sistema realice en respuesta a éstos.

Una vez identificados los eventos que el usuario genera pasaremos a describirlos más detalladamente mediante los contratos. Especificando los cambios que realiza en el sistema y la información mostrada.

### 6.1 CASO DE USO GIRAR FILA

#### 6.1.1 Diagrama de secuencia del sistema



*Ilustración 8: Diagrama de secuencia del caso de uso girar fila*

### 6.1.2 Contratos

- Nombre: Pulsar(x,y)
  - Responsabilidades: Controla el punto en el que se pulsa sobre la pantalla. El cual es indicado mediante los parámetros x e y.
  - Precondición: No debe de estar pulsada la pantalla.
  - Postcondición: Se almacena la posición (X,Y) en la que se pulsa la pantalla.
  - Salida:
- 
- Nombre: Mover(x,y)
  - Responsabilidades: Se arrastra el dedo por la pantalla en una determinada dirección. Cada vez que se desplace el dedo por la pantalla se genera un punto diferente representado por x e y.
  - Precondición: Debe de estar pulsada la pantalla desde un punto anterior.
  - Postcondición: Se girará la fila adecuada en la dirección en la que vaya el movimiento del dedo.
  - Salida: Se muestra el movimiento de toda la fila según la dirección que siga el movimiento del dedo por la pantalla.
- 
- Nombre: Soltar(x,y)
  - Responsabilidades: Al levantar el dedo de la pantalla se debe de ajustar la fila que se está moviendo a la posición más cercana para que el cubo siga manteniendo su forma.
  - Precondición: Se debe de tener el dedo sobre la pantalla.
  - Postcondición: Se ajusta la fila seleccionada a la posición del cubo, se almacenará la posición en la que se encuentran los cubitos del cubo.
  - Salida: Se muestra el estado del cubo con todas las filas ajustadas, el cubo de rubik deberá de seguir manteniendo su forma de cubo.

## 6.2 CASO DE USO VOLVER ESTADO INICIAL

### 6.2.1 Diagrama de secuencia del sistema



*Ilustración 9: Diagrama de secuencia del caso de uso salir*

### 6.2.2 Contratos

Nombre: EstadoInicial()

Responsabilidades: Se encarga de volver al estado inicial del cubo en cualquier momento.

Precondición: La aplicación está arrancada.

Postcondición: Se eliminan todos los movimientos del cubo de rubik volviendo a un estado inicial del mismo con todas las caras del mismo color.

Salida:





## 7 DISEÑO

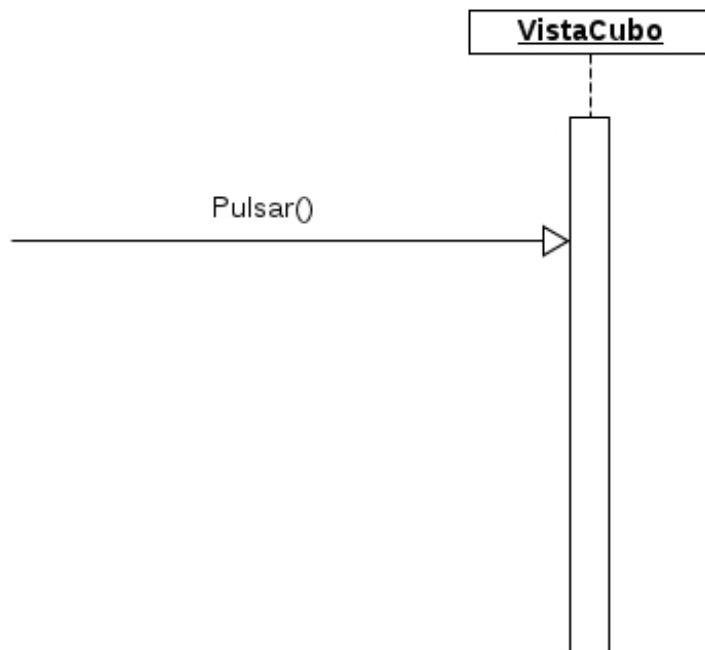
Se procurará obtener un diseño que conserve el encapsulamiento de los objetos consiguiendo un bajo acoplamiento. Las clases serán lo mas sencillas posible para un mejor mantenimiento y mayor facilidad de comprensión. Con esto obtendremos una alta cohesión entre ellas y reduciremos el tiempo para localizar los problemas. Se describirán los diseños y se definirán los diagramas de secuencia de todas las funciones.

### 7.1 *DIAGRAMA DE SECUENCIA*

Los diagramas de secuencia se obtendrán de las funciones que se han conseguido mediante el análisis. Se explicarán una a una y se mostrará su diagrama.

#### 7.1.1 Pulsar(x,y)

Es la primera de las acciones que realizaremos en un uso normal de la aplicación. Al pulsar la pantalla se podrá obtener la posición x e y de la zona de la pantalla sobre la que hemos hecho la pulsación y sabiendo esto, esta función podrá calcular sobre qué fila y a qué cara pertenece esa fila en la que se ha realizado la pulsación. Esto nos ayudara en el siguiente paso, el de Mover, para decirle cuál ha sido la posición del primer toque. Siguiendo el modelo vista controlador esta acción la realizara la vista de la aplicación (Capa de presentación), se observa su esquema en ilustración 10.



*Ilustración 10: Diagrama de secuencia de la función `Pulsar(x,y)`*

### 7.1.2 Mover(x,y)

Para que se pueda llegar a esta función previamente se tiene que haber pasado por la de Pulsar(). Por lo tanto ya tendremos cargadas la cara y la línea sobre la que se pulsó previamente desde el diseño Pulsar(). Hay que tener en cuenta que la acción Mover se va a repetir tantas veces como el usuario mantenga el dedo sobre la pantalla.

Para controlar la primera pulsación de este conjunto de movimientos utilizaremos el booleano Ajustar. Si es el primer movimiento que se realiza, esto es, Ajustar esta en True, habrá que realizar un AjustarReferencias. Esto sirve para Ajustar al cubo la fila que se va a mover y de esta forma poder saber su ángulo inicial. Se realiza un getCubitoReferenciado sobre el cubo (0,0, línea) siendo (x,y,z) para obtener el cubito uno de los cubitos sobre los que vamos a realizar el giro. Se podría hacer sobre todos los cubitos que se van a girar pero el resultado sería el mismo, por lo tanto para tener una mejor eficiencia solamente se realiza sobre uno cualquiera de todos. Una vez se tiene el cubito se calcula el ángulo más cercano al que se encuentra el cubito (0, 90, 180 o 270 grados). Estos ángulos son los que harían encajar al cubito en la estructura del cubo (para que siga formando un cubo el conjunto de cubitos). Tras esto simplemente hay que añadirles el ángulo correspondiente mediante la función AnadirAngulo.

Por otro lado cada vez que se realice esta acción se girará gráficamente el cubo, por lo tanto habrá que añadirle el ángulo de giro actual a todos los cubitos de la fila que se está girando, para que se pueda ver la animación mientras se está moviendo la fila. Para ello llamamos a la función GirarReferenciasGL, la cual a todos los cubitos de esa fila (i,j, fila) les realizará un añadido de su ángulo de giro sumándole la diferencia. La diferencia en este caso es la distancia que hay desde que pulsamos la pantalla hasta donde tenemos el dedo actualmente. Solamente necesitaremos consultar a los objetos Cubo y Cubito ya que serán los que almacenen las variables que necesitaremos cambiar para realizar el movimiento.

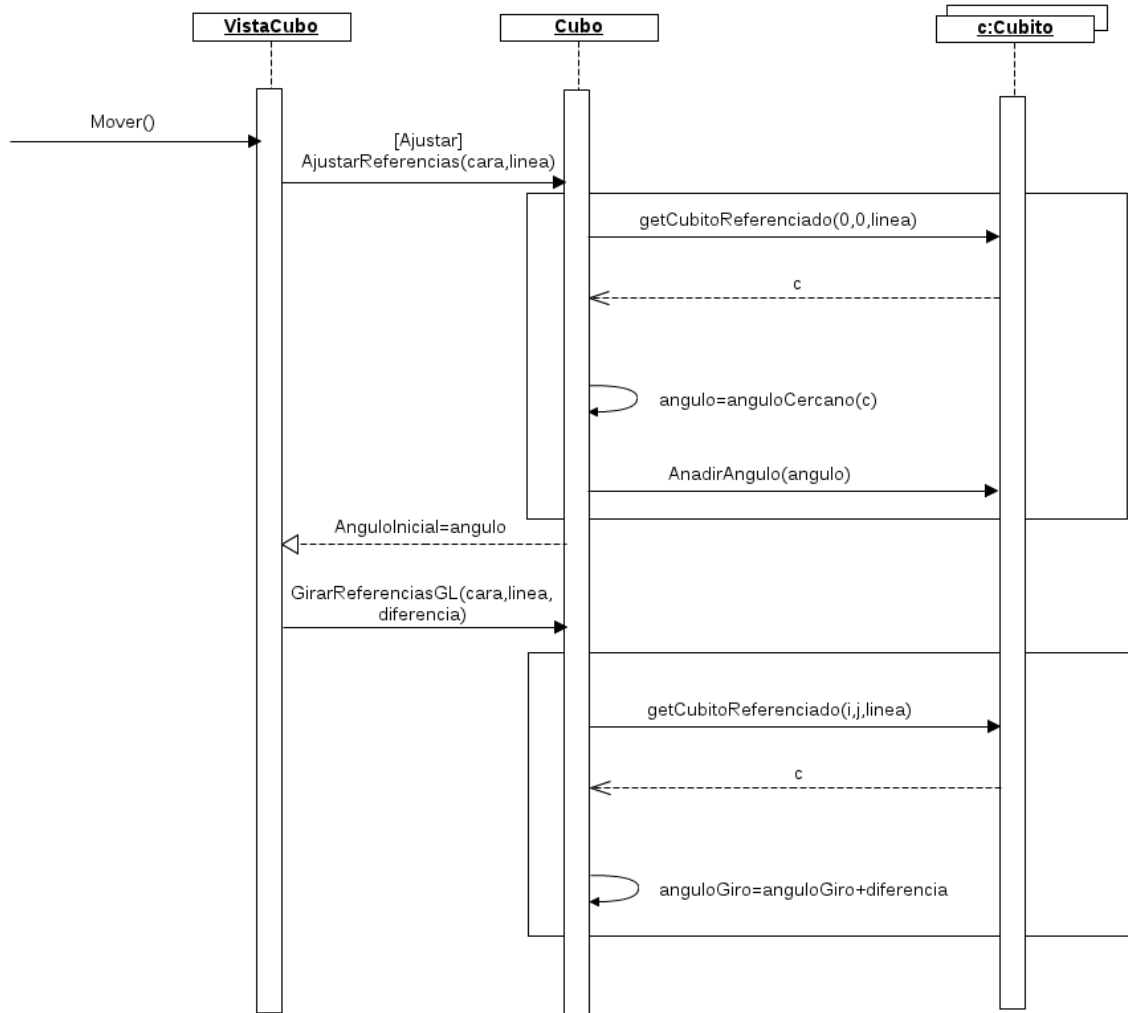


Ilustración 11: Diagrama de secuencia de la función  $Mover(x,y)$

### 7.1.3 Soltar(x,y)

Por último, una vez realizada la pulsación sobre la pantalla y desplazado el dedo sobre ella tanto como se quiera, se procederá a levantar el dedo de la pantalla. En cuanto se realice esta acción lo primero que habrá que hacer es ajustar las referencias de todos los cubitos de la fila que se estuviese moviendo en ese momento. Por ello se llama a la función de AjustarReferencias, lo cual calculará cual es el ángulo más cercano a la posición actual y girará todos los cubitos hasta ese ángulo. En este momento el cubo de rubik deberá de volver a tener otra vez la forma de un cubo.

Todo esto puede haber hecho cambiar la estructura del cubo, o sea, que los cubitos lo más probable es que hayan cambiado de posición unos con respecto de otros. Por lo cual hay que situar a los cubitos en su posición correspondiente. Para ello llamamos a la función del Cubo llamada GirarReferencias que realizará este giro lógico sobre la estructura de datos.

