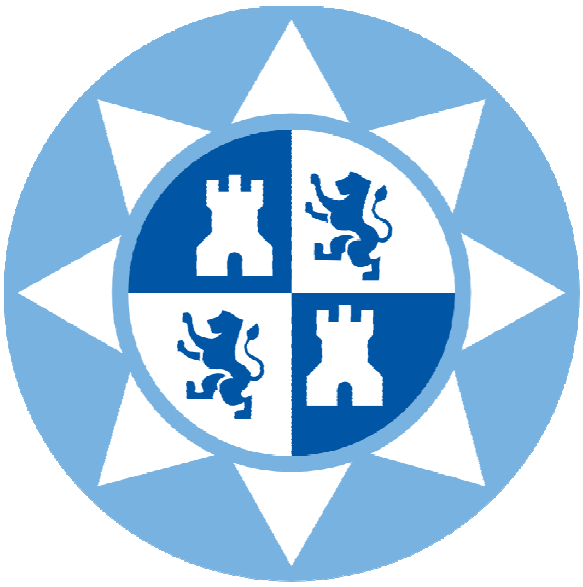
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA



Proyecto Fin de Carrera

Aplicación Android para el posicionamiento y seguimiento mediante códigos QR

AUTOR: Víctor Manuel Fernández López

DIRECTOR: Diego Alonso Cáceres

Abril / 2013

Índice

Índice de figuras

Capítulo 1 – Introducción

La localización *indoor* ofrece múltiples soluciones para diversos ámbitos, como puede ser el seguimiento de los pacientes en hospitales o aplicaciones domóticas para el encendido o apagado de luces o electrodomésticos en función de si la persona está o no en la habitación. Otro objetivo a conseguir es que la solución desarrollada sea fácilmente desplegable y de fácil acceso a los usuarios: facilidad de manejo, funcionamiento fiable y aprovechamiento de la infraestructura ya existente.

En el presente proyecto fin de carrera se tiene como objetivo diseñar una aplicación móvil para dispositivos Android que nos ofrezca poder realizar seguimiento de ubicaciones en espacios interiores. Será una aplicación lo suficientemente flexible como para poder reaccionar de manera correcta a las posibles contingencias que pudieran surgir durante su uso, pero también tendrá como fin ser lo más sencilla posible para mejorar en todo lo posible la experiencia de los usuarios.

Es importante mencionar que la localización *indoor* también presenta problemas, por ejemplo la imposibilidad de determinar la posición mediante señal GPS puesto que ésta no posee la potencia necesaria para llegar al interior de los edificios, o la situación de que a menudo no se dispone de la infraestructura necesaria para poder desplegar una solución o es demasiado gasto para el beneficio que pudiera ofrecer. Por motivos como estos se hace necesario algún otro sistema que permita realizar esta labor de forma sencilla.

Aunque en la presente memoria se aborda desde el principio un nuevo sistema para la localización *indoor*, existen tecnologías que se podrían utilizar para nuestro propósito. Algunos ejemplos son:

- RFID *(Radio Frequency IDentification)*: Es un sistema que transmite y recibe información mediante ondas de radio, sirviéndose de las llamadas etiquetas RFID, unos pequeños dispositivos similares a una pegatina que se pueden adherir a donde se estime oportuno. Contienen antenas para poder recibir y responder a peticiones por radiofrecuencia desde un emisor-receptor RFID.
- Localización wifi: Se basa en recopilar información de las redes wifi que tiene cerca el dispositivo para poder averiguar su posición.
- Redes de sensores: Se trata de redes de pequeños ordenadores equipados con sensores y que colaboran en una tarea común. Se caracterizan por su facilidad de despliegue y por ser autoconfigurables.

En el primer capítulo se abordará el estado de la técnica: se tratará con más detalle las tres alternativas recién propuestas, así como el interés y motivos de la opción que se va a desarrollar. Se hablará también de distintos algoritmos de *pathfinding* (búsqueda de caminos).

Para que esta solución funcione, intervienen diversas tecnologías que trabajan de forma conjunta para lograr el objetivo deseado, y es en el segundo capítulo donde se detallarán todos los detalles inherentes a ellas.

En el tercer capítulo se describirá con todo lujo de detalles la solución desarrollada. Ya que ésta presenta una arquitectura cliente-servidor tendrá un apartado dedicado al primero y otro apartado dedicado al segundo.

A pesar de todo, la solución propuesta tiene algunas limitaciones. En el cuarto y último capítulo se hablará de ellas. También se plantearán posibles ampliaciones y trabajos futuros que puedan continuar el trabajo realizado. Finalmente, se explicarán las conclusiones personales derivadas de este proyecto.

Capítulo 2 – Estado de la técnica

En la introducción se mencionaron tres alternativas para abordar la localización *indoor*. A continuación se explican de manera más detallada.

2.1. RFID

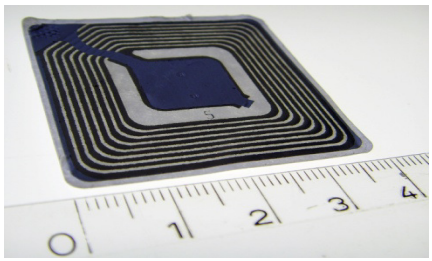


Figura 1: Etiqueta RFID

En la actualidad, la tecnología más extendida para la identificación de objetos es la de los códigos de barras. Sin embargo, éstos presentan algunas desventajas, como la escasa cantidad de datos que pueden almacenar y la imposibilidad de ser reprogramados. La mejora ideada constituyó el origen de la tecnología RFID; consistía en usar chips de silicio que pudieran transferir los datos que almacenaban al lector sin contacto físico, de forma equivalente a los lectores de infrarrojos utilizados para leer los códigos de barras.

RFID (siglas de ***R**adio **F**requency **ID**entification*, en español identificación por radiofrecuencia) es un sistema de almacenamiento y recuperación de datos remoto que usa dispositivos denominados etiquetas, tarjetas, transpondedores o tags RFID. El propósito fundamental de la tecnología RFID es transmitir la identidad de un objeto (similar a un número de serie único) mediante ondas de radio ([6]).

Las etiquetas RFID son unos pequeños dispositivos, similares a una pegatina, que pueden ser adheridas o incorporadas a un producto, un animal o una persona. Contienen antenas para permitirles recibir y responder a peticiones por radiofrecuencia desde un emisor-receptor RFID. Las etiquetas pasivas no necesitan alimentación eléctrica interna, mientras que las activas sí lo requieren. Una de las ventajas del uso de radiofrecuencia (en lugar, por ejemplo, de infrarrojos) es que no se requiere visión directa entre emisor y receptor.

2.1.2. Arquitectura

El modo de funcionamiento de los sistemas RFID es simple. La etiqueta RFID, que contiene los datos de identificación del objeto al que se encuentra adherido, genera una señal de radiofrecuencia con dichos datos. Esta señal puede ser captada por un lector RFID, el cual se encarga de leer la información y enviarla en formato digital a la aplicación específica que utiliza RFID.

Un sistema RFID consta de los siguientes tres componentes:

- Etiqueta RFID o transpondedor: compuesta por una antena, un transductor radio y un material encapsulado o chip. El propósito de la antena es permitirle al chip, el cual contiene la información, transmitir la información de identificación de la etiqueta. Existen varios tipos de etiquetas. El chip posee una memoria interna con una capacidad que depende del modelo y varía de una decena a millares de bytes. Existen varios tipos de memoria:

  - Sólo lectura: el código de identificación que contiene es único y es personalizado durante la fabricación de la etiqueta.
  - De lectura y escritura: la información de identificación puede ser modificada por el lector.
  - Anticolisión: Se trata de etiquetas especiales que permiten que un lector identifique varias al mismo tiempo (habitualmente las etiquetas deben entrar una a una en la zona de cobertura del lector).

- Lector de RFID o transceptor: compuesto por una antena, un transceptor y un decodificador. El lector envía periódicamente señales para ver si hay alguna etiqueta en sus inmediaciones. Cuando capta una señal de una etiqueta (la cual contiene la información de identificación de esta), extrae la información y se la pasa al subsistema de procesamiento de datos.
- Subsistema de procesamiento de datos o middleware RFID: proporciona los medios de proceso y almacenamiento de datos.

2.1.3 Tipos de etiquetas RFID

Las etiquetas RFID pueden ser activas, semipasivas (también conocidas como semiactivas o asistidos por batería) o pasivas. Las etiquetas pasivas no requieren ninguna fuente de alimentación interna y son dispositivos puramente pasivos (sólo se activan cuando un lector se encuentra cerca para suministrarles la energía necesaria. Los otros dos tipos necesitan alimentación, típicamente una pila pequeña.

Etiquetas pasivas

Las etiquetas pasivas no poseen alimentación eléctrica. La señal que les llega de los lectores induce una corriente eléctrica pequeña y suficiente para operar el circuito integrado de la etiqueta, de forma que puede generar y transmitir una respuesta. Estas etiquetas suelen tener distancias de uso práctico comprendidas entre los 10 cm y llegando hasta unos pocos metros, según la frecuencia de funcionamiento y el diseño y tamaño de la antena. Por su sencillez conceptual, son obtenibles por medio de un proceso de impresión de las antenas.

Debido a las preocupaciones por la energía y el coste, la respuesta de una etiqueta pasiva RFID es necesariamente breve, normalmente de apenas un número de identificación. La falta de una fuente de alimentación propia hace que el dispositivo pueda ser bastante pequeño: pueden incluirse en una pegatina o incluso insertarse bajo la piel.

Etiquetas activas

A diferencia de las etiquetas pasivas, las activas poseen su propia fuente autónoma de energía, que utilizan para dar corriente a sus circuitos integrados y propagar su señal al lector. Éstas son mucho más fiables (tienen menos errores) que las pasivas debido a su capacidad de establecer sesiones con el lector. Gracias a su fuente de energía son capaces de transmitir señales más potentes que las de las pasivas, lo que les lleva a ser más eficientes en entornos dificultosos para la radiofrecuencia, como el agua o el metal. También son efectivas a distancias mayores

pudiendo generar respuestas claras a partir de recepciones débiles (al contrario que las pasivas).

Las principales ventaja de las etiquetas RFID activas respecto a las pasivas son el elevado rango de lectura, del orden de decenas de metros, la mayor capacidad de almacenamiento y la habilidad de guardar información adicional enviada por el transceptor. Algunas de ellas integran sensores de registro de temperatura, humedad, luz, vibración y radiación, y es precisamente por motivos como estos que las etiquetas activas sean muy comunes hoy en día a pesar de tener mayor coste. Como desventajas cabe destacar el precio, muy superior al de las pasivas y la dependencia de alimentación por baterías. El tiempo de vida de las baterías depende del modelo de etiqueta y también de su actividad, normalmente del orden de años. Para facilitar la gestión de las baterías es habitual que las etiquetas RFID activas envíen al lector información del nivel de batería, lo que permite sustituir con antelación aquellas que están a punto de agotarse.

Etiquetas semipasivas

Las etiquetas semipasivas se parecen a las activas en que poseen una fuente de alimentación propia, aunque en este caso se utiliza principalmente para alimentar el microchip y no para transmitir una señal. La energía contenida en la radiofrecuencia se refleja hacia el lector como en una etiqueta pasiva. Un uso alternativo para la batería es almacenar información propagada desde el lector para emitir una respuesta en el futuro (las etiquetas sin batería deben responder reflejando energía de la portadora del lector al vuelo).

La batería puede permitir al circuito integrado de la etiqueta estar constantemente alimentado y eliminar la necesidad de diseñar una antena para recoger potencia de una señal entrante. Las etiquetas RFID semipasivas responden más rápidamente, por lo que son más fuertes en el ratio de lectura que las pasivas.

Estas etiquetas tienen una fiabilidad comparable a la de las activas, a la vez que pueden mantener el rango operativo de una pasiva, y también suelen durar más tiempo que las activas.

2.1.4. Tipos de antena

El tipo de antena utilizado en la etiqueta depende de la aplicación para la que esté diseñado y de la frecuencia de operación. Las etiquetas de baja frecuencia (*LF*, del inglés *low frequency*) normalmente se sirven de la inducción electromagnética. Como el voltaje inducido es proporcional a la frecuencia, se puede producir el necesario para alimentar un circuito integrado utilizando un número suficiente de espiras. Existen etiquetas LF compactas (como las encapsuladas en vidrio, utilizadas para identificación humana y animal) que utilizan una antena en varios niveles (tres de 100-150 espiras cada uno) alrededor de un núcleo de ferrita.

En alta frecuencia (*HF*, 13.56 MHz) se utiliza una espiral plana con 5-7 vueltas y un factor de forma parecido al de una tarjeta de crédito para lograr distancias de decenas de centímetros. Estas antenas son más baratas que las LF ya que pueden producirse por medio de litografía en lugar de espiración, aunque son necesarias dos superficies de metal y una aislante para realizar

la conexión cruzada del nivel exterior al interior de la espiral, donde se encuentran el condensador de resonancia y el circuito integrado.

Las etiquetas pasivas en frecuencia ultra alta (*UHF*) y de microondas suelen acoplarse por radio a la antena del lector y utilizar antenas clásicas de dipolo. Sólo es necesaria una capa de metal, lo que reduce el coste. Las antenas de dipolo, no obstante, no se ajustan muy bien a las características de los circuitos integrados típicos (con alta impedancia de entrada, ligeramente capacitiva). Se pueden utilizar dipolos plegados o bucles cortos como estructuras inductivas complementarias para mejorar la alimentación. Los dipolos de media onda (16 cm a 900 MHz) son demasiado grandes para la mayoría de aplicaciones (por ejemplo las etiquetas RFID para uso en etiquetas no pueden medir más de 10 cm), por lo que hay que doblar las antenas para satisfacer las necesidades de tamaño. También pueden usarse estructuras de banda ancha. La ganancia de las antenas compactas suele ser menor que la de un dipolo (menos de 2 dB) y pueden considerarse isótropas en el plano perpendicular a su eje.

Los dipolos experimentan acoplamiento con la radiación que se polariza en sus ejes, por lo que la visibilidad de una etiqueta con una antena de dipolo simple depende de su orientación. Las etiquetas con dos antenas ortogonales (etiquetas de doble dipolo) dependen mucho menos de ella y de la polarización de la antena del lector, pero suelen ser más grandes y caras que sus contrapartidas simples.

Pueden usarse antenas de parche (*patch*) para dar servicio en las cercanías de superficies metálicas, aunque es necesario un grosor de 3 a 6 mm para lograr un buen ancho de banda, además de que es necesario tener una conexión a tierra que incrementa el coste comparado con estructuras de una capa más sencillas.

2.1.5. Uso actual y aplicaciones potenciales

Dependiendo de las frecuencias utilizadas en los sistemas RFID, el coste, el alcance y las aplicaciones son diferentes. Los sistemas que emplean frecuencias bajas tienen igualmente costes bajos, pero también baja distancia de uso. Los que emplean frecuencias más altas proporcionan distancias mayores de lectura y velocidades de lectura más rápidas. Así, las de baja frecuencia se utilizan comúnmente para la identificación de animales, seguimiento de barricas de cerveza, o como llave de automóviles con sistema antirrobo. En ocasiones se insertan en pequeños chips en mascotas, para que puedan ser devueltas a su dueño en caso de pérdida. Las etiquetas RFID de alta frecuencia se utilizan en bibliotecas y seguimiento de libros, seguimiento de palés, control de acceso en edificios, seguimiento de equipaje en aerolíneas, seguimiento de artículos de ropa y últimamente en pacientes de centros hospitalarios para hacer un seguimiento de su historia clínica. Otro uso muy extendido de las etiquetas de alta frecuencia es como identificación de acreditaciones, sustituyendo a las tarjetas de banda magnética: sólo es necesario acercar estas insignias a un lector para identificar al portador.

Las etiquetas RFID se ven como una alternativa que reemplazará a los códigos de barras UPC o EAN, puesto que tienen un importante número de ventajas importantes sobre la arcaica tecnología de código de barras, aunque quizás no logren sustituir en su totalidad a los códigos de barras, debido en parte a su coste más alto. Para algunos artículos con un coste más bajo la

capacidad de cada etiqueta de ser única se puede considerar exagerada, aunque tendría algunas ventajas tales como una mayor facilidad para llevar a cabo inventarios.

También se debe reconocer que el almacenamiento de los datos asociados al seguimiento de las mercancías a nivel de artículo ocuparía muchos terabytes. Es mucho más probable que las mercancías sean seguidas a nivel de palés usando etiquetas RFID, y a nivel de artículo con producto único, en lugar de códigos de barras únicos por artículo.

Los códigos RFID son tan largos que cada etiqueta RFID puede tener un código único, mientras que los códigos UPC actuales se limitan a un solo código para todos los casos de un producto particular. La unicidad de las etiquetas RFID significa que un producto puede ser seguido individualmente mientras se mueve de lugar en lugar, terminando finalmente en manos del consumidor. Esto puede ayudar a las compañías a combatir el hurto y otras formas de pérdida del producto. También se ha propuesto utilizar RFID para comprobación de almacén desde el punto de venta, y sustituir así al encargado de la caja por un sistema automático que no necesite ninguna captación de códigos de barras. Sin embargo no es probable que esto sea posible sin una reducción significativa en el coste de las etiquetas actuales. Se está llevando a cabo una investigación sobre la tinta que se puede utilizar como etiqueta RFID, que reduciría costes de forma significativa, pero faltan todavía algunos años para que esto dé sus frutos.

Una posible utilización en el sector sanitario es la localización de expedientes clínicos, dentro de un entorno masivo o de almacenes descentralizados, es decir, en almacenes fuera del hospital. La gestión de inventario y la localización se pueden mejorar altamente obteniendo resultados increíbles con sólo poner un chip de RFID en los mismos. Además con los dispositivos de lectura masiva, se puede garantizar el 100% de lectura de los expedientes clínicos y conseguir la trazabilidad completa sin problemas y de una manera muy sencilla.

Otra aplicación propuesta es el uso de RFID para señales de tráfico inteligentes en la carretera. Se basa en el uso de transpondedores RFID enterrados bajo el pavimento (radiobalizas) que son leídas por una unidad que lleva el vehículo que filtra las diversas señales de tráfico y las traduce a mensajes de voz o lo muestra en el salpicadero. Su principal ventaja frente a los sistemas basados en satélite es que las radiobalizas no necesitan de mapeado digital ya que proporcionan el símbolo de la señal de tráfico y la información de su posición por sí mismas. Las radiobalizas RFID también son útiles para complementar sistemas de posicionamiento de satélite en lugares como los túneles o interiores, o en el guiado de personas ciegas.

2.1.6. Polémicas sobre su utilización

El uso de la tecnología RFID ha causado una considerable polémica e incluso boicots de productos. Las tres razones principales por las que RFID resulta preocupante en lo que a privacidad se refiere son:

- El comprador de un artículo no tiene por qué saber de la presencia de la etiqueta o ser capaz de eliminarla.
- La etiqueta puede ser leída a cierta distancia sin el consentimiento del individuo.
- Si un artículo etiquetado es pagado mediante tarjeta de crédito o conjuntamente con el uso de una tarjeta de fidelidad, entonces sería posible enlazar la ID única de ese artículo con la identidad del comprador.

La mayoría de las preocupaciones giran alrededor del hecho de que las etiquetas RFID puestas en los productos siguen siendo funcionales incluso después de que se hayan comprado los productos y se hayan llevado a casa, y esto puede utilizarse para vigilancia y otros propósitos cuestionables sin relación alguna con sus funciones de inventario en la cadena de suministro. Aunque la intención es emplear etiquetas RFID de corta distancia, éstas pueden ser interrogadas a mayores distancias por cualquier persona con una antena de alta ganancia, permitiendo de forma potencial que el contenido de una casa pueda ser explorado desde cierta distancia. Incluso un escaneado de rango corto es preocupante si todos los artículos detectados aparecen en una base de datos cada vez que una persona pasa un lector, o si se hace de forma malintencionada (por ejemplo, un robo empleando un escáner de mano portátil para obtener una evaluación instantánea de la cantidad de víctimas potenciales). Con números de serie RFID permanentes, un artículo proporciona información inesperada sobre una persona incluso después de su eliminación, por ejemplo los artículos que se revenden o se regalan, pudiendo permitir trazar la red social de una persona.

Otro problema referente a la privacidad es debido al soporte para un protocolo de anticolisión. Esta es la razón por la cual un lector puede enumerar todas las etiquetas que responden a él sin que se interfieran entre sí. La estructura de la versión más común de este protocolo es tal que todos los bits del número de serie de la etiqueta salvo el último se pueden deducir por *eavesdropping* (detección a distancia) pasivo tan sólo en la parte del protocolo que afecta al lector. Por esta razón, si las etiquetas RFID están cerca de algún lector, la distancia en la cual la señal de una etiqueta puede ser *escuchada* es irrelevante. Lo que importa es la distancia a la que un lector de mucho más alcance puede recibir la señal. Independientemente de que esto dependa de la distancia a la que se encuentre el lector y de qué tipo sea, en un caso extremo algunos lectores tienen una salida de energía máxima (4 W) que se podría recibir a diez kilómetros de distancia.

Blindajes Faraday como contramedida a RFID

Si se rodeara un dispositivo RFID con una caja de Faraday tendría señales entrantes y salientes muy atenuadas, hasta el punto de que no podrían ser utilizables. Un blindaje de Faraday muy sencillo, válido para la mayoría de los propósitos, sería un envoltorio de papel de aluminio. Uno más efectivo sería un rectángulo de cobre alrededor del objeto. Un RFID implantado sería más difícil de neutralizar con dicho blindaje, pero incluso una simple cubierta de papel de aluminio atenuaría la componente de campo eléctrico de las señales.

Neutralizar permanentemente el RFID podría necesitar una fuerte corriente eléctrica alterna adyacente al RFID, que sobrecargue la etiqueta y destruya su electrónica. En algunos casos, dependiendo de la composición del RFID, un imán fuerte puede servir para destruir mecánicamente la bobina o la conexión del chip por la fuerza mecánica ejercida en la bobina. Con el desarrollo de la tecnología RFID, pueden ser necesarios otros métodos.

Las etiquetas de 125 y 134 KHz (baja frecuencia), y en varios casos de 13.56 MHz (alta frecuencia) están unidas por un campo magnético en lugar de un campo eléctrico, es lo que se denomina acoplamiento inductivo. Como la caja de Faraday blinda solamente del campo electromagnético, el blindaje de papel de aluminio es ineficaz. Cualquier blindaje magnético,

como por ejemplo una hoja fina de hierro o acero, encapsulando la bobina de la antena de la etiqueta, será eficaz.

