UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

BisiRacing

Proyecto Final de Carrera

Ingeniería Técnica en Informática de Gestión

Autores: José Manuel Martínez Expósito

Cristian Roca Badia

Director: Francisco Javier Piris Ruano

26/02/2013

Resumen

Este documento se corresponde con la memoria de proyecto final de carrera de Ingeniería Técnica en Informática de Gestión (ITIG), que describe paso por paso el proceso de desarrollo del mismo.

BisiRacing, trata de una aplicación para dispositivos móviles con Sistema Operativo Android. Este tiene dos objetivos, donde uno es obtener información real y actualizada de Valenbisi con la intención de que un usuario conozca cuales son los puntos de recogida/deposición más cercanos a su posición con las bicicletas y anclajes allí disponibles seleccionando la pestaña de estaciones.

El segundo objetivo pretende fomentar el Valenbisi como unidad de entretenimiento y ejercicio. Para ello la aplicación constará de un juego con dos posibilidades. El primero de ellos consistiría en un juego de contrarreloj, cogiendo nuestra posición mediante GPS; es decir, seleccionando la estación origen, un cronometro se activará, y este finalizará cuando se llegue a la estación destino quedando registrado el tiempo con la dificultad de la aparición de enemigos sobre el mapa, los cuales te perseguirán por el recorrido y tendremos que huir de ellos y llegar hasta la meta, por lo cual la puntuación se determinara por el tiempo transcurrido, distancia recorrida y si logramos el objetivo de llegar a la meta "vivos". El segundo será similar al juego anterior, pero con la diferencia de que la posición no se obtiene mediante GPS, sino que será "un modo sofá", sin salir de casa, consistirá en ir dando instrucciones al jugador mediante el Touch, por lo que se diferencia solamente por la posición actual que será ficticia.

La aplicación ha sido desarrollada mediante la tecnología Eclipse como principal herramienta de desarrollo de código, tras haber instalado una serie de plugins para preparar al entorno para el desarrollo de aplicaciones para Android.

Ilustración 1. Icono de aplicación

Tabla de contenidos

1. INTRODUCCIÓN

1.1- Explicación del proyecto

La aplicación móvil que se desarrolla permite consultar la localización e información de todas las estaciones ValenBisi, pudiendo informarnos sobre el estado de las bicicletas y de los anclajes de cada estación mediante un servicio que nos proporciona la información en formato XML.

Por otra parte la aplicación móvil nos permitirá interactuar mediante Google Maps, convirtiéndolo en un juego donde se tendrá que escapar de una serie de enemigos hasta llegar a la meta seleccionada.

En el caso de utilizar la versión de "Modo Real" la aplicación nos proporcionara mediante geolocalización GPS nuestra posición sobre el mapa, y mostrarnos las 5 estaciones más cercanas a nosotros. En el caso del "Modo Sofá" no recibiremos nuestras posición por GPS, simplemente tocaremos donde queremos aparecer, así sin movernos de casa podemos jugar igualmente mediante órdenes táctiles.

Una vez acabada la partida, se guardara la puntuación obtenida, pudiendo reversarla mediante la pestaña de Puntuaciones.

1.2- Motivación

Como bien dice el propio título, un PFC o Proyecto Final de Carrera es la aplicación y culminación de todos los conceptos aprendidos a lo largo de los cursos y asignaturas de la carrera para la obtención de un producto.

En los últimos años, las nuevas tecnologías en dispositivos móviles han avanzado hasta un nivel en el que se puede disponer de aplicaciones de gestionar grandes cantidades de información y de realizar costosas tareas y operaciones. Además el abaratamiento de los costes en una conexión de datos para estos dispositivos supone un abanico de posibilidades en cuanto a obtener información en tiempo real de cualquier tipo. Si a ello añadimos la posibilidad que hoy en día es posible conocer nuestra propia posición en tiempo real, se da la posibilidad de desarrollar una herramienta capaz de conocer, analizar y almacenar datos sobre lo que nos interesa y que en ese mismo momento esté a nuestro alrededor.

En nuestro proyecto nos decantamos por la plataforma para dispositivos móviles Android.

El gran potencial de esta tecnología y las grandes facilidades que aporta al desarrollo de aplicación motiva a la realización de este proyecto. Android se encuentra prácticamente

al inicio de sus pasos y sin embargo ya tiene en su cartera muchos seguidores, tanto de usuarios como de compañías.

Además de esto su código abierto hace que cualquier desarrollador pueda modificar las secciones internas de una aplicación, Esto significa que se puede compartir el código y desarrollar aplicaciones de manera más rápida.

También nos decantamos por la posibilidad que nos ofrece para aprender un lenguaje en el que el presente ya tiene muchas posibilidades, y que en el futuro pensamos que puede llegar muy lejos, por ello aprender este lenguaje y todas las posibilidades que el mismo nos ofrece, pensamos que con gran seguridad nos puede servir.

Todas estas razones hacen que la tarea de estudiar, desglosar y clasificar la plataforma Android, así como el desarrollo de sus aplicaciones, sea algo importante y el factor primordial de la motivación de este proyecto.

Otros datos que nos han ayudado a decidirnos a programar aplicaciones en Android han sido que, aunque iPhone comenzó el año liderando el mercado de los navegadores de móvil con un 48,77%, éste fue cayendo a lo largo del 2011 hasta finalizar con una cuota de usuarios del 35,99%. El caso de Android fue justamente el contrario. Comenzó con un 19,01% y acabó con un 53,29%, por lo que podemos deducir que Android se perfila como potencialmente el sistema operativo más usado para móviles en un futuro cercano. Esto se debe a algunas de sus principales características: libre, gratuito y multiplataforma; proporcionando también una serie de herramientas OpenSource para su desarrollo.

Por lo que pensamos que aprovechar la realización del Proyecto Final de Carrera para aplicar los conocimientos adquiridos en la realización de nuestros estudios universitarios, incorporando nuevos conocimientos asequibles gracias a nuestro conocimiento de Java, será una valiosa experiencia para incorporarnos en el mercado laboral en un futuro.

1.3- Objetivos

Aunque los objetivos se han podido vislumbrar prácticamente en su totalidad en el apartado anterior, los recopilaremos aquí para mayor claridad:

- Desarrollar aplicaciones para el sistema operativo Android con el entorno de desarrollo Eclipse.
- Introducirse en el uso del lenguaje y las Apis para Android.
- Usar medios de geolocalización como GPS.
- Aplicar la interoperabilidad para interactuar servicios ofrecidos desde internet.

1.4- Metodología utilizada:

Para el desarrollo de la aplicación, se ha utilizado el modelo de ciclo de vida en espiral, parecido al que se puede ver en la siguiente imagen:

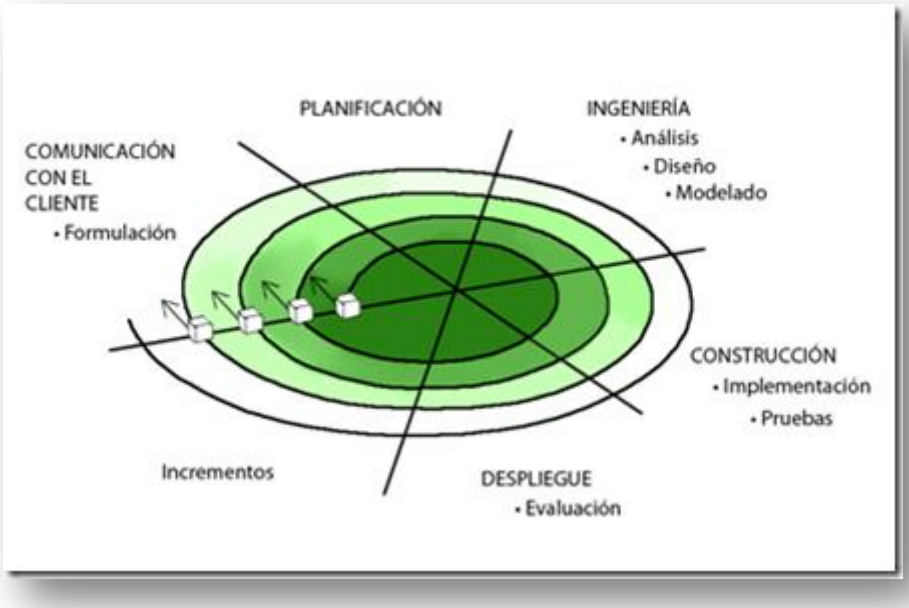


Ilustración 2. Ciclo de vida en espiral.

Por ello en este modelo hemos pasado una serie de fases hasta llegar de manera iterativa a la versión final del producto. Las fases son las siguientes:

- **Análisis de requerimientos:** Durante esta etapa de estudia detalladamente los requerimientos que cada objetivo conlleva. Aquí establecen todos los detalles funcionales deseados.

- **Diseño del sistema:** Con los datos de la etapa anterior, se diseña el sistema. Se realiza la interfaz de usuario, entorno , etc..

- **Etapas de construcción:** La etapa de construcción comprende básicamente la codificación y test de unidades. Esta etapa es un trabajo de programación pura.

- **Test y evaluación:** En esta etapa se realiza un test del módulo completo así como su evaluación frente al estudio de requerimientos. En muchos casos en es esta etapa los usuarios finales participan de manera activa aportando información decisiva para la usabilidad del sistema.

1.5- Requerimientos

Hay varios requisitos previos que hay que tener en cuenta en la aplicación, se enumeran a continuación:

- ***Compatibilidad***: La aplicación está desarrollada para la versión 2.3.3 de Android, por lo que no habrá problemas para versiones posteriores, pero para las versiones inferiores no podrán ejecutar esta aplicación.

- ***El servicio web debe estar accesible:*** El servicio web que nos proporciona la información de las estaciones, bicis y anclajes debe estar disponible en todo momento para no ocasionar fallos.

- ***El usuario deberá tener el GPS activado*:** En el caso de no estar utilizando el Modo sofá, para poder coger la localización sobre el mapa.

- **El usuario deberá disponer de conexión a internet:** En el caso de no tener conexión a internet no podremos recibir la información de Google Maps, ni de las estaciones.

1.6- Esquema de funcionamiento de la aplicación

A continuación se hará una breve descripción sobre los pasos que sigue la aplicación, con el fin de que el lector pueda entender de una forma más clara las explicaciones posteriores.

- Al meternos en nuestra aplicación, lo primero será arrancar el mapa de Google Maps.