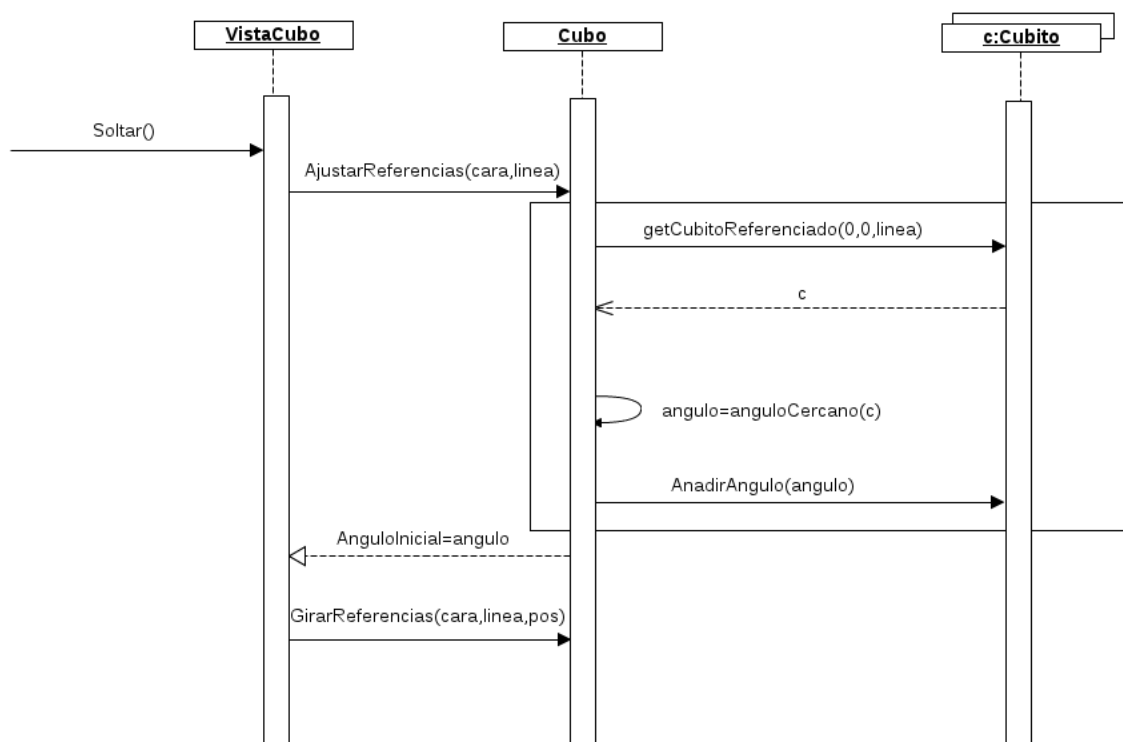
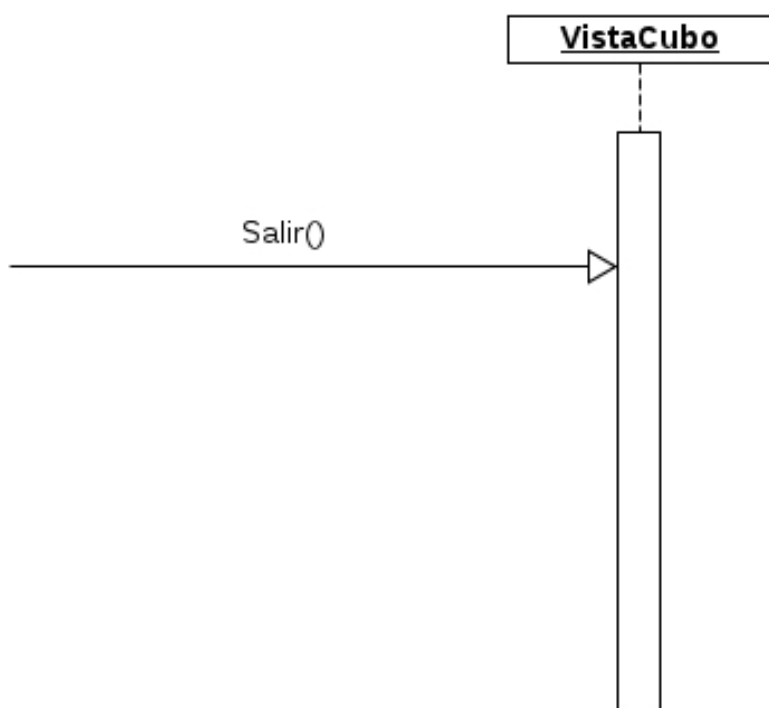


Ilustración 12: Diagrama de secuencia de la función Soltar(x,y)

#### 7.1.4 EstadoInicial()

En cualquier momento del programa se puede pulsar la tecla de Back con la que cuentan los dispositivos Android. Este evento será captado por la clase VistaCubo la cual se encargará de deshacer todos los movimientos realizados sobre el cubo de rubik. El cubo debería de volver al estado inicial con todas las caras de su color correspondiente.



*Ilustración 13: Diagrama de secuencia de la función Estado Inicial()*

## **8 IMPLEMENTACIÓN**

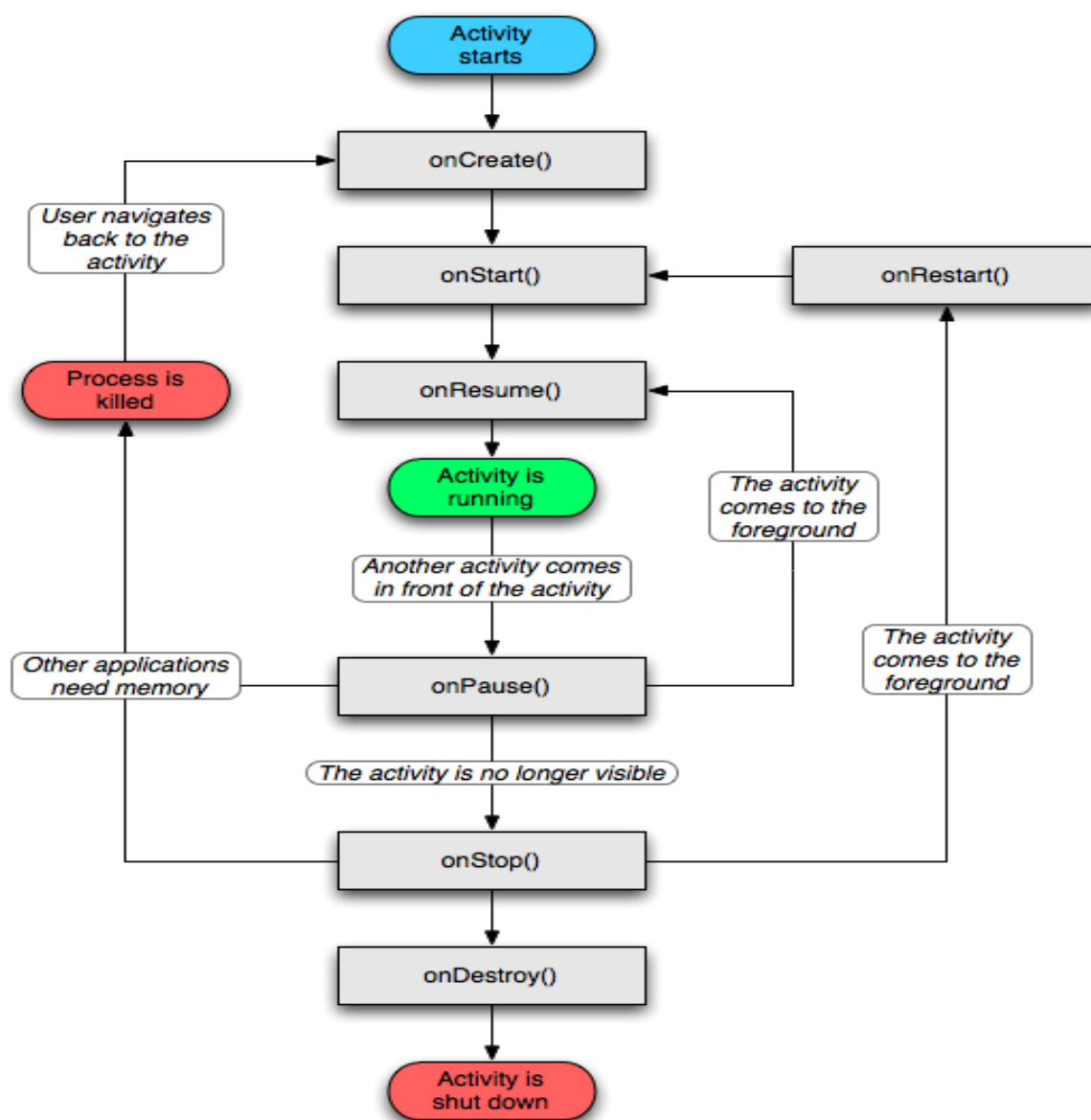
En este capítulo se detalla la forma en la que se decidió desarrollar la aplicación. La descripción de las clases implementadas, las estructuras utilizadas, sus funcionalidades y demás aspectos para la consecución del producto final. También se mostrará el diagrama de todas las clases utilizadas para la consecución de este proyecto.

### **8.1 MANUAL DE IMPLEMENTACIÓN**

La forma de implementar la aplicación se puede dividir en dos partes. Por un lado estarían las clases que se encargan de gestionar las opciones de la aplicación, serían unas clases que nos servirían de enlace entre el dispositivo y la aplicación. Y por otro lado las clases destinadas a modelar el programa. Todas las clases que se detallarán a continuación vienen recogidas de forma ordenada en la ilustración 15. En el primer lugar se detallarán las tres primeras clases implementadas: CuboRubik, VistaCubo y RenderizadoCubo en el apartado de preparación de las clases de OpenGL + Android. A continuación se detallarán todas las demás clases utilizadas para modelar el cubo de rubik. Tras esto se explicarán las principales estructuras de datos utilizadas por estas clases para gestionar las relaciones entre todas estas clases mediante objetos.

#### **8.1.1 Preparación de las clases de OpenGL + Android**

Se parte de la base de que la aplicación será para dispositivos Android, por lo cual hay que tener en cuenta el funcionamiento de las aplicaciones de Android. El ciclo de vida de las aplicaciones Android sigue el esquema de la ilustración 14 (<http://droideando.blogspot.com/2011/02/ciclo-de-vida-de-una-aplicacion-android.html>).



*Ilustración 14: Ciclo de vida de las aplicaciones de Android*

De todo el esquema para el desarrollo de esta aplicación nos interesa especialmente el evento `onCreate()`. Ya que será el evento con el que comience nuestra aplicación. Para que nuestra aplicación pueda ejecutarse correctamente deberíamos de crear una clase que extienda a la clase `Activity` del paquete `Android.app` la cual nos hará heredar las funciones `onCreate()`, `onResume()` y `onPause()`. Esta clase será la denominada `CuboRubik`, que sería por así decirlo la que llevaría el `main` del programa.

Ahora será necesario controlar los eventos del dispositivo, tales como las pulsaciones en la pantalla o los botones. Como se toma la decisión de que la capa de presentación de la aplicación será tramitada por `OpenGL` se utilizará una clase de `OpenGL` que se encarga de ofrecernos esas características. Esta será la clase `GLSurfaceView` del package `Android.opengl`.



Esta clase la deberemos de extender y nos ofrecerá diferentes tipos de funciones que nos permitirán la detección y gestión de todos los eventos que le lleguen al dispositivo. Para extender esta clase se utiliza la clase VistaCubo que será nuestro intermediario entre usuario y programa.

Todavía nos falta una última clase para poder comenzar a modelar nuestro programa mediante OpenGL. Necesitamos un renderizador, que se encargue de convertir lo que modelemos en 3 dimensiones a las 2 dimensiones en las que nos lo mostrará la pantalla del dispositivo. Para ello creamos la clase RenderizadoCubo que extenderá a GLSurfaceView.Renderer del package de OpenGL. Esta clase se encargará de pintar en la pantalla todo lo que nosotros deseemos. Aquí también definiremos todas las opciones de OpenGL, posicionaremos la cámara desde la que visualizaremos el cubo, definiremos el fondo... Al extender heredaremos tres funciones: onDrawFrame(), onSurfaceChanged() y onSurfaceCreated(). Estas funciones son eventos que se ejecutarán en determinados momentos. OnDrawFrame() se ejecutará cada vez que se dibuje una pantalla, onSurfaceChanged() cada vez que se cambie de orientación la pantalla (de horizontal a vertical y viceversa) y onSurfaceCreated() en cuanto se cree la primera pantalla (ideal para crear los objetos).

### 8.1.2 Clases del Cubo de Rubik

Ahora es la parte de modelar el cubo de rubik. Lo primero es que necesitamos tener en cuenta que vamos a dividir en dos capas nuestro programa: presentación y negocio. Las clases de negocio se encargarán de cargar los datos del objeto en cuestión y de ofrecer funciones para trabajar con ellos, mientras que las de presentación se encargarán de ofrecer funciones que tengan que ver con la forma en la que se dibujará en pantalla el objeto. Para diferenciar unas de otras, las clases que comiencen por GL serán las de la capa de presentación y las demás de la parte de negocio.

Se comienza a implementar desde arriba hacia abajo, esto es, desde lo más general hasta lo más específico. Por lo tanto se comienza implementando lo que sería el cubo de rubik en sí. Esto englobaría todas las funciones más generales a las que la parte de el usuario llamaría. Por ejemplo, una función importante sería la de girar fila, ya que es una de las cosas que el usuario puede realizar con el cubo de rubik. Esta clase será extendida por la clase GLCubo que será la representación gráfica del cubo de rubik. En sí esta clase lo único que realizará a la hora de dibujarse, es dibujar tantos cubitos como sean necesarios.

El cubo a su vez está compuesto por cubitos, los cuales serán los objetos que giren, ya que el cubo de rubik no debería de girar. Uno de los aspectos más importantes que deberán de controlar los cubitos será su posición, esto es, el giro que ha realizado. También cuenta con su clase de representación gráfica, llamada GLCubito, que se encargará de dibujarlo.

Como parte final, se hace una última descomposición de los cubitos. Estas serán las caras. Cada cubito contara con 6 caras que dependiendo cual sea el cubito en cuestión, serán 2 o 3 de uno de los colores del cubo de rubik y todas las demás negras. De igual manera que los anteriores objetos éste también contará con una clase que se encargue de dibujarlo, llamada GLCara. La parte más delicada de este objeto es la función que se encarga de pintarlo en pantalla. En este momento nos encontramos con el primer problema. Como la versión que los dispositivos Android incluyen, no es la completa, sino una versión reducida, llamada OpenGL ES, esta versión solamente nos permite dibujar triángulos. Por lo cual para dibujar una cara

de un cubo (un cuadrado) se necesitará dibujar mediante dos triángulos que tengan la hipotenusa común a ambos, y que sus lados catetos tengan todos la misma longitud, para que formen un cuadrado.

### 8.1.3 Estructuras de Datos

Una vez descritas las clases y sus características generales se explicará cómo se van a organizar los objetos, las decisiones que se toman para almacenarlos en diferentes estructuras y sus motivos.