2.2. Localización WI-FI

Mediante este sistema, cualquier dispositivo equipado con tecnología Wi-Fi puede determinar su posición en base a las redes vecinas que esté recibiendo en un momento concreto. Este sistema es muy útil para el posicionamiento en zonas urbanas y dentro de edificios en los que la cobertura GPS es escasa.

Si se opera con un teléfono móvil o PDA se puede ver que es capaz de determinar su posición sobre un mapa sin necesidad de usar GPS. Pero, ¿cómo es esto posible? La razón está detrás de la empresa Skyhook Wireless y su tecnología WPS (*WiFi Position System*) que comenzó a implementar con éxito en la primera generación del iPhone y en los iPod Touch (se debe mencionar también que Skyhook no es la única empresa que se dedica a esta tarea, hay otras como pueden ser Google, Apple o Microsoft).

Figura 2: Teléfono con localización WIFI

Cada punto de acceso WiFi (el router de una vivienda, el de una oficina, el de lugares públicos, etc) está emitiendo continuamente una señal con su dirección MAC, que es un número único para cada uno de ellos que permite distinguir unos de otros. Aunque para tener una buena cobertura y poder conectarnos tenemos que estar a menos de 50 metros, la señal es recibida por los dispositivos WiFi, aunque sea débilmente, hasta unos 500 metros de distancia.

Aquí se encuentra la clave de esta técnica, en una calle de cualquier ciudad o en el interior de un edificio podemos encontrar un gran número de puntos WiFi en 500 metros a la redonda, con lo que si tenemos acceso a internet mediante alguno de ellos, tan sólo tendríamos que consultar en qué zona del mapa se encuentran los puntos recibidos y esa será nuestra posición.

Para tener esta información desde Skyhook se han encargado de recorrer miles de calles de ciudades de todo el mundo detectando las señales WiFi procedentes de todos los routers de domicilios, tiendas, oficinas, plazas públicas, etc. y guardando en una base de datos la posición de cada uno de ellos, según el GPS de sus vehículos.

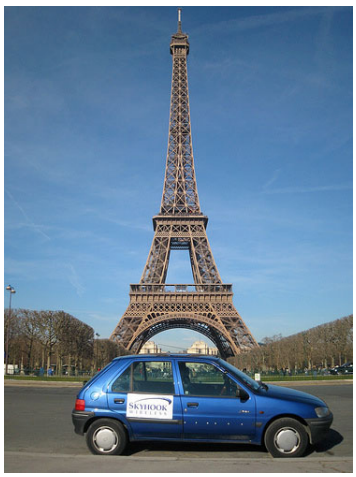


Figura 3: Coche de Skyhook

De esta forma, para averiguar la posición, nuestro sistema WiFi tan sólo tendrá que escanear todas las redes WiFi a su alrededor, consultar con la base de datos de Skyhook esas redes y ésta le devolverá su posición aproximada.

La base de datos de posiciones está en continuo crecimiento

pues si alguien se mudara o bien comprara un nuevo router no catalogado, al conectar un equipo y detectar sus redes vecinas le enviaría también la información sobre la nueva red detectada, añadiéndose su posición si es un elemento nuevo o bien modificándola si ya existía en otro lugar.

Evidentemente esta técnica de posicionamiento tiene sus ventajas pero también desventajas frente a otras como el GPS:

- Es una tecnología muy barata y extendida, todos los ordenadores portátiles y muchos teléfonos móviles tienen WiFi y hay gran número de routers extendidos por todo el mundo.
- La precisión de este sistema es de 20-30 metros frente a los 8-10 del GPS, sin embargo la cobertura en zonas urbanas y en el interior de edificios es mucho mayor.
- El sistema tiene poca o nula cobertura en zonas rurales donde no existen puntos de acceso WiFi de referencia y donde el GPS si llega.

Se puede probar el posicionamiento por WiFi desde el ordenador o el teléfono móvil mediante Google Maps, y para ello sólo se necesita estar conectado a una red WiFi y usar un navegador compatible como Firefox 3.5 o Google Chrome.

Otra alternativa para la localización mediante Wi-Fi es la solución desarrollada por la empresa Ekahau. Ekahau es una empresa con sede en Estados Unidos fundada en 2000 como una *spin-off* de la universidad de Helsinki (Finlandia), y que se dedica a la manufactura de sistemas de localización en tiempo real. Ha desarrollado una solución para la localización *indoor* utilizando redes Wi-Fi, con la excepcional característica de que se adapta a la totalidad de ellas (es compatible con todos los routers y puntos de acceso). Este sistema se denomina RTLS (*Real Time Location System*) y se detalla en [5].

La principal característica de esta tecnología es que no se necesita instalar infraestructuras adicionales, y que trabaja en cualquier punto donde haya cobertura Wi-Fi. Además tiene el valor añadido de que es fácilmente desplegable ya que sólo son necesarios cuatro pasos:

- Recorrer el edificio con un ordenador portátil ejecutando un software (también desarrollado por Ekahau) cuyo propósito es crear un mapa del entorno Wi-Fi.
- Una vez recopilados todos los datos necesarios, se transfieren a un *RTLS Controller* (un servidor central) que se encargará de coordinar el sistema a la vez que realiza el seguimiento.
- Activar las etiquetas Wi-Fi (diferentes tipos de dispositivos utilizados para que el servidor pueda monitorizar la posición).
- Utilizar el software Ekahau Vision para configurar las reglas de la lógica de negocio, como las alertas, la información adicional de las etiquetas y los puntos de interés.

Al no necesitarse cables para implantar este sistema, repercute directamente en menores costes y tiempo de despliegue (hacerlo mediante cables implicaría semanas o incluso meses).

¿Cómo funciona RTLS?

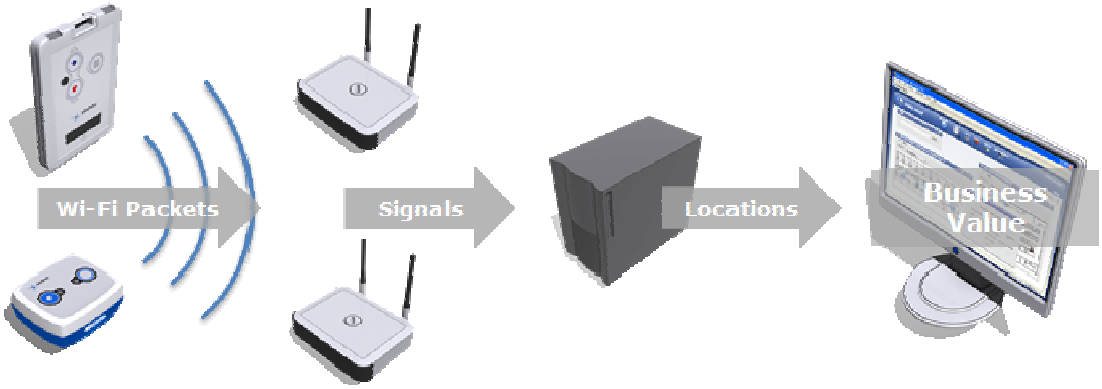


Figura 4: Sistema RTLS

Las etiquetas Wi-Fi transmiten periódicamente unos pocos paquetes Wi-Fi bajo el estándar 802.11, o cuando se pulsa un botón o hay movimiento. La red Wi-Fi mide la señal y envía esa información al *RTLS Controller*, que calculará la posición precisa de las etiquetas mediante algoritmos estadísticos de cálculo de potencia de señal. Una vez el servidor tiene los datos los transmite a la aplicación Ekahau Vision, que se encargará de recibir los datos de localización, alertas y mapas para mostrarlos en tiempo real, así como realizar avisos y generar informes cuando sea necesario.

Los dispositivos Wi-Fi seguidos son en la mayoría pequeñas etiquetas alimentadas por baterías, y que incluyen características avanzadas como botones, luces, sensores de movimiento y sonidos de alarma. También se pueden seguir *smartphones*, tabletas, ordenadores portátiles y teléfonos *VoIP.*

Etiquetas Wi-Fi

La empresa ofrece una amplia selección de etiquetas Wi-Fi RFID para el seguimiento de la localización de bienes o personas y monitorización de temperatura. Se pueden usar desde en sanidad hasta en aplicaciones industriales para seguir bienes o personas, proveyendo comunicación en doble sentido. Ofrecen un largo tiempo de batería y operan incluso en entornos en los que el equipamiento es de distintos fabricantes. Algunos tipos de etiquetas ofrecidos se detallan a continuación.

Figura 5: Etiqueta A4

La A4 es una etiqueta activa de localización que ofrece una alta precisión utilizando las redes Wi-Fi existentes. Es resistente al polvo y a las salpicaduras y tiene un pequeño tamaño, comparable al tamaño de una pequeña caja de cerillas. Dispone de dos botones configurables que se pueden usar para enviar alertas o notificaciones. También incluye un sensor de forzado que envía alertas al software de que la etiqueta ha sido despegada del objeto al que estuviera colocada. Esta etiqueta puede ser configurada y utilizada de forma remota, y su baja potencia de radio y sensor de movimiento inteligente pueden ofrecer una duración de batería de hasta cinco años.

Figura 6: Etiqueta B4

Esta etiqueta, en forma de placa, es también un buscapersonas que permite a los usuarios enviar y recibir mensajes de texto. Estos mensajes pueden basarse en eventos, estados, localización u otras reglas de negocio establecidas en el sistema. Las aplicaciones de terceros también pueden enviar y recibir mensajes hacia el buscapersonas o hacia él, gracias a las API abiertas de Ekahau. Además es una etiqueta recargable, lo que elimina la necesidad de almacenar y cambiar baterías.

Figura 7: Etiqueta T301W

La T301W es la primera etiqueta transportable basada en Wi-Fi de la industria. Está diseñada para ser llevada por pacientes, empleados, niños u otros individuos, y permite visibilidad a tiempo real de la localización exacta de una persona. Su diseño permite llevarla cómodamente en la muñeca o en el tobillo y al ser a prueba de agua permite su desinfección a fondo después de su uso, así como también permite al usuario ducharse o bañarse tranquilamente llevándola puesta. Incluye un LED bicolor, alarma por vibración para alertar a la persona que la lleva y un botón de llamada personalizable para enviar mensajes de emergencia o de estado.

Figura 8: Etiqueta L4

La etiqueta L4 es un dispositivo de seguimiento de localizaciones diseñado para evitar el robo o extravío de ordenadores portátiles, que se inserta en la ranura ExpressCard. Cuando el equipo abandona una zona designada o su ubicación difiere de otra manera de un patrón preestablecido, inmediatamente se envía una señal al personal designado como administradores o seguridad. La alerta también incluye la identidad y ubicación del ordenador, y puede ser enviada a varios equipos incluyendo los buscapersonas B4, ordenadores fijos o portátiles, correo electrónico o teléfonos inalámbricos. La etiqueta está equipada con una batería interna recargable que permite el seguimiento de la ubicación incluso con el ordenador portátil apagado, así como un sensor de forzado para detectar que está siendo desconectada del equipo.

Figura 9: Etiqueta HS1

Monitorizar la humedad y la temperatura es vital en áreas sensibles ambientalmente en habitaciones de hospital, salas de abastecimiento quirúrgico, salas de almacenamiento médico y en diversos procesos industriales. Asegurando la humedad y temperatura correctas se puede prevenir el crecimiento de moho, ácaros y hongos. La etiqueta de humedad HS1 proporciona una manera automatizada para

medir y monitorizar la humedad y temperatura relativas a su alrededor. Combinada con la aplicación Ekahau Vision, avisa automáticamente cuando la humedad o temperatura monitorizada se sale de un rango establecido.

Figura 10: Etiqueta TS2

Las etiquetas TS1 y TS2 son etiquetas inalámbricas para la medición, la monitorización y los informes de temperaturas de congeladores y frigoríficos en hospitales y otros entornos donde el control de la temperatura es crítico. La TS1 viene con una sonda que se introduce en el congelador o frigorífico mientras la etiqueta en sí se localiza fuera del entorno frío para mejores mantenimiento y conectividad de red. Estos dispositivos cuentan con sensores integrados para gestionar el muestreo frecuente de la temperatura y poder informar inmediatamente de cualquier anomalía detectada.

Controlador RTLS

El *Ekahau RTLS Controller* (ERC) es el punto neurálgico de esta tecnología. Dispone tanto de una API para desarrolladores de aplicaciones, software de terceros y *middleware* como de otra API para la integración con redes Wi-Fi, etiquetas, Tablet PCs, PDAs y *smartphones* entre otros. El ERC se basa completamente en web y ofrece capacidades completas para la monitorización de sistemas y gestión de etiquetas. Sus características clave son:

- Funciona sobre cualquier red Wi-Fi: El sistema soporta todos los estándares 802.11a/b/g/n de puntos de acceso Wi-Fi y no necesita hardware propietario.
- Rendimiento: Un único servidor puede seguir más de 20000 objetos e individuos en tiempo real a través de múltiples plantas, edificios o incluso edificios geográficamente dispersos.
- API abierta: El ERC ha sido diseñado para integrarse en el *middleware*, flujo de trabajo, bases de datos u otros sistemas ya existentes en las empresas. La API funciona mediante HTTP/XML y proporciona acceso a todas las funciones de RTLS.
- Configuración y gestión de etiquetas: Proporciona visibilidad total del estado y configuración de todas las etiquetas Wi-Fi de la red. Permite también gestión *over-the-air* de las etiquetas para cambiar los ajustes de los sensores, tasas de parpadeo, etc. y aprovecha la red Wi-Fi para enviar alertas a las etiquetas.

Aplicación Ekahau Vision

La aplicación Ekahau Vision es una plataforma que junto con las etiquetas Wi-Fi da a los administradores una visibilidad sin precedentes de la localización, condición y estado de objetos y personas a través de los edificios mediante un navegador web. Tiene algoritmos y análisis de localización Wi-Fi e iconos visuales para mostrar la localización en el edificio, planta, zona y habitación de los objetos y personas seguidas, utilizando los planos reales del edificio. Cada usuario puede personalizar la interfaz de la aplicación para proporcionar inteligencia empresarial que tiene la facultad de mejorar los flujos de trabajo, la eficiencia operacional y la seguridad del espacio de trabajo.

Entre los múltiples beneficios y características que ofrece esta aplicación se encuentran los siguientes:

- Gestión mejorada de inventarios
- Gestión automatizada del flujo de trabajo
- Informes de tendencias y cuadros de mando personalizados
- Cumplimiento automático de la normativa ambiental
- Garantía de seguridad con comunicación bidireccional instantánea
- Escalabilidad para cualquier tamaño de empresa
- Facilidad de manejo y activación
- Buscador flexible de objetos
- Informes exhaustivos
- Diversas opciones de alarma
- Entrega confiable de alertas
- Monitorización efectiva de temperatura y humedad

2.3. Redes de sensores

Una red de sensores (del inglés *sensor network*) es una red de pequeños ordenadores (denominados nodos) equipados con sensores que colaboran en una tarea común. Estas redes están formadas por un grupo de sensores con ciertas capacidades sensitivas y de comunicación inalámbrica, los cuales permiten formar redes *ad hoc* sin infraestructura física preestablecida ni administración central.

Las redes de sensores son un concepto relativamente nuevo en adquisición y tratamiento de datos con múltiples aplicaciones en distintos campos tales como entornos militares o industriales, domótica o detección ambiental. Se caracterizan por su facilidad de despliegue y por ser autoconfigurables, pudiendo convertirse en todo momento en emisor y receptor, ofrecer servicios de encaminamiento entre nodos sin visión directa, así como registrar datos referentes a los sensores locales de cada nodo. Otra de sus características es su gestión eficiente de la energía, que les permite obtener una alta tasa de autonomía que las hacen plenamente operativas.

La creciente miniaturización de ordenadores ha dado a luz la idea de desarrollar computadores extremadamente pequeñas (del tamaño de un grano de arena o incluso una partícula de polvo) y baratas que se comunican de forma inalámbrica y se organizan autónomamente. Mediante sus sensores, circuitos y tecnología de comunicación bidireccional sin hilos los dispositivos recopilarían datos, realizarían cómputos y se comunicarían por radio con otros en distancias que se acercan a los 300 metros, y que cuando están muy cercanos crean automáticamente redes altamente flexibles de baja potencia.

Actualmente las redes de sensores son un tema muy activo en investigación en varias universidades, aunque ya empiezan a existir aplicaciones comerciales basadas en ellas. Una de las redes de sensores más grandes desplegadas es una que consistió en 800 nodos y que se

desplegó el 27 de agosto de 2001 en la universidad de Berkeley para demostrar la potencia de esta tecnología.

2.3.1. Áreas de aplicación

La evolución de las redes de sensores tiene su origen en iniciativas militares. Como predecesor de las redes de sensores modernas se considera a *Sound Surveillance System (SOSUS)*, una red de boyas sumergidas instaladas en los Estados Unidos durante la guerra fría para detectar submarinos usando sensores de sonido.

Aunque su origen sea militar, las redes de sensores tienen usos civiles interesantes, como los siguientes:

- Eficiencia energética: Las redes de sensores se utilizan para controlar el uso eficaz de la electricidad.
- Entornos de alta seguridad: Existen lugares que requieren altos niveles de seguridad para evitar ataques terroristas, tales como centrales nucleares, aeropuertos, edificios del gobierno de paso restringido. Gracias a una red de sensores se pueden detectar situaciones que con una simple cámara sería imposible.
- Sensores ambientales: El control ambiental de vastas áreas de bosque o de océano, sería imposible sin las redes de sensores que controlen múltiples variables, como temperatura, humedad, fuego, actividad sísmica u otras. También ayudan a los expertos a diagnosticar o prevenir un problema o urgencia y además minimizan el impacto ambiental de la presencia humana.
- Sensores industriales: Dentro de las fábricas existen complejos sistemas de control de calidad y el tamaño de estos sensores les permite estar allí donde se requiera.
- Automoción: Las redes de sensores son el complemento ideal a las cámaras de tráfico, ya que pueden informar de la situación del tráfico en ángulos muertos que no cubren las cámaras y también pueden informar a conductores de la situación, en caso de atasco o accidente, con lo que estos tienen capacidad de reacción para tomar rutas alternativas.
- Medicina: Es otro campo bastante prometedor. Con la reducción de tamaño que están sufriendo los nodos, la calidad de vida de pacientes que tengan que tener controladas sus constantes vitales (pulsaciones, presión, nivel de azúcar en sangre, etc), podrá mejorar sustancialmente.
- Domótica: Su tamaño, economía y velocidad de despliegue la hacen una tecnología ideal para domotizar el hogar a un precio asequible.

2.3.2. Características de las redes de sensores

Estas redes tienen una serie de características, unas propias y otras adaptadas de las redes *ad hoc*. Son las siguientes:

- Topología Dinámica: En una red de sensores, la topología siempre es cambiante y éstos tienen que adaptarse para poder comunicar nuevos datos adquiridos.

- Variabilidad del canal: El canal radio es un canal muy variable en el que existen una serie de fenómenos como pueden ser la atenuación, desvanecimientos rápidos, desvanecimientos lentos e interferencias que puede producir errores en los datos.

- No se utiliza infraestructura de red: Una red de sensores no tiene necesidad alguna de infraestructura para poder operar, ya que sus nodos pueden actuar de emisores, receptores o enrutadores de la información. Sin embargo, hay que destacar en el concepto de red de sensores la figura del nodo recolector (también denominados *sink node*), que es el nodo que recolecta la información y por el cual se recoge la información generada normalmente en tiempo discreto. Esta información generalmente es adquirida por un ordenador conectado a este nodo y es sobre el ordenador que recae la posibilidad de transmitir los datos por tecnologías inalámbricas o cableadas según sea el caso. Esto presenta un problema: si se usa una topología centralizada y el nodo central fallara, la red entera se colapsaría. Así, se puede incrementar la fiabilidad de la red utilizando arquitectura distribuida. Ésta se usa cuando los nodos son susceptibles de fallar, para mejorar la recogida de datos y para dotar a los nodos de copia de seguridad en caso de fallas o averías en el nodo central.

- Tolerancia a errores: Un nodo dentro de una red de sensores tiene que ser capaz de seguir funcionando a pesar de tener errores en el sistema propio.

- Comunicaciones multisalto o broadcast: Siempre es característico el uso de algún protocolo que permita comunicaciones multisalto, aunque también es muy común utilizar mensajería basada en broadcast.

- Consumo energético: Es uno de los factores más sensibles debido a que tienen que conjugar autonomía con capacidad de proceso, ya que actualmente cuentan con una unidad de energía limitada. Un nodo sensor tiene que contar con un procesador de consumo ultra bajo así como de un transceptor radio con la misma característica, a esto hay que agregar un software que también conjugue esta característica haciendo el consumo aún más restrictivo.

- Limitaciones hardware: Para poder conseguir un consumo ajustado, se hace indispensable que el hardware sea lo más sencillo posible, así como su transceptor radio, esto nos deja una capacidad de proceso limitada.

- Costes de producción: Dado que la naturaleza de una red de sensores tiene que ser mediante el despliegue de un número muy elevado de nodos para poder obtener datos con fiabilidad, una vez definida su aplicación los dispositivos son económicos de producir si son fabricados en grandes cantidades.

2.3.3. Software

La energía es el recurso más escaso de las redes de sensores, y es quien determina su tiempo de vida. Las redes de sensores están pensadas para ser desplegadas en gran número en varios

entornos, incluyendo zonas remotas y hostiles donde las comunicaciones son *ad hoc* son clave. Por este motivo los algoritmos y protocolos deben lidiar con estas cuestiones:

- Maximización del tiempo de vida: El consumo de energía del sensor debe minimizarse y los nodos deben ser muy eficientes debido a que su energía limitada determina su tiempo de vida. Para ahorrar energía el nodo debería desactivar la alimentación de radio cuando no se esté utilizando.
- Robustez y tolerancia a fallos: Debido a que las redes de sensores se despliegan en gran número pero en entornos muy dispares, se hace necesario que el software sea capaz de responder adecuadamente a los fallos que puedan producirse.
- Autoconfiguración: Por el mismo motivo que la robustez y tolerancia a fallos, el software debe ser capaz de adaptarse a las distintas características de la red a medida que estas cambien su configuración para adaptarse al medio.

2.3.4. Sistema operativo

Los sistemas operativos para redes de sensores suelen ser menos complejos que los sistemas operativos de propósito general. Se parecen mucho a los sistemas embebidos por dos razones: la primera es que estas redes se suelen desplegar con una aplicación particular en mente más que como una plataforma general; la segunda es que la necesidad de bajo coste y baja potencia conduce a que los sensores tengan microcontroladores de baja potencia que aseguren que mecanismos como la memoria virtual son o bien innecesarios o bien demasiado caros de implementar.