- Seguidamente conectaremos con el Servicio web alojado en http://www.valenbisi.es/service/carto

- Del servicio web anterior, se descargara la información en un XML, y seguidamente se deberá insertar los datos en las diferentes listas y así poder ir rellenando las tablas, poder posicionar las diferentes estaciones, en definitiva toda la parte de información externa en la aplicación.

- Después en el caso del Modo Real, obtendremos la localización del GPS. Gracias a la localización GPS se posiciona con Latitud y Longitud nuestro posicionamiento, y con nuestra posición y la de las estaciones, calcular sus

distancias a nosotros para poder sacar las estaciones más cercanas, y posicionarlas en Google Maps.

Este sería a groso modo un resumen del funcionamiento de la aplicación. De esta forma cualquier detalle en futuros apartados será mejor comprendido.

2. SISTEMA OPERATIVO ANDROID

2.1- ¿Qué es Android?

Android es una plataforma de software de código abierto que incluye un sistema operativo para dispositivos móviles basado en Linux. Desarrollado por Google y por la Open Handset Alliance.

Android ha nacido con una filosofía de código abierto, esperando que programadores de todo el mundo contribuyan de manera libre al constante desarrollo del sistema operativo. Además Google también dejará libertad a los fabricantes para que desarrollen aplicaciones para sus teléfonos y, que la licencia sea de código abierto, no descarta la opción de que hayan empresas que cobren por los programas que desarrollen. De hecho cualquier operadora puede vender terminales con Android, siempre que ésta cumpla lo siguiente:

- Precio bajo
- Acceso abierto a Internet
- Libertad del usuario para descargar cualquier aplicación

Para animar a los programadores a desarrollar para Android, Google ha ofrecido 10 millones de dólares a las mejores aplicaciones creadas. Esto nos muestra la estrategia de Google: recopilar un montón de aplicaciones nuevas y atractivas para impresionar al mercado y aumentar la demanda de terminales. Para que los programadores puedan desarrollar, Google dispone de una página Web para que los usuarios puedan descargarse el SDK, bajarse manuales, ver ejemplos y leer instrucciones para que estos puedan familiarizarse con la plataforma. Algo realmente lógico si quieren que los programadores se predispongan a programar de manera voluntaria.

2.2- Historia

En julio de 2005, Google compró Android Inc., una pequeña empresa situada en Palo Alto, California, y entonces empezaron a surgir rumores acerca de que Google estaba planeando construir su propio teléfono móvil libre y posiblemente gratis, centrándose en ganancias de publicidad en las búsquedas de las personas para mover un poco el mercado móvil. Obviamente, los rumores de un móvil gratis fueron falsos pero al final Android resultó ser algo mucho más interesante y revolucionario: un sistema operativo móvil open source propulsado por Google, así que repasemos la historia de Android por corta que sea.

El lanzamiento inicial del Android Software Development Kit apareció en noviembre de 2007 y algun tiempo después - mediados de agosto de 2008- apareció el Android 0.9 SDK en beta. A finales de septiembre 2008, finalmente lanzaron Android 1.0 SDK (Release 1). Seis meses después - principios de marzo 2009-, Google presentó la versión 1.1 de Android para el "dev phone" y la actualización incluía algunos cambios estéticos menores, además de soporte para "búsquedas por voz", aplicaciones de pago en Android Market, arreglos en el reloj alarma, mejoras en Gmail y algunas otras características.

A mediados de mayo 2009, Google lanza la versión 1.5 de Android (llamada Cupcake) con su respectivo SDK que incluía: grabación de video, soporte para Estéreo Bluetooth, sistema de teclado personalizable en pantalla, reconocimiento de voz y el AppWidget framework que permitió que los desarrolladores puedan crear sus propios widgets para la página principal. Android 1.5 fue la versión que más personas usaron para iniciarse en Android (con el T-Mobile G1 y HTC Dream en USA) y sigue siendo actualmente una versión que se encuentra disponible en muchos móviles Android como el HTC Hero o varios de los nuevos MOTOBLUR como el Motorola Backflip o Motorola Dext.

Después apareció Android 1.6 "Donut" en septiembre de 2009 con mejoras en las búsquedas, indicador de uso de batería y hasta el VPN control applet. De hecho, esta versión fue tan buena que todos los Android que no tienen una interfaz personalizada como HTC Sense o Motoblur ahora corren 1.6, incluyendo el T-Mobile G1, y en la actualidad sigue siendo la versión más popular.

Para llevar las cosas más allá, el Motorola Droid (Motorola Milestone) fue lanzado con Android 2.0 "Eclair" en el que se podían encontrar aplicaciones precargadas que requerían un hardware mucho más rápido que la generación anterior de teléfonos móviles con Android (un mes más tarde, salió 2.0.1, una pequeña actualización).

Poco después, el Google Nexus One (el cuál marcó un antes y un después ya que Google trató de venderlo por su cuenta y liberado, además de en algunas operadoras) llegó con Android 2.1 (el cual algunos llamaron "Flan" pero Google sigue considerándolo parte de "Eclair") con nuevas capacidades 3D, live wallpapers y lo que significó una gran mejora de la plataforma desde la versión 1.6.

Más adelante, El 6 de diciembre de 2010, el SDK 2.3 (Gingerbread) fue liberado incluyendo numerosas mejoras como por ejemplo Teclado multi-táctil rediseñado, soporte mejorado para desarrollo de código nativo, mejoras en la entrada de datos, audio y gráficos para desarrolladores de juegos, recolección de elementos concurrentes para un mayor rendimiento, soporte nativo para más sensores (como giroscopios y barómetros), un administrador de descargas para descargar archivos grandes, administración de la energía mejorada y control de aplicaciones mediante la administrador de tareas, soporte nativo para múltiples cámaras, etc.

Y siguiendo en esta línea de "releases" aparecieron también la versión 3.0 (HoneyComb) y sus posteriores versiones 3.1 y 3.2. Más adelante aparecería la versión 4.0 (Ice Cream Sandwich) y actualmente la versión 4.1 (Jelly Bean).Como es de esperar todas ellas ofrecen numerosas ventajas y nuevas características.

Mirando el futuro crecimiento de Android podremos ver como Android Market es la tienda de aplicaciones que más ha crecido, llegando a las 40 mil aplicaciones. Ahora mismo Android es el sistema operativo que más está creciendo en Estados Unidos rivalizando con el de iPhone.

2.3- Arquitectura de Android

La arquitectura de Android está formada por capas de software donde cada una puede utilizar los servicios de la capa inferior. Empezando por la capa superior, tenemos la capa de aplicaciones que, junto al Framework de aplicaciones, son totalmente públicas y los usuarios pueden acceder libremente. Un nivel más abajo tenemos un cojunto de librerías que no son accesibles directamente, pero si lo son a través del nivel superior. Por último en la capa inferior tenemos el conjunto de drivers basados en Linux.

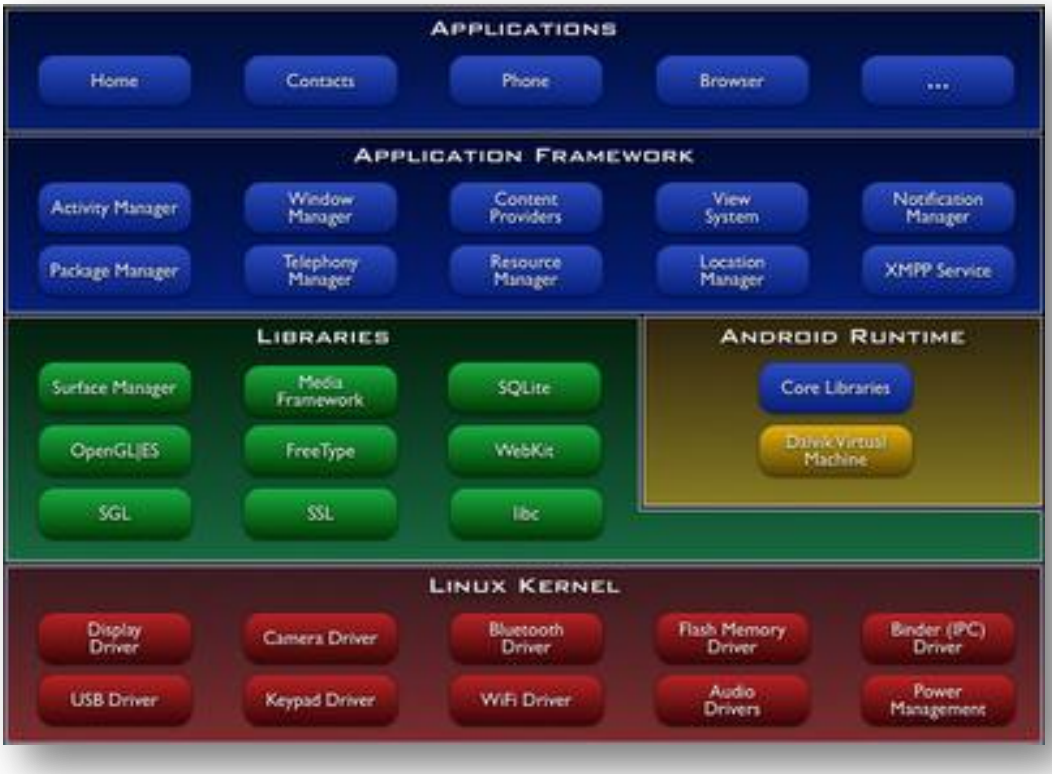


Ilustración 3. Arquitectura Android.

- **Applications:** Un conjunto de aplicaciones base de Android, entre las que encontramos un navegador Web, un cliente de email, un calendario etc. Todas hechas en lenguaje Java.

- **Application Framework:** Es una base para las aplicaciones donde los desarrolladores tienen acceso completo. Pensado para la reutilización de componentes, es decir, una aplicación puede coger funcionalidades de otra creada anteriormente para su desarrollo. Éste incluye:

    - **Telephony manager**: Gestor de hardware del teléfono.

    - **View system:** Conjunto de vistas para poder desarrollar una aplicación.

- **Content providers**: Para datos que son compartidos entre varias aplicaciones, como por ejemplo la agenda de teléfono.

- **Resource Manager:** Administrador de recursos que permite acceder a recursos como Strings, gráficos, archivos de layout…

- **Notification Manager**: Administrador de notificaciones para mostrar alertas. Las aplicaciones pueden añadir eventos en una barra de notificaciones.

- **Activity Manager:** Administrador de actividades, éste maneja el ciclo de vida de las aplicaciones y la navegación entre ellas.