Siguiendo el orden anterior, desde arriba hacia abajo, la clase cubo va a tener dos estructuras destacables. La primera es un array tridimensional del tamaño de la dimensión del cubo que contendrá todos los objetos cubitos que tiene el cubo, en total serán un total de  $\text{dimensión}^3$ . Sobre este array no se realizará ninguna gestión, por lo tanto se podría haber usado un vector pero como un array tridimensional se parece más a lo que en la realidad se quiere representar, se decidió hacerlo así. Además, de esta forma permite saber cual es la posición inicial en la que estaba el cubo al comenzar, ya que la posición que cada cubo ocupa dentro del array es la inicial.

Por otro lado se crea otro array tridimensional llamado `ArrayReferenciaCubitos`. Cada posición de este array almacena un objeto simple que solamente contiene 3 variables (x,y,z). Este array es de la misma dimensión que el anterior y de lo que se encarga es de señalar la posición de los cubitos. Al inicializar, cada uno apuntará a la posición de su cubito correspondiente del array de cubitos. Con este array sera con el que se realicen los giros lógicos. Al girar una fila se girará dentro de este array, cambiando también los objetos y por lo tanto la referencia a los cubitos. De esta forma se podrá controlar la posición de cada cubito.

En la clase Cubito nos encontramos con el mismo problema anterior. Necesitamos almacenar las caras del cubito. Por lo tanto se crea un vector de tamaño 6 que almacenará todas estas caras. Según la posición de éstas, se sabe qué cara es:

0-Frontal

1-Izquierda

2-Atrás

3-Derecha

4-Abajo

5-Arriba

En la clase Cara se utilizarán vectores para almacenar los vértices de las caras, ya que OpenGL para dibujar los triángulos, necesita que los vértices sean indicados de esta forma.

## 8.2 ALTERNATIVAS DE IMPLEMENTACIÓN

En este apartado se estudiará la información que se ha encontrado, así como otras posibles formas de resolución de los problemas que ha planteado la implementación del proyecto.

Como se ha explicado anteriormente, la información en la red sobre el tema de Android es inmensa. Una de las ventajas de Android es que al ser una comunidad muy grande, hay mucha

gente que desarrolla aplicaciones, tanto para uso personal como comercial. De esta forma hay una gran variedad de manuales en internet. La única pega que podía tener era que está casi todo en inglés, pero no a sido un gran problema ya que el alumno tiene bastante facilidad con este idioma. La implementación se ha realizado siguiendo distintos manuales y ejemplos encontrados, por lo cual podría haberse hecho de maneras diferentes.

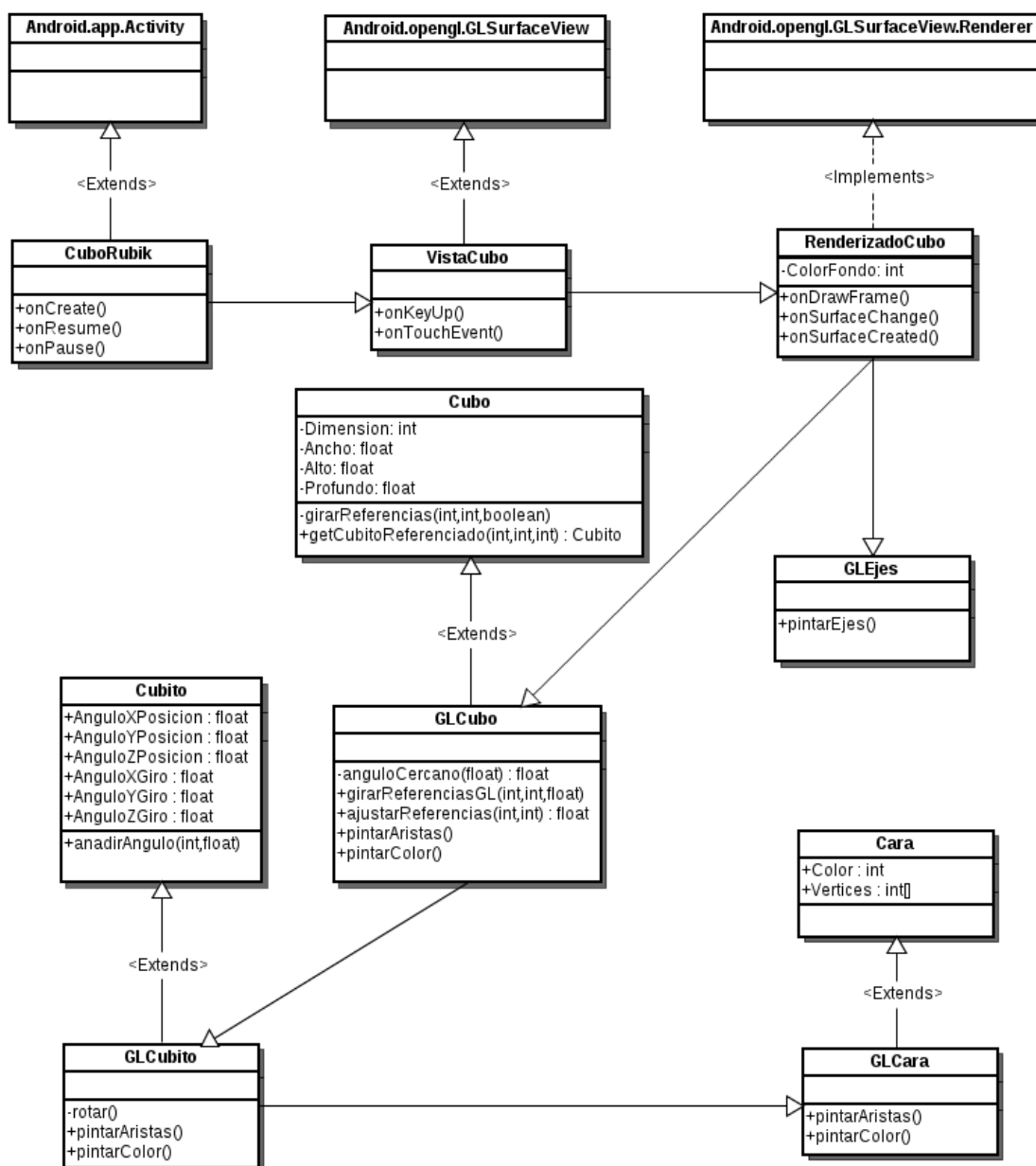
Al haber tantísima información, surgía el problema de que muchas veces era bastante difícil encontrar la solución a un problema específico, y más sobre algoritmos para, por ejemplo los, giros del cubo, ya que en las búsquedas la mayoría de resultados solían ser sobre el algoritmo para la resolución del cubo de rubik, y no lo que interesaba.

Otro problema es que la mayoría del código no estaba comentado o estaba muy poco comentado y era bastante costoso seguir la línea de lo que se estaba realizando, y más cuando se usaban clases o funciones que no se conocían, ya que la API de Android es inmensa. Por esta razón, leer código se hacía bastante pesado y se tardaba muchísimo tiempo en conseguir entender pequeñas partes de código.

Se llegó a encontrar el código de un programa que hacia algo parecido al cubo de rubik, que giraba filas y columnas. Este programa se analizo bastante pero la forma de almacenar los objetos los hacía todos de forma individual, esto es, creaba objetos cada uno con un identificador y a la hora de girarlos los cambiaba según su nombre. Esta idea no parecía correcta y por eso en este proyecto se utiliza un array donde se almacenan los objetos y se mueve según la posición en la que esté, independientemente de qué cubito sea. De esta forma además, se permite que el numero de cubitos varíe, pudiendo crear cubos de rubik de N cubos por lado en un futuro.

En cuanto al tema de OpenGL para la implementación se han seguido unos cuantos tutoriales que estaban bastante bien explicados (Ref. [4], Ref. [9]) algunas incluso en castellano. Con estos tutoriales se consiguieron las nociones básicas sobre OpenGL para poder comenzar con la realización del proyecto. También existen bastantes manuales en internet sobre OpenGL detallando todas sus funcionalidades, aunque casi todo es sobre su versión completa, la que usan los ordenadores. El problema que existía al principio era que al ser para dispositivos Android la versión que traen de OpenGL no es la versión completa sino una reducida. Esta versión difería algo en la completa, especialmente a la hora de dibujar los objetos.

### 8.3 DIAGRAMA DE CLASES



*Ilustración 15: Diagrama de clases*

Todas las clases descritas anteriormente son las plasmadas en la Ilustración 15. Se puede ver claramente como se ha separado en dos capas la aplicación y cuales son las clases de preparación del programa para poder funcionar en Android mediante OpenGL y cuales las que representan el dominio del programa. También se aprecian todas sus relaciones entre unas y otras explicadas anteriormente, así como sus funciones y variables más importantes.

## **9 PRUEBAS**

Tras la implementación se han realizado unas determinadas pruebas sobre el programa para comprobar el correcto funcionamiento del mismo.

Se intenta buscar casos de pruebas que tengan altas probabilidades de hacer fallar el programa para una depuración correcta del mismo. Se prueba cualquier parte del programa tanto botones como movimientos en la pantalla o diferentes tipos de configuraciones.

Aún así todas estas pruebas no garantizan un programa perfecto, ya que nos encontramos con un programa que tiene infinitos movimientos por lo que nunca se podrían llegar a probar todas las posibilidades. Por esta razón todos los proyectos deberían de tener un proceso después de su finalización que se encargue de corregir los posibles errores que puedan surgir.

### **9.1 PRUEBAS DE CAJA NEGRA**

Este tipo de pruebas nos permiten encontrar errores en la asignación de responsabilidades y la interfaz gráfica de la aplicación. Se prueban los botones y las posibilidades que ofrece el programa.

#### **9.1.1 Pulsar el botón “Back”**

Este botón debe de permitirnos salir del programa en cualquier momento. El programa se cierra y la única forma de volver al programa es ejecutándolo de nuevo. Así mismo también realiza el caso de uso de volver a estado inicial, por lo cual al volver a entrar al programa el cubo de rubik vuelve a iniciarse desde el principio. En un principio esta prueba al salir mantenía el estado del cubo y al volver a iniciar el programa el cubo estaba tal y como lo habíamos dejado en el uso anterior. Se corrige este fallo evitando que la aplicación se quede en modo zombie.

#### **9.1.2 Pulsar el botón “Home”**

El botón home permite perder el foco del programa (parecido a minimizarlo en un PC) y de esta forma volver a la pantalla principal de Android. Podríamos volver al estado en el que dejamos el programa. Se debe de conservar el estado en el que dejamos el programa antes de pulsar este botón. El funcionamiento de este botón es correcto y no es necesario ningún cambio en el programa.

#### **9.1.3 Pulsar otros botones**

Todos los demás botones del dispositivo no deben de realizar ninguna acción en especial con el programa. Aunque por ejemplo, botones como subir o bajar volumen deben de realizar su función normal pero sin afectar el funcionamiento del programa. Se verifica un correcto funcionamiento de todos los botones y se comprueba que no existe ningún problema con ellos.

#### **9.1.4 Un Toque en la Pantalla**

Se tiene que poder tocar la pantalla en cualquier parte de ella. Esto, en principio, no debería de

realizar ninguna acción a no ser que se desplace el dedo. Como es prácticamente imposible tocar con el dedo todas las posiciones de la pantalla ya que la pantalla del dispositivo en el que se realizan las pruebas es de 480x854 pixels se utiliza el programa durante mucho tiempo a posiciones al azar y ninguna da problemas.

### **9.1.5 Girar una fila desplazando el dedo**

Una vez tocada la pantalla se tiene que permitir desplazar el dedo, esta acción debe de girar la fila sobre la que se ha pulsado inicialmente en la dirección y sentido en el que se desplace el dedo sobre la pantalla. Hay que tener en cuenta a la hora de realizar esta acción, que el sentido en el que se realizará el movimiento es el que la pantalla lo detecta, y para una persona puede tener un poco de dificultad poder representar con el movimiento en la pantalla el sentido en el que el desea que sea. Con esta prueba se detecta que el desplazamiento horizontal cuesta un poco y se mejora.

### **9.1.6 Girar varias filas**

Una vez girada una fila se comprueba que esta acción se puede repetir un número indefinido de veces en cualquier dirección y en cualquiera de las filas visibles. Siguiendo en todo momento los movimientos que realiza un cubo físico normal. Se comprueba que el funcionamiento es correcto.

## **9.2 PRUEBAS DE CAJA BLANCA**

Estas pruebas se realizan analizando y ejecutando el código. Permitirán detectar errores de flujo de control y caminos de tratamiento de errores.

### **9.2.1 Correcta visualización**

Esta prueba prueba que el cubo de rubik se vea al completo en toda la pantalla, que se puedan visualizar las 3 caras correctamente y que todo lo mostrado en pantalla es correcto. Cualquier fallo debería de ser responsabilidad del dibujado en pantalla.

### **9.2.2 Capa por Capa**

Como se ha definido una estructura de capas se pueden realizar pruebas a cada capa independientemente de las demás, por ejemplo, se podría testear que el cubo gira correctamente simplemente en la capa de negocio, sin necesidad de ver nada por pantalla, visualizando el estado del array mediante el debugger. Estas pruebas han sido utilizadas durante todo el desarrollo del programa y funciona correctamente.

## **9.3 OTRAS PRUEBAS**

### **9.3.1 Pruebas de Implantación**

Se debe de probar el correcto funcionamiento de la aplicación en diferentes dispositivos, con diferentes tipos de hardware. También deberían de tener diferentes versiones de software.

Como testarlo en todos los dispositivos del mercado y con todas sus versiones es una tarea prácticamente imposible, se prueba en varios dispositivos diferentes. Esta prueba está explicada más detalladamente en el capítulo de Implantación.

### **9.3.2 Pruebas de Configuración**

También se comprueba con diferentes configuraciones. Por ejemplo, en el modo avión de los dispositivos, con el brillo muy bajo en la pantalla, desactivando internet... Ninguna de las configuraciones da ningún problema al programa.