Sería posible utilizar sistemas operativos como *eCos* o *µC/OS* en las redes de sensores. Sin embargo estos sistemas están a menudo diseñados con características de tiempo real. Otros sistemas son *LiteOS*, un sistema desarrollado recientemente que aporta abstracción similar a UNIX y soporte al lenguaje de programación.

Mención especial merece el sistema *TinyOS*, por ser quizá el primer sistema operativo diseñado específicamente para redes de sensores inalámbricos. Este sistema está basado en una arquitectura dirigida por eventos en lugar de multihilo. Los programas en *TinyOS* están compuestos por manejadores de eventos y tareas con semánticas de "ejecutar hasta terminar". Cuando ocurre un evento externo, como la recepción de un paquete de datos o la lectura de un sensor, el sistema operativo envía una señal al manejador apropiado para procesarlo.

2.4. ¿Por qué se ha elegido la opción propuesta?

La solución propuesta se compone de varias tecnologías, que se explicarán más adelante. Cobra especial interés la tecnología del sistema operativo de Google para teléfonos móviles: Android. Se ha decidido utilizar esta tecnología para desarrollar la solución por varios motivos:

- Difusión: la gran penetración que ha tenido Android en el mercado es muy grande. Según IDC (empresa dedicada a estudios de mercado en tecnologías de la información y las comunicaciones) el tercer trimestre de 2012 tres de cada cuatro *smartphones* vendidos contaba con el sistema de Google (más concretamente, en dicho trimestre se

vendieron 181.1 millones de terminales y el 75% de ellos contaba con Android, como se muestra en [1]).

- Apertura: al ser un sistema de base Linux, y por tanto libre, deja todas las puertas abiertas a que cualquier persona que lo desee pueda modificar el sistema como estime oportuno, dando lugar a una personalización nunca vista anteriormente en los teléfonos móviles.
- Adaptabilidad: puede que Android esté diseñado principalmente para teléfonos móviles, pero al estar basado en Linux (sistema de propósito general) las opciones de integración en otros sistemas aumentan. Como ejemplo se pueden mencionar ordenadores portátiles como el que se expone en [3], el sistema Google TV mostrado en [2], tabletas electrónicas o incluso cámaras fotográficas como la reciente Samsung Galaxy Camera ([4]).
- Base Java: las aplicaciones para Android se programan en una combinación de Java y XML, por lo que se facilita enormemente su desarrollo. Es también este un motivo para decantarse por Android: aprovechar la sólida base Java ya poseída.

Es evidente el interés que despierta Android para desarrollar esta solución, pero el sistema de Google no es el único motivo. Existen aplicaciones Android para leer y decodificar códigos QR (de los que hablaremos más adelante) de las que se hará uso en la aplicación. Estos códigos QR también han sido un motivo de peso pues serán la clave del proyecto, además de ser muy sencillo y económico colocar un código QR en prácticamente cualquier lugar.

Uno de los aspectos fundamentales de la aplicación es su capacidad para generar rutas, como si de un GPS se tratara, mediante algoritmos de *pathfinding* (búsqueda de caminos). Los algoritmos de *pathfinding* se basan en el llamado "problema de los caminos más cortos". En la teoría de grafos, este problema consiste en encontrar un camino entre dos vértices (o nodos) de tal manera que la suma de los pesos de las aristas que lo constituyen es mínima. Un ejemplo claro de este problema es encontrar el camino más rápido para ir de una ciudad a otra, también extrapolable al tráfico de datos en Internet: cuando un dispositivo envía una petición a un servidor, los routers que encaminan ese paquete de datos deben ser capaces de saber cómo y por donde enviarlo para que llegue a su destino.

Debido a que la aplicación crea rutas para ir de un punto a otro, se hace necesario el uso de estos algoritmos. Existen múltiples algoritmos para tratar este asunto, como pueden ser el algoritmo de Dijkstra o el de Bellman-Ford. Para esta solución concreta, se implementa el algoritmo A* pero con ciertas adaptaciones al programa.

Capítulo 3 – Tecnologías utilizadas

Para desarrollar el presente proyecto se ha hecho necesario utilizar diversas tecnologías para que trabajen de forma conjunta y lograr nuestro objetivo. En este capítulo se hablará detalladamente de ellas.

3.1. Android

Android es un sistema operativo móvil basado en Linux, que junto con aplicaciones middleware (software utilizado para comunicar diferentes aplicaciones entre sí) está enfocado para ser utilizado en dispositivos móviles como teléfonos inteligentes, tabletas, Google TV y otros dispositivos. Es desarrollado por la Open Handset Alliance, la cual está liderada por Google. Este sistema por lo general maneja aplicaciones como Google Play.

3.1.1. ¿Qué hace Android especial?

Existen muchas plataformas para móviles (iPhone, Symbian, Windows Phone, BlackBerry, Java Mobile Edition, Linux Mobile…); sin embargo Android presenta una serie de características que lo hacen diferente ([7]). Es el primero que combina en una misma solución las siguientes cualidades:

- Plataforma realmente abierta. Es una plataforma de desarrollo libre basada en Linux y de código abierto. Una de sus grandes ventajas es que se puede usar y personalizar el sistema sin pagar *royalties*.
- Portabilidad asegurada. Las aplicaciones finales son desarrolladas en Java, lo que nos asegura que podrán ser ejecutadas en gran variedad de dispositivos tanto presentes como futuros. Esto se consigue gracias al concepto de máquina virtual.
- Arquitectura basada en componentes inspirados en internet. Por ejemplo, el diseño de la interfaz de usuario se hace en XML, lo que permite que una misma aplicación se ejecute en un móvil de pantalla reducida o en netbook.
- Filosofía de dispositivo siempre conectado a Internet.
- Gran cantidad de servicios incorporados: por ejemplo, localización basada tanto en GPS como en torres de telefonía móvil. Incorpora potentes bases de datos con SQL. Reconocimiento y síntesis de voz, navegador, mapas…
- Alto nivel de seguridad. Los programas se encuentran aislados unos de otros gracias al concepto de ejecución dentro de una caja que incorpora la máquina virtual. Cada aplicación dispone de una serie de permisos que limitan su rango de actuación (servicios de localización, acceso a Internet…)
- Optimización para baja potencia y poca memoria. Por ejemplo, Android utiliza la Máquina Virtual Dalvik. Se trata de una implementación de Google de la máquina virtual de Java optimizada para dispositivos móviles.
- Alta calidad de gráficos y sonido: gráficos vectoriales suavizados, animaciones inspiradas en Flash, gráficos en tres dimensiones basados en OpenGL. Incorpora los códecs estándar más comunes de audio y vídeo, incluyendo H.264 (AVC), MP3, AAC…

Como se ve, Android ofrece una forma sencilla y novedosa de implementar potentes aplicaciones para móviles.

3.1.2 Los orígenes

Como se explica en [10], Google adquiere Android Inc. en el año 2005. Se trataba de una pequeña compañía que acababa de ser creada, orientada a la producción de aplicaciones para terminales móviles. Ese mismo año empiezan a trabajar en la creación de una máquina virtual Java optimizada para móviles (Dalvik VM).

En el año 2007 se crea el consorcio Handset Alliance (http://www.openhandsetalliance.com) con el objetivo de desarrollar estándares abiertos para móviles. Está formado por Google, Intel, Texas Instruments, Motorola, T-Mobile, Samsung, Ericsson, Toshiba, Vodafone, NTT DoCoMo, Sprint Nextel y otros. Una pieza clave de los objetivos de esta alianza es promover el diseño y difusión de la plataforma Android. Sus miembros se han comprometido a publicar una parte importante de su propiedad intelectual como código abierto bajo licencia Apache v2.0.

En noviembre de 2007 se lanza una primera versión del Android SDK (*Software Development Kit*). Al año siguiente aparece el primer móvil con Android (T-Mobile G1). En octubre Google libera el código fuente de Android principalmente bajo licencia de código abierto Apache (licencia GPL v2 para el núcleo). Ese mismo mes se abre Android Market (ahora Google Play), para la descarga de aplicaciones. En abril del 2009 Google lanza la versión 1.5 del SDK que incorpora nuevas características como el teclado en pantalla. A finales del 2009 se lanza la versión 2.0 y durante el 2010 las versiones 2.1, 2.2 y 2.3.

Durante el año 2010 Android se consolida como uno de los sistemas operativos para móviles más utilizados, con resultados cercanos al iPhone e incluso superando al sistema de Apple en EE.UU.

Posteriormente, Google lanza nuevas versiones de Android: las versiones 3.0, 3.1 y 3.2 en febrero, mayo y junio de 2011, respectivamente. Más tarde lanza más nuevas versiones: la versión 4.0 en octubre de 2011 y la versión 4.1 en julio de 2012.

3.1.3. Arquitectura de Android

El siguiente gráfico muestra la arquitectura de Android, explicada en [10]. Como se puede ver está formada por cuatro capas. Una de las características más importantes es que todas las capas están basadas en software libre.

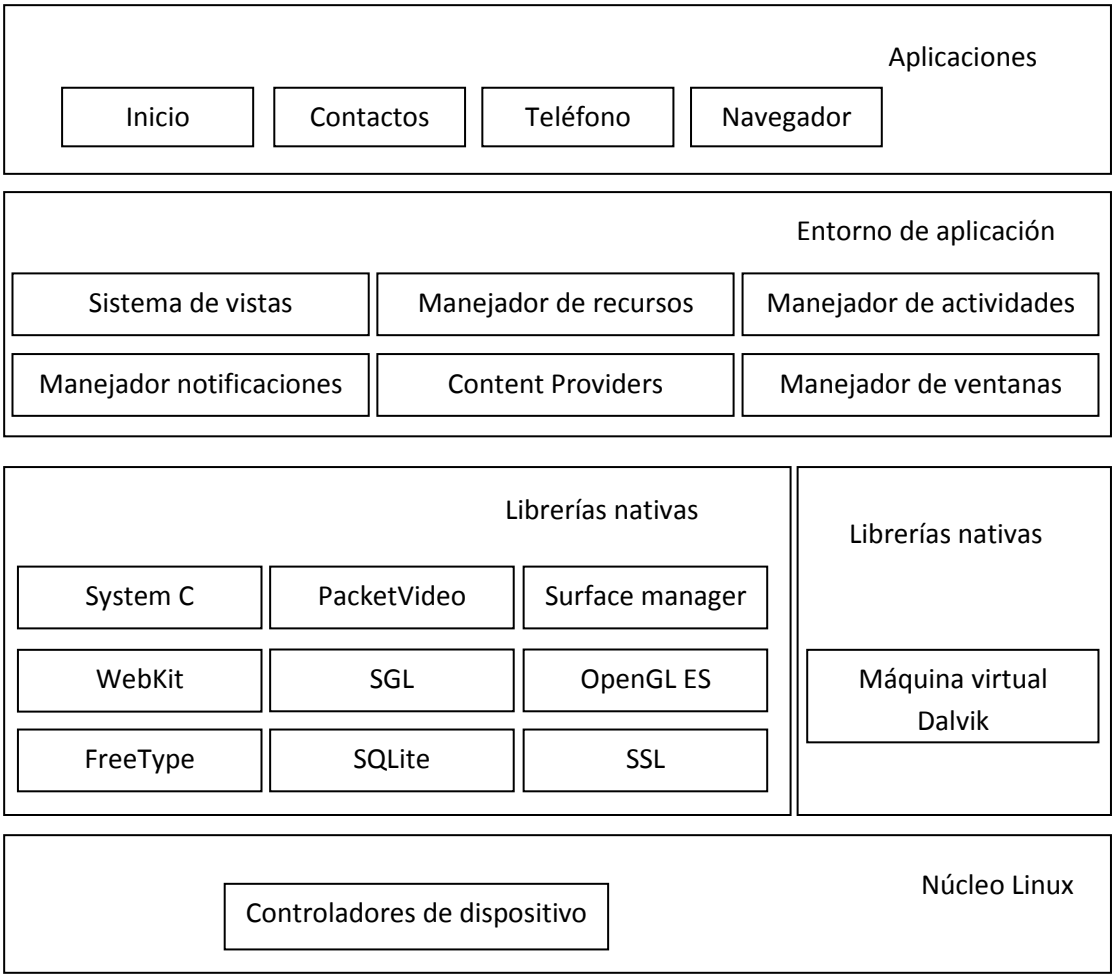


Figura 11: Arquitectura Android

El núcleo Linux

El núcleo de Android está formado por el sistema operativo Linux versión 2.6. Esta capa proporciona servicios como la seguridad, el manejo de la memoria, el multiproceso, la pila de protocolos y el soporte de *drivers* para dispositivos.

Esta capa del modelo actúa como capa de abstracción entre el hardware y el resto de la pila. Por lo tanto, es la única que es dependiente del *hardware*.

***Runtime* de Android**

Está basado en el concepto de máquina virtual utilizado en Java. Dadas las limitaciones de los dispositivos donde ha de correr Android (poca memoria y procesador limitado) no fue posible utilizar una máquina virtual Java estándar. Google tomó la decisión de crear una nueva, la máquina virtual Dalvik, que respondiera mejor a estas limitaciones.

Algunas características de la máquina virtual Dalvik que facilitan esta optimización de recursos son: que ejecuta ficheros Dalvik ejecutables (.dex) – formato optimizado para ahorrar memoria. Además está basada en registros. Cada aplicación corre en su propio proceso Linux con su propia instancia de la máquina virtual Dalvik. Delega al kernel de Linux algunas funciones como *threading* y el manejo de la memoria a bajo nivel.

También se incluye en el *runtime* de Android el "core libraries" con la mayoría de las librerías disponibles en el lenguaje Java.

Librerías nativas

Incluye un conjunto de librerías en C/C++ usadas en varios componentes de Android. Están compilados en código nativo del procesador. Muchas de las librerías utilizan proyectos de código abierto. Algunas de estas librerías son:

- **System C library**: una derivación de la librería BSD de C estándar (libc), adaptada para dispositivos embebidos basados en Linux.
- **Media Framework**: librería basada en PacketVideo's OpenCore; soporta codecs de reproducción y grabación de multitud de formatos de audio, vídeo e imágenes MPEG4, H.264, MP3, AAC, AMR, JPG y PNG.
- **Surface Manager**: maneja el acceso al subsistema de representación gráfica en 2D y 3D.
- **WebKit**: soporta un moderno navegador web utilizado en el navegador Android y en la vista webview. Se trata de la misma librería que utiliza Google Chrome y Safari de Apple.
- **SGL**: motor de gráficos 2D.
- **Librerías 3D**: implementación basada en OpenGL ES 1.0 API. Las librerías utilizan el acelerador hardware 3D si está disponible, o el software altamente optimizado de proyección 3D.
- **FreeType**: fuentes en bitmap y renderizado vectorial.
- **SQLite**: potente y ligero motor de bases de datos relacionales disponible para todas las aplicaciones.
- **SSL**: proporciona servicios de cifrado *Secure Socket Layer*.

Entorno de aplicación

Proporciona una plataforma de desarrollo libre para aplicaciones con gran riqueza e innovaciones (sensores, localización, servicios, barra de notificaciones). También se conoce como Java SDK.

La arquitectura ha sido diseñada para simplificar la reutilización de componentes. Las aplicaciones pueden publicar sus capacidades y otras pueden hacer uso de ellas (sujetas a las restricciones de seguridad). Este mismo mecanismo permite a los usuarios reemplazar componentes.

Una de las mayores fortalezas del entorno de aplicación Android es que se aprovecha el lenguaje de programación Java. El SDK de Android no acaba de ofrecer todo lo disponible para

su estándar del entorno de ejecución Java (JRE), pero es compatible con una fracción muy significativa de la misma.

Los servicios más importantes que incluye son:

- **Views**: extenso conjunto de vistas (parte visual de los componentes).
- **Resource Manager**: proporciona acceso a recursos que no son en código.
- **Activity Manager**: maneja el ciclo de vida de las aplicaciones y proporciona un sistema de navegación entre ellas.
- **Notification Manager**: permite a las aplicaciones mostrar alertas personalizadas en la barra de estado.
- **Content Providers**: mecanismo sencillo para acceder a datos de otras aplicaciones (como los contactos).

Aplicaciones

Este nivel está formado por el conjunto de aplicaciones instaladas en una máquina Android. Todas las aplicaciones han de correr en la máquina virtual Dalvik para garantizar la seguridad del sistema.

Normalmente las aplicaciones Android están escritas en Java. Para desarrollar aplicaciones en Java podemos utilizar el Android SDK. Existe otra opción consistente en desarrollar las aplicaciones utilizando C/C++. Para esta opción podemos utilizar el Android NDK (Native Developers Kit).

3.1.4. Plataformas de desarrollo

A continuación, y como se hace en [10], se describen las plataformas lanzadas hasta la fecha con una breve descripción de las novedades introducidas. Las plataformas se identifican de tres formas alternativas: versión, nivel de API y nombre. El nivel de API corresponde a números enteros comenzando desde 1. Para los nombres se han elegido postres en inglés en orden alfabético. Las dos primeras versiones, que hubieran correspondido a las letras A y B, no recibieron nombre.

Android 1.0 Nivel de API 1 (septiembre 2008)

Primera versión de Android. Nunca se utilizó comercialmente, por lo que no tiene mucho sentido desarrollar para esta plataforma.

Android 1.1 Nivel de API 2 (febrero 2009)

No se añadieron apenas funcionalidades, simplemente se fijaron algunos errores de la versión anterior. Es la opción a escoger si queremos desarrollar una aplicación compatible con todos los dispositivos Android. No obstante apenas existen usuarios con esta versión.

Android 1.5 Nivel de API 3 (abril 2009, Cupcake)

Es la primera versión con un número significativo de usuarios (un 6% a principios de 2011). Como novedades, se incorpora la posibilidad de teclado en pantalla con predicción de texto, los terminales ya no tienen que tener un teclado físico, así como la capacidad de grabación avanzada de audio y vídeo. También aparecen los *widgets* de escritorio y *live folders*. Incorpora soporte para *bluetooth* estéreo, por lo que permite conectarse automáticamente a auriculares *bluetooth*. Las transiciones entre ventanas se realizan mediante animaciones.

Android 1.6 Nivel de API 4 (diciembre 2009, Donut)

Permite capacidades de búsqueda avanzada en todo el dispositivo. También se incorpora *gestures* y *multi-touch*. Permite la síntesis de texto a voz. También se facilita que una aplicación pueda trabajar con diferentes densidades de pantalla. Soporte para resolución de pantallas WVGA. Aparece un nuevo atributo XML, `onClick`, que puede especificarse en una vista. Android Market se mejora permitiendo una búsqueda más sencilla de aplicaciones. Soporte para CDMA/EVDO, 802.1x y VPNs. Mejoras en la aplicación de la cámara.

Android 2.0 Nivel de API 5 (octubre 2009, Éclair)

Esta versión de API apenas cuenta con usuarios, dado que la mayoría de fabricantes pasaron de la versión 1.6 a la 2.1. Como novedades cabría destacar que incorpora un API para manejar el *bluetooth* 2.1. Nueva funcionalidad que permite sincronizar adaptadores para conectarlo a cualquier dispositivo. Ofrece un servicio centralizado de manejo de cuentas. Mejora la gestión de contactos y ofrece más ajustes en la cámara. Se ha optimizado la velocidad de *hardware*. Se aumenta el número de tamaños de ventana y resoluciones soportadas. Nueva interfaz del navegador y soporte para HTML5. Mejoras en el calendario y soporte para Microsoft Exchange. La clase `MotionEvent` ahora soporta eventos en pantallas multitáctiles.

Android 2.1 Nivel de API 7 (enero 2010, Éclair)

Se considera una actualización menor, por lo que le siguieron llamando Éclair. Destacamos el reconocimiento de voz, que permite introducir un campo de texto dictando sin necesidad de utilizar el teclado. También permite desarrollar fondos de pantalla animados. Se puede obtener información sobre la señal de la red actual que posea el dispositivo. En el paquete WebKit se incluyen nuevos métodos para manipular bases de datos almacenadas en Web. También se permite obtener permisos de geolocalización, y modificarlos en WebView. Se incorporan mecanismos para administrar la configuración de la caché de aplicaciones, almacenamiento web, y modificar la resolución de la pantalla. También se puede manejar vídeo, historial de navegación, vistas personalizadas…

Android 2.2 Nivel de API 8 (mayo 2010, Froyo)

Como característica más destacada se puede indicar la mejora de velocidad de ejecución de las aplicaciones (ejecución del código de la CPU de 2 a 5 veces más rápido que en la versión 2.1 de acuerdo a varios *benchmarks*). Esto se consigue con la introducción de un nuevo compilador JIT de la máquina Dalvik.

Se añaden varias mejoras relacionadas con el navegador Web, como el soporte de Adobe Flash 10.1 y la incorporación del motor Javascript V8 utilizado en Chrome o la incorporación del campo "subir fichero" en un formulario.

El desarrollo de aplicaciones permite las siguientes novedades: se puede preguntar al usuario si desea instalar una aplicación en un medio de almacenamiento externo (como una tarjeta SD), como alternativa a la instalación en la memoria interna del dispositivo. Las aplicaciones se actualizan de forma automática cuando aparece una nueva versión. Proporciona un servicio para la copia de seguridad de datos que se puede realizar desde la propia aplicación para garantizar al usuario el mantenimiento de sus datos. Por último, se facilita que las aplicaciones interaccionen con el reconocimiento de voz y que terceras partes proporcionen nuevos motores de reconocimiento.

Se mejora la conectividad: ahora podemos utilizar nuestro teléfono para dar acceso a Internet a otros dispositivos (*tethering*), tanto por USB como por Wi-Fi. También se añada el soporte a Wi-Fi IEEE 802.11n.

Se añaden varias mejores en diferentes componentes: En el API gráfica OpenGL ES se pasa a soportar la versión 2.0. También se puede realizar fotos o vídeos en cualquier orientación (incluso vertical) y configurar otros ajustes de la cámara. Para finalizar, permite definir modos de interfaz de usuario ("automóvil" y "noche") para que las aplicaciones se configuren según el modo seleccionado por el usuario.

Android 2.3 Nivel de API 9 (diciembre 2010, Gingerbread)

Debido al éxito de Android en las nuevas tabletas ahora soporta mayores tamaños de pantalla y resoluciones (WXGA y superiores).

Incorpora un nuevo interfaz de usuario con un diseño actualizado. Dentro de las mejoras de la interfaz de usuario destacamos la mejora de la funcionalidad de "cortar, copiar y pegar" y un teclado en pantalla con capacidad multitáctil.