- **Location Manager:** Servicio de localización. Permite al móvil recibir avisos, notificaciones, eventos etc., de un lugar específico o por nuestra localización actual. Una posibilidad de las muchísimas que se podrían dar con esta API sería que cada vez que nos acercásemos a un cine, recibamos las novedades de éste en nuestro teléfono.

- **Servicio XMPP:** Envío de mensajes para aplicaciones entre terminales Android. Se podría utilizar en juegos multiusuario por ejemplo.

- La capa **Libraries** está formada por un conjunto de librerías escritas en C/C++. Todas expuestas a los desarrolladores a través del Framework de aplicaciones.

- **Android Runtime:** Está compuesto por el núcleo de librerías y la máquina virtual Dalvik. Aquí disponemos de un conjunto de librerías que incluyen las funcionalidades que solemos encontrar en las librerías básicas de Java.

Como se ha explicado anteriormente, esto no quiere decir que la máquina virtual sea Java, pero si su lenguaje usado para programar. En la parte más inferior tenemos la **Kernel de Linux**. Basado en el núcleo de Linux 2.6 y base de la pila de software del sistema, se encarga de las funciones más básicas: gestión de drivers, seguridad, gestión de memoria, administración de procesos etc.

2.4- Estructura de un proyecto Android

Cada vez que creamos un nuevo proyecto de una aplicación Android en Eclipse, se genera automáticamente un sistema de archivos definido para todas las aplicaciones de dicho sistema operativo.

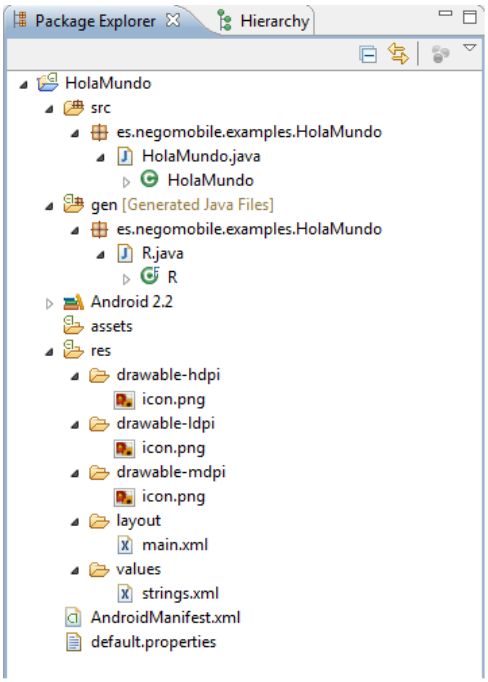


Ilustración 4. Estructura proyecto Android.

Dentro de la carpeta principal, que lleva el nombre del proyecto en cuestión, se encuentra la estructura de archivos que pasamos a definir a continuación:

- **En la carpeta *src*** es donde se guarda el paquete que contiene las clases necesarias para hacer funcionar la aplicación. Son los archivos de origen *.java* que una vez compilados se transformarán en archivos *.class*.

- **En la carpeta *bin***, que no aparece en la estructura que muestra eclipse, pero sí que está dentro del proyecto, es donde se guardan las clases compiladas que hemos descrito en el punto anterior, los fichero *.class*.

- En el **directorio *gen*** se ubica una clase muy peculiar de las aplicaciones en Android, la clase *R.java*. Esta clase se genera automáticamente cada vez que modificamos los ficheros de recursos y no podemos retocarla manualmente. Se podría considerar como un nexo de unión entre los recursos de la aplicación y los ficheros *.xml* que se encuentran en el proyecto y definiremos en breve.

- Dentro del proyecto también se encuentran las **librerías de Android** que son las que dan funcionalidad al sistema. Las librerías también dependen de la versión del SDK que estemos utilizando.

- En la **carpeta *assets***, inicialmente vacía y que en general no suele usarse, se pueden ubicar cualquier tipo de ficheros que represente un origen de datos para la aplicación. Para poder manejar esos datos, en la aplicación se crearía un gestor de assets, o *AssetManager*, que se encargaría de asociar los datos dentro de la propia aplicación.

- En el **directorio *res*** sí que se genera automáticamente una estructura definida para gestionar los recursos de origen de la aplicación. En el subdirectorio ***drawable*** se almacenan todos los ficheros de imágenes necesarias para la aplicación. Dentro de ***layout*** se guardan los ficheros *.xml* que es con lo que más comúnmente se trabaja en las aplicaciones de Android, y que componen la interfaz gráfica del usuario. En el subdirectorio ***values*** se almacena un fichero *string.xml* donde se asocian todas las cadenas de texto de la aplicación, y también se pueden tener valores de atributos o estilos de los elementos de la aplicación.

- En el fichero **AndroidManifest.xml** que es indispensable para todas las aplicaciones de Android. En este fichero se almacenan los parámetros de configuración de la aplicación, tal como las Activities principales (a continuación describiremos qué es una Activity), los privilegios asignados, etc.

- El fichero **default.properties** no nos influirá en la realización del proyecto, ya que sólo es un fichero que se genera automáticamente indicando las propiedades por defecto del SDK con el que estamos trabajando

2.5- Componentes de una aplicación Android

Toda aplicación puede constar de varias componentes que interactúan entre sí, y cada una de ellas tiene una función determinada a realizar. Las componentes se implementan en las clases que tengamos creadas en la carpeta ***src*** de nuestro proyecto. Las más importantes de toda aplicación las pasamos a definir a continuación:

- ***Activity*:** En general, corresponde a una pantalla específica de la aplicación, como por ejemplo la pantalla inicial que representa el punto de entrada de dicha aplicación. Desde ahí se pueden invocar a las principales tareas que el usuario puede llevar a cabo.

Las actividades pasan por una serie de estados, desde que se crean hasta que se destruyen, que conviene conocer para poderlos capturar y actuar en ellos en determinadas circunstancias.

Una actividad tiene esencialmente 4 estados:

- Si una actividad está en el primer plano de la pantalla (en la parte superior de la pila), está activa o en marcha (ejecutándose).

- Si una actividad ha perdido el foco, pero sigue siendo visible (es decir, una nueva actividad que no ocupe toda la pantalla o una transparente que tiene el foco en la parte superior de su actividad), está en pausa. Una actividad pausada está completamente viva (que mantiene toda la información del estado y del miembro y permanece unido al gestor de ventanas), pero puede ser eliminada por el sistema en situaciones extremas de poca memoria.

- Si una actividad es completamente oscurecida por otra actividad, se detiene. Conserva toda la información del estado y miembros; sin embargo, ya no es visible para el usuario por lo que su ventana se oculta y a menudo esta actividad será eliminada por el sistema cuando la memoria sea necesaria en cualquier otro sitio.

- Si una actividad está en pausa o se detiene, el sistema puede dejar la actividad sin memoria, ya sea esperando a que termine, o simplemente matar el proceso. Cuando se vuelva a mostrar al usuario, ésta debe ser completamente renovada y restaurada a su estado anterior.

Por tanto, para terminar de entender claramente las actividades, destacaremos las tres fases en la que toda actividad se mueve:

1. La vida entera de una actividad ocurre entre la primera llamada a **onCreate()** y la única llamada final a **onDestroy()**. Una actividad será lanzada por primera vez y, además, establecerá toda la configuración general de su estado en onCreate(), y liberará todos los recursos que quedan en OnDestroy (). Por ejemplo, si tiene un hilo conductor en un segundo plano para descargar datos de la red, puede crear ese hilo en onCreate () y luego se detiene el hilo en OnDestroy ().

2. El curso de la vida visible de una actividad ocurre entre una llamada al método **onStart()** y la correspondiente llamada a **onStop()**. Durante este tiempo el usuario puede ver la actividad en pantalla, aunque podría no ser en primer plano e interactuando con el usuario. Entre estos dos métodos se pueden mantener los recursos que se necesitan para mostrar el resultado de la actividad al usuario. Por ejemplo, puede registrar un BroadcastReceiver en OnStart() para controlarlos cambios que ocurren en tu interfaz de usuario, y anular el registro en onStop() cuando el usuario ya no vea lo que está mostrando. El OnStart() y onStop() se pueden llamar varias veces, así como la actividad se hace visible y se oculta para el usuario.

3. El tiempo de vida en primer plano de una actividad ocurre entre una llamada a **onResume()** y la correspondiente llamada a **onPause()**. Durante este tiempo la actividad se encuentra delante de todas las demás actividades y en la interacción con el usuario. Una actividad con frecuencia puede estar en esta situación - por ejemplo, cuando el dispositivo se va a dormir, cuando el resultado de la actividad se entrega - por lo que el código de estos métodos debe ser bastante ligero. El método onPause() es el estado en el que entra la actividad cuando se lanza otra. La actividad permanece en segundo plano hasta que se vuelva de la nueva que se acaba de lanzar. El método onResume() es el estado que indica que

la actividad ha vuelto a entrar en primer plano tras estar en estado de pausa. Hay que destacar que las actividades no terminan, a no ser que se haga de forma explícita. Normalmente permanecen en segundo plano (no en pausa, sino en stop) hasta que el sistema operativo las cierra para liberar la memoria que ocupan en caso de necesidad.

Otras componentes:

- ***Service***: Representa una aplicación que corre internamente en el sistema y debe ejecutarse sin interacción con el usuario. Es una aplicación que se está ejecutando pero sin necesidad de ser vista, como puede ser el caso de reproducir una pista de audio. Es una aplicación que puede mandar los mensajes que sean necesarios a un determinado servicio que esté activo. Para utilizar un recurso, como por ejemplo el GPS, es necesario solicitarlo al manejador de servicios, que nos proporciona una instancia del mismo y así podemos utilizarlo.

- ***IntentReceiver***: Este componente permite a la aplicación realizar llamadas internas que responden a cambios de estado de nuestro terminal. Por ejemplo una llamada, un mensaje recibido o un cambio en la geolocalización.

- ***ContentProvider***: Representa una capa que permite compartir datos a las distintas aplicaciones de Android. Es necesario declarar este tipo de componentes para poner a disposición de los procesos los datos que se consideren oportunos.

3. TECNOLOGÍAS UTILIZADAS

En este apartado vamos a comentar brevemente las tecnologías usadas. Pretendemos dar una breve descripción de qué son y cómo las hemos usado.

3.1- GPS

El Sistema de Posicionamiento Global (NAVSTAR-GPS o más comúnmente GPS) es el más común y conocido a la hora de enfrentarse al problema de localizar un dispositivo móvil. Se trata de una constelación de 24 satélites que giran alrededor de la Tierra a una distancia de 20200 Km, con trayectorias sincronizadas para cubrir toda la superficie del planeta.