## **10 IMPLANTACIÓN**

### **10.1 ESPECIFICACIONES**

La aplicación se ha programado utilizando la API de Android en su versión 7, lo cual permite que se pueda ejecutar en dispositivos con versión de Android 2.1 o superior. La única característica especial que tiene la aplicación es que usa OpenGL, lo cual viene incorporado en la API de Android desde la versión 3 (Android 1.5). Esta versión es la más antigua con la que vienen incorporados los dispositivos, ya que es la que compró Google en su momento. En cuanto a tema de hardware en principio solo tendría el requisito de contar con pantalla táctil. Ya que cualquier dispositivo que sea capaz de correr Android 2.1 sería capaz de poder utilizar la aplicación.

### **10.2 COMPILACIÓN**

La compilación de la aplicación se realiza mediante eclipse. El SDK nos generara un archivo de extensión .apk con el nombre del proyecto. Este archivo sera el que podremos copiar para luego instalarlo en los dispositivos que deseemos. Este archivo contendrá el manifiesto de Android, los binarios y los archivos necesarios, tales como imágenes, sonidos etc...

### **10.3 INSTALACIÓN EN DISPOSITIVOS**

La instalación de la aplicación en cualquier dispositivo esta más detalladamente explicada en el apéndice de la guía de instalación para usuario. Este proyecto a sido probado en 6 dispositivos móviles Android diferentes y en los 6 ha funcionado a la perfección sin ningún problema:

- Motorola Defy con Android 2.1 y Android 2.2
- Samsung Galaxy S con Android 2.1, Android 2.2 y Android 2.3
- Samsung Galaxy S SCL con Android 2.2 y 2.3
- Samsung Galaxy S II con Android 2.3
- Sony Ericsson Xperia Neo V con Android 2.3
- HTC WildFire con Android 2.2

Todos ellos cumplían con los requisitos y se veía la aplicación en la pantalla a la perfección, a pesar de que cada uno tiene un tamaño de pantalla y resolución distinta. Estaría bien haberlo probado en algún tablet y netbook para comprobar su total funcionamiento con cualquier tipo de dispositivo, pero no contábamos con acceso a ningún dispositivo de este tipo. Aunque si que se han probado en la máquina virtual con las especificaciones de tablet y netbook y la aplicación funcionaba perfectamente.



## 11 GESTIÓN

Desde el momento en el que se aprobó el proyecto se sabía que iba a ser un proyecto difícil y largo. Que iba a tener muchos problemas y que su desarrollo sería complejo. Prueba de ello ha sido la imposibilidad de acabar el proyecto para la fecha establecida en un principio (Septiembre de 2011). Por lo cual el proyecto se tuvo que alargar hasta la siguiente convocatoria de Febrero de 2012.

En la primera parte del desarrollo, desde Noviembre hasta Febrero, se tuvieron bastantes problemas para poder seguir la planificación aunque al final se consiguieron alcanzar los objetivos en los plazos indicados. A partir de ahí las cosas se empezaron a complicar más aún. Con el comienzo del segundo cuatrimestre la carga de las demás asignaturas del alumno imposibilitaron seguir con el ritmo llevado hasta el momento. Por lo que se decidió dejar apartado el proyecto, de momento, mientras se le dedicaba más tiempo a otras asignaturas. En ese momento se vio que iba a ser imposible conseguir los plazos y la segunda iteración planificada se pospuso para después de los exámenes ordinarios del segundo cuatrimestre. Por este motivo se ha realizado una segunda planificación siguiendo lo establecido en el plan de contingencia.

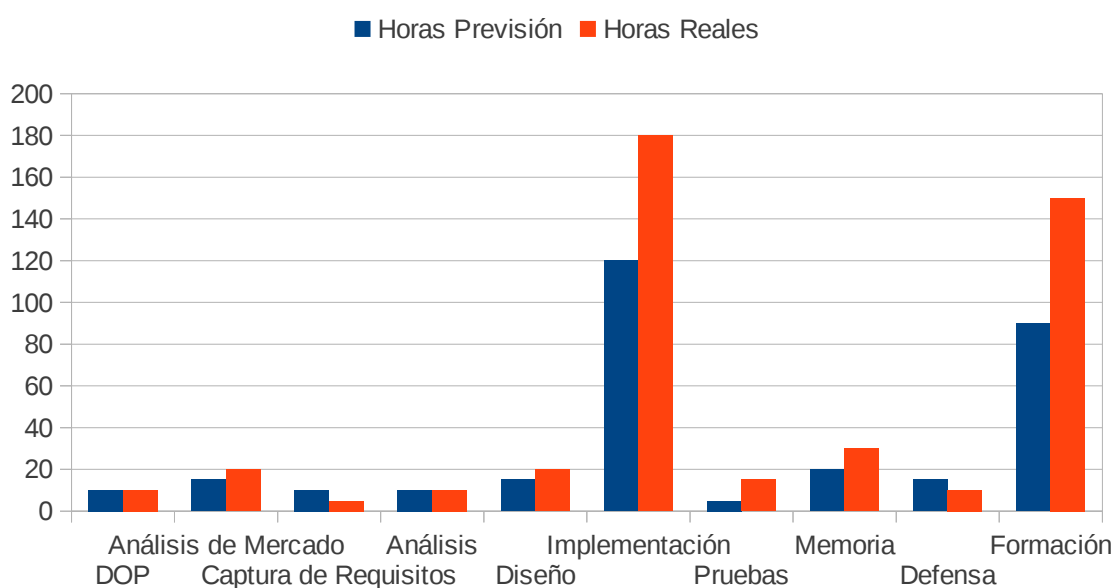
La segunda iteración comenzaría para principios de Julio y se le dedicaría todo el verano hasta principios de Octubre. De esta forma la tercera iteración debería de comenzar tras la finalización de la segunda, y finalizaría para mediados de Enero. Así pues se debería de llegar a la fecha de entrega de la memoria del 27 de Enero con todo finalizado.

Con esta nueva planificación y gracias a que el alumno no suspendió ninguna asignatura se contaba con total disponibilidad para el desarrollo del proyecto. Por lo cual adaptarse a los plazos no fue un gran problema en estas dos últimas iteraciones.

En cuanto a las horas invertidas se ha necesitado invertir bastantes más de las previstas, especialmente en la primera fase para alcanzar los objetivos a tiempo. En un principio se calcularon un total de cerca de unas 300 horas lo que al final ha resultado en más de 400 horas para todo el proyecto. Muchas de ellas fueron causadas por la inexperiencia en el campo de dispositivos Android y en gráficos en 3 dimensiones.

A continuación se puede observar como algunas previsiones del tiempo de las tareas si que se consiguieron, mientras que en otras, como en la Implementación o en la formación, ni se acercaron a la previsión. Es por lo cual que el número de horas al final fue bastante mayor del esperado.

	Horas Previsión	Horas Reales
DOP	10	10
Análisis de Mercado	15	20
Captura de Requisitos	10	5
Análisis	10	10
Diseño	15	20
Implementación	120	180
Pruebas	5	15
Memoria	20	30
Defensa	15	10
Formación	90	150
TOTAL	310	450



*Ilustración 16: Gráfico Horas Previsión vs Horas Reales*

De esta misma forma como los plazos de proyecto cambiaron en la segunda iteración, el Gantt real es bastante diferente al previsto. En la ilustración 17 se muestra en la parte superior la previsión realizada en el DOP y en la parte inferior la duración del proyecto real.

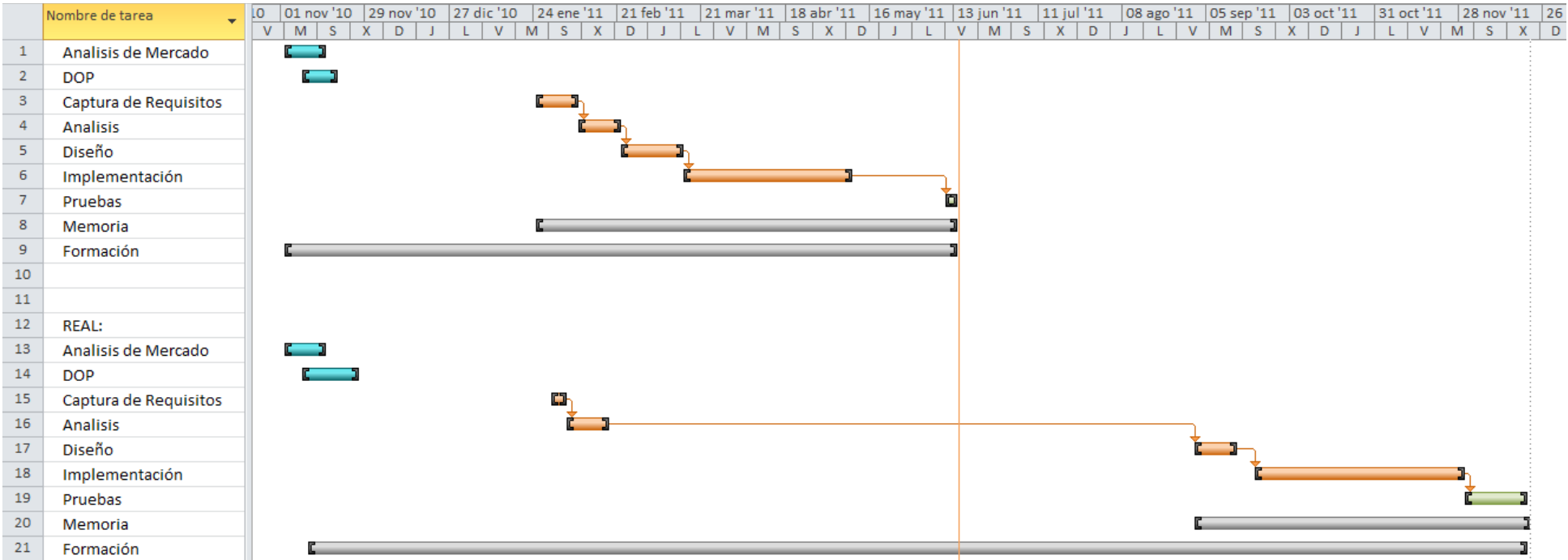


Ilustración 17: Diagrama de Gantt previsión vs realidad

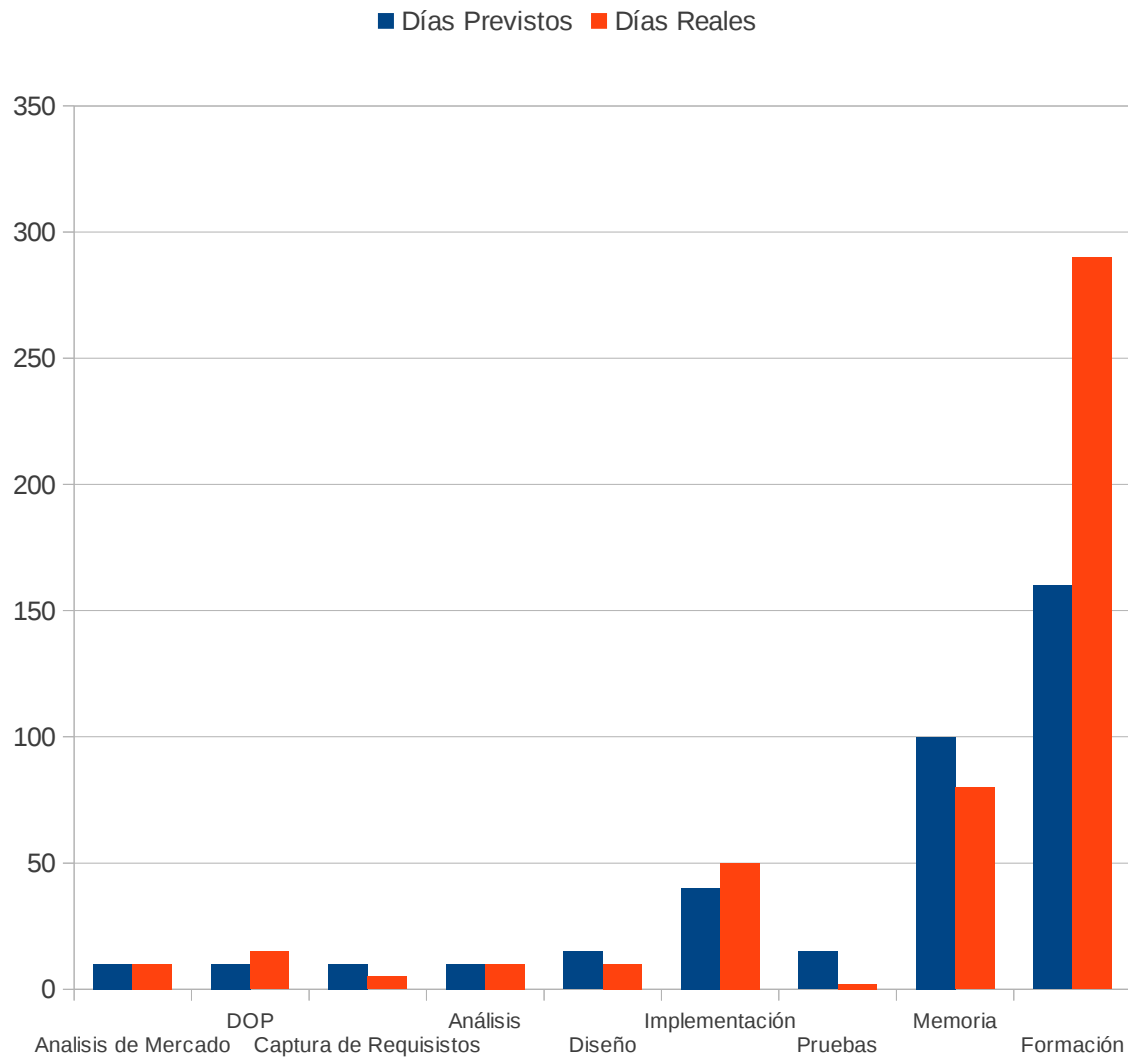
Como se observa claramente el proyecto se terminó alargando el doble de tu tiempo de duración estimado, a causa del parón en el segundo cuatrimestre, aunque los tiempos previstos de duración de cada tarea no son muy diferentes de los previstos. En la ilustración 18 podemos observar la comparativa de los tiempos de duración de cada tarea.