Se incluye soporte nativo para varias cámaras, pensando en la segunda cámara usada en videoconferencia. La incorporación de esta segunda cámara ha propiciado la inclusión de reconocimiento facial para identificar el usuario del terminal.

La máquina virtual de Dalvik para Android introduce un nuevo recolector de basura que minimiza las pausas de la aplicación, ayudando a garantizar una mejor animación y el aumento de la respuesta en juegos y aplicaciones similares. Se trata de corregir así una de las lacras de

este sistema operativo móvil, que en versiones previas no ha sido capaz de cerrar bien las aplicaciones en desuso. Se dispone de mayor apoyo para el desarrollo de código nativo (NDK). También se mejora la gestión de energía y control de aplicaciones. Y se cambia el sistema de ficheros, que pasa de YAFFS a ext4.

Entre otras novedades destacamos el soporte nativo para telefonía sobre Internet VoIP/SIP. El soporte para reproducción de vídeo WebM/VP8 y codificación de audio AAC. El soporte para la tecnología NFC. Las facilidades en el audio, gráficos y entradas para los desarrolladores de juegos. El soporte nativo para más sensores (como giroscopios y barómetros). Un gestor de descargas las descargas largas.

Android 3.0 Nivel de API 11 (febrero 2011, Honeycomb)

Para mejorar la experiencia de Android en las nuevas tabletas se lanza la versión 3.0 optimizada para dispositivos con pantallas grandes. La nueva interfaz de usuario ha sido completamente rediseñada con paradigmas nuevos para la interacción, navegación y personalización. La nueva interfaz se pone a disposición de todas las aplicaciones, incluso las construidas para versiones anteriores de la plataforma.

Con el objetivo de adaptar la interfaz de usuario a pantallas más grandes se incorporan las siguientes características: resolución por defecto WXGA (1280*800), escritorio 3D con widgets rediseñados, nuevos componentes y vistas, notificaciones mejoradas, arrastrar y soltar, nuevo cortar y pegar, barra de acciones para que las aplicaciones dispongan de un menú contextual siempre presente y otras características para aprovechar las pantallas más grandes.

Se mejora la reproducción de animaciones 2D/3D gracias al renderizador OpenGL acelerado por hardware. El nuevo motor de gráficos Renderscript saca un gran rendimiento de los gráficos en Android e incorpora su propia API.

Primera versión de la plataforma que soporta procesadores multinúcleo. La máquina virtual Dalvik ha sido optimizada para permitir multiprocesado, lo que permite una ejecución más rápida de las aplicaciones, incluso aquellas que son de hilo único.

Se incorporan varias mejoras multimedia, como listas de reproducción M3U a través de HTTP Live Streaming, soporte a la protección de derechos musicales (DRM) y soporte para la transferencia de archivos multimedia a través de USB con los protocolos MTP y PTP.

En esta versión se añaden nuevas alternativas de conectividad, como las nuevas APIs de Bluetooth A2DP y HSP con streaming de audio. También, se permite conectar teclados completos por USB o Bluetooth.

El uso de los dispositivos en un entorno empresarial es mejorado. Entre las novedades introducidas destacamos las nuevas políticas administrativas con encriptación del almacenamiento, caducidad de contraseña y mejoras para administrar los dispositivos de empresa de forma eficaz.

A pesar de la nueva interfaz gráfica optimiza para tabletas, Android 3.0 es compatible con las aplicaciones creadas para versiones anteriores. La tecla de menú, inexistente en las nuevas tabletas, es reemplazada por un menú que aparece en la barra de acción.

Android 4.0 Nivel de API 14 (octubre 2011, Ice Cream Sandwich)

Se trata de una versión que toma muchos de los aspectos más útiles y populares de Honeycomb, como los botones virtuales, la facilidad de la multitarea con la lista de aplicaciones recientes y la barra de acciones dentro de cada aplicación, y la tecnología NFC.

Ice Cream Sandwich también modifica el teclado con unas mejoras notables en los nuevos diccionarios, mayor número de sugerencias y más velocidad de dictado. Otro aspecto que ha experimentado fuertes cambios es la cámara, tanto su interfaz como el nuevo sistema de captación. También incluye estabilizador de imágenes y editor fotográfico muy básico.

El acceso a las notificaciones es más rápido ya que ahora se puede extraer cada una de las notificaciones de la lista, deslizarla fuera y acceder a ella. Otra de las novedades son la posibilidad crear carpetas arrastrando los iconos y una bandeja de favoritos completamente personalizable por el usuario.

En cuanto al desbloqueo del dispositivo, además de las ya existentes Ice Cream Sandwich incorpora la opción de desbloqueo facial. Esta opción de reconocimiento facial también es útil y abre nuevas puertas al sector de los juegos, por ejemplo.

3.1.5. Ciclo de vida de una aplicación

El ciclo de vida de una aplicación Android es bastante diferente al ciclo de vida de una aplicación en otros sistemas operativos. La mayor diferencia es que en Android el ciclo de vida es controlado completamente por el sistema, en lugar de ser controlado directamente por el usuario. En [10] se detalla este ciclo de vida.

Una aplicación en Android va a estar formada por un conjunto de elementos básicos de visualización, conocidos como actividades. Además de varias actividades una aplicación también puede contener servicios. Son las actividades las que realmente controlan el ciclo de vida de una aplicación, dado que el usuario no cambia aplicación, sino de actividad. El sistema va a mantener una pila con las actividades previamente visualizadas, de forma que el usuario va a poder regresar a la actividad anterior pulsando la tecla “atrás”.

Una aplicación Android corre dentro de su propio proceso Linux. Este proceso es creado para la aplicación y continuará vivo hasta que ya no sea requerido y el sistema reclame su memoria para asignársela a otra aplicación.

Una característica importante y poco usual de Android es que la destrucción de un proceso no es controlada directamente por la aplicación. En lugar de esto, es el sistema quien determina cuándo destruir el proceso, basándose en el conocimiento que tiene el sistema de las partes de la aplicación que están corriendo (actividades y servicios), lo importantes que son para el usuario y cuánta memoria disponible hay en un determinado momento.

Si tras eliminar el proceso de una aplicación, el usuario vuelve a ella, se crea de nuevo el proceso, pero se habrá perdido el estado que tenía esta aplicación. En estos casos, va a ser responsabilidad del programador almacenar el estado de la aplicación, si se desea que cuando sea reiniciada conserve su estado.

Como se ve, Android es sensible al ciclo de vida de una aplicación, por lo tanto se necesita comprender y manejar los eventos relacionados con el ciclo de vida si se quieren crear aplicaciones estables.

Una actividad en Android puede estar en uno de estos cuatro estados:

- **Activa** (*Running*): La actividad está encima de la pila, lo que quiere decir que es visible y tiene el foco
- **Visible** (*Paused*): La actividad es visible pero no tiene el foco. Se alcanza este estado cuando pasa a activa otra actividad con alguna parte transparente o que no ocupa toda la pantalla. Cuando una actividad está tapada por completo, pasa a estar parada.
- **Parada** (*Stopped*): Cuando la actividad no es visible, se recomienda guardar el estado de la interfaz de usuario, preferencias, etc.
- **Destruida** (*Destroyed*): Cuando la actividad termina, o es matada por el sistema Android, sale de la pila de actividades.

Cada vez que una actividad cambia de estado se van a producir eventos que podrán ser capturados por ciertos métodos de la actividad. A continuación se muestra un esquema que ilustra los métodos que capturan estos eventos.

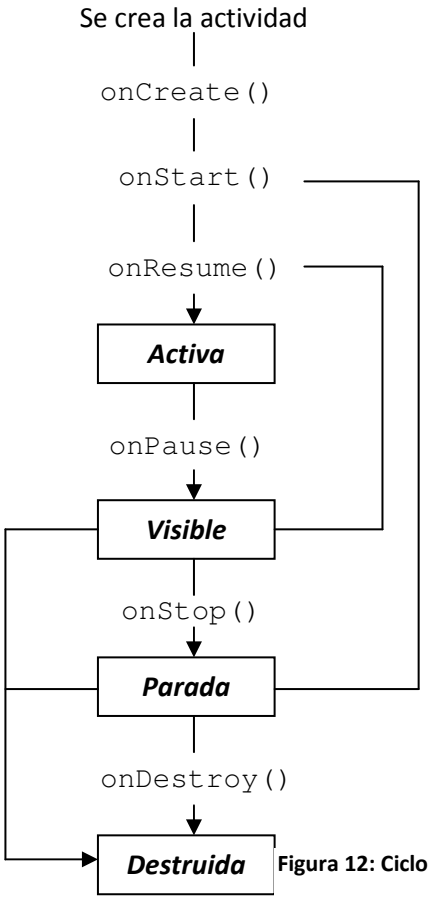


Figura 12: Ciclo de vida

- **`OnCreate(Bundle)`**: Se llama en la creación de la actividad. Se utiliza para realizar todo tipo de inicializaciones, como la creación de la interfaz de usuario. Puede recibir información de estado de instancia (en una instancia de la clase `Bundle`), por si se reanuda desde una instancia pausada anterior.
- **`onStart()`**: Indica que la actividad está a punto de ser mostrada al usuario.
- **`onResume()`**: Se llama cuando la actividad va a comenzar a interactuar con el usuario. Es un buen lugar para lanzar las animaciones y la música.
- **`onPause()`**: Indica que la actividad está a punto de ser lanzada a segundo plano, normalmente porque otra aplicación es lanzada. Es el lugar adecuado para almacenar los datos que estaban en edición.
- **`onStop()`**: La actividad ya no va a ser para el usuario. Si hay muy poca memoria es posible que la actividad se destruya sin llamar a este método.
- **`onRestart()`**: Indica que la actividad va a volver a ser representada después de haber pasado por `onStop()`.
- **`onDestroy()`**: Se llama antes de que la actividad sea totalmente destruida. Si hay muy poca memoria es posible que la actividad se destruya sin llamar a este método.

¿Qué proceso se elimina?

Como se ha comentado Android mantiene en memoria todos los procesos que quepan aunque éstos no se estén ejecutando. Una vez que la memoria está llena y el usuario decide ejecutar una nueva aplicación, el sistema ha de determinar qué proceso de los que están ejecución ha der ser eliminado. Android ordena los procesos en una lista jerárquica, asignándole a cada uno una determinada importancia. Esta lista se confecciona basándose en los componentes de la aplicación que están corriendo (actividades y servicios) y el estado de estos componentes.

Para establecer esta jerarquía de importancia se distinguen los siguientes tipos de procesos:

- **Proceso de primer plano** (*Foreground process*): Hospeda una actividad en la superficie de la pantalla y con la cual el usuario está interactuando (su método `onResume()` ha sido llamado). Debería haber sólo uno o unos pocos procesos de este tipo. Sólo serán eliminados como último recurso, si es que la memoria está tan baja que ni siquiera estos procesos pueden continuar corriendo.
- **Proceso de servicio** (*Service process*): Hospeda un servicio que ha sido inicializado con el método `startService()`. Aunque estos procesos no son directamente visibles al usuario, generalmente están haciendo tareas que para el usuario son importantes (tales como reproducir un archivo MP3 o mantener una conexión con un servidor de contenidos). El sistema siempre tratará de mantener estos procesos corriendo, a menos que los niveles de memoria comiencen a comprometer el funcionamiento de los procesos de primer plano o visibles.
- **Proceso de fondo** (*Background process*): Hospeda una actividad que no es actualmente visible al usuario (su método `onStop()` ha sido llamado). Si estos procesos son eliminados no tendrán un impacto directo en la experiencia del usuario.
- **Proceso vacío** (*Empty process*): No hospeda a ningún componente de aplicación activo. La única razón para mantener ese proceso es tener un "caché" que permita mejorar el

tiempo de activación en la próxima vez que un componente de su aplicación sea ejecutado.

Guardando el estado de una actividad

Cuando el usuario ha estado utilizando una actividad, y tras cambiar a otras, regresa a la primera, lo habitual es que esta permanezca en memoria y continúe su ejecución sin alteraciones. Como se ha explicado, en situaciones de escasez de memoria, es posible que el sistema haya eliminado el proceso que ejecutaba la actividad. En este caso, el proceso será ejecutado de nuevo, pero se habrá perdido su estado, es decir, se habrá perdido el valor de sus variables y el puntero de programa. Como consecuencia, si el usuario estaba rellenando un cuadro de texto o estaba reproduciendo un audio en un punto determinado perderá esta información. En este apartado se estudiará un mecanismo sencillo que proporciona Android para resolver este problema.

***Nota**: cuando se ejecuta una aplicación sensible a la inclinación del teléfono, puede verse en horizontal o en vertical, se presenta un problema similar al anterior. La actividad es destruida y vuelva a construir con las nuevas dimensiones de pantalla y por lo tanto se llama de nuevo al método `onCreate()`. Antes de que la actividad sea destruida también resulta fundamental guardar su estado.*

Dependiendo de lo sensible que sea la pérdida de información de estado se podrá actual de diferentes maneras. Una posibilidad es por ejemplo no hacer nada. Siguiendo con el ejemplo anterior el usuario tendría que volver a rellenar el cuadro de texto o el audio volvería a reproducirse desde el principio.

En caso de tratarse de información extremadamente sensible, se recomienda el uso del método `onPause()` para guardar el estado y `onResume()` para recuperarlos. Por supuesto, la información sensible ha de almacenarse en un medio permanente como un fichero o una base de datos. Se trata del método más fiable, como se vio en el ciclo de vida de una actividad, los métodos `onStop()` y `onDestroy()` no tienen la garantía de ser llamados.

También se tiene una tercera posibilidad que, aunque no tenga una fiabilidad tan grande, resulta mucho más sencilla. Consiste en utilizar los dos métodos siguientes:

- **onSaveInstanceState(Bundle)**: Se invoca para permitir a la actividad guardar su estado. Si hay muy poca memoria es posible que la actividad se destruya sin llamar a este método.
- **onRestoreInstanceState(Bundle)**: Se invoca para recuperar el estado guardado por `onSaveInstanceState()`.

La ventaja de usar estos métodos es que el programador no ha de buscar un método de almacenamiento permanente, es el sistema quien hará este trabajo. El inconveniente es que el sistema no garantiza que sean llamados. En casos de extrema necesidad de memoria se podría destruir el proceso sin llamarlos.

3.1.6. Patrón de diseño Modelo-Vista-Controlador

El Modelo-Vista-Controlador (en adelante, MVC) es un modelo de abstracción de desarrollo de software que separa los datos de una aplicación, la interfaz de usuario y la lógica de negocio en tres componentes distintos, y que se desarrolló por primera vez en SmallTalk como se explica en [11]. Este patrón se ve con frecuencia en aplicaciones web, donde la vista es la página HTML (junto con el código que proporciona datos dinámicos a la página), el modelo es el sistema de gestión de bases de datos junto con la lógica de negocio, y el controlador es el responsable de recibir los eventos de entrada desde la vista. Cobra especial importancia en Android debido al diseño que tiene.

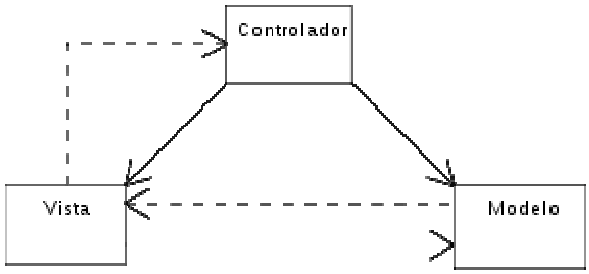


Figura 13: Modelo-Vista-Controlador

El diagrama, obtenido de [12], muestra las relaciones entre el modelo, la vista y el controlador. Las líneas continuas representan asociación directa, y las punteadas asociación indirecta (como podría ser un patrón Observador por ejemplo). Cada componente tiene una función específica:

- Modelo: Es la representación de la información en el sistema. Trabaja junto a la vista para mostrar la información al usuario y es accedido por el controlador para añadir, eliminar, consultar o actualizar datos.
- Vista: Este componente presenta el modelo en un formato adecuado para interactuar, normalmente la interfaz de usuario.
- Controlador: Este componente responde a eventos, usualmente acciones del usuario. Interactúa con la vista y el modelo.

Este patrón se utiliza en múltiples *frameworks*, como por ejemplo:

- Java Swing
- Java Enterprise Edition (J2EE)
- XForms (Formato XML estándar del W3C para la especificación de un modelo de datos XML e interfaces de usuario como formularios web)
- GTK+ (escrito en C, toolkit creado por Gnome para construir aplicaciones gráficas, inicialmente para el sistema X Window)
- ASP.NET (Microsoft)
- Google Web Toolkit (GWT, para crear aplicaciones Ajax con Java)
- Apache Struts (*framework* para aplicaciones web J2EE)
- Ruby on Rails (*framework* para aplicaciones web con Ruby)

La utilidad de MVC viene dada por la separación que permite realizar, de manera que se puedan implementar los tres componentes de manera totalmente independiente y

reutilizable. Concretamente, el modelo es lo más reutilizable y menos susceptible a sufrir cambios. Lo que más cambia es la vista, que variará en función de la presentación que se quiera implementar. En un punto intermedio está el controlador.

Se puede deducir que el modelo debe ser independiente: las clases (o funciones o estructuras) del modelo no deben ver a ninguna clase de los otros dos componentes. De esta manera se podría compilar el modelo en una librería aparte que añadir a futuros proyectos. El controlador suele ver clases del modelo, pero no de la vista (si hay un cambio en la vista, ésta avisará al modelo para que la actualice). Finalmente, como la vista es lo más cambiante, puede ver clases tanto del modelo como del controlador.

Se utilizará como ejemplo un juego de ajedrez para que la comprensión de este patrón sea más fácil. Tiene tres partes bien diferenciadas:

- Todas las reglas del ajedrez son independientes de si se dibuja un tablero plano o un tablero 3D, o si las figuras son las tradicionales blancas y negras o de otro tipo. Este código es el modelo. En un juego de ajedrez podría ser una clase (o conjunto de clases) que mantenga un array 8x8 con las piezas, que controle si los movimientos son legales, que detecte el jaque y el jaque mate, las tablas, y demás situaciones.
- El juego tendrá una presentación visual. Normalmente será una interfaz gráfica, pero también podría ser de texto, de comunicaciones con otro programa, etc… Esta es la vista, que en caso de ser una interfaz gráfica, dibuja el tablero con las piezas o permite arrastrar una pieza con el ratón para moverla.
- La tercera parte es el algoritmo, el código que toma decisiones; es el controlador. No tiene que ver con las ventanas visuales ni con las reglas de nuestro modelo. En el juego de ajedrez el algoritmo para pensar las jugadas formaría parte del controlador.

Flujo de MVC

Aunque se pueden encontrar diferentes implementaciones de MVC, el flujo que sigue el control generalmente es el siguiente:

1. El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, pulsa un botón o un enlace).
2. El controlador recibe (por parte de los objetos de la vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (*handler*) o callback.
3. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el carro de la compra del usuario).
4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se reflejan los cambios en el modelo (por ejemplo, produce un listado del contenido del carro de la compra). El modelo no debe tener conocimiento directo sobre la vista.
5. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

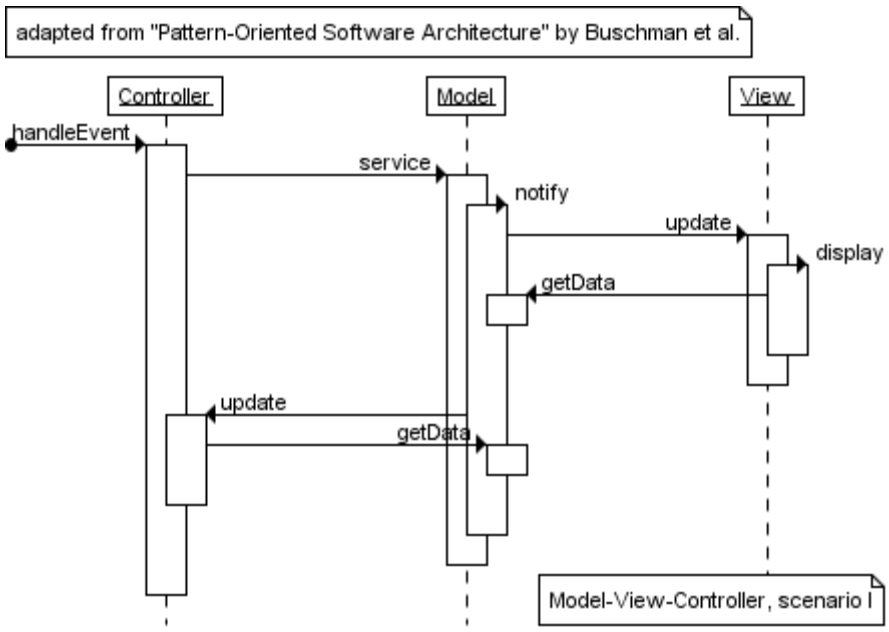


Figura 14: Funcionamiento de MVC

3.2. SQLite

SQLite es un sistema de dominio público de gestión de bases de datos relacional, contenida en una biblioteca relativamente pequeña (alrededor de 275 kb) escrita en C. Su creador es Richard Hipp.

Este sistema es compatible con la especificación ACID, lo que significa que cada secuencia de operaciones se trata como una transacción (es decir, como un todo). Más específicamente, un sistema compatible con ACID cumple las siguientes características (que se especifican en [9] junto con otras características):

- Atomicidad (*atomicity*): Es la propiedad que asegura que la operación se ha realizado o no, y por lo tanto ante un fallo del sistema no puede quedar a medias.
- Consistencia (*consistency*): Es la propiedad que asegura que sólo se empieza aquello que se puede acabar. Por lo tanto se ejecutan aquellas operaciones que no van a romper las reglas y directrices de integridad de la base de datos.
- Aislamiento (*isolation*): Es la propiedad que asegura que una operación no puede afectar a otras. Esto asegura que la realización de dos transacciones sobre la misma información sean independientes y no generen ningún tipo de error.
- Durabilidad (*durability*): Es la propiedad que asegura que una vez realizada la operación, ésta persistirá y no se podrá deshacer aunque falle el sistema.

Como se explica en [8], a diferencia de los sistemas de gestión de bases de datos tradicionales basados en el esquema cliente-servidor, el motor de SQLite no es un proceso independiente con el cual se comunica el programa principal. En lugar de eso, la biblioteca SQLite se enlaza con el programa pasando a formar parte de él, y éste utiliza llamadas simples a funciones y

subrutinas para acceder a la funcionalidad SQLite. Con esto se consigue reducir la latencia para acceder a la base de datos, puesto que es más eficiente realizar llamadas a funciones que comunicar procesos independientes. El conjunto de la base de datos (definiciones, tablas, índices y los propios datos) son guardados como un único fichero estándar en la máquina host. Esto es posible gracias al bloqueo que se realiza al fichero de base de datos al principio de cada transacción.