Para su funcionamiento, el terminal del usuario debe disponer básicamente de un módulo GPS.

Con estos elementos, el dispositivo es capaz de recibir los datos que envían los satélites que tenga en línea de visión, que son fundamentalmente la posición del satélite y una

marca de tiempo. Sincronizando esas marcas de tiempo se puede ver el retardo que han sufrido las señales hasta llegar al terminal, y por tanto la distancia a la que se encuentran los satélites. Ahora, triangulando según esas distancias se puede hallar la posición del dispositivo. El número de satélites necesarios para conseguir el punto en el que nos encontramos es variable, según la calidad del módulo GPS del terminal: teóricamente son necesarios 3, pero en la práctica pueden ser necesarios hasta 9 satélites para obtener una buena aproximación.

En cuanto a los datos que proporciona el sistema son varios, entre los que se encuentran las coordenadas geográficas (latitud y longitud) de nuestra posición, la altura sobre el nivel del mar a la que nos hallamos, marcas de tiempo, etc. La precisión de las coordenadas geográficas en condiciones óptimas tiene un error de aproximadamente 15 metros, siendo la tecnología comercial más precisa que podemos encontrar, funcionando por sí sola.

El sistema GPS es, por tanto, ideal para establecer la posición de un dispositivo, ya que prácticamente en cualquier espacio abierto tenemos cobertura. Sin embargo, el sistema GPS adolece de un problema importante en núcleos urbanos y lugares montañosos, y es que en situaciones en las que permanecemos rodeados de edificios u obstáculos altos en general, no disponemos de línea de visión directa con el número suficiente de satélites necesarios para fijar nuestra posición. Por tanto, el sistema no es suficiente para ello. Esto puede solucionarse con el sistema A-GPS, como veremos más adelante.

Por último, otro problema al que nos enfrentamos con esta tecnología es el del consumo de batería. Los dispositivos móviles se caracterizan por alimentarse de una batería. Los requisitos de ligereza hacen que las mismas deban ser del mínimo tamaño posible con un rendimiento adecuado. El contratiempo se produce porque el modulo GPS tiene un consumo de batería elevado, limitando la duración de operatividad del terminal, que, en algunos casos como en los teléfonos, es un factor crítico: ¡Podemos quedarnos incomunicados! Para paliar esto se han desarrollado algoritmos que ahorran batería, conectando el GPS sólo cuando sea estrictamente necesario.

3.2- XML

Antes de explicar que es la tecnología XML, aclarar que en el proyecto, XML tiene dos usos, uno que es para la recopilación de información del servicio web sobre el estado de las estaciones, bicicletas y anclajes, y el otro uso es sobre la programación de Android.

XML, siglas en inglés de *eXtensible Markup Language* (lenguaje de marcas ampliable), es un metalenguaje extensible de etiquetas desarrollado por el *World Wide Web Consortium*. Es una simplificación y adaptación del *SGML* y permite definir la gramática de lenguajes específicos (de la misma manera que *HTML* es a su vez un lenguaje definido por *SGML*). Por lo tanto *XML* no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de

estos lenguajes que usan *XML* para su definición son *XHTML, SVG, MathML*. *XML 1.0* se convirtió en una recomendación del *W3C* el 10 de febrero de 1998.

XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

3.2.1- Puntos fuertes de *XML*

- Permite separar contenido de presentación.
- Es extensible mediante la adición de nuevas etiquetas.
- El analizador es un componente estándar, no es necesario crear un analizador específico para cada lenguaje. Esto posibilita el empleo de uno de los tantos disponibles. De esta manera se evitan bugs y se acelera el desarrollo de la aplicación.
- Si un tercero decide usar un documento creado en *XML*, es sencillo entender su estructura y procesarlo. Mejora la compatibilidad entre aplicaciones.

3.2.2- Puntos débiles de *XML*

- *XML* no es especialmente bueno manejando grandes cantidades de datos.
- *XML* se vuelve complicado de entender si hay mucha información en un único documento.
- Ciertos tipos de información (imágenes, otros tipos de información binaria) no están muy bien representados en *XML*.

A la hora de diseñar la UI, *Android* permite hacerlo tanto por código *Java* como por ficheros en lenguaje *XML*. La primera opción puede llegar a ser muy compleja y confusa, mientras que la segunda añade los beneficios y la potencia del uso del lenguaje *XML*. *Android* define un gran número de elementos personalizados, cada uno representando una subclase de *View*. Para diseñar una interfaz, se anidan los diferentes elementos y se guardan en un fichero *XML* dentro del directorio *res/layout* de la aplicación cada fichero describe una sola pantalla, pero esta puede estar compuesta tanto por un elemento simple como un conjunto de ellos.

Android dibuja los elementos en el orden que aparecen en el *XML*, por lo que si algún elemento se solapa, se dibujará el último que se lea. Cada fichero *XML* se compila a partir del árbol de elementos por lo que el fichero sólo debe contener un *tag root*.

3.3- Geolocalización y Mapas

El SDK de Android incluye dos *packages* que proporcionan soporte primario para montar servicios de localización: *android.location* y *com.google.android.maps.*

El primero contiene varias clases relacionadas con los servicios de localización y, además, incluye el servicio *LocationManager* el cual proporciona una *API* para determinar la localización del dispositivo. *LocationManager* no puede ser instanciado directamente sino que se debe obtener un controlador.

myLocationManager=
(LocationManager)getSystemService(Context.LOCATION_SERVICE);

Cuando se obtiene el control de un *LocationManager*, se pueden realizar las tres siguientes acciones:

- Solicitar una lista de todos o algunos de los *LocationProviders* disponibles.
- Registrarse o des registrarse para actualizaciones periódicas de la posición.
- Registrarse o des registrarse de eventos cuando el dispositivo se encuentra en determinada ubicación (por ejemplo, cerca de otros dispositivos).

- Solicitación de un *LocationProvider* concreto (gps):

myLocation = myLocationManager.getLastKnownLocation("gps");

- Registro para actualizaciones periódicas (2º argumento y para eventos 3er argumento).

myLocationManager.requestLocationUpdates(strProvider,
MINIMUM_TIME_BETWEEN_UPDATE,MINIMUM_DISTANCECHANGE_FOR_UP
DATE, listener);

Cuando se usa el emulador se deben introducir los datos de la ubicación. Se puede utilizar tanto el DDMS como el comando *geo.*

geo fix -121.45356 46.51119 4392

En el segundo *package* (*com.google.android.maps*), se obtienen varias clases relacionadas con el *rendering*, control y dibujo sobre mapas. La más importante es la

clase *MapView* que automáticamente dibuja un mapa básico de *Google Maps* cuando lo añades al *layout* de la aplicación.

- Crea un Mapa y lo muestra por pantalla:

myMapView = new MapView(this, "");

setContentView(myMapView);

Para poder obtener un *MapView*, se necesita previamente un *API Key* de *Google Maps*. Además, para poder hacer uso de esta clase, se ha de añadir el permiso en el manifiesto dado que no es un *package* estándar de *Android*. La librería que utiliza Android es <uses-library android:name="com.google.android.maps" />

Después de todo el procedimiento, se puede trabajar sobre la vista del mapa con la clase *Overlay*.

- Se obtiene el control de capas del objeto MapView

overlays = myMapView.getOverlays();

- Se crea un objeto *Overlay* y se define una capa para el mapa

MyLocationOverlay myLocationOverlay = new MyLocationOverlay();

- Una vez definida la capa, se añade a la vista

overlays.add(myLocationOverlay);

4. REQUISITOS

A continuación, respecto a la especificación de requisitos de la aplicación, vamos a definir dos tipos de requisitos, que son los siguientes:

- **Requisitos funcionales**, a nivel de uso y/o diseño para cada tipo de usuario posible en ambas partes del proyecto.
- **Requisitos no funcionales**, a nivel de gestión y manipulación de los datos y el soporte para éstos y otras características adicionales del sistema.

Todos los requisitos expuestos a continuación han ido refinándose a lo largo de todo el proceso de desarrollo.

4.1- Requisitos funcionales

Los requisitos funcionales son aquellos que consisten en una característica requerida del sistema que expresa una capacidad de acción del mismo - una funcionalidad, generalmente expresada en una declaración en forma verbal.

- **Requisito 1:** Se deberá mostrar las cinco estaciones más cercanas a la posición del jugador.
- **Requisito 2:** Al tocar las estaciones, se deberá mostrar si se requiere la estación como inicial, o como final.
- **Requisito 3:** Se deberán descargar los datos correctamente del servicio web para su posterior uso.
- **Requisito 4:** Se dispone de dos botones para la elección de Modo de Juego, y corresponder con las funcionalidades de cada uno.
- **Requisito 5:** Los enemigos deberán aparecer en un radio lógico cercano al jugador y a la estación destino**.**
- **Requisito 6:** En el modo sofá, debe corresponder la orden con el touch para que el jugador vaya donde le indicamos.
- **Requisito 7:** Dependiendo si se tiene el Gps activado, deberá aparecer la opción de solo "Modo Sofá", o también el "Modo Real" si se tiene activado el GPS

4.2- Requisitos no funcionales

Los requisitos no funcionales son aquellos que consisten en una característica requerida del sistema, del proceso de desarrollo, del servicio prestado o de cualquier otro aspecto del desarrollo, que señala una restricción del mismo.

- **Requisito 1**: El servicio web que nos proporciona la información de las estaciones, debe de estar disponible.
- **Requisito 2**: El GPS deberá estar activo en el caso de elegir el modo de juego Real, para poder detectar la localización.

- **Requisito 3**: El dispositivo móvil debe tener conexión a internet para poder descargarse la información y poder mostrar el Google Maps.

- **Requisito 4**: El dispositivo móvil debe tener la versión Android 2.3.3 o superior.

4.3- Requisitos de eficiencia

Por motivos que anteriormente se ha comentado, como los problemas que pueden tener los dispositivos móviles respecto a su consumo de batería y su consumo de datos de internet, la aplicación deberá ser implementada de forma que consuma la mínima cantidad de ambos, tanto como sea posible. Por ello tendremos varios aspectos en cuenta:

- Minimizar el número de conexiones para descargar el fichero XML.
- Minimizar el número de consultas al GPS.
- Evitar ejecutar código de forma repetitiva que no añada funcionalidad necesaria.

4.4- Restricciones de diseño

No se ha especificado ninguna restricción a destacar en cuanto al desarrollo de la aplicación.