	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1	Analisis de Mercado	10 días	mar 02/11/10	lun 15/11/10	
2	DOP	10 días	lun 08/11/10	vie 19/11/10	
3	Captura de Requisitos	10 días	mar 25/01/11	lun 07/02/11	
4	Analisis	10 días	mar 08/02/11	lun 21/02/11	3
5	Diseño	15 días	mar 22/02/11	lun 14/03/11	4
6	Implementación	40 días	mar 15/03/11	lun 09/05/11	5
7	Pruebas	2 días	vie 10/06/11	lun 13/06/11	6
8	Memoria	100 días	mar 25/01/11	lun 13/06/11	
9	Formación	160 días	mar 02/11/10	lun 13/06/11	
10					
11					
12	REAL:				
13	Analisis de Mercado	10 días	mar 02/11/10	lun 15/11/10	
14	DOP	15 días	lun 08/11/10	vie 26/11/10	
15	Captura de Requisitos	5 días	dom 30/01/11	jue 03/02/11	
16	Analisis	10 días	vie 04/02/11	jue 17/02/11	15
17	Diseño	10 días	jue 01/09/11	mié 14/09/11	16
18	Implementación	50 días	mié 21/09/11	mar 29/11/11	17
19	Pruebas	15 días	mié 30/11/11	mar 20/12/11	18
20	Memoria	80 días	jue 01/09/11	mié 21/12/11	
21	Formación	290 días	mié 10/11/10	mar 20/12/11	

*Ilustración 18: Duración de las tareas previstas vs reales*

Se aprecia claramente que exceptuando la memoria y la formación que dependían de la duración total del proyecto, las demás tareas duran prácticamente lo mismo que la previsión. Esto también se debe a que el alumno tuvo bastante en cuenta este apartado a la hora de la realización del proyecto, intentando ajustarse a el y en muchos casos dedicando más horas al día de las que en un principio se preveían.

Mediante la ilustración 19 se aprecia más claramente este detalle.



*Ilustración 19: Gráfico de horas de las tareas previstas vs reales*





## **12 CONCLUSIONES**

### **12.1 OBJETIVOS LOGRADOS**

El objetivo principal del proyecto era crear una base sobre la que se podría más tarde crear un juego completo. Incluso este juego podría tener cabida en el Android Market ya que en este momento no hay ninguna aplicación con las características de este proyecto. Por lo tanto el objetivo se ha cumplido y estoy satisfecho. El resultado ha sido un éxito ya que aparte de que funciona, estéticamente ha quedado bonito gracias a los gráficos 3D y teniendo en cuenta que no utiliza texturas.

En cuanto a Android la verdad es que me ha gustado la forma de implementar las aplicaciones. Los primeros pasos sobre todo, son muy lentos y cuestan muchísimo, pero en cuanto vas entendiendo el sistema se pueden crear programas de gran potencia. OpenGL me ha parecido aún más difícil que Android ya que tiene infinidad de opciones y llegar a controlarlas en ocasiones resulta muy costoso.

### **12.2 VALORACIÓN PERSONAL**

Me siento muy satisfecho de la realización de este proyecto. Sinceramente pensé que no iba a ser tan complicado, pero la verdad es que para alguien que nunca ha trabajado con gráficos, y mucho menos en 3 dimensiones y que tiene que funcionar en un dispositivo del que se partía sin saber nada, hizo que las cosas no fuesen fáciles. Todos los conocimientos que he adquirido con este proyecto han sido de forma autodidacta, ya que no había recibido ninguna clase previa de nada que tuviera que ver con el tema que aquí se tocaba.

Por todo esto uno se siente orgulloso de haber llegado a este punto. Por todas las horas invertidas y por haber conseguido lo que en un principio se marcó que se realizaría.

En resumen me siento muy feliz por haber finalizado este proyecto de la forma que lo he hecho. Está claro que este tipo de aplicaciones es una apuesta segura para el futuro. Si pudiese volver atrás y elegir otro proyecto, no lo cambiaría por ninguno.

### **12.3 TRABAJOS FUTUROS**

El estado de la aplicación, tras finalizar este proyecto, es simplemente una demo. Solamente está implementado el núcleo del programa, el motor de movimientos. Un siguiente paso sería el que realmente se convierta en un juego. Para esto sería interesante añadirle un menú, con opciones como “empezar una partida nueva” o que el propio programa nos permitiese la opción de empezar con un cubo mezclado e intentar ordenarlo nosotros.

Todo esto debería de tener un cronómetro y contador del número de movimientos, para así poder realizar un ranking con los mejores tiempos, menos cantidad de movimientos han necesitado para la resolución del cubo.

Otra parte importante es el tema de gráficos. Estaría bien poder utilizar texturas en lugar de colores así como añadir una imagen de fondo y que las caras de cubitos pudieran ser dibujos.

También se podría ampliar en cuanto a funciones. Ahora mismo casi todos los dispositivos

cuentan con diversos sensores de movimiento, giroscopios, acelerómetros, etc. Se podría utilizar esta tecnología para poder realizar giros en el cubo, por ejemplo de caras completas.

## 13 APÉNDICES

### 13.1 MANUAL DEL USUARIO

Tras cargar la aplicación podremos ver que nos muestra un cubo de rubik con todas sus caras cargadas por defecto. Al ser una demo, al iniciarse el cubo tendrá cada una de sus caras de uno de los colores del cubo (ver Ilustración 20).

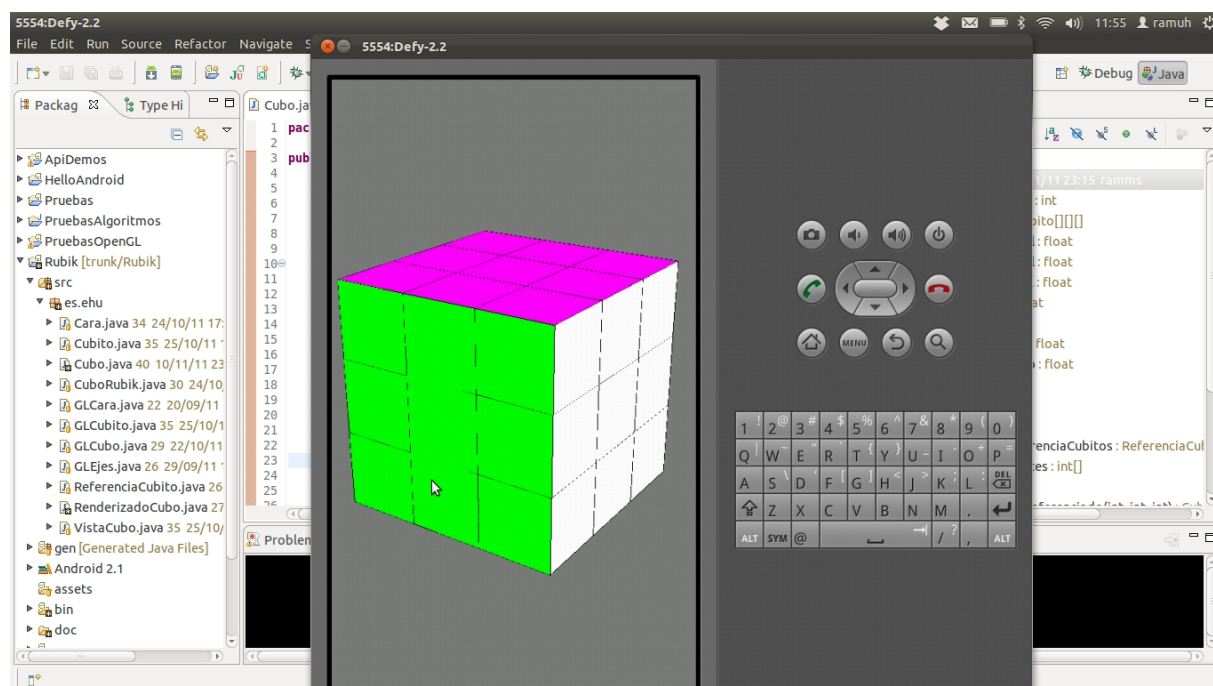


Ilustración 20: Inicio de la aplicación

A partir de este momento podríamos empezar a realizar movimientos. Solamente tendríamos que desplazar el dedo por la pantalla y moveríamos una de las filas más cercanas a donde pulsémos con el dedo, en la dirección en la que hagamos el movimiento. Por ejemplo, si pulsásemos en la mitad de la pantalla y desplazásemos el dedo hacia arriba, realizaría el movimiento de la ilustración 21.

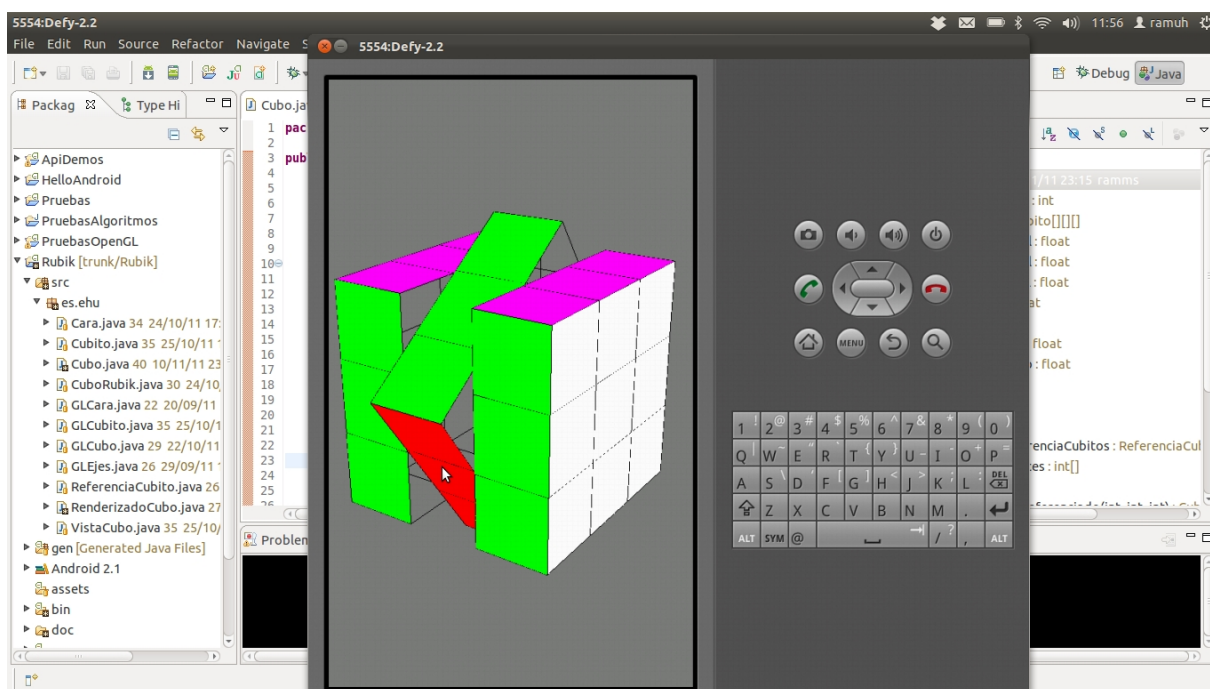


Ilustración 21: Primer movimiento en la aplicación

Mientras no se levante el dedo de la pantalla esa fila en cuestión giraría tanto como el desplazamiento del dedo. De esta forma podemos realizar un giro de más de 90 grados con un solo movimiento. En cuanto levantemos el dedo de la pantalla la fila que estamos girando se ajustará a la forma del cubo para seguir manteniendo su forma cuadrada. Se ajustará a la parte que mas cercana quede de donde se suelta la fila (ver Ilustración 22).