A diferencia de SQL, SQLite maneja un tipo de datos inusual. En lugar de asignar un tipo a una columna como en la mayor parte de sistemas SQL, los tipos se asignan a los valores individuales. El tipo de dato de un valor está asociado con el valor en sí mismo, no con su contenedor (tipado dinámico). El sistema de tipado dinámico de datos SQLite es compatible con los sistemas de tipado estático más comunes de otros motores de bases de datos, en el sentido de que las sentencias SQL que trabajan en bases de datos con tipado estático deberían trabajar de la misma manera en SQLite. Sin embargo el tipado dinámico en SQLite permite hacer cosas que no son posibles en las bases de datos tradicionales de tipado estático.

Para maximizar la compatibilidad entre SQLite y otros motores de base de datos, SQLite soporta el concepto de "Afinidad Tipo" en las columnas. La afinidad tipo de una columna es el tipo recomendado para almacenar el valor en esa columna. La idea central es que este tipo es el recomendado pero no obligatorio, cualquier columna puede almacenar cualquier tipo de dato. La cuestión está en que algunas columnas tienen que escoger usar una clase de almacenamiento sobre otra. Esta clase de almacenamiento es la afinidad.

Cada valor almacenado en una base de datos SQLite tiene una de las siguientes clases de almacenamiento:

- INTEGER
- TEXT
- NONE
- REAL
- NUMERIC

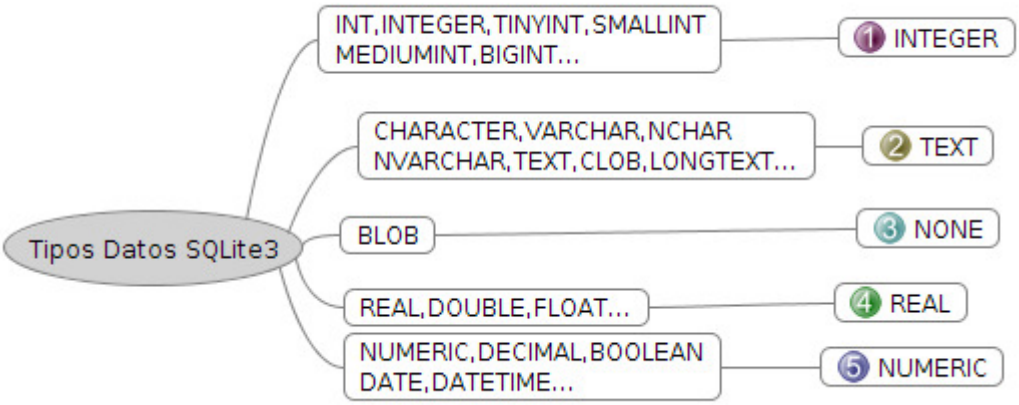


Figura 15: Tipos de datos SQLite

Hay que hacer una mención especial al tipo BLOB (Binary Large OBjects). Éste es un tipo utilizado para almacenar datos de gran tamaño que cambian de forma dinámica. Generalmente se guardan como BLOB imágenes, archivos de sonido y otros objetos multimedia.

Pueden acceder sin problema ninguno varios hilos o procesos a la misma base de datos, así como se pueden servir en paralelo varios accesos de lectura. No ocurre lo mismo con la escritura, ya que un acceso de escritura sólo puede ser servido si no se está sirviendo ningún otro acceso concurrentemente. En caso contrario, el acceso de escritura falla devolviendo un código de error (o puede reintentarse automáticamente hasta que expira un tiempo de expiración configurable).

Actualmente se utiliza SQLite en una gran cantidad de software, ya sea comercial o libre. Algunos ejemplos son:

- Adobe Photoshop Elements
- Mozilla Firefox
- Opera
- Skype
- Aperture

Además, debido a su tamaño reducido, SQLite es muy adecuado para su uso en los sistemas integrados, y de hecho está incluido en:

- Android
- BlackBerry
- Google Chrome
- iOS
- Maemo
- MeeGo
- Symbian OS
- webOS

El sistema operativo Android dispone de serie de todas las herramientas necesarias para la creación y gestión de bases de datos SQLite, y entre ellas una completa API para llevar a cabo de manera sencilla todas las tareas necesarias.

El SDK de Android ofrece métodos y clases Java para crear, actualizar y conectar con una base de datos SQLite. Concretamente, se hará uso de la clase `SQLiteOpenHelper`. Se creará una clase que derive de ella y que se adaptará como sea conveniente para adaptarse a las necesidades del programa.

Esta clase sólo tiene un constructor, que normalmente no hará falta sobreescribir, y los métodos abstractos `onCreate()` y `onUpgrade()`, que habrá que implementar con el código necesario para crear y actualizar la base de datos, respectivamente.

El método `onCreate()` se ejecutará automáticamente por parte de la clase hija de `SQLiteOpenHelper` cuando se necesite crear la base de datos (cuando no exista). Dentro

de este método se crearán las tablas que definan la base de datos, y se insertarán los datos iniciales si los hay.

El método `onUpgrade()` se ejecutará también de forma automática cuando se tenga que actualizar la estructura de la base de datos o haya que hacer una conversión sobre los datos. En este momento habría que programar toda la lógica necesaria para convertir la base de datos de una versión a otra, y de colocar los datos de la base antigua en la base actualizada.

A continuación hay que crear un objeto de la clase hija de `SQLiteOpenHelper`. Al crear este objeto pueden pasar tres cosas:

- Si la base de datos existe y su versión coincide con la solicitada, simplemente conectaremos con ella.
- Si la base de datos existe pero la versión solicitada es posterior a la actual, se llamará automáticamente al método `onUpgrade()` para convertir la base de datos a la nueva versión. Después de la actualización, se conectará con ella.
- Si la base de datos no existe, se llamará automáticamente al método `onCreate()` y se conectará con ella.

Se acaba de obtener acceso a la base de datos, pero todavía no se puede obtener datos de ella o insertarle otros nuevos. Para ello primero se debe acceder a la base de datos, en modo lectura o en modo escritura, según se necesite únicamente consultar datos o también añadir o realizar modificaciones. Se emplearán para ello los métodos `getReadableDatabase()` o `getWritableDatabase()`, respectivamente.

Una vez es posible extraer datos, insertar datos nuevos o modificar los ya existentes, se dispone de varios métodos para ello:

- `execSQL`
- `insert`
- `update`
- `delete`
- `rawQuery`
- `query`

En este caso se usará, por simplicidad, el método `execSQL` para realizar instrucciones de inserción, modificación o eliminación de datos, y el método `rawQuery` para instrucciones que devuelvan datos. En el caso de consultas que devuelvan datos, el método `rawQuery` devuelve un objeto `Cursor`, que contiene el resultado de la consulta. Esta clase contiene también los métodos `moveToFirst` y `moveToNext`, que serán de gran utilidad a la hora de recorrer el `Cursor` para obtener los datos.

Una vez se ha terminado de trabajar con la base de datos, se llama al método `close()` para cerrar la conexión con ella.

A continuación se muestra un ejemplo práctico donde se muestran todos estos conceptos.

```
public class UsuariosSQLiteHelper extends SQLiteOpenHelper {
```

```
    String sqlCreate = "CREATE TABLE Usuarios (codigo INTEGER, nombre TEXT)";

    public UsuariosSQLiteHelper(Context contexto, String nombre,
                                CursorFactory factory, int version) {
        super(contexto, nombre, factory, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(sqlCreate);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int versionAnterior, int
       versionNueva) {

        db.execSQL("DROP TABLE IF EXISTS Usuarios");
        db.execSQL(sqlCreate);
    }
}
```

Como se ha explicado anteriormente, se crea una clase, `UsuariosSQLiteHelper`, que hereda de la clase `SQLiteOpenHelper`. Más tarde se usará esta clase. Aquí es importante indicar que en el método `onUpgrade` se elimina la tabla y luego se crea de nuevo. Se realiza así por simplicidad, pero lo más correcto sería implementar cierta lógica para que se actualizara la tabla y se añadieran a la nueva tabla los datos de la antigua, sin eliminarla.

Seguidamente se muestra mostramos una clase que ya es un programa Android, que hace uso de esta clase recién creada.

```
public class AndroidBaseDatos extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //Abrimos la base de datos 'DBUsuarios' en modo escritura
        UsuariosSQLiteHelper usdbh =
            new UsuariosSQLiteHelper(this, "DBUsuarios", null, 1);

        SQLiteDatabase db = usdbh.getWritableDatabase();
         Cursor c = null;
         String usuario = null, email = null;

        //Si hemos abierto correctamente la base de datos
        if(db != null)
        {
            //Insertamos 5 usuarios de ejemplo
            for(int i=1; i<=5; i++)
            {
                //Generamos los datos
                int codigo = i;
                String nombre = "Usuario" + i;

                //Insertamos los datos en la tabla Usuarios
                db.execSQL("INSERT INTO Usuarios (codigo, nombre) " +
                           "VALUES (" + codigo + ", '" + nombre +"')");
            }
             c = db.rawQuery("SELECT * FROM Usuarios");
             c.moveToFirst();

             do{
```

```
                usuario = c.getString(0);
                email = c.getString(1);
                System.out.println("" + usuario + " - " + email);
            while(c.moveToNext());

            //Cerramos la base de datos
            db.close();
        }
    }
}
```

Aquí se crea un objeto de la clase `UsuariosSQLiteHelper` definida antes. Al constructor hay que pasarle como primer argumento un `Context` (le pasamos una referencia a la actividad principal), como segundo argumento el nombre de la base de datos, como tercer argumento un objeto `CursorFactory` que normalmente no necesitaremos (por eso le pasamos `null`) y como cuarto parámetro la versión de la base de datos.

A continuación se declara un objeto de la clase `SQLiteDatabase` que recogerá el resultado de `getWritableDatabase`. A través de este objeto ya se puede realizar operaciones sobre la base de datos. Junto a él se declara también un objeto `Cursor` que se utilizará más adelante para recoger el resultado de una consulta.

Ahora se crea un bucle para insertar datos dentro de la base de datos, utilizando el método `execSQL`, cuyo argumento es simplemente la instrucción a ejecutar. Después, se utiliza el método `moveToNext()que` devuelve `true` si ha ejecutado bien el movimiento para controlar el bucle que recorre el `Cursor`. Esta clase contiene métodos para obtener los tipos de datos primitivos como el `getString` que se utiliza en este caso, y cuyo argumento es la posición de la columna cuyo dato se quiere obtener, empezando por el 0. Una vez obtenidos los datos se imprimen por pantalla para después cerrar la conexión con la base de datos.

3.3. Códigos QR

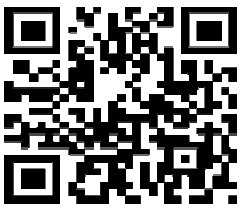


Figura 16: Código QR

El código QR (del inglés *Quick Response*) es la marca registrada de un tipo de código de barras matricial (en dos dimensiones), que fue diseñado primeramente para la industria automovilística ([6]). Recientemente este sistema se ha vuelto popular fuera de la industria por su rápida lectura y gran capacidad de almacenamiento en comparación con los códigos de barras tradicionales. El código se compone de módulos negros (puntos cuadrados) dispuestos en un patrón cuadrado sobre un fondo blanco. La información codificada puede estar compuesta por cuatro tipos estandarizados (los "modos") de datos (numéricos, alfanuméricos, bytes/binarios, kanji) o, mediante extensiones que lo soporten, virtualmente cualquier tipo de datos.

Estos códigos fueron inventados por Denso Wave (subsidiaria de Toyota) en 1994 para el seguimiento de vehículos durante el proceso de manufactura, y se diseñaron para que su contenido pudiera ser decodificado a una gran velocidad.

A diferencia de los códigos de barras tradicionales, diseñados para ser escaneados mediante un fino rayo de luz, los códigos QR se detectan como una imagen en dos dimensiones, que luego es analizada digitalmente por un software. Este software localiza los tres cuadros destacados de las esquinas de la imagen y normaliza el tamaño, orientación, ángulo y

visionado de la imagen, con la ayuda del pequeño cuadro de la cuarta esquina. Entonces se convierten los puntos pequeños en números binarios y se comprueba la validez con un código de corrección de errores.

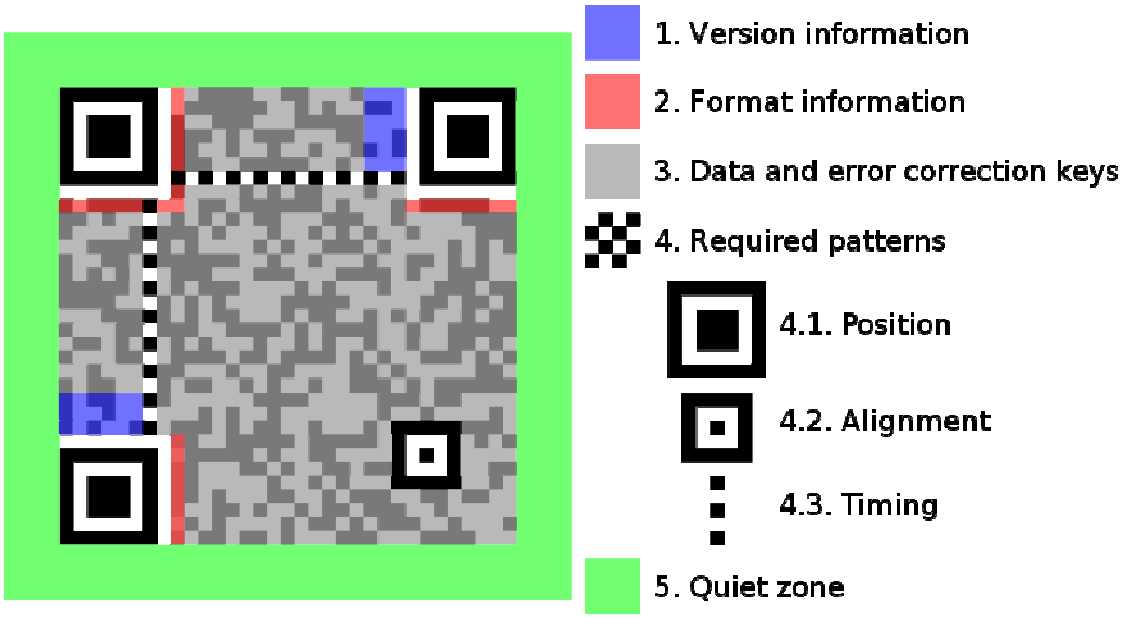


Figura 17: Estructura del código QR

3.3.1. Usos

Aunque antes estaban confinados a uso industrial, en los últimos años los códigos QR se han convertido en algo común en el embalaje y la publicidad al consumidor, debido a que la proliferación de smartphones ha producido que todo el mundo tenga un lector de código de barras en el bolsillo. Como consecuencia, el código QR ha llegado a ser un foco de la estrategia publicitaria, pues ofrece un acceso rápido y sin esfuerzo a la página web de la marca. Más allá de la mera comodidad del consumidor, la importancia de esta capacidad es que aumenta la tasa de conversión (esto es, aumenta la posibilidad de que el contacto con el anuncio se convierta en una venta) persuadiendo a clientes potenciales sin ningún retraso o esfuerzo, llevando a quien ve el anuncio al emplazamiento del anunciante inmediatamente, donde se puede continuar una mayor persuasión dirigida. Aunque inicialmente se utilizaron para el seguimiento de piezas en la manufactura de vehículos, los códigos QR ahora se usan en una gama mucho más amplia de aplicaciones, incluyendo el seguimiento comercial, los billetes en transportes y el etiquetado de productos en la propia tienda. También pueden usarse para almacenar información personal.

Muchas de las aplicaciones de estos códigos van dirigidas a los usuarios de teléfonos móviles. Estos usuarios pueden recibir texto, añadir un contacto a la agenda de su teléfono, abrir un URI o crear un mensaje de texto o correo electrónico después de escanear un código QR. Estas aplicaciones decodificadores se encuentran fácilmente en prácticamente todos los smartphones.

Se pueden encontrar códigos QR conteniendo enlaces web en revistas, señales, buses, tarjetas de visita o en prácticamente cualquier objeto sobre el que los usuarios puedan necesitar más

información. Los usuarios con un teléfono con cámara que tengan la aplicación lectora adecuada pueden escanear un código para mostrar un texto o una información de contacto, conectar con una red inalámbrica o abrir una página web en el navegador del teléfono. Este enlazado con objetos físicos se conoce como *hardlinking* o *object hyperlinking.*

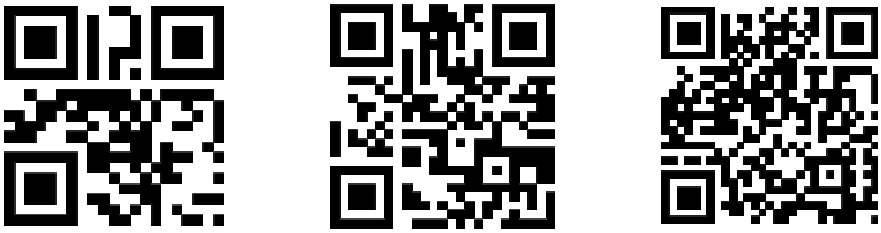
Las direcciones URL han ayudado a las tasas de conversión para el marketing incluso en la era anterior a los smartphones, pero durante ese tiempo han tenido que enfrentarse a varias limitaciones: quien ve el anuncio normalmente tenía que escribir la URL y a menudo no tenía un navegador web delante en el momento en que ve el anuncio. Las posibilidades de que olvidaran visitar la web más tarde, sin necesidad de escribir una URL, u olvidaran qué URL escribir eran altas. Las llamadas "URL amigables" (URL que son más cortas, facilitando así tanto su comprensión humana como su procesado) redujeron estos riesgos pero no los eliminaron. Algunas de estas desventajas para la tasa de conversión se están desvaneciendo ahora que los smartphones ponen constantemente al alcance del usuario el acceso web y el reconocimiento de voz. De esta forma, quien ve el anuncio sólo necesita alcanzar el teléfono y decir la URL en el momento en que lo ve, en lugar de recordar escribirla más tarde en un ordenador.

3.3.2. Almacenamiento

La cantidad de datos que se puede almacenar en un código QR depende del tipo de datos (el *modo* o juego de caracteres de entrada), la versión (1…40, indicando las dimensiones del código) y el nivel de corrección de errores. La máxima capacidad posible se da en en el caso 40-L (versión 40 y nivel L de corrección de errores). Según el tipo de datos a almacenar, hay diferentes capacidades:

- Datos numéricos: Máximo 7089 caracteres (0, 1, 2, 3, 4, 5, 6, 7, 8, 9).
- Datos alfanuméricos: Máximo 4296 caracteres(0–9, A–Z [sólo mayúsculas], espacio, $, %, *, +, -, ., /, :)
- Datos binarios/byte: Máximo 2953 caracteres (bytes de 8 bits) (23624 bits)
- Datos Kanji/Kana: Máximo 1817 caracteres.

A continuación se muestran ejemplos de las diferentes versiones de códigos QR:



Versión 1 (21x21) Versión 2 (25x25) Versión 3 (29x29)

Figura 18: Versiones de códigos QR

Versión 4 (33x33) Versión 10 (57x57) Versión 40 (177x21)

Figura 19: Versiones de códigos QR

3.3.3. Corrección de errores

Las palabras-código tienen 8 bits de largo y usan el algoritmo de corrección de errores de Reed-Solomon, con cuatro niveles de corrección de errores. Cuanto mayor sea el nivel de corrección de errores, menor será la capacidad de almacenamiento. Estos son las capacidades aproximadas de corrección de errores para cada nivel:

- Nivel L (Low): Se puede corregir el 7% de las palabras-código
- Nivel M (Medium): Se puede corregir el 15% de las palabras-código
- Nivel Q (Quartile): Se puede corregir el 25% de las palabras-código
- Nivel H (High): Se puede corregir el 30% de las palabras código.

Debido al diseño de los códigos Reed-Solomon y el uso de palabras-código de 8 bits, un único bloque no puede tener una longitud mayor de 255 palabras-código. Ya que los códigos QR más grandes contienen mucha mayor cantidad de datos, es necesario dividir el mensaje en múltiples bloques. La especificación QR no usa el mayor tamaño de bloque posible. En su lugar, define los tamaños de bloque de tal forma que no haya más de 30 símbolos de corrección de error en cada bloque. Esto implica que se podrán corregir al menos 15 errores por bloque, lo que limita la complejidad de ciertos pasos del algoritmo de decodificación. A continuación se intercalan los bloques, haciendo menos probable que el daño localizado en un código QR abrume la capacidad de un único bloque.

Gracias a la corrección de errores, es posible crear códigos QR artísticos que se pueden escanear correctamente pero contienen errores intencionales para hacerlos más legibles o atractivos para el ojo humano, así como incorporar colores, logotipos y otras características.

Figura 21: QR dañado pero todavía decodificable

Figura 20: QR artístico decodificable por corrección de errores

3.3.4. Codificación

La información de formato recuerda dos cosas: el nivel de corrección de errores y el patrón de enmascaramiento usado para el código. El enmascaramiento se usa para dividir patrones en el área de datos que podrían confundir al escáner, como grandes áreas vacías o propiedades engañosas que se parezcan a las marcas de localización. Los patrones de enmascaramiento se definen en matrices 6x6 que se repiten como haga falta para cubrir el código entero. Los módulos correspondientes a las áreas oscuras de la máscara se invierten. La información de formato está protegida contra errores mediante un código BCH, y se incluyen dos copias completas en cada QR.

Los datos del mensaje se colocan de derecha a izquierda en zigzag, como se muestra debajo. En códigos más grandes esto es más complicado por la presencia de patrones de alineamiento y el uso de múltiples bloques de corrección de errores entrelazados.