5. ANÁLISIS

Seguidamente entramos en la etapa del análisis del sistema, que tiene como objetivo la obtención de los diagramas de cómo se va a comportar el proyecto.

5.1- Casos de uso

El diagrama de casos de uso de BisiRacing, plasma las diferentes posibilidades de uso de nuestra aplicación. El usuario (actor) será el que va a utilizar la aplicación. Interactuara de manera directa con la aplicación móvil.

Caso de Uso: Uso de la aplicación

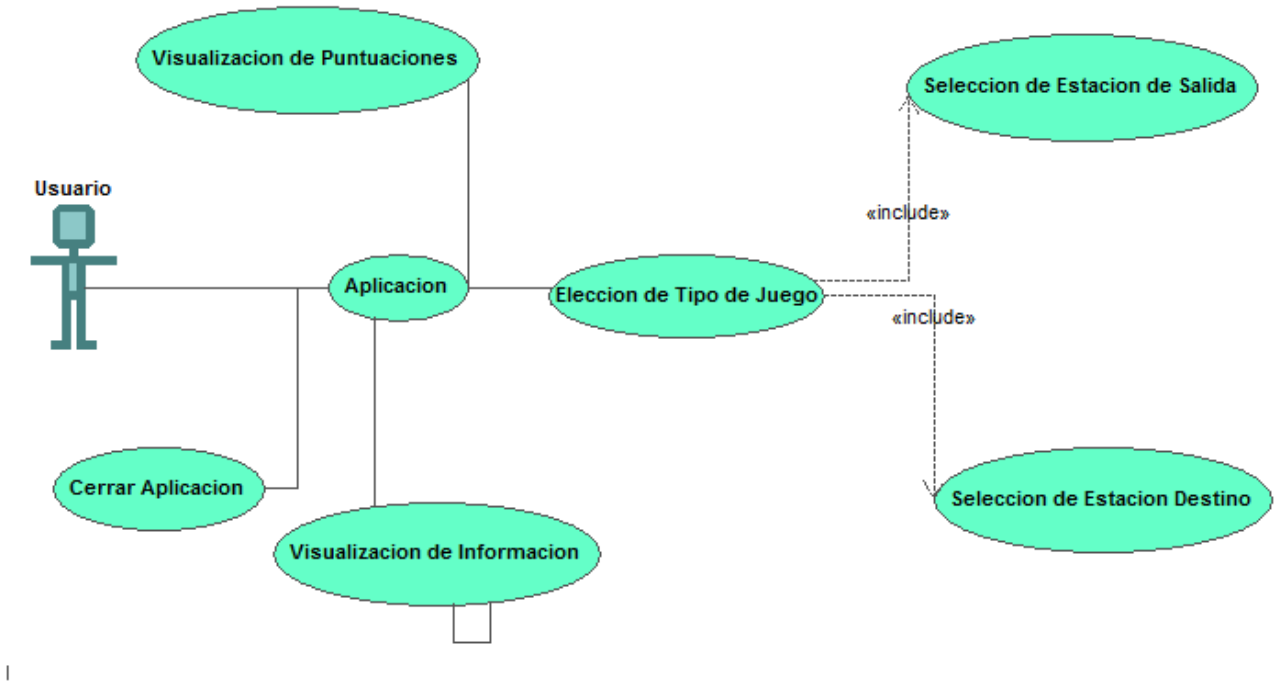


Ilustración 5. Caso de Uso de la aplicación.

Descripción: El Usuario (actor) es capaz de interactuar con la aplicación, visualizando las puntuaciones obtenidas, visualizando la información de las diferentes estaciones, en el momento que se elige el tipo de juego, tendremos las dos opciones a elegir, que estación establecemos como salida y como llegada. Por último, el actor tiene la posibilidad de cerrar la aplicación.

6. DISEÑO

Respecto al diseño de la aplicación se especializó principalmente en la interacción con el usuario y las diferentes secuencias y acciones que éste podría ejecutar con respecto a la funcionalidad. A continuación se muestra el diagrama de clases.

6.1-Diagrama de clases

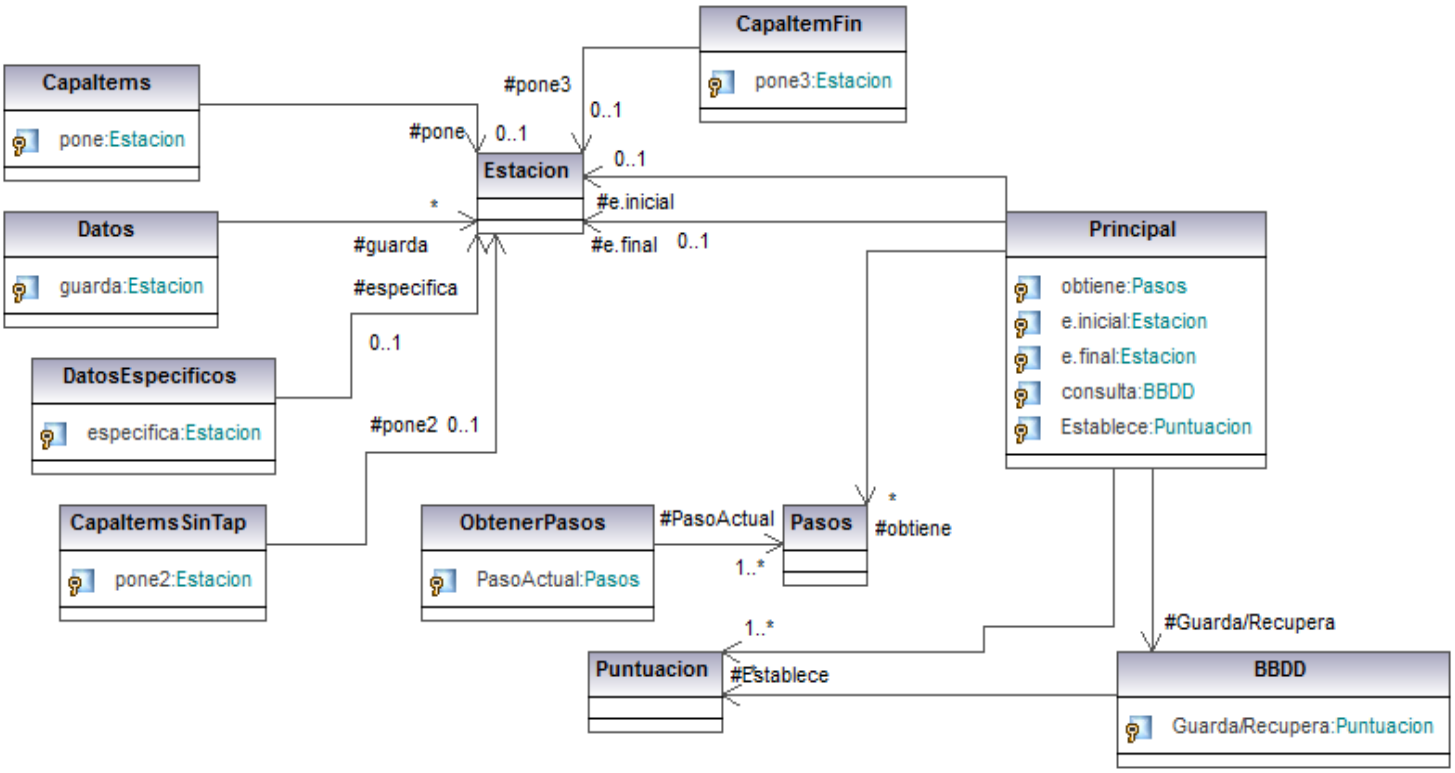


Ilustración 6. Diagrama de clases.

6.2-Diagrama de colaboraciones

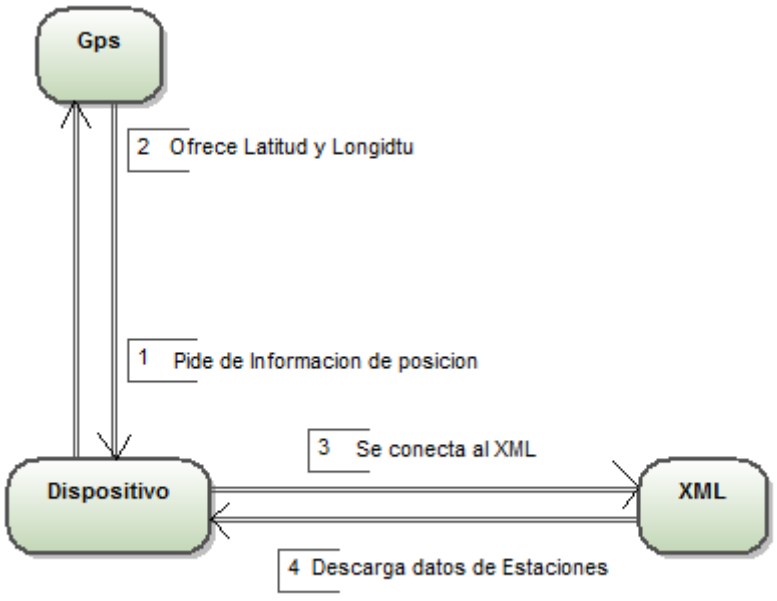


Ilustración 7. Diagrama de colaboraciones.

7. IMPLEMENTACIÓN DE LA APLICACIÓN

Este apartado es el apartado más importante de la memoria, ya que aplicaremos todo lo que hemos ido explicando en apartados anteriores para desarrollar la aplicación. Todo el desarrollo de la aplicación se ha realizado en Local.

Se explicara las partes más relevantes de la aplicación, por ello al final de la memoria se hará referencia al código de las clases.

Mapa en el Proyecto

Este tipo de controles tienen la particularidad de que sólo pueden ser añadidos a una actividad de tipo MapActivity, por lo que pantalla de la aplicación en la que queramos incluir el mapa debe heredar de esta clase.

```
public class Principal extends MapActivity{
```

Como nuestra clase hereda de MapActivity debemos implementar obligatoriamente el método isRouteDisplayed(), cuyo valor de retorno debe ser false porque no vamos a representar ningún tipo de información de ruta sobre el mapa.

```
protected boolean isRouteDisplayed() {
            return false;}
```

Además de esto, en el método onCreate() se hará referencia al método creado para Iniciar el Mapa, y dentro de IniciarMapa(), llamaremos al método setBuiltInZoomControls() sobre la referencia al control MapView para mostrar los controles de zoom estándar sobre el mapa, de forma que podamos acercar y alejar la vista del mapa. Además podemos darle diferentes opciones como son, la vista aérea, el centrar el mapa respecto a nuestra posición todo el rato, y darle un Zoom predeterminado al mapa.

```
mapa.setBuiltInZoomControls(true);
mapa.setSatellite(false);
MapController controlMapa = mapa.getController();
controlMapa.setCenter(new GeoPoint(miLatitud,miLongitud));
controlMapa.setZoom(15);
```

A partir de aquí se podrán añadir los diferentes Overlays que sean necesarios para la aplicación.