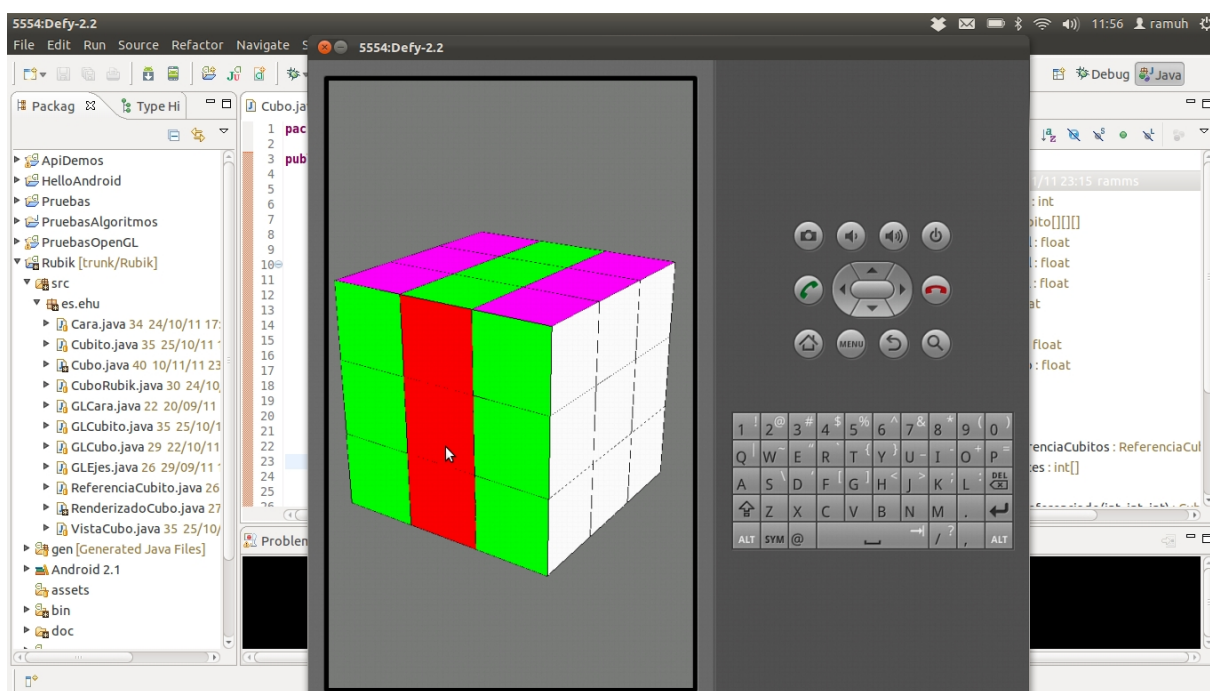


Ilustración 22: Soltar el primer movimiento de la aplicación

Podemos seguir haciendo todos los movimientos que deseemos en las direcciones que queramos de las 3 caras que se muestran en la aplicación. El cubo seguirá manteniendo su forma y mostrando los colores. En la Ilustración 23 vemos el cubo tras varios giros.

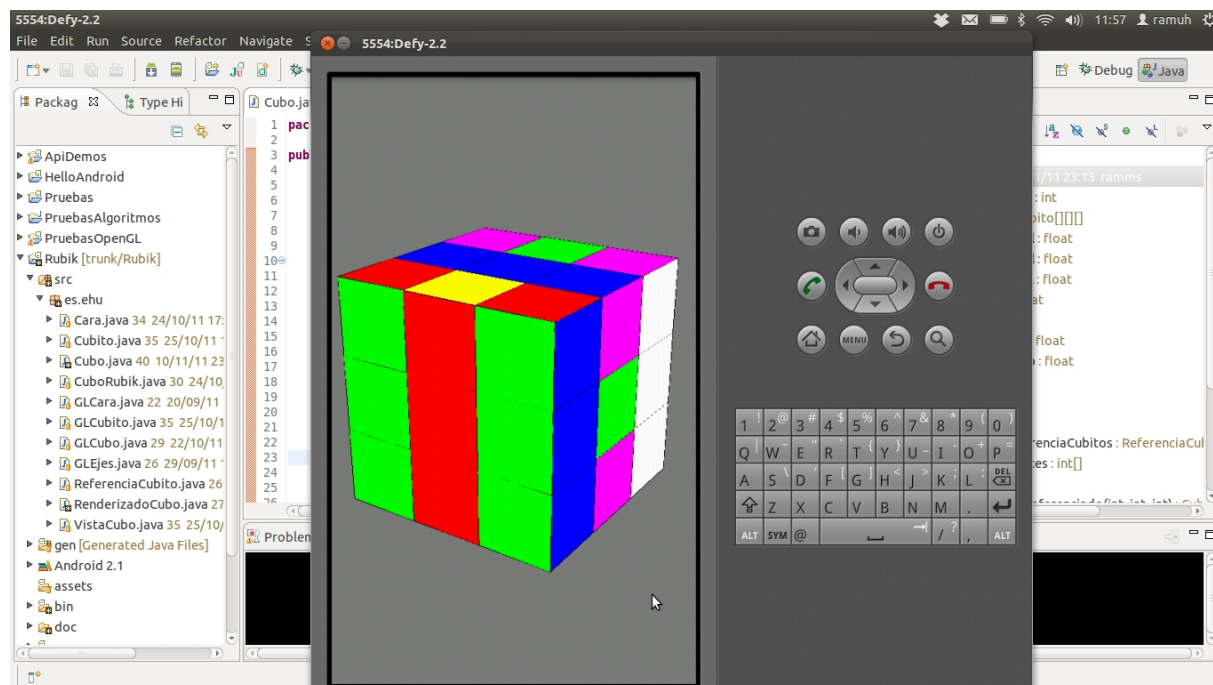


Ilustración 23: La aplicación tras varios movimientos

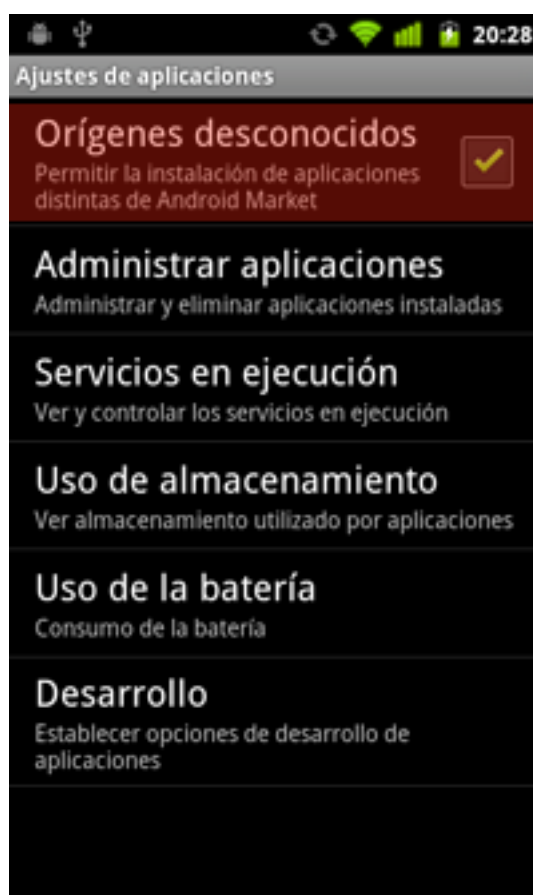
En cualquier momento podemos pulsar el botón de salir y el programa finalizaría. Pudiendo volver a entrar al programa y volviendo a tener el cubo desde el principio, esto es, con su forma por defecto.

## 13.2 GUÍA DE INSTALACIÓN PARA EL USUARIO

En la decisión tecnológica se eligió usar SourceForge para almacenar toda la información del proyecto. Por ello cuenta el proyecto con una página desde la que poder acceder a sus fuentes y a sus ejecutables binarios: <http://sourceforge.net/projects/acr/>

Desde aquí el usuario podrá descargarse el binario que puede instalar en cualquier dispositivo con Android ya sea un móvil, un tablet, un netbook, etc. En concreto estará en la sección files de la página del proyecto (<http://sourceforge.net/projects/acr/files/>). El archivo es de extensión .apk ya que es la extensión de los binarios de Android y su nombre completo es Rubik.apk

Como uno de los requisitos del proyecto era que no tuviera dependencia de ningún otro software la aplicación puede instalarse sin instalar nada previamente en el dispositivo. El único requisito que tiene, previo a la instalación es que debe de habilitarse la instalación de aplicaciones que no estén en el Android Market de Google (Orígenes Desconocidos). Para ello nos dirigimos a Ajustes → Aplicaciones y nos aparecerá una pantalla como la de la ilustración 24.



*Ilustración 24: Habilitar orígenes desconocidos en el dispositivo*

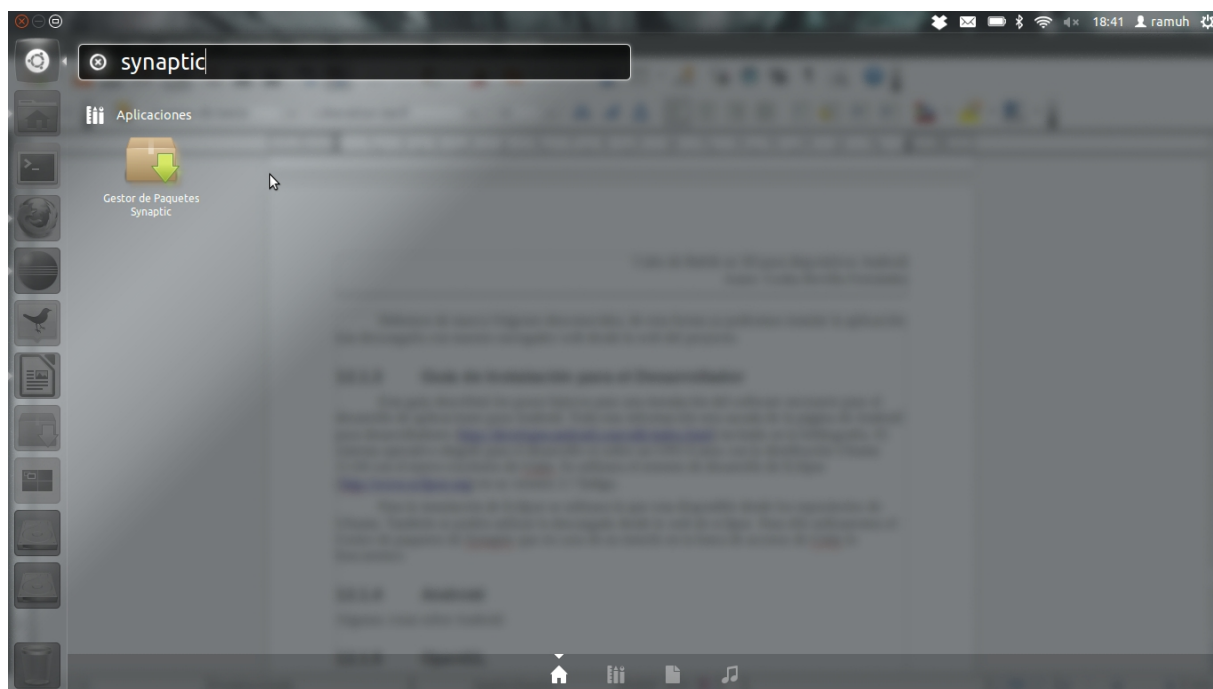
Debemos de marcar Orígenes desconocidos, de esta forma ya podremos instalar la aplicación tras descargarla con nuestro navegador web desde la web del proyecto.

### **13.3 GUÍA DE INSTALACIÓN PARA EL DESARROLLADOR**

Esta guía describirá los pasos básicos para una instalación del software necesario para el desarrollo de aplicaciones para Android. Toda esta información esta sacada de la página de Android para desarrolladores (Ref. [2]). El sistema operativo elegido para el desarrollo es sobre un GNU/Linux con la distribución Ubuntu 11.04 con el nuevo escritorio de Unity. Se utilizará el entorno de desarrollo de Eclipse en su versión 3.7 Índigo.

#### **13.3.1 Instalación de Eclipse:**

Para la instalación de Eclipse se utilizará la que está disponible desde los repositorios de Ubuntu. También se podría utilizar la descargada desde la web de Eclipse. Para ello utilizaremos el Gestor de paquetes de Synaptic, que en caso de no tenerlo en la barra de accesos de Unity, lo buscaremos con la búsqueda de Unity como se muestra en la ilustración 25.



*Ilustración 25: Buscar Synaptic en el equipo*

Una vez localizado Synaptic entramos. Tras iniciar el programa presionamos sobre el botón que dice buscar y pondremos eclipse. Esto nos devolverá la búsqueda y marcaremos el paquete eclipse para su instalación. Nos marcará que depende de varios paquetes los cuales también tendremos que instalar y pulsaremos sobre aplicar.



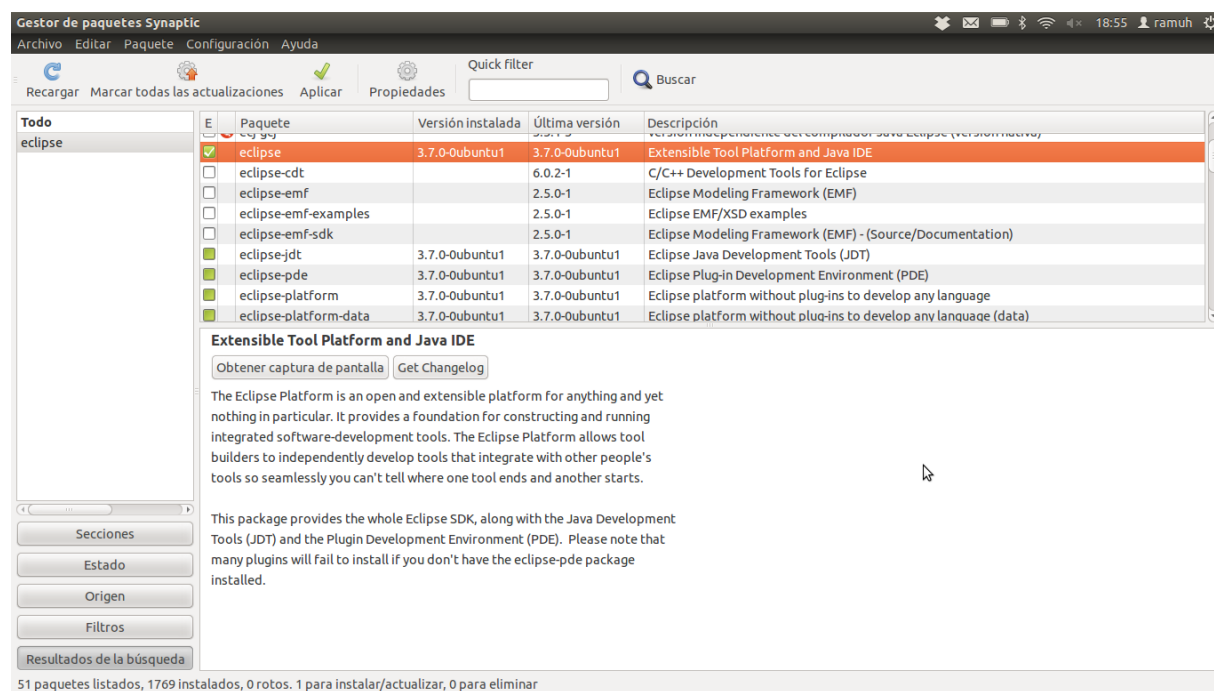


Ilustración 26: Buscar eclipse en Synaptic

Una vez finalice la instalación, podremos ejecutar eclipse desde el buscador de Unity.