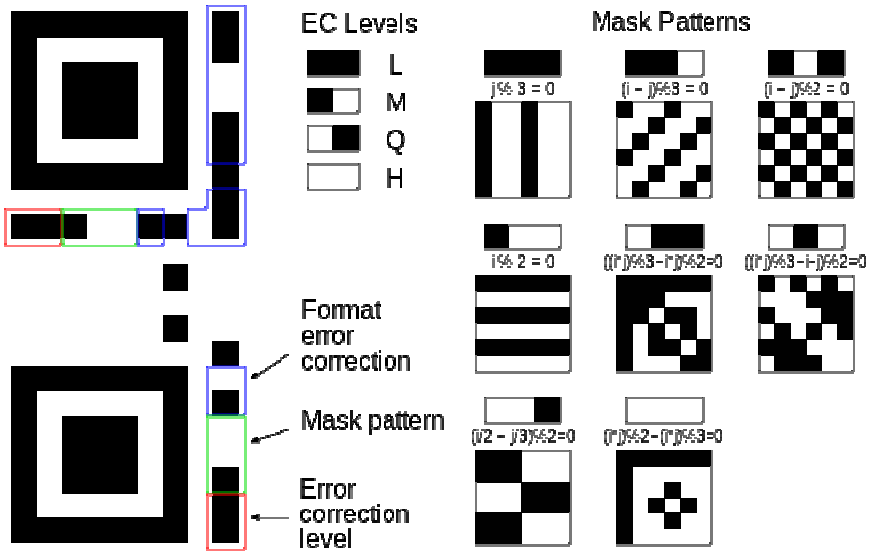


Figura 22: Información de formato

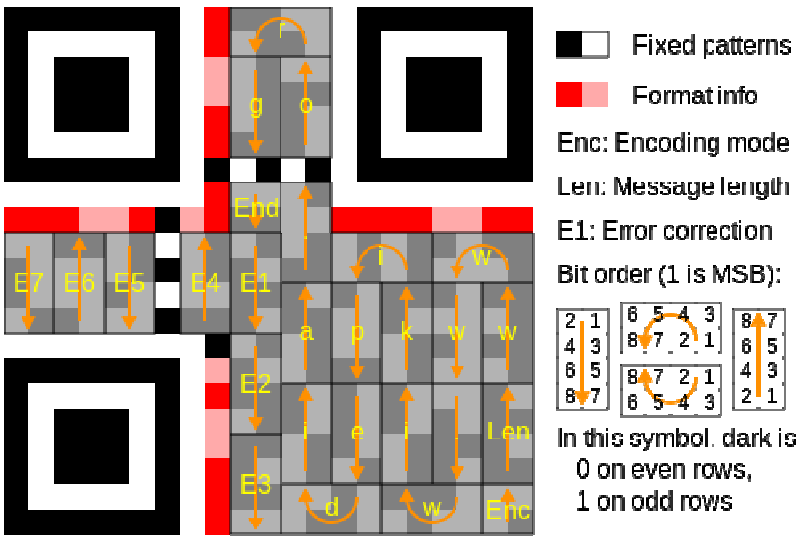


Figura 23: Posicionamiento del mensaje dentro del código QR

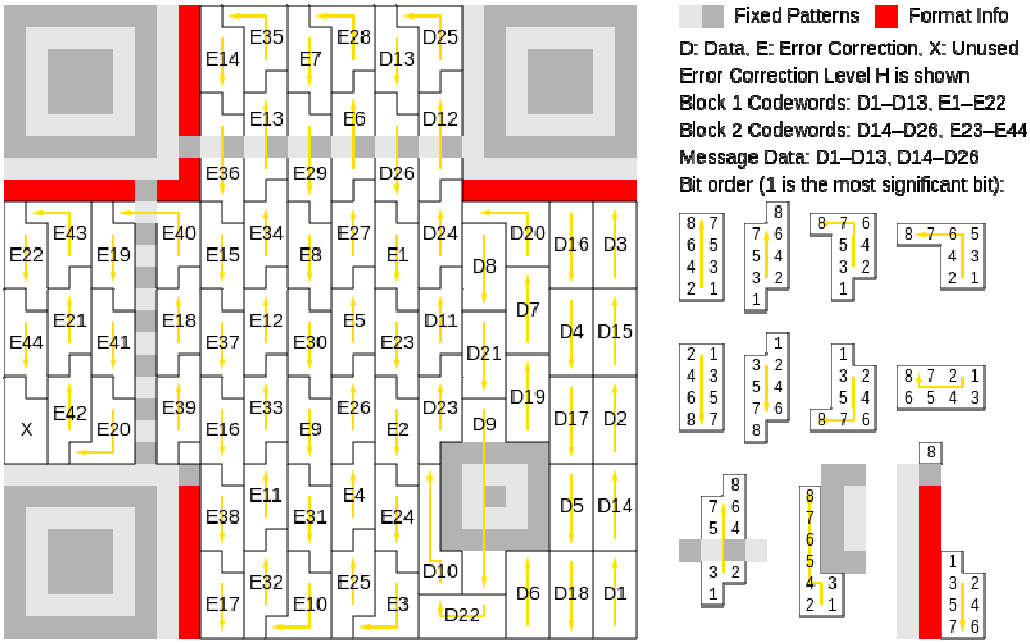


Figura 24: Bloques entrelazados en un código QR

3.4. Algoritmo A*

El algoritmo de búsqueda A* fue presentado por primera vez en 1968 por Peter E. Hart, Nils J. Nilsson y Bertram Raphael. El A* es una extensión del algoritmo de Dijkstra y se utiliza para encontrar, siempre y cuando se cumplan unas determinadas condiciones, el camino de menor coste entre un nodo origen y un nodo destino. Es un algoritmo completo, así que si existe un camino, siempre lo encontrará. Se puede encontrar mucha información adicional en [14] y [15]

3.4.1. Historia

En 1968 Nils Nilsson sugirió un enfoque heurístico para el robot Shakey (el primer robot móvil multipropósito con capacidad de actuar por sí solo) para moverse a través de una habitación que contuviera obstáculos. El algoritmo de búsqueda de caminos, llamado A1, era una versión más rápida del entonces bien conocido algoritmo de Dijkstra. Bertram Raphael sugirió algunas mejoras significativas a este algoritmo, llamando A2 a la nueva versión. Después Peter E. Hart argumentó que el algoritmo A2, con sólo unos cambios menores, sería el mejor algoritmo posible para encontrar el camino más corto. Los tres conjuntamente desarrollaron una prueba de que el algoritmo A2 revisado era óptimo para encontrar el camino más corto bajo ciertas condiciones definidas. Así, utilizaron la sintaxis de la estrella de Kleene para nombrar el nuevo algoritmo como A*, para así poder contemplar todos los posibles números de versión.

3.4.2. Propiedades

Este algoritmo utiliza la función de evaluación $f(n) = g(n) + h(n)$, donde $h(n)$ representa el coste heurístico (longitud estimada al nodo destino) del nodo actual, y $g(n)$ representa el coste real para llegar al nodo actual n desde el nodo inicial. Así, la función $f(n)$ evalúa el coste

total de cada nodo a la hora de su evaluación para incluirlo en la ruta. Utiliza dos estructuras auxiliares, que se llamarán lista abierta y lista cerrada. La primera sirve para almacenar nodos que todavía hay que evaluar, mientras que la segunda almacena aquellos nodos que ya se han procesado y que no se volverán a comprobar.

El mayor problema de este algoritmo es la memoria que necesita, ya que necesita almacenar todos los posibles nodos siguientes en cada estado. La cantidad de memoria que necesitará será exponencial con respecto al tamaño del problema.

3.4.3. Funcionamiento

El primer paso es dividir el área en una cuadrícula de forma que se convierta en una malla de casillas (nodos).

El algoritmo va siguiendo estos pasos:

a) Añade el nodo inicial a la lista
b) Repite el siguiente proceso:
    a) Busca el cuadro con el coste F más bajo en la lista abierta. Este es el nodo actual.
    b) Lo mueve a lista cerrada
    c) Para cada nodo adyacente al actual hace lo siguiente:
        - Si no es transitable o si está en la lista cerrada, lo ignora.
        - Si no está en la lista abierta, lo añade y hace que el nodo actual sea su padre. Almacena los costes f, g y h.
        - Si ya está en lista abierta, utiliza el coste g para evaluar si el camino hacia él es mejor que el actual (una g menor implica un mejor camino). De ser así, marca el nodo actual como su padre y recalcula f,g y h.
    d) Se detiene cuando añade el nodo objetivo a la lista abierta (hay camino) o cuando trate de encontrar el nodo objetivo y la lista abierta esté vacía (no hay camino).
c) Se va moviendo hacia atrás desde el cuadro objetivo, mediante la referencia al padre de cada nodo. El camino seguido es la ruta buscada.

A continuación se explica el algoritmo mediante un ejemplo para que su comprensión sea más fácil. Se seguirá el siguiente código de colores:

- Nodo inicial: verde claro
- Nodo final: rojo
- Nodo en lista abierta: amarillo
- Nodo en lista cerrada: verde oscuro
- Nodo actual: azul claro
- Nodo no transitable: negro

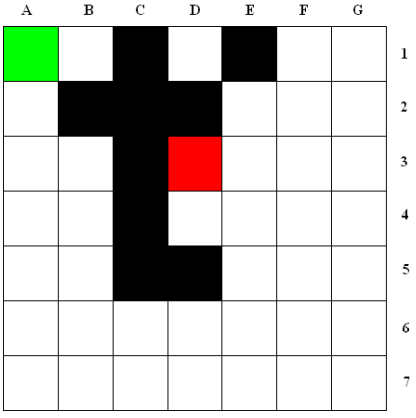


Figura 25: Mapa de ejemplo

El primer paso es añadir el nodo inicial a la lista abierta:

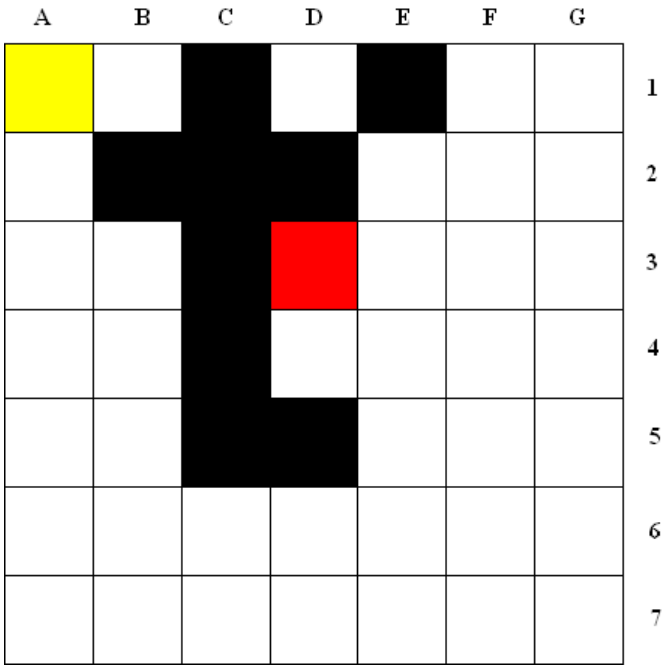


Figura 26: Nodo inicial a lista abierta

Ahora se busca el nodo con el coste f más bajo de la lista abierta. Como ahora mismo sólo hay un nodo, él mismo será el que se moverá a la lista cerrada:

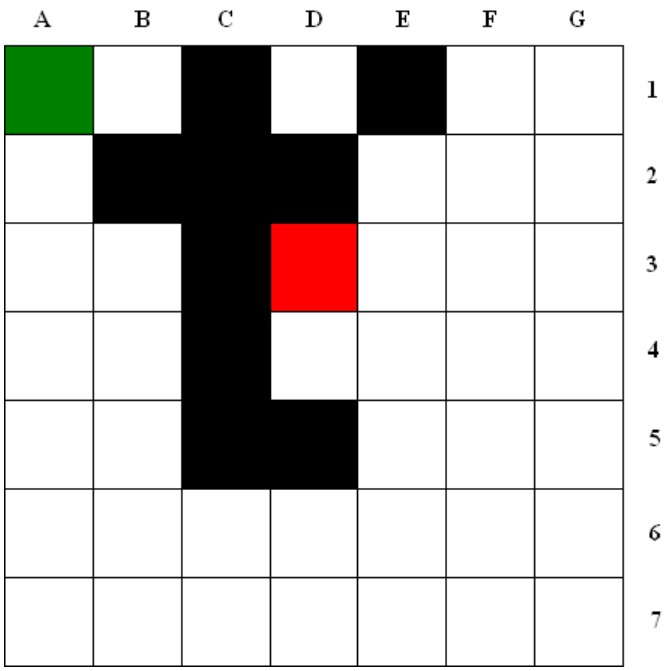


Figura 27: Búsqueda de nodo con coste f más bajo

Ahora se marca como el nodo actual:

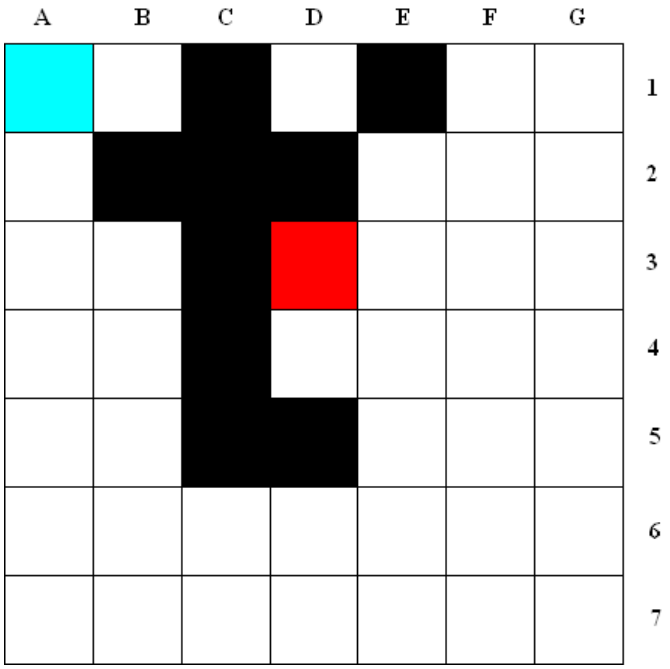


Figura 28: Nodo marcado como nodo actual

Una vez se tiene el nodo actual, se añaden a la lista abierta sus nodos adyacentes, se marca el nodo actual como su padre y se calculan sus costes:

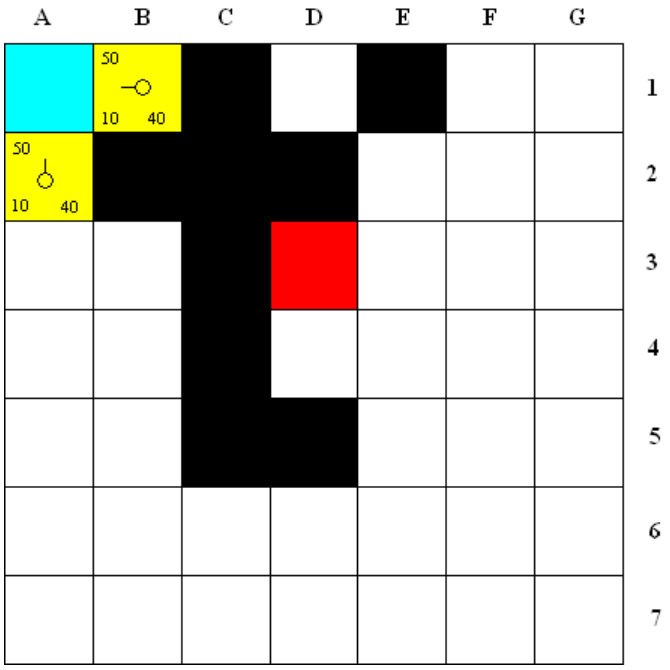


Figura 29: Adición de nodos adyacentes a lista abierta

Como hay dos nodos con F 50, se elige el nodo B1, por tener más antigüedad en la lista:

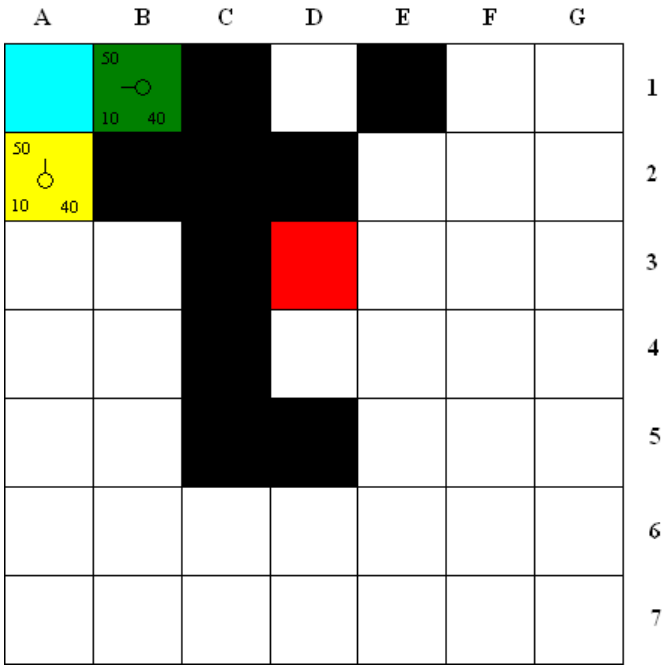


Figura 30: Selección de nodo

Como no tiene nodos adyacentes que añadir a la lista abierta, y los adyacentes que tiene en lista abierta no hacen un mejor camino, no se hace nada. Se vuelve a empezar, buscando en la lista abierta el nodo con la F más baja. Como sólo hay uno, se selecciona como actual, y se añaden sus adyacentes a la lista abierta:

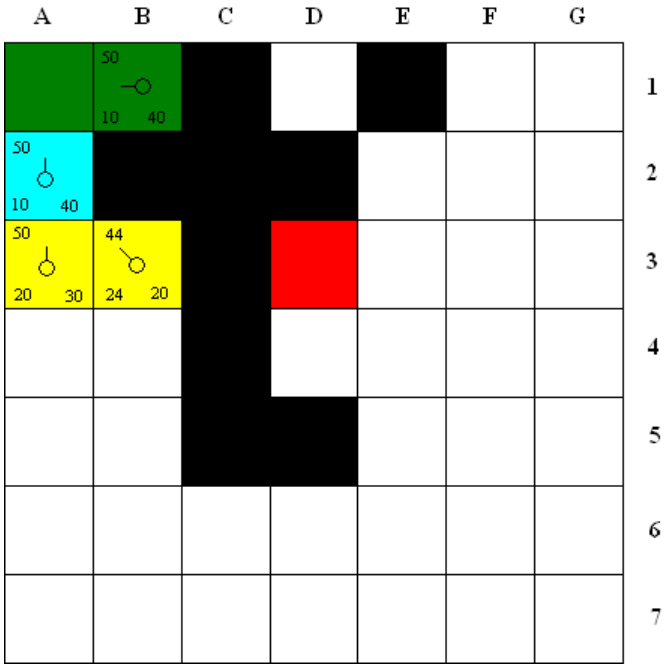


Figura 31: Adición de nodos adyacentes a lista abierta

Se selecciona de nuevo el nodo en la lista abierta con el coste f más bajo, en este caso B3, y se añaden a lista abierta sus adyacentes, marcando B3 como el padre y calculando los costes:

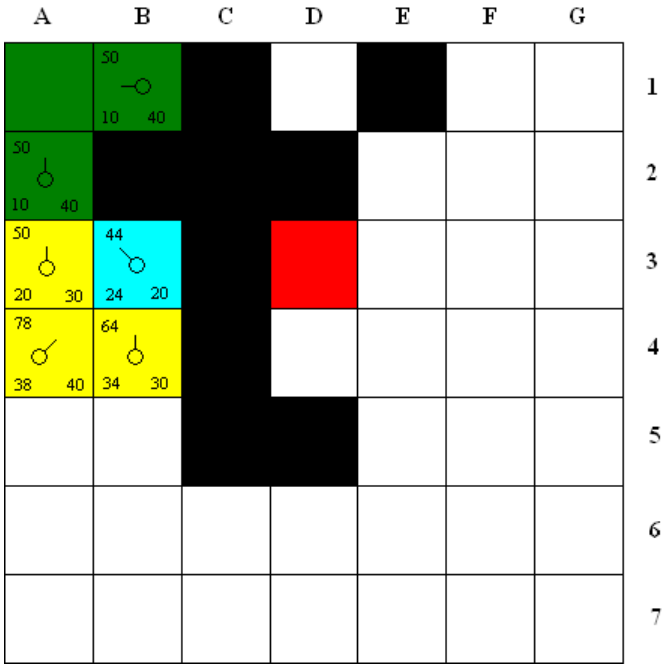


Figura 32: Adición de nodos adyacentes a lista abierta

Ahora el nodo en lista abierta con el coste f más bajo es A3. Al evaluar sus adyacentes, como ya están en lista abierta, se evaluará si el camino es mejor desde el actual a ese nodo. Se verá cómo el nodo A4 cambiará de padre.

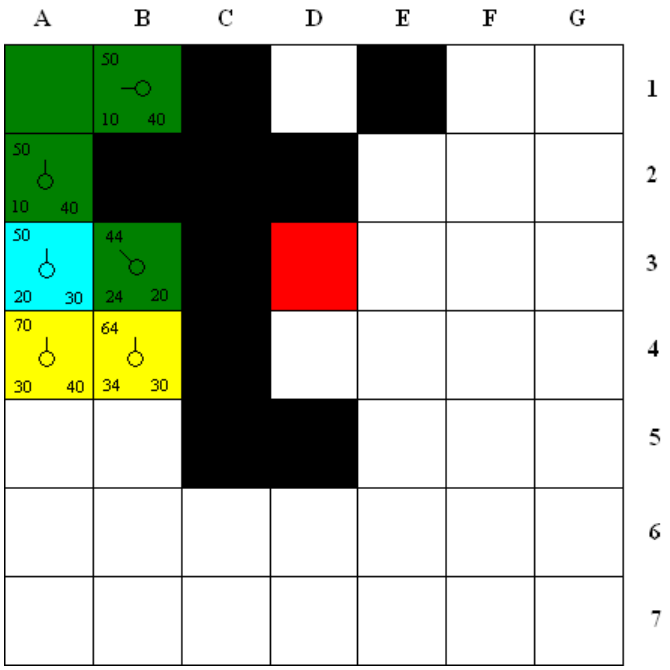


Figura 33: Cambio de padre

Así, el algoritmo se sigue ejecutando hasta que se llega al nodo objetivo:

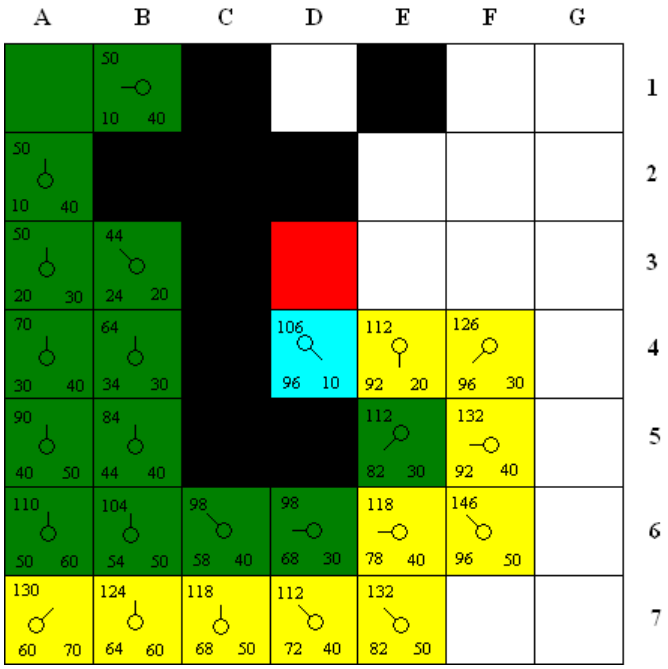


Figura 34: Repetición de iteraciones

Ahora, al investigar los nodos adyacentes al actual (D4), el nodo final será añadido a la lista abierta. A partir de ahí se puede crear la ruta que buscada. Sólo hay que ir hacia detrás, siguiendo las flechas que marcan el padre de cada nodo, hasta llegar al nodo inicial.