Overlays

Un Overlay es una capa que se coloca encima del mapa para así poder destacar algo del mismo. Los overlays se pueden colocar por pixel encima del mapa, con lo que se quedará fijo en la misma posición aunque desplacemos la posición del usuario; o se pueden situar, como en nuestro caso, por posición geográfica, con lo que la marca permanecerá en la posición del mapa donde se situe.
En nuestra aplicación hemos creado tres clases distintas de Overlays dependiendo de la acción a realizar al pulsar sobre cada una de ellas: la primera invitará a empezar el juego desde dicha estación, la segunda a colocar esa estación como meta de nuestro juego, y una tercera que inhabilitará cualquier tipo de acción al hacer click sobre ella.

```
public class CapaItems extends ItemizedOverlay{
public class CapaItemFin extends ItemizedOverlay{
public class CapaItemSinTap extends ItemizedOverlay{
```

Podemos darles diferentes imágenes para que pinten sobre el mapa al inicializar una capa de estos tipos, llamando al constructor

```
public CapaItemSinTap(Drawable defaultMarker, Context context) {
```

De esta manera, podemos darle una imagen diferente a cada estación dependiendo de la disponibilidad de bicicletas en las mismas.

Captación de datos de las estaciones ValenBisi

Para la lectura del archivo XML proporcionado, se ha utilizado el método SAX simplificado, ya que el anterior método SAX a pesar de funcionar perfectamente y de forma bastante eficiente, tiene claras desventajas. Por un lado se hace necesario definir una clase independiente para el handler. Adicionalmente, la naturaleza del modelo SAX implica la necesidad de poner bastante atención a la hora de definir dicho handler, ya que los eventos SAX definidos no están ligados de ninguna forma a etiquetas concretas del documento XML sino que se lanzarán para todas ellas, algo que obliga entre otras cosas a realizar la distinción entre etiquetas dentro de cada evento y a realizar otros chequeos adicionales.

A continuación se muestra la clase donde se aplica SAX:

```
public class Datos
{
    private URL rssUrl;
    private Estacion estacionActual;

    public Datos(String url)
    {
        try
```

```
            {
                this.rssUrl = new URL(url);
            }
            catch (MalformedURLException e)
            {
                throw new RuntimeException(e);
            }
        }

        public List<Estacion> parse()
        {
            final List<Estacion> estaciones = new ArrayList<Estacion>();

            RootElement root = new RootElement("carto");
            Element channel = root.getChild("markers");
            Element item = channel.getChild("marker");


            item.setStartElementListener(new StartElementListener(){
                public void start(Attributes attrs) {
                      estacionActual = new Estacion();
                      estacionActual.setNombre(attrs.getValue("name"));

          estacionActual.setNumero(Integer.valueOf(attrs.getValue("number"
)));

          estacionActual.setLatitud(Double.valueOf(attrs.getValue("lat"))*
1E6);

          estacionActual.setLongitud(Double.valueOf(attrs.getValue("lng"))
*1E6);
                      if(attrs.getValue("open") == "1")
                            estacionActual.setAbierta(true);
                      else
                            estacionActual.setAbierta(false);
                      if(attrs.getValue("bonus") == "1")
                            estacionActual.setAbonos(true);
                      else
                            estacionActual.setAbonos(false);
                }
            });

            item.setEndElementListener(new EndElementListener(){
                public void end() {
                      estaciones.add(estacionActual);
                }
            });
            try
            {
                Xml.parse(this.getInputStream(),
                        Xml.Encoding.UTF_8,
                        root.getContentHandler());
            }
            catch (Exception ex)
            {
                throw new RuntimeException(ex);
            }

            return estaciones;
        }
```

```
    private InputStream getInputStream()
    {
        try
        {
            return rssUrl.openConnection().getInputStream();
        }
        catch (IOException e)
        {
            throw new RuntimeException(e);
        }
    }
}
```

Debemos atender principalmente al método parse(). En el modelo SAX clásico nos limitamos a instanciar al *handler* definido en una clase independiente y llamar al correspondiente métodoparse() de SAX. Por el contrario, en este nuevo modelo SAX simplificado de Android, las acciones a realizar para cada evento las vamos a definir en esta misma clase y además asociadas a etiquetas concretas del XML. Y para ello lo primero que haremos será recorrer la estructura del XML hasta llegar a las etiquetas que nos interesa tratar, y una vez allí, asignarle algunos de los *listeners* disponibles [de apertura (StartElementListener) o cierre (EndElementListener) de etiqueta] incluyendo las acciones oportunas de lo que deseamos. De esta forma, para el elemento <item> (markers en el XML) navegaremos hasta él obteniendo en primer lugar el elemento raíz del XML (<rss>) declarando un nuevo objeto RootElement (carto en el XML) y después accederemos a su elemento hijo(<channel> markers en el XML) y a su vez a su elemento hijo (<item>markers en el XML) , utilizando en cada paso el métodogetChild(). Una vez hemos llegado a la etiqueta deseada, asignaremos los listeners necesarios, en nuestro caso uno de apertura de etiqueta y otro de cierre, donde inicializaremos la noticia actual y la añadiremos a la lista final respectivamente, de forma análoga a lo que hacíamos para el modelo SAX clásico. Para el resto de etiquetas actuaremos de la misma forma, accediendo a ellas con getChild() y asignado los listeners necesarios.

En el método "**public void** start(Attributes attrs)" se guardara la información deseada en la lista de estaciones, para poder recuperar esa información localmente cuando se desee y sea necesario.

Finalmente, iniciaremos el proceso de parsing simplemente llamando al método parse(), al que pasaremos como parámetros el stream de entrada, la codificación del documento XML y un handler SAX obtenido directamente del objetoRootElement definido anteriormente.

Localización Gps

Para la implementación de la parte del Gps, principalmente se basara en métodos de clase LocationManager, donde nos proporciona una serie de métodos que nos permite hacer lo que deseamos con el GPS

Necesitaremos algún mecanismo para saber si éste está activado o no el Gps en el dispositivo. Para esta tarea, la clase LocationManager nos proporciona otro método llamado "isProviderEnabled()" que nos permite hacer exactamente lo que necesitamos. Para ello, debemos pasarle el nombre del provider que queremos consultar.

```
if (locManager.isProviderEnabled(LocationManager.GPS_PROVIDER))
```

Y a continuación creamos el Listener:

```
LocationListener mLocationListener = new LocationListener()
```

En cuanto al listener, éste será del tipo LocationListener y contendrá una serie de métodos asociados a los distintos eventos que podemos recibir del proveedor:

- onLocationChanged(location): Lanzado cada vez que se recibe una actualización de la posición.

```
public void onLocationChanged(Location arg0) {
// TODO Auto-generated method stub
            Double aux_lat = arg0.getLatitude()*1000000;
            Double aux_long = arg0.getLongitude()*1000000;
            miLatitud = aux_lat.intValue();
            miLongitud = aux_long.intValue();
            if(primera){
                  mapa.getOverlays().clear();
                  IniciarMapa();
                  primera1 = true;
                  IniciarTabla();
                  pd.dismiss();
                  primera = false;
            }
            loc = arg0;
            LocationChan();
            mapa.postInvalidate();
      }
```

- onProviderDisabled(provider): Lanzado cuando el proveedor se deshabilita.

```
public void onProviderDisabled(String arg0) {
// TODO Auto-generated method stub}
```

- onProviderEnabled(provider): Lanzado cuando el proveedor se habilita.

```
public void onProviderEnabled(String arg0) {
// TODO Auto-generated method stub}
```

- onStatusChanged(provider, status, extras): Lanzado cada vez que el proveedor cambia su estado.

```
public void onStatusChanged(String arg0, int arg1, Bundle arg2){
// TODO Auto-generated method stub}
```

La forma en la que podemos obtener la posición real actualizada, va a consistir en algo así como activar el proveedor de localización y suscribirnos a sus notificaciones de cambio de posición. O dicho de otra forma, vamos a suscribirnos al evento que se lanza cada vez que un proveedor recibe nuevos datos sobre la localización actual. Y para ello, vamos a darle previamente unas *indicaciones* sobre cada cuanto tiempo o cada cuanta distancia recorrida necesitaríamos tener una actualización de la posición.

- ```
  locManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
  0, 0, mLocationListener);
        Looper.loop();
        Looper.myLooper().quit();
  ```

Implementación de Base de Datos Local

Se ha utilizado una herramienta de Base de datos llamada SQLite, para el almacenamiento y consulta de datos estructurados, que utilizaremos para guardar y recuperar las distintas puntuaciones que vaya consiguiendo el usuario de la aplicación.

Utilizamos la típica forma para crear, actualizar y conectar una base de datos SQLite, será a través de nuestra propia clase pero con la particularidad de que deriva de la clase auxiliar SQLiteOpenHelper.