### 13.3.2 Instalación del SDK de Android

El siguiente paso será instalar en el entorno de Eclipse el SDK de Android. Para ello nos basaremos en la información obtenida de la web oficial de desarrollo de Android (Ref. [2]). En esta misma página tendremos los enlaces para descargarnos su SDK. Nos bajaremos el del sistema operativo Linux. Esto nos descargará un archivo de extensión .tgz el cual descomprimiremos. Nos creará una carpeta llamada android-sdk-linux, la cual deberemos de copiar a la carpeta /opt/ para un mejor acceso por parte de otros usuarios del sistema. Desde consola sería tan fácil como (Se utiliza sudo porque en la carpeta /opt/ solo puede escribir por defecto root):

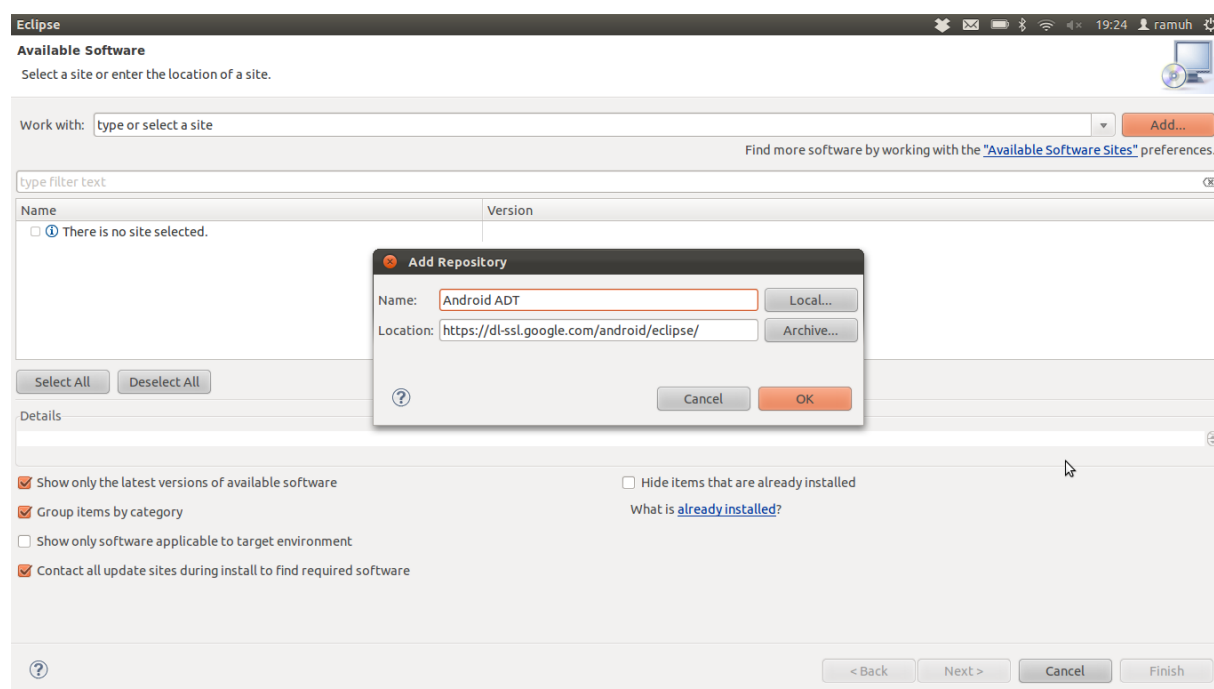
```
sudo cp -r ./android-sdk-linux /opt/
```

Nos pedirá la contraseña de superusuario, la introducimos y nos copiará la carpeta. Tras esto ya tenemos la carpeta donde irán los archivos del SDK de Android.

### 13.3.3 Instalación del ADT para Eclipse

Ahora nos falta la instalación del Plugin para Eclipse. Este Plugin será el que nos permita usar un emulador para Eclipse, entre otras cosas. Para ello abrimos Eclipse y nos dirigimos a Help → Install new Software. Nos abrirá una nueva ventana en la que en la parte derecha pulsaremos sobre el botón Add y añadiremos los datos de la ilustración 27.





*Ilustración 27: Añadir el repositorio de Android*

Tras esto pulsamos OK y nos cargará los paquetes que podremos instalar. Marcaremos Developer Tools y pulsaremos sobre Next. Ahora nos mostrará todo lo que vamos a instalar, volvemos a pulsar en Next. Aceptamos la licencia y pulsamos sobre Finish. Tras finalizar la descarga e instalación necesitaremos reiniciar Eclipse.

### 13.3.4 Configuración del Plugin ADT e Instalación de la API

Por último nos queda configurar el Plugin e instalar las API que deseemos junto con sus emuladores. Para ello abrimos de nuevo Eclipse y nos dirigimos a Window → Preferences. En la ventana que nos abre en el panel de la izquierda buscamos Android y deberemos de indicar en el SDK location la carpeta en la que tenemos el SDK de Android que descargamos en el primer paso. En nuestro caso en /opt/android-sdk-linux. Quedaría como se muestra en la ilustración 28(La parte del recuadro que aparece en la imagen no debería de aparecer).

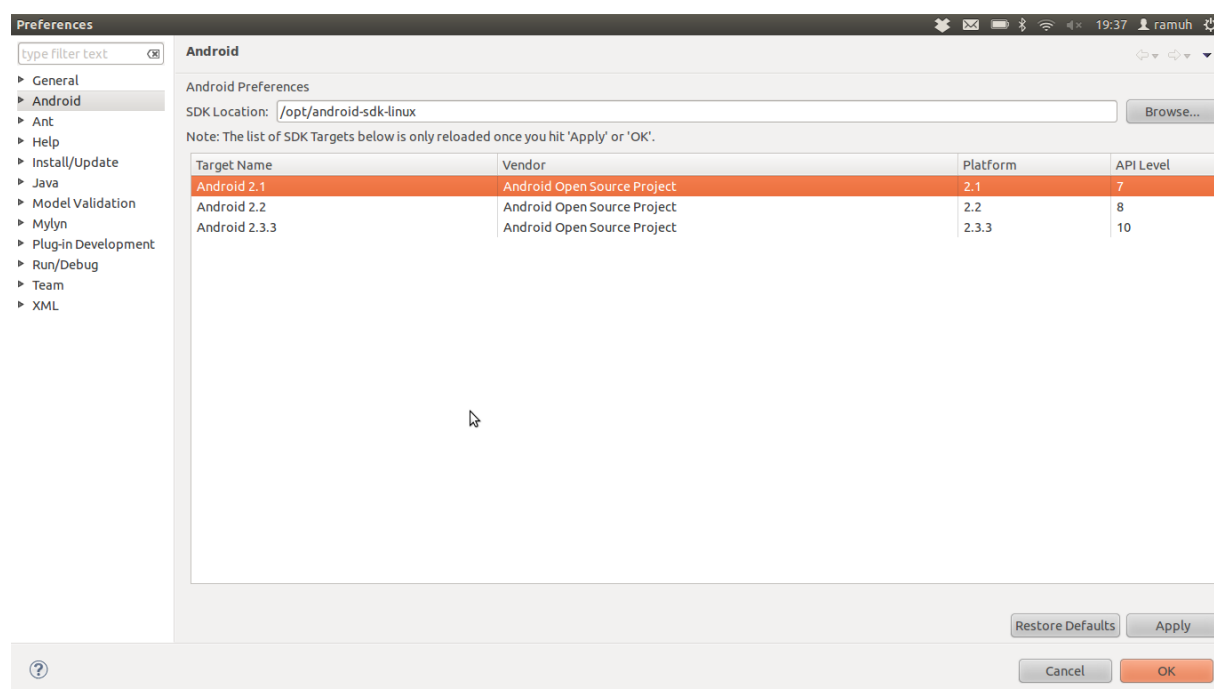


Ilustración 28: Determinar la ubicación del SDK

Ahora instalaremos la API que necesitamos. Para ello vamos a Window → Android SDK Manager. En esta ventana nos aparecerán todas las APIs de las versiones de Android disponibles. En el momento de escribir estas líneas casi todos los dispositivos móviles vienen con la versión 2.2 de Android y el Dispositivo con el que cuenta el alumno (Motorola Defy) también. Por lo tanto se instalará la API de esta versión. Se pueden instalar tantas versiones como se desee y de esta forma se podrán realizar pruebas sobre diferentes dispositivos. Seleccionamos el que deseemos instalar y pulsamos sobre install.

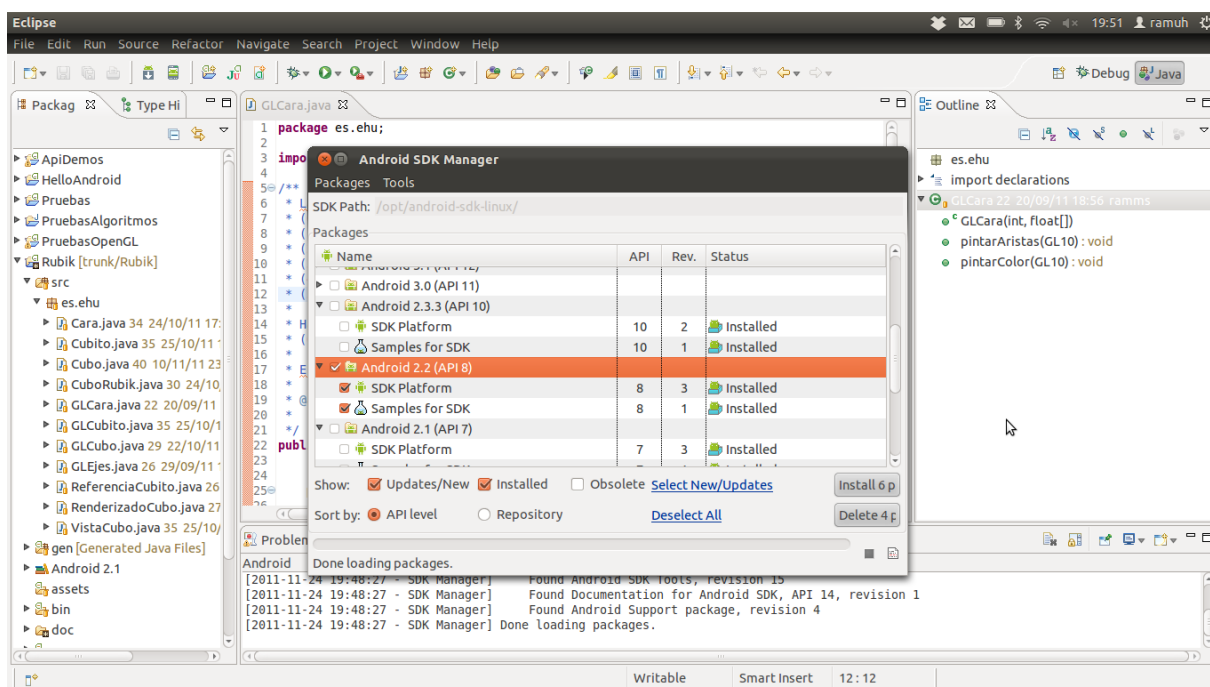


Ilustración 29: Instalar las APIs deseadas

Una vez instalado reiniciamos Eclipse. Ahora ya tendríamos la API instalada y nos quedaría crear los dispositivos virtuales (el emulador). Para ello vamos a Window → AVD Manager. En esta ventana pulsamos sobre new y creamos un nuevo dispositivo. Se le pueden añadir muchas cosas como acelerómetros, tarjetas de memoria, etc. Aunque para el desarrollo de esta aplicación no necesitaremos ningún extra. Con esto sería suficiente.