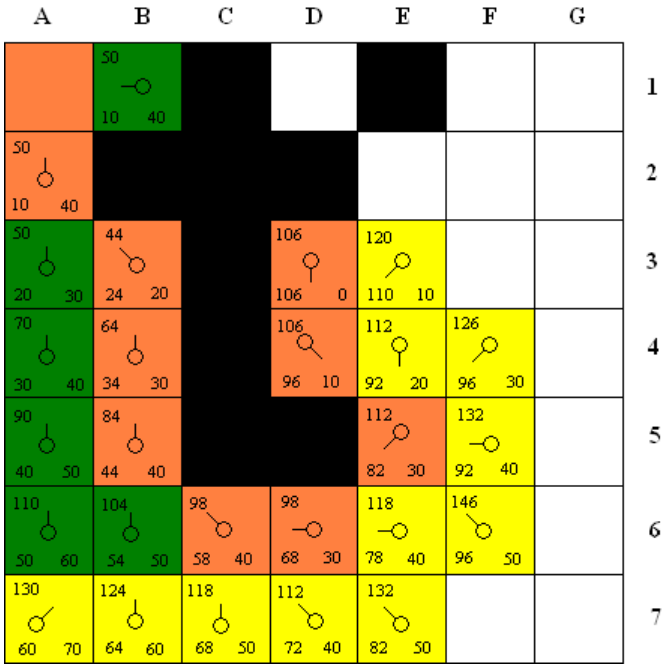


Figura 35: Vuelta atrás

Capítulo 4 – Implementación de la solución

El objetivo de esta aplicación es dotar al usuario de una solución para el guiado y seguimiento de ubicaciones en interiores mediante el uso de códigos QR definidos y establecidos previamente, de tal forma que el usuario sólo tenga que escanear el código y la aplicación le proporcionará la ruta que necesita.

4.1. Descripción del servidor

Para lograr el propósito marcado, se ha decidido utilizar una arquitectura cliente-servidor, de tal forma que sea el servidor quien contiene la información que utilizará el cliente (la aplicación móvil) para crear las rutas. La labor del cliente será conectarse a este servidor y en función de lo que el servidor le responda, decidir qué hacer.

Debido a que el cliente utiliza una base de datos para poder crear las rutas, será labor del servidor enviársela. Esto se consigue mediante unos ficheros de texto plano que lo acompañan y que contienen la definición de la base de datos. Se ha diseñado de esta forma para dar la mayor facilidad posible a los administradores de hacer cambios en ella cuando sea necesario (únicamente tendrá que modificar esos ficheros), pues la aplicación será adaptable a estas contingencias.

Como la base de datos es del tipo SQLite, el servidor contendrá un fichero con las instrucciones CREATE necesarias y otro con las instrucciones INSERT. La clave de la adaptabilidad de la aplicación reside en que siempre va a tener actualizadas las fechas de última modificación de los ficheros, y el cliente se va a valer de ello para decidir qué ficheros solicitarle para tener la base de datos actualizada en todo momento. Se puede ver que la conexión con el servidor es necesaria únicamente para redefinir la base de datos del cliente, y que una vez conocida funciona *offline*. Además, al ser una aplicación diseñada para su uso en entornos universitarios, corporativos o empresariales, se puede configurar el servidor de tal forma que sólo acepte peticiones que provengan de equipos que pertenezcan a su red.

Una vez el servidor se ponga en marcha, se ejecutará en bucle infinito a la espera de peticiones. Una vez reciba una petición de conexión, enviará al cliente las fechas de última modificación de los ficheros que definen la base de datos, y el cliente le enviará un mensaje en consecuencia. Es ahora cuando se bifurca el funcionamiento del servidor, y pueden darse distintas situaciones:

- El cliente no tiene base de datos ninguna, por lo que el servidor tiene que enviarle toda la base de datos.
- El cliente tiene base de datos pero debe actualizarla, así que el servidor le envía los cambios pertinentes.
- El cliente tiene la base de datos actualizada, no hay que hacer nada.

El diagrama de flujo correspondiente al servidor es este:

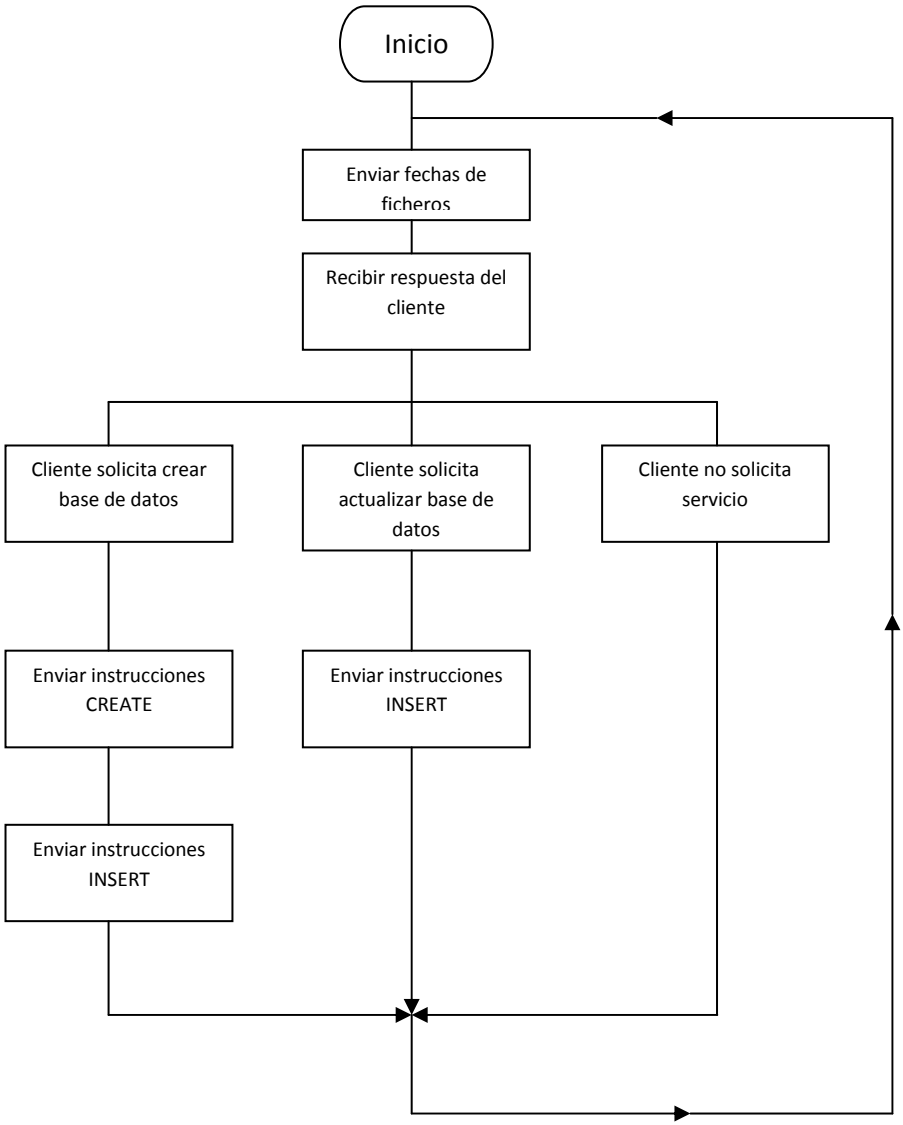


Figura 36: Esquema del servidor

Los ficheros que acompañan al servidor se llaman `instruccionesCreate.txt` e `instruccionesInsert.txt` y son los mencionados anteriormente que contienen las instrucciones necesarias para crear y actualizar la base de datos. Se han predefinido tres tipos de mensaje que condicionarán el funcionamiento tanto del servidor como del cliente:

- CLIENTE_CREAR: Cuando se recibe este mensaje por parte del cliente, el servidor lo entiende como una solicitud de ambos ficheros, y procede a enviárselos.
- CLIENTE_ACTUALIZAR: Al recibir una petición de actualización, se envía al cliente el fichero `instruccionesInsert.txt`, pues la base de datos ya está creada.
- CLIENTE_NO_ACTUAR: El cliente tiene la base de datos actualizada, por lo que no hay que realizar ninguna acción.

Como se puede ver, tenemos tres posibles situaciones. En la última simplemente terminará la iteración en curso del bucle infinito y comenzará otra nueva. La única diferencia entre la

primera y la segunda será que la primera enviará dos ficheros mientras que la segunda sólo enviará uno. Tanto para el caso de un fichero como el de dos ficheros el proceso de envío será el mismo:

- Se usa la clase `File` para abrir el fichero
- Se recorre el fichero contando el número de líneas que contiene
- Se envía al cliente el número de líneas
- Se vuelve a abrir el fichero. Como ahora tanto cliente como servidor saben el número de mensajes que van a recibir y enviar respectivamente, es posible crear un bucle controlado por esa variable encargado de enviar los mensajes en un lado y otro bucle en el otro lado para recibirlos.

Un ejemplo en código Java sería como el que sigue, utilizando la documentación existente en [13]:

```
fichero = new File("./instruccionesInsert.txt");
fechaServidorInsert = fichero.lastModified();

br = new BufferedReader(new FileReader(new File(
       "./instruccionesCreate.txt")));

numeroLineas = 0;
while(br.readLine() != null)
       numeroLineas++;
salidaDatos.writeInt(numeroLineas);

fichero = new File("./instruccionesCreate.txt");
fr = new FileReader(fichero);
br = new BufferedReader(fr);

for(int i = 1; i <= numeroLineas; i++) {
       linea = br.readLine();
       salidaDatos.writeUTF(linea);
}
```

La base de datos contendrá una única tabla, llamada Profesores, con el formato que se muestra a continuación. Cada fila de la tabla ofrece datos sobre un profesor.

id	apellidos	nombre	despacho	correo	teléfono	planta	departamento	mapa	nodo
INTEGER PRIMARY KEY	TEXT	TEXT	INTEGER	TEXT	INTEGER	INTEGER	TEXT	INTEGER	INTEGER

La primera fila muestra el nombre de cada campo, y la segunda su tipo de dato SQLite asociado. Son especialmente importantes los campos id, mapa y nodo, pues son los que necesitará el algoritmo de *pathfinding* (los motivos de la importancia de estos campos se explicarán en el capítulo 4). El campo teléfono también puede ser importante pues será necesario si el usuario indica que quiere llamar a un despacho.

4.2. Descripción del cliente

El cliente utiliza de manera conjunta varias tecnologías, que trabajando entre ellas logran el propósito deseado de guiar al usuario en su camino. Tiene especial importancia el algoritmo de

encaminamiento utilizado para generar las rutas que el usuario debe seguir, y que se explica a continuación.

4.2.1. Implementación del algoritmo A*

Se mencionó en el capítulo 2 que se han hecho unas ligeras modificaciones al algoritmo en aras de la eficiencia. Son las siguientes:

- No hay lista cerrada. En su lugar, habrá una variable booleana en cada nodo (casilla) que marca si ha sido procesado o no.
- Los nodos no transitables simplemente no se tendrán en cuenta.

Se ilustrará mejor con un ejemplo:

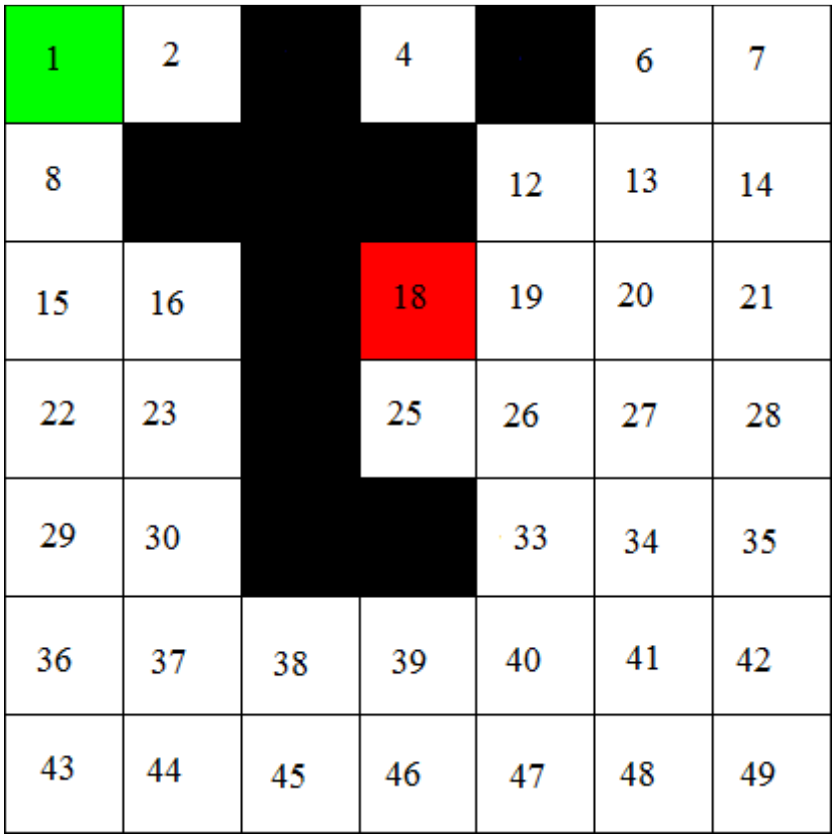


Figura 37: Mapa de ejemplo

En este mapa de ejemplo se ven las adyacencias entre cada nodo: 1 tiene como adyacentes a 2 y 8, 2 tiene como adyacentes a 1 y a 8, 8 tiene como adyacentes a 1, 2,15 y 16, etc… Además, será necesario conocer la ubicación de cada nodo respecto a los demás. Esto se hará considerando la malla como una matriz (el nodo 1 tendrá coordenadas (1,1), el nodo 16 tendrá coordenadas (3,2), el nodo 40 tendrá coordenadas (6,5), etc…).

Los mapas físicos se modelarán mediante un fichero de texto. Para el mapa concreto puesto de ejemplo el fichero contendría lo siguiente:

```
39

1 2 8-1 1
```

```
2 1 8-1 2

6 7 12 13 14-1 6

7 6 13 14-1 7

8 1 2 15 16-2 1

12 6 13 18 19 20-2 5

13 6 7 12 14 19 20 21-2 6

14 6 7 13 20 21-2 7

15 8 16 22 23-3 1

16 8 15 22 23-3 2

18 12 19 25 26-3 4

19 12 13 18 20 25 26 27-3 5

20 12 13 14 19 21 26 27 28-3 6

21 13 14 20 27 28-3 7

22 15 16 23 29 30-4 1

23 15 16 22 29 30-4 2

25 18 19 26 33-4 4

26 18 19 20 25 27 33 34-4 5

27 19 20 21 26 28 33 34 35-4 6

28 20 21 27 34 35-4 7

29 22 23 30 36 37-5 1

30 22 23 29 36 37 38-5 2

33 25 26 27 34 39 40 41-5 5

34 26 27 28 33 35 40 41 42-5 6

35 27 28 34 41 42-5 7

36 29 30 37 43 44-6 1

37 29 30 36 38 43 44 45-6 2

38 30 37 39 44 45 46-6 3

39 33 38 40 45 46 47-6 4

40 33 34 39 41 46 47 48-6 5

41 33 34 35 40 42 47 48 49-6 6

42 34 35 41 48 49-6 7

43 36 37 44-7 1
```

```
44 36 37 38 43 45-7 2

45 37 38 39 44 46-7 3

46 38 39 40 45 47-7 4

47 39 40 41 46 48-7 5

48 40 41 42 47 49-7 6

49 41 42 48-7 7
```

Estos ficheros tienen un formato específico que indica la posición de unos nodos respecto de otros. La primera línea contiene el número de nodos transitables que contiene el mapa, 39 en este caso. Para el resto de líneas el primer número indica un nodo del mapa, mientras que el resto de números hasta el guión son sus adyacentes. Los dos números que hay tras el guión representan la ubicación del nodo en el eje *x* y en el eje *y*, respectivamente. Así, el nodo 1 tiene al 2 y al 8 como adyacentes y tiene coordenadas (1,1), el 2 tiene como adyacentes al 1 y al 8 y tiene coordenadas (1,2), etc… De esta forma en un único fichero se puede tener definido todo el mapa y establecer las adyacencias, de tal forma que con el procesado del fichero es posible crearlo.

Generación del mapa físico

Se harán dos pasadas sobre el fichero. En la primera se leerá la primera línea para así poder crear un array del tamaño necesario, sin sobredimensionar ni subdimensionar, para almacenar todos los nodos. A continuación se leerá línea por línea pero teniendo en cuenta únicamente el primer número (el identificador del nodo) y los dos números tras el guión (sus coordenadas), para así ir creando todos los nodos del mapa y añadiéndolos al array. Después, en una segunda pasada, se ignorará la primera línea y de la segunda en adelante se ignorará también el primer número, y se tendrán en cuenta los siguientes, son sus adyacentes. Lo que se hará a la hora de añadir nodos adyacentes es recorrer el array creado en la primera pasada buscando ese nodo, para así no tener que crear más objetos. De esta forma se consigue tener todos los nodos y sus adyacentes en memoria para poder trabajar con ellos.

Para generar las rutas primero es necesario crear el mapa físico en que se encuentre el usuario, y será la clase `ListaNodos` la encargada de esta labor. Puesto que el diseño elegido implica extraer información del fichero en varias pasadas, hay que encontrar alguna forma de poder "rebobinarlo" (de forma similar a la función `rewind` del lenguaje C).

Las interfaces de lectura de datos no permiten volver atrás, sino únicamente leer hacia adelante. Sin embargo, la clase `BufferedReader` (que se describe en [17]) provee de un buffer intermedio de lectura que se podrá utilizar para realizar esa vuelta atrás necesaria. Para ello dispone de los métodos `mark` y `reset`, que son idóneos para el propósito deseado. Como el mapa físico está modelado por un fichero de texto y en el uso del patrón decorador para abrir ficheros intervienen objetos `BufferedReader`, se ha diseñado el constructor de `ListaNodos` de tal forma que su parámetro sea precisamente un objeto de este tipo. Con este enfoque se logra que cuando se invoque a este constructor reciba directamente el objeto sobre el que tendrá que realizar los barridos y vueltas atrás, pues al ser un objeto BufferedReader ya es posible usar `mark` y `reset`.

El método `mark` establece una marca en el buffer de entrada, y recibe como parámetro un entero que representa el número de caracteres que pueden leerse antes de anular la marca; si se excediera el límite de caracteres que establece este parámetro, no se podrá realizar la vuelta atrás. Por su parte, el método `reset` no recibe parámetros y devuelve el puntero al punto donde se estableció la marca, y si por algún motivo fuera invocado con la marca invalidada, provocará un error.

Otro aspecto importante es el parámetro que va a recibir `mark`. La solución más sencilla sería poner un tamaño muy grande de buffer para no tener que preocuparse por invalidar la marca, pero eso sería ineficiente. Hay que adquirir un compromiso entre la eficiencia (el tamaño de buffer) y el funcionamiento que se espera de la aplicación. En este caso concreto se ha decidido establecer un tamaño de 3072 caracteres (3 KB), pero modificar este valor es una tarea muy simple ya que sólo hay que modificar una línea en el programa.

4.2.2. Estructuración y funcionamiento del algoritmo utilizado

Para desarrollar el algoritmo se ha realizado un diseño en el que intervienen varias clases. La primera se llama `Camino` y tiene como objetivo encapsular las rutas que genere el algoritmo. La segunda es `ListaEnlazada`, modela la lista abierta y contiene métodos que el algoritmo usará con frecuencia para la comprobación de existencia y búsqueda de nodos. La tercera clase se llama `ListaNodos` y su cometido es modelar el mapa físico del entorno en el que se encuentre el usuario, además contiene también métodos para el diagnóstico y búsqueda de nodos, que al igual que con `ListaEnlazada` serán muy utilizados por el algoritmo. Por último, la clase `Nodo` contiene todos los parámetros necesarios para el desempeño de la generación de rutas, así como métodos para establecer u obtener esos parámetros y otros métodos que también serán útiles. En el diagrama anterior se muestra la relación entre ellas así como sus métodos más importantes y significativos.

El algoritmo ejecuta una serie de pasos de forma iterativa, aunque para ello necesita realizar unos pasos previos. Cuando sea invocado por primera vez comenzará creando una representación en memoria de todos los nodos que componen la ubicación donde se encuentre el usuario así como las adyacencias entre ellos, es decir, el mapa; esta tarea se realizará de la forma detallada en el apartado anterior.

El siguiente paso es obtener los identificadores del nodo en el que está el usuario en el momento de lanzar el algoritmo y del nodo al que se quiere dirigir: son el nodo origen y el nodo destino. El nodo origen se obtendrá al escanear un código QR y el nodo destino se obtendrá consultando en la base de datos después de que el usuario seleccione un lugar al que ir entre todos los posibles. Los mapas están diseñados de tal forma que cada nodo contiene un identificador único, que es uno de los múltiples atributos que contiene la clase `Nodo`. Al ser un identificador único es un parámetro especialmente importante puesto que todas las funciones de diagnóstico y búsqueda se basan en éste, y es por eso que cuando se dice que se obtiene un nodo en realidad se refiere a obtener su identificador.