A continuación se muestra la clase de la Base de Datos:

```
public class BBDD extends SQLiteOpenHelper{

      String sqlCreatePuntuaciones = "CREATE TABLE puntuaciones
(tiempo TEXT, fecha TEXT, puntuacion TEXT, vivo TEXT)";

      public BBDD(Context context, String name, CursorFactory factory,
int version) {
            super(context, name, factory, version);
            // TODO Auto-generated constructor stub
      }

      @Override
      public void onCreate(SQLiteDatabase db) {
            // TODO Auto-generated method stub
```
```

```
            db.execSQL(sqlCreatePuntuaciones);
      }

      @Override
      public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
            // TODO Auto-generated method stub
            db.execSQL("DROP TABLE IF EXISTS puntuacion");
        db.execSQL(sqlCreatePuntuaciones);
      }

      public void insertPuntuaciones(SQLiteDatabase db, String tiempo,
String fecha, String puntuacion, String vivo)
      {
            ContentValues nuevoRegistro = new ContentValues();
            nuevoRegistro.put("tiempo", tiempo);
            nuevoRegistro.put("fecha", fecha);
            nuevoRegistro.put("puntuacion", puntuacion);
            nuevoRegistro.put("vivo", vivo);

            db.insert("puntuaciones", null, nuevoRegistro);
      }

      public List<Puntuacion> selectPuntuaciones(SQLiteDatabase db)
      {
            List<Puntuacion> puntuaciones = new
ArrayList<Puntuacion>();
            String[] campos = new String[]
{"tiempo","fecha","puntuacion","vivo"};

            if(db != null)
            {
                  Cursor c = db.query("puntuaciones", campos, null,
null, null, null, null);
                  if(c.moveToFirst()){
                        do{
                              Puntuacion punt = new Puntuacion();
                              punt.setTiempo(c.getString(0));
                              punt.setFecha(c.getString(1));
                              punt.setPuntuacion(c.getString(2));
                              punt.setVivo(c.getString(3));

                              puntuaciones.add(punt);
                        }while(c.moveToNext());
                  }
            }
            return puntuaciones;
      }
}
```

Lo primero que hacemos es definir una variable llamado sqlCreatePuntuaciones donde almacenamos la sentencia SQL para crear una tabla llamada “puntuaciones” con los campos alfanuméricos tiempo, fecha, puntuación y vivo.

El método onCreate() será ejecutado automáticamente por nuestra clase, cuando aún no exista la BBDD. Para la creación de la tabla utilizaremos la sentencia SQL ya definida y la ejecutaremos contra la base de datos utilizando el método llamado execSQL(). Este

método se limita a ejecutar directamente el código SQL que le pasemos como parámetro.

Por su parte, el método onUpgrade() se lanzará automáticamente cuando sea necesaria una actualización de la estructura de la base de datos o una conversión de los datos. Pues este tipo de cosas son las que se encargará de hacer automáticamente el método onUpgrade().

- Si la base de datos ya existe y su versión actual coincide con la solicitada simplemente se realizará la conexión con ella.
- Si la base de datos existe pero su versión actual es anterior a la solicitada, se llamará automáticamente al método onUpgrade() para convertir la base de datos a la nueva versión y se conectará con la base de datos convertida.
- Si la base de datos no existe, se llamará automáticamente al método onCreate() para crearla y se conectará con la base de datos creada.

Una vez tenemos una referencia al objeto UsuariosSQLiteHelper, llamaremos a su métodogetReadableDatabase() o getWritableDatabase() para obtener una referencia a la base de datos, dependiendo si sólo necesitamos consultar los datos o también necesitamos realizar modificaciones, respectivamente. Desde la clase principal llamaremos de esta forma:

```
private static void IniciarBBDD()
	{
		bbdd = new BBDD(actividad, "BBDD", null, 1);
		dbWritter = bbdd.getWritableDatabase();
		dbReader = bbdd.getReadableDatabase();
	}
```

Ahora que ya hemos conseguido una referencia a la base de datos (objeto de tipoSQLiteDatabase) ya podemos realizar las inserciones de las puntuaciones y datos de ellas con las sentencias INSERT correspondientes. Por último cerramos la conexión con la base de datos llamando al método close().

Para poder leer los datos de las puntuaciones e insertar las distintas puntuaciones conseguidas, llamaremos desde la clase principal con estas dos líneas respectivamente.

```
List<Puntuacion> puntuaciones=bbdd.selectPuntuaciones(dbReader);

bbdd.insertPuntuaciones(dbReader, String.valueOf(tiempo),
fecha.toGMTString(), String.valueOf(puntuacion), "N");
```

Polylines

Podemos definir Polyline, como un conjunto de puntos que definen una línea. Podemos comparar el método de Polylines usado por Google para definir una ruta con los juegos de unir los puntos: tenemos una serie de puntos ordenados de manera que trazando una línea recta entre cada punto, conseguiremos recrear una línea.

De esta manera utilizando un servicio de Google, obtendremos estos Polylines con los que poder obtener una ruta posible; posible quiere decir que no atraviese edificios, por carretera…

Todo esto se obtiene en las clase 'Pasos' y 'ObtenerPasos'

En nuestro caso, para obtener los pasos llamamos al servicio con los datos de punto inicial; ya sea del jugador o del enemigo, y los datos de posición final; ya sea un punto aleatorio en el caso de los enemigos, o donde el usuario a pulsado en el mapa en el modo sofá. Además también enviamos unos parámetros fijos para ajustar mas la veracidad de la ruta: los enemigos van andando (mode=walking), unidad de medida métrica (unit=metric) lenguaje del XML obtenido (language=en) y si el dispositivo desde el que enviamos tiene sensor GPS (sensor=true). Con lo que nuestra consulta quedaría definida de la siguiente manera:

```
this.rssUrl = new
URL("https://maps.googleapis.com/maps/api/directions/xml?origin=" +
                    (double) (usuarioLat) / 1000000 + "," +
(double)(usuarioLon)/1000000 + "&destination=" +
                    (double) (enemigoLat) /1000000 + "," + (double)
(enemigoLon)/1000000 + "&mode=walking&unit=metric&sensor=true&language=en");
```

Una vez obtenidos estos puntos hacemos que el enemigo (o el usuario en modo sofá) se desplace con una cierta velocidad hasta el siguiente punto, haciendo así que se muevan exclusivamente por lugares por los que puede hacerlo.

8. PRUEBAS

Después de implementar la aplicación, llegamos a la fase de hacer las pruebas sobre varias partes de la aplicación, y poder comprobar el correcto funcionamiento de la aplicación.

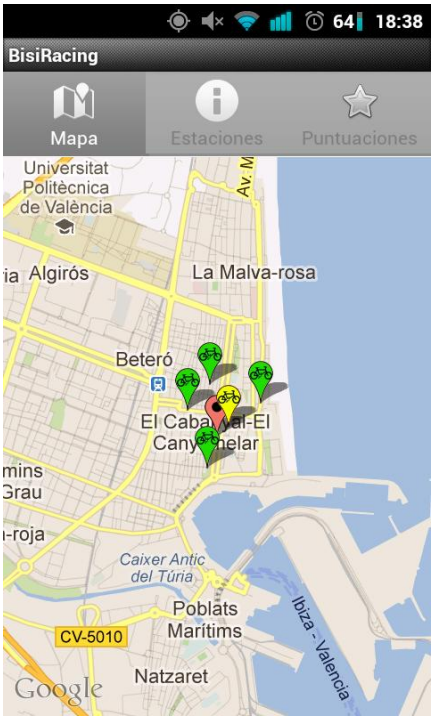
- **Prueba nº 1:** En esta prueba vamos a comprobar que si el GPS de nuestro móvil no está activado, al arrancar la aplicación, solo nos aparecerá la opción de "Modo Real", en el caso de que tengamos el GPS activado, de lo contrario aparecerá tan solo la opción de "Modo Sofá".

Para saber si el GPS está activado solo hace falta visualizar la barra de herramientas del dispositivo y ver que el icono del GPS no está activo:

Ilustración 8. Opción con Gps desactivado.

Ilustración 9. Opciones con Gps activado.

Prueba nº2: En esta prueba, dada una posición en el mapa, deberán aparecer en el mapa las estaciones más cercanas a la posición del usuario(con un radio de 1 Kilometro de distancia), y en la pestaña de estaciones, deberá mostrarse ordenado por distancia.



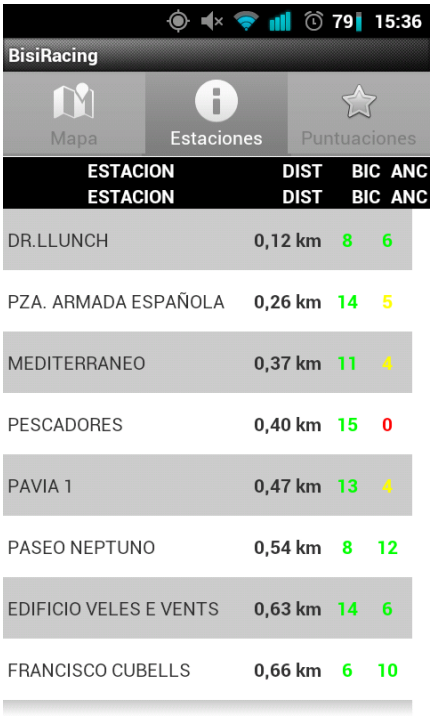


Ilustración 10. Punto adquirido con GPS

Ilustración 11. Estaciones más cercanas.

Como podemos ver aparecemos nosotros en el punto central, las estaciones más cercanas a nosotros en un 1 kilometro. Por otra parte, tenemos una lista con las estaciones ordenadas por distancia de nosotros.

Prueba nº3: En esta prueba, comprobaremos, que si en una estación no hay bicicletas disponibles, no la podremos seleccionar para estación de salida, en Modo Real.

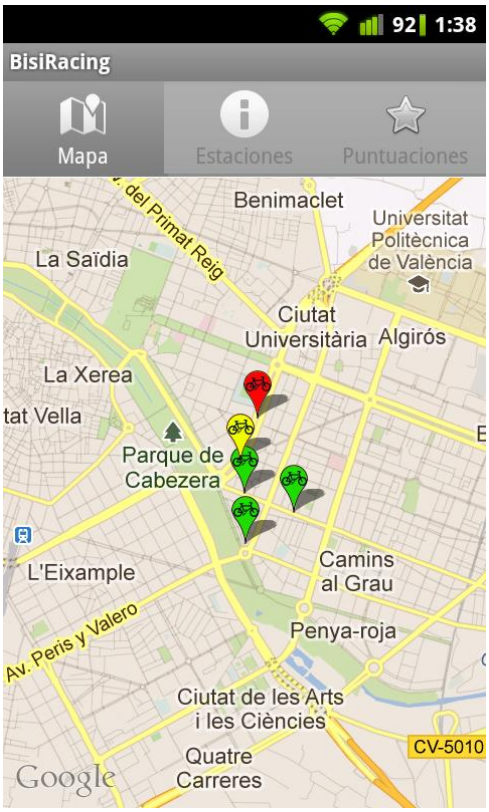


Ilustración 12. Toque en estación Roja.

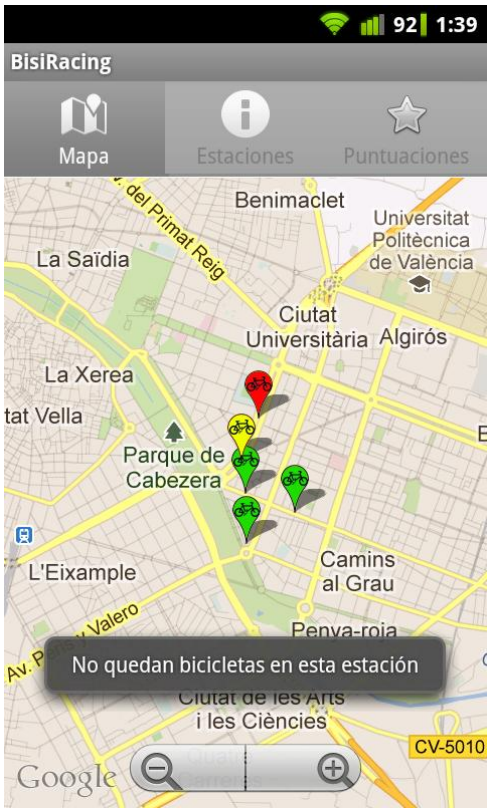


Ilustración 13. Toast Sin bicicletas.

Podemos observar que al tocar sobre una estación de color rojo (No quedan bicicletas), no nos dejara seleccionarla como una estación de salida, y nos notificara mediante un Toast, que no la podemos seleccionar para jugar desde ella.

Prueba nº4: Esta prueba consistirá en que los enemigos se acerquen hacia nosotros a medida que nos vamos moviendo por el mapa.

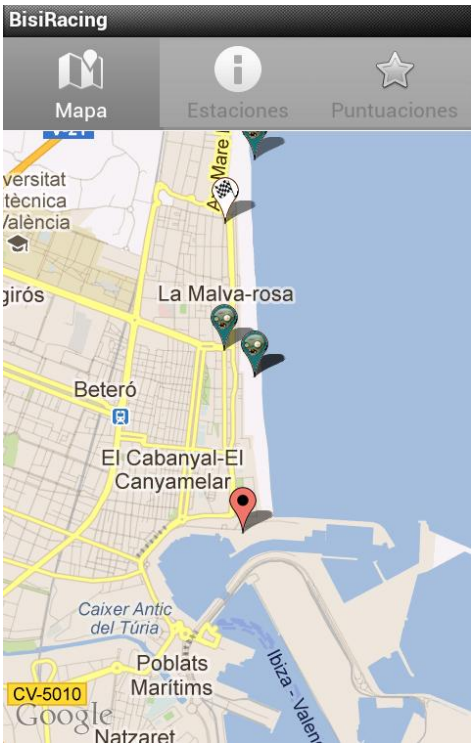


Ilustración 13. Posición inicial.

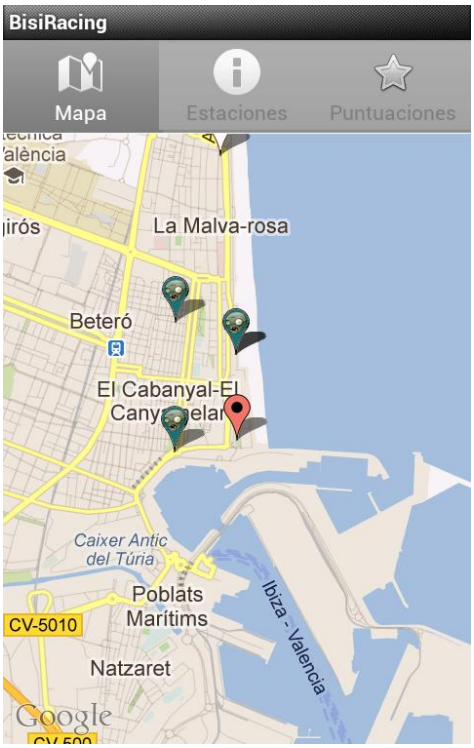


Ilustración 14. Zombis acercándose.

En las siguientes 2 imágenes podemos observar, como cada vez los enemigos se deben acercar más al usuario.

Prueba nº5: Esta prueba consistirá en que, en el modo sofá, donde nosotros toquemos del mapa, la posición deberá seguir el camino hasta donde hemos tocado en el mapa.

Ilustración 15. Toque en un punto.

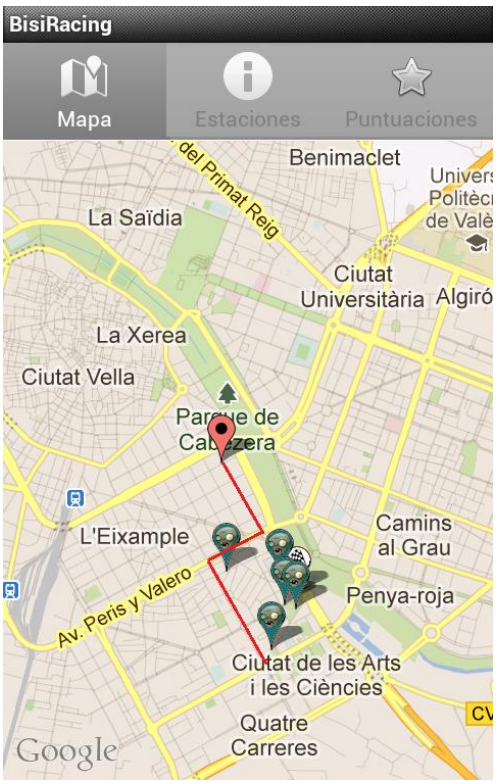


Ilustración 16. Camino del usuario.

Se puede observar en la primera imagen donde hemos tocado la pantalla, y en la segunda imagen como se dirige hacia donde hemos tocado en el mapa.

Prueba nº6: En esta prueba, comprobaremos que las diferentes puntuaciones que consigamos, deberán aparecer en la pestaña de "puntuaciones".

Ilustración 17. Puntuaciones del Usuario.

Podemos observar que las puntuaciones con fondo blanco y gris, estas corresponden a puntuaciones de otras partidas, y las puntuaciones con fondo amarillo corresponden a las puntuaciones actuales de la partida en curso.

9. MANUAL DE USUARIO

Este manual servirá para que el usuario tenga unas nociones de cómo utilizar la aplicación:

Antes que nada, habrá que instalar la aplicación. Para ello nos haremos con el archivo con extensión .apk que obtenemos al compilar nuestros proyecto. Cuando abramos la .apk, aparecerá una pantalla de instalación, detallándonos la información sobre los permisos que la aplicación necesita para ejecutarse.

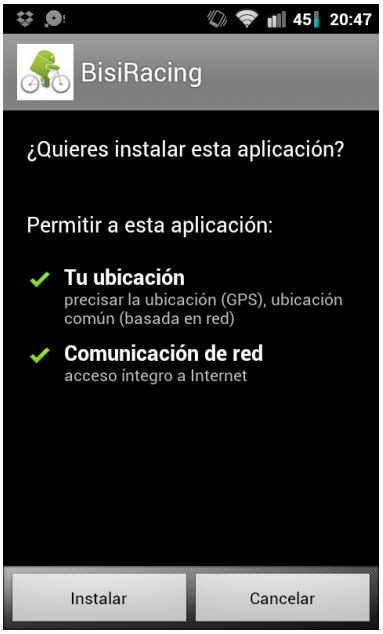


Ilustración 18. Pantalla de instalación.

En la instalación nos informa sobre los siguientes permisos:

- Poder usar el GPS para obtener nuestra posición.
- Poder usar la comunicación de red para poder acceder a Internet.

Una vez aquí deberemos de pulsar el botón Instalar para instalar la aplicación.

Será imprescindible arrancar la aplicación con internet, ya sea mediante 3G o WiFi, ya que la aplicación requiere de internet para poder conectarse con Google Maps, y con la Api de Direcciones de Google para poder generar los diferentes caminos.

Sera necesario tener el Gps conectado si queremos acceder al “Modo Real”.

10. BIBLIOGRAFIA

La bibliografía de este proyecto consta de una serie de páginas web, donde hemos obtenido toda la información sobre las dudas del proyecto, información sobre programación Android, leer XML en Android, Apis de Google Maps y Google Directions. A continuación enumeramos algunas de las paginas que hemos consultado:

- Mapas en Android (I): http://www.sgoliver.net/blog/?p=1949
- Programación de juegos para móviles: **http://www.programacion.com/articulo/programacion_de_juegos_para_moviles_con_j2me_286/8**
- XML datos ValenBisi: http://www.valenbisi.es/service/carto
- Tratamiento de XML en Android (II) SAX Simplificado: http://www.sgoliver.net/blog/?p=1580
- Mapas en Android (III), Overlays (Capas): http://www.sgoliver.net/blog/?p=2004
- Mapas en Android (II), Control MapView: http://www.sgoliver.net/blog/?p=1979
- Localización geográfica en Android (I): http://www.sgoliver.net/blog/?p=1887
- Diálogos: http://developer.android.com/guide/topics/ui/dialogs.html
- Google Directions API: https://developers.google.com/maps/documentation/directions/?hl=es
- Google Maps API Key Android v1 registro: https://developers.google.com/maps/documentation/android/v1/maps-api-signup
- Dibujo múltiples marcadores de ubicación en un MapaVer: http://androidcookbook.com/Recipe.seam;jsessionid=509EB1AD88ECB395598B6F1F5DC8361D?recipeId=2308
- Decodificación polilíneas de Google Maps API Dirección con Java : http://jeffreysambells.com/2010/05/27/decoding-polylines-from-google-maps-direction-api-with-java
- Búsqueda de ubicaciones cercanas a un punto geográfico: http://www.mapanet.eu/resources/Script-Locations.asp#EjemploPractico
- ¿Cómo obtener la latitud y longitud en el mapa en Android? http://stackoverflow.com/questions/4446811/how-to-get-the-latitude-and-longitude-on-map-in-android
- Method Touch Release en Android: http://stackoverflow.com/questions/5765904/touch-release-method-in-android
- ¿Cómo firmar aplicaciones Android?: http://androideity.com/2011/08/25/%C2%BFcomo-firmar-aplicaciones-android/
- Ficheros en Android (I), Memoria Interna: http://www.sgoliver.net/blog/?p=2019

Se accedido por última vez a todos los enlaces a fecha de 22/02/2013.

11. AMPLIACIONES

Ahora vamos a exponer unas posibles ampliaciones que por falta de tiempo no se han podido realizar, pero que sin embargo consideramos que tendrían muy buena cabida en el futuro:

- **Puntuaciones compartidas:** Esta ampliación propuesta, trataría de que cada puntuación de cada jugador que se obtiene, seria guardada en una base de datos global, en la que estarían todas las puntuaciones guardadas de todos los usuarios de la aplicación. En la aplicación se podrían consultar las 25 primeras puntuaciones mundiales, o poder consultar tus propias puntuaciones, para saber a cuanto están tus puntuaciones del Top 25.