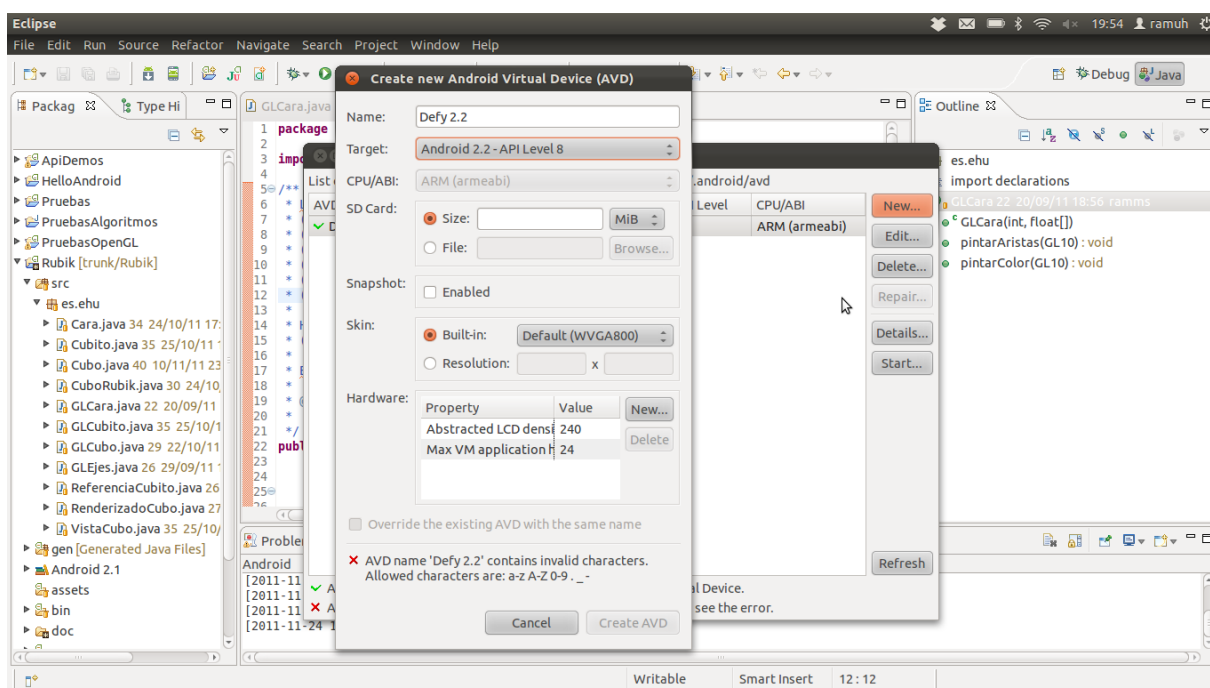


Ilustración 30: Crear un nuevo dispositivo virtual de Android

Ahora ya tenemos el emulador configurado y listo para funcionar. Ya solo nos quedaría crear una aplicación y ejecutarla. No haría falta que la versión de la API que usamos para crear el programa sea la misma que la que tiene el emulador, de hecho para probar la aplicación sería recomendable probarlo en varios emuladores diferentes, con diferentes resoluciones de pantalla, diferentes versiones de Android, etc. En la ilustración 31 se ve como se vería el emulador ejecutándose.

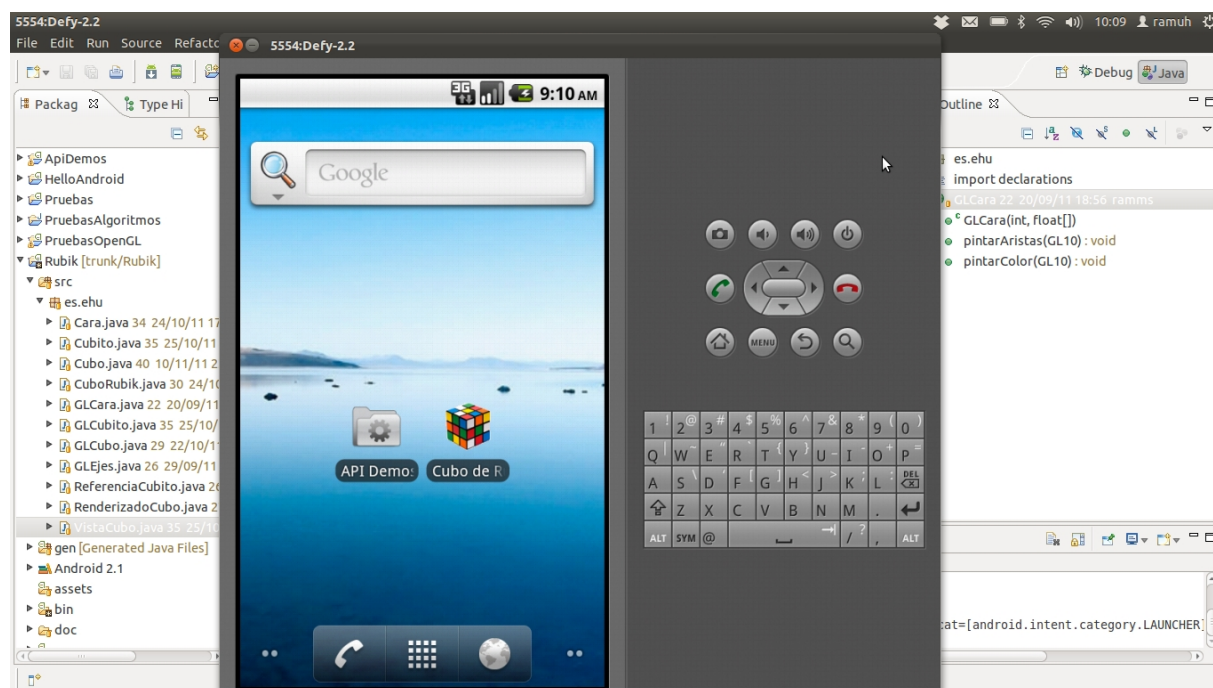
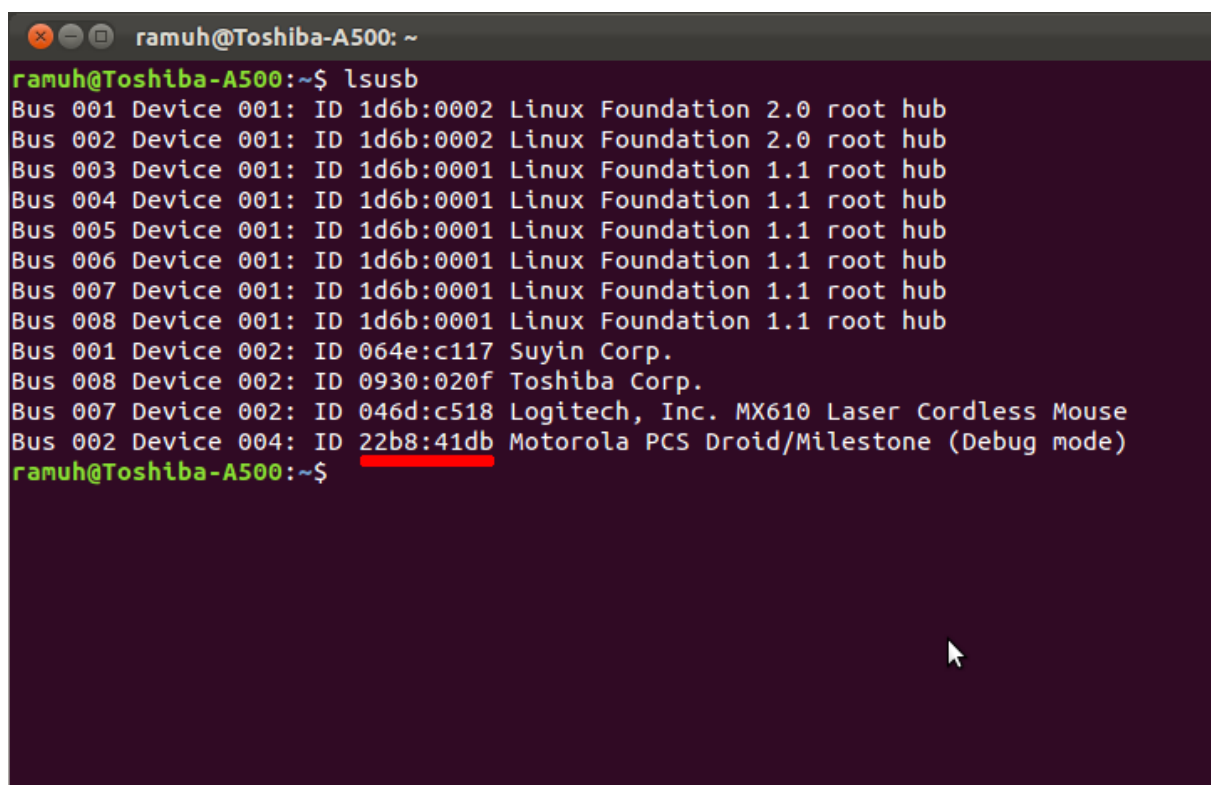


Ilustración 31: Ejecutando el emulador

### 13.3.5 Debugger con dispositivos físicos

Como hemos visto en el apéndice de instalación para desarrolladores, Android nos ofrece un emulador sobre el que probar nuestros programas desarrollados. Existe la posibilidad de que surjan problemas para un determinado dispositivo, ya sea por culpa del hardware, por una configuración diferente a lo habitual, por los programas instalados... Para solucionar este problema Google nos ofrece la posibilidad de poder conectar nuestro dispositivo al ordenador y poder usarlo de sustituto del emulador, dicho de otra forma, nos permitirá probar nuestras aplicaciones directamente en nuestro propio dispositivo físico. Este detalle es de gran utilidad a la hora de debuggear cualquier error. Para ello la única configuración que tenemos que hacer en el dispositivo es habilitar el debugger por usb en Ajustes → aplicaciones → Desarrollo y activar Depuración por USB. Mientras en el ordenador tendremos que permitirle escribir sobre el dispositivo a la API, para ello con el dispositivo conectado al PC por usb buscamos mediante el comando `lsusb` el id del fabricante y del producto. Estos son los indicados en la parte subrayada de la Ilustración 32.



```
ramuh@Toshiba-A500: ~$ lsusb
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 003 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 004 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 005 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 006 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 007 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 008 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 001 Device 002: ID 064e:c117 Suyin Corp.
Bus 008 Device 002: ID 0930:020f Toshiba Corp.
Bus 007 Device 002: ID 046d:c518 Logitech, Inc. MX610 Laser Cordless Mouse
Bus 002 Device 004: ID 22b8:41db Motorola PCS Droid/Milestone (Debug mode)
ramuh@Toshiba-A500: ~$
```

*Ilustración 32: Comando lsusb mostrando el idVendor y el idProduct*

Una vez identificados creamos en /etc/udev/rules.d un archivo por ejemplo de nombre 99-android.rules con el siguiente contenido.

```
SUBSYSTEM=="usb", ATTRS{idVendor}=="22b8", ATTRS{idProduct}=="41da",  
SYMLINK+="android_adb", MODE="0666", OWNER="root"
```

Siendo 22b8 y 41da los datos obtenidos del comando lsusb. Ahora, cuando vayamos a ejecutar nuestra aplicación desde eclipse, podremos elegir nuestro dispositivo.

## **13.4 OPENGL**

Este apartado describirá diferentes nociones que han sido necesarias conocer para poder realizar este proyecto sobre OpenGL.

### **13.4.1 ¿Que es OpenGL?**

OpenGL es una librería de gráficos para poder realizar aplicaciones que dibujen gráficos en 2D y en 3D. La ventaja de OpenGL es que es multiplataforma y multilenguaje. En este caso para Android se nos ofrecía la posibilidad de poder usarla con C++ o Java. Otras de las ventajas es que viene incorporado con la API de Android y por lo tanto no es necesaria ninguna instalación aparte.

### **13.4.2 Versiones**

Hay multitud de versiones de OpenGL, dependiendo de la tarjeta gráfica con la que contemos empezando desde la 1.0 hasta la 4.2 ahora mismo. Cada una tiene sus ventajas y desventajas y son soportadas por distintas tarjetas gráficas. Todas estas versiones son para ordenadores, por lo cual en este proyecto se ha necesitado usar una versión distinta a todas ellas de OpenGL. En concreto esta versión se llama OpenGL ES 2.0.

OpenGL ES (Embedded Systems) es una variante simplificada de OpenGL, que está diseñada para utilizarse en dispositivos integrados. Por lo cual podemos verlo en Iphones o Ipad, en dispositivos Android e incluso en algunas videoconsolas como en la Play Station 3. En concreto Android incorpora su versión más reciente, la 2.0.

Tiene algunas diferencias la versión para PC y la versión para dispositivos empotrados, que hace que el tiempo de cálculo de esta última sea más corto, debido a que está destinado a dispositivos menos potentes que el primero.

### **13.4.3 Frameworks**

Existe una gran variedad de frameworks para OpenGL que ofrecen mayor facilidad a la hora de trabajar con las funciones que ofrece OpenGL. La mayoría de frameworks son para versiones completas de PC, aunque también podemos encontrar algunas de OpenGL ES para dispositivos Android. Algunas de ellas son muy útiles a la hora de desarrollar juegos ya que por ejemplo, nos ofrecen physics, lo cual es ideal para representar colisiones entre objetos y cosas por el estilo.

Cuando se realizó el análisis de mercado se investigó durante un tiempo el framework orientado a juegos AndEngine. Aunque se prefirió no utilizar ninguno de estos frameworks para poder conocer más a fondo OpenGL.

## 14 BIBLIOGRAFIA

Aquí se recogen todas las fuentes utilizadas para documentación sobre los temas referentes al proyecto. Los clasificamos según su contenido:

Android:

- [1] Android, guía para desarrolladores.  
de Frank Ableson, Charlie Collins y Robi Sen
- [2] Guía de Instalación de la API de Android en Eclipse.  
<http://developer.android.com/sdk/index.html>
- [3] Documentación de la API de Android.  
<http://developer.android.com/sdk/android-2.1.html>
- [4] Tutorial básico sobre juegos en Android.  
<http://knol.google.com/k/juan-de-dios-maldonado-s%C3%A1nchez/gu%C3%ADa-tutorial-para-programar-juegos-2d/yf7xkfmg7vie/3#>
- [5] Blog de desarrollo de aplicaciones sobre Android.  
<http://droideando.blogspot.com/>

OpenGL ES:

- [6] Documentación sobre OpenGL ES.  
<http://www.khronos.org/opengles/sdk/docs/man/>
- [7] Programación de un videojuego en 3D para PC utilizando OpenGL.  
PFC de Rubén García Moreno de la UPV/EHU en 2008
- [8] Página sobre programación mediante OpenGL.  
<http://www.programaciongrafica.com/>
- [9] Tutorial sobre OpenGL ES OpenGL de Per-Erik Bergman.  
<http://blog.jayway.com/2009/12/03/opengl-es-tutorial-for-android-part-i/>

Programación en general:

- [10] Foro StackOverFlow sobre todo tipo de dudas sobre programación: Android, OpenGL, Java...  
<http://stackoverflow.com/>

Documentación:

- [11] Wikipedia.  
<http://wikipedia.org/>