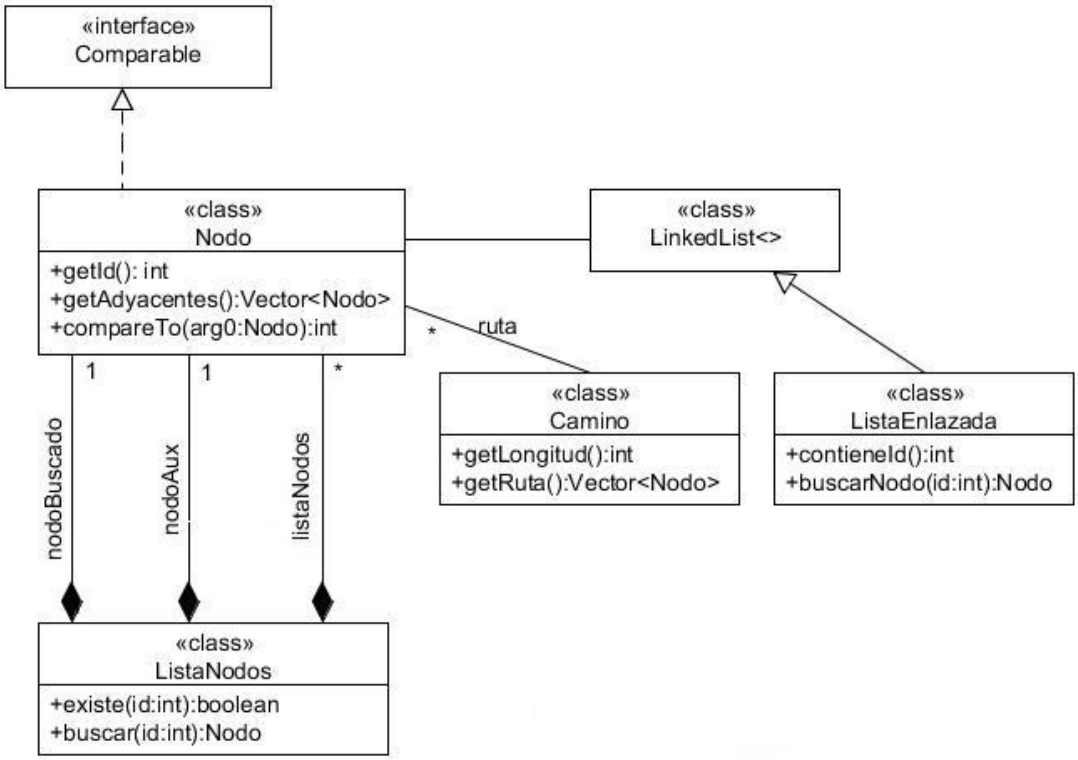


Figura 38: Diagrama UML del algoritmo

Por último, antes de empezar a iterar, se añadirá el nodo inicial a la lista abierta. Para esto se utiliza el método `add`, porque `ListaEnlazada` hereda de la clase `LinkedList` parametrizada con objetos `Nodo`. Esto permitirá utilizar otra serie de métodos cuyo objetivo será la gestión de la lista abierta.

Ahora, en cada iteración del algoritmo la primera tarea será obtener el primer nodo de la lista abierta. El diseño escogido hará que se utilice el método `poll` para extraer el primer nodo (que se designará como nodo actual) pero también eliminarlo de la lista abierta porque será marcado como procesado (equivalente a moverlo a lista cerrada). A continuación se utilizará el método `getAdyacentes` para obtener un `Vector` de objetos `Nodo` que contiene todas las casillas a evaluar y que están alrededor de la actual.

Después de obtener los nodos adyacentes al actual, se recorrerá el conjunto que forman, comprobando si cada nodo está o no procesado (en lista cerrada). Si está procesado simplemente se ignorará y se avanzará al siguiente; en caso contrario produce una bifurcación: se utilizarán los métodos `contieneId` de `ListaEnlazada` y `getId` de `Nodo` para discernir si el nodo evaluado pertenece o no a la lista abierta. Si no está en lista abierta se añade a ella, se marca el nodo actual como su padre y se calculan sus costes de movimiento; si ya está en lista abierta se comprueba si el coste hasta este nodo es menor que el que se tenía antes: en caso de serlo se actualizan los costes y si no lo es no se hace nada. Sea como sea, tanto si se actualizan costes como si se añade un nuevo nodo a la lista abierta, ésta después siempre va a ser reordenada.

La función que calcula el coste de movimiento tiene tres parámetros: G es el coste de movimiento hasta un nodo determinado, H es la distancia estimada hacia el destino y F es la suma de ambos valores. Se ha establecido que un movimiento ortogonal (hacia arriba/abajo o izquierda/derecha) tenga un coste G=10, mientras que un movimiento en diagonal tenga un coste G=14. Para calcular la distancia hacia el destino se utilizará una aproximación heurística, la llamada "Regla Manhattan", consistente en que la distancia entre dos puntos es la diferencia absoluta de sus coordenadas (las rectas que sigue la trayectoria son paralelas a los ejes de coordenadas).

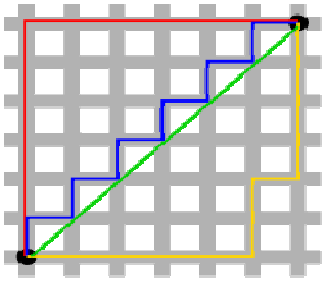


Figura 39: Dstancia Manhattan

Es muy importante el aspecto de la reordenación de la lista abierta. En cada comienzo de iteración el algoritmo necesita extraer un nodo de la lista abierta: aquel que tenga un menor coste y que designará como nodo actual. Se ha decidido utilizar una implementación de lista enlazada ya definida por Java en la clase `LinkedList` porque existe el método `sort` de la clase `Collections` que será muy útil para reordenar la lista abierta cuando se produzca algún cambio en ella o en alguno de sus nodos. De esta forma el nodo con el menor coste siempre será el primero y podrá obtenerse fácilmente mediante `poll`. El método `sort`, como se detalla en [18], recibe una `LinkedList` y devolverá otra `LinkedList` ordenada de forma ascendente (de menor a mayor). Este método tiene una particularidad: para poder utilizarse los elementos que componen la `LinkedList` deben implementar la interfaz `Comparable`. Esta interfaz contiene el método `compareTo`, que será utilizado por `sort` para ordenar la lista mediante una variación de la ordenación por mezcla.

Antes de finalizar cada iteración, se comprueba mediante `contieneId` si se ha alcanzado el nodo de destino, es decir, si la lista abierta contiene el nodo al que quiere llegar el usuario. En el caso de que se haya alcanzado, se marcará como `true` una variable booleana que es precisamente quien controla la repetición del bucle. Una vez que hayan terminado las iteraciones se habrá alcanzado el nodo final, y como cada nodo contiene una referencia a su anterior, se puede construir la ruta mediante esas referencias hasta que encontremos un nodo que no tenga padre, sabiendo así que será el nodo inicial.

El diagrama de flujo utilizado para desarrollar el algoritmo es el que se muestra en la página siguiente.

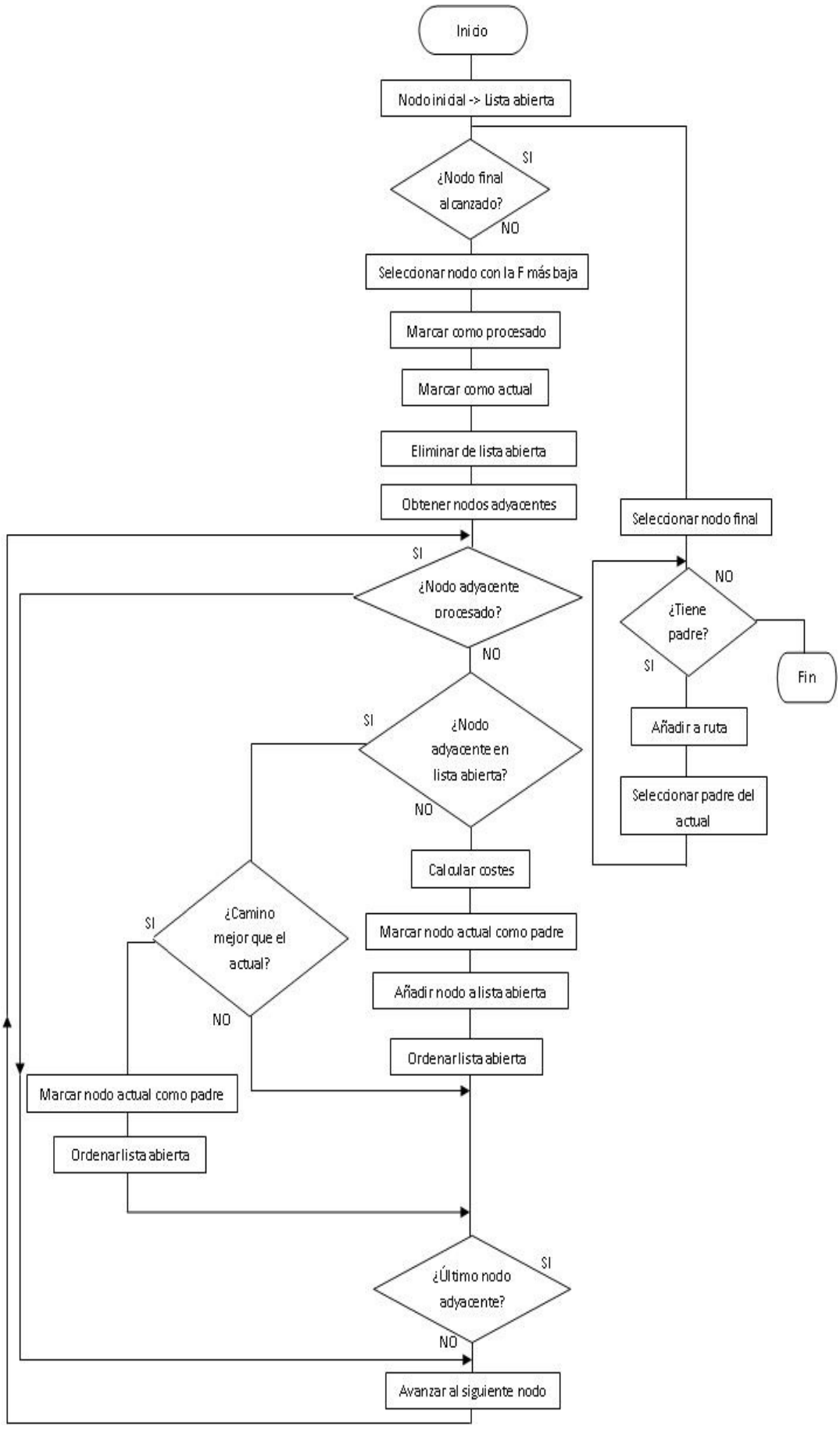


Figura 40: Flujograma de A*

4.2.3. Funcionamiento de la aplicación

El primer paso de la aplicación será crear o actualizar la base de datos, pues ésta es clave para su correcto funcionamiento. Esto se producirá al iniciarse la aplicación, mostrando un mensaje de bienvenida y un botón para que el usuario lo pulse y la aplicación se conecte al servidor para obtener la definición más reciente de la base de datos. Tal y como se explicó en el capítulo del servidor, lo primero que hará tras conectar con el servidor será recibir las últimas fechas de modificación de las instrucciones CREATE e INSERT que tiene éste, para después compararlas con las que tiene ella localmente y en función de ellas decidirá si tiene que crear/actualizar la base de datos o no hacer nada.

Después de conseguir la versión más reciente de la base de datos, la aplicación mostrará un campo en el que el usuario podrá escribir el lugar al que quiere ir o el profesor al que quiere visitar. Previamente, se hace un barrido sobre la base de datos recogiendo todos los nombres y apellidos de los profesores y los nombres de las ubicaciones que no sean despachos de profesores. Con esta operación se consigue dotar al campo de texto de la función de autocompletado: a medida que el usuario vaya escribiendo la aplicación filtrará entre todas las posibles opciones para mostrar en forma de lista desplegable aquellas que contengan lo que vaya escribiendo el usuario.

Cuando se seleccione una opción de la lista desplegable se abrirá un cuadro de diálogo que podrá presentar al usuario una o dos opciones. Si la opción elegida es un profesor dará la opción de llamarle por teléfono o de ir a verle personalmente a su despacho y si es otro tipo de ubicación dará únicamente la opción de ir en persona. Si se marca la opción de llamar por teléfono la aplicación hará una consulta a la base de datos para obtener el número y automáticamente realizar la llamada.

El caso de que el usuario quiera ver personalmente al profesor se va a explicar por separado porque implica un mayor grado de complejidad en la lógica implementada. En primer lugar abrirá otro cuadro de diálogo avisando al usuario para que se disponga a escanear un código QR en los alrededores. Para esta tarea se hará uso de una aplicación de terceros (QR Droid) al permitir su fácil integración en otras aplicaciones, tal como se muestra en [20]. El código QR escaneado contendrá un ID único ya preestablecido y que codifica una ubicación, representando una celdilla, una casilla, un nodo del mapa físico. Tras el escaneo la aplicación sabe dónde se encuentra el usuario y hacia dónde se quiere dirigir, por lo que tiene los dos parámetros que necesita para poder lanzar el algoritmo que creará la mejor ruta posible para llegar al destino. La aplicación sabrá en todo momento el destino deseado por el usuario, permitiendo siempre escanear más códigos QR para recalcular la ruta, porque tendrá presente en todo momento cuál es el destino y permitirá escanear códigos mientras no se haga con el que identifica el destino.

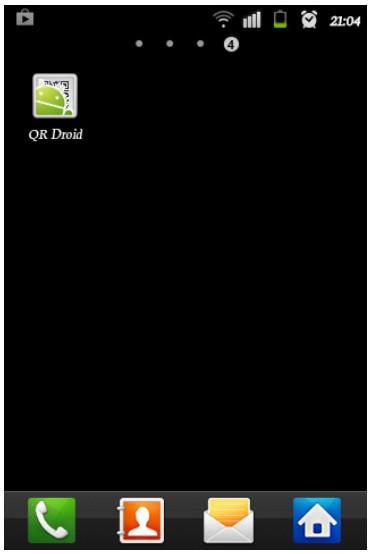


Figura 41: Aplicación QR Droid

Se ha diseñado una implementación flexible del algoritmo con la aplicación, logrando así crear rutas incluso a destinos que no estén en el mismo piso en que se encuentra el usuario. La aplicación hará uso de lo que se ha denominado "supermapa", que establece las conexiones entre los pisos. Estas conexiones se logran mediante los llamados nodos enlace: son unos nodos concretos de cada piso en los que hay una escalera. De esta manera, si el usuario quiere ir a un lugar que no está en el mismo piso que la ubicación de inicio, la aplicación le llevará a la escalera más cercana.

```
0 1-108 118
0 2-108 118
1 0-52 74
1 2-52 74
2 0-59 82
2 1-59 82
```

La serie de números superior es el contenido del fichero supermapa. En cada línea los dos primeros números indican la transición de piso (del 0 al 1, del 0 al 2, del 1 al 0, etc.), mientras que los números a la derecha del guión indican los nodos enlace. Cuando se cree una ruta cuyo destino no esté en la misma planta, la aplicación buscará el nodo enlace más cercano y llevará al usuario hasta allí. Una vez haya cambiado de piso simplemente tendrá que escanear otro código QR y repetir el proceso. A continuación se presenta un caso de uso de la aplicación.

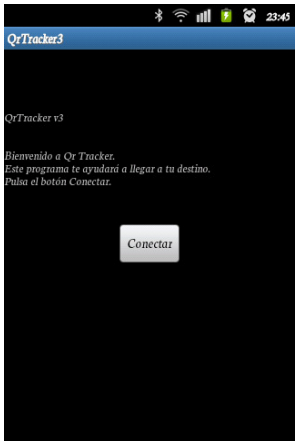


Figura 42: Conexión

1. En primer lugar, el usuario inicia el programa, y se muestra una pantalla de bienvenida con un botón para conectar con el servidor.

Figura 43: Selección de ubicación

2. Se muestra la pantalla de selección. El usuario escribe "García" y la aplicación filtra todos los profesores mostrando los que tengan García como primer o segundo apellido.

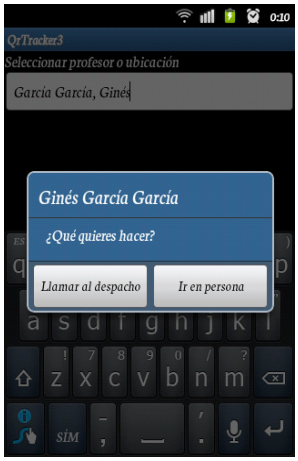


Figura 44: Decisión del usuario

3. Selecciona una opción de la lista desplegada y se abre un cuadro de diálogo preguntando qué hacer.

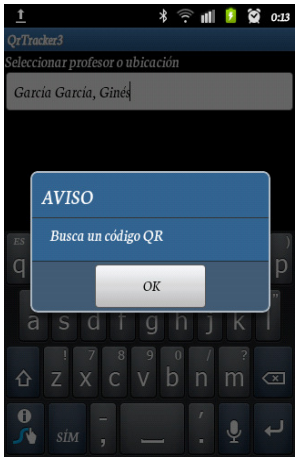


Figura 45: Aviso para buscar código QR cercano

4. La aplicación avisa al usuario para que busque un código QR en los alrededores.

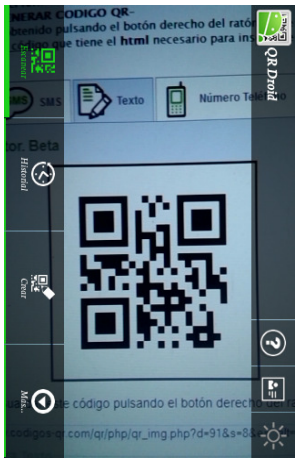


Figura 46: Escaneo del código

5. Se abre la aplicación QR Droid para escanear el código.

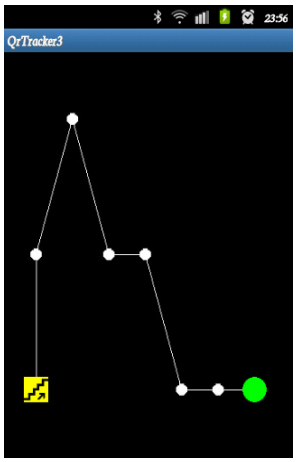


Figura 47: Ruta hacia otro piso

6. Tras escanear el código QR la aplicación obtiene toda la información que necesita de la base de datos y dibuja una ruta. El punto verde es el nodo origen y cada punto blanco es un nodo del mapa. En este caso hay que subir una planta, y se marca mediante un gráfico.

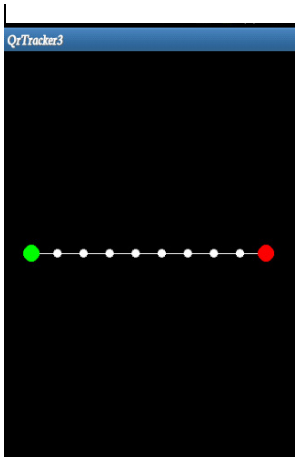


Figura 48: Ruta hacia el mismo piso

7. Tras subir las escaleras el usuario escanea otro código QR cercano. El punto rojo indica el destino y que está en la misma planta.

Figura 49: Destino alcanzado

8. El usuario ha llegado al despacho que buscaba, escanea el código QR que hay en la puerta y la aplicación confirma que ha llegado a su destino. Desde aquí vuelve otra vez a la pantalla de bienvenida.

Se ha querido dotar a la aplicación de la posibilidad de que el usuario pueda cambiar su decisión; esto es, si decide ir a un sitio determinado pero a mitad de camino se arrepiente y

quiere ir a otro lugar la aplicación podrá responder de forma adecuada para satisfacer su necesidad. En Android al darle a la flecha hacia atrás no se cierra la aplicación ni se mata el proceso, sino que se vuelve a la pantalla anterior siendo el sistema operativo quien mata el proceso cuando esté falto de recursos. Esto puede provocar una larga lista de actividades y procesos en la pila si se ejecuta muchas veces la aplicación y hay muchos cambios de decisión por parte del usuario. Para evitar esto, se ha implementado código para limpiar la pila de actividades en momentos concretos de la ejecución. Su máquina de estados sería la que se muestra en la página siguiente.

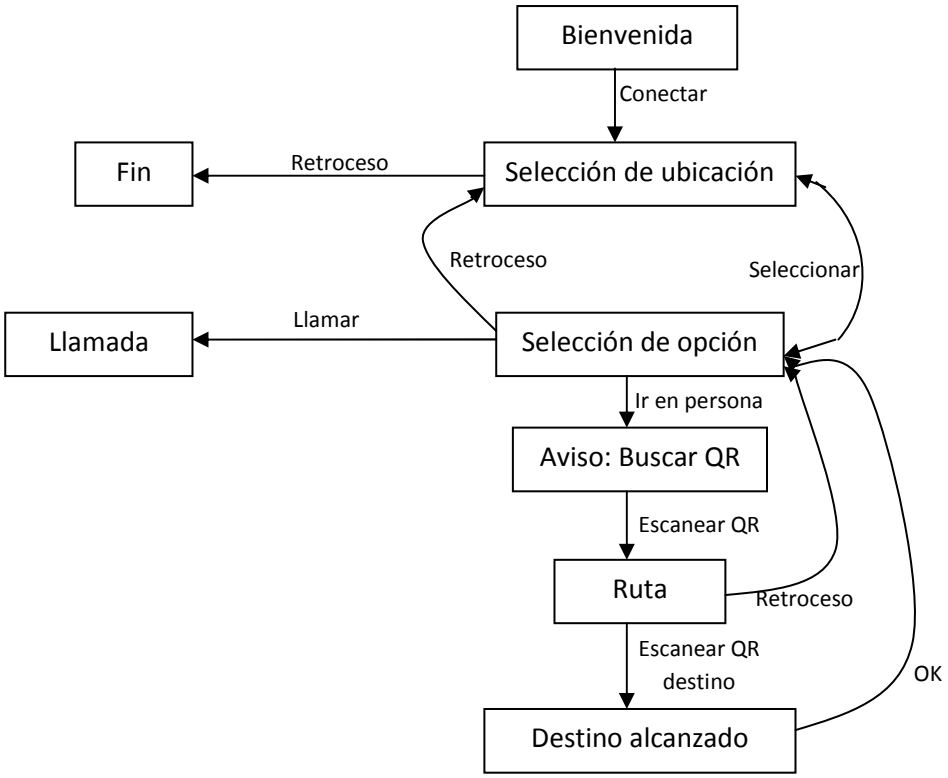


Figura 50: Máquina de estados de la aplicación

4.2.4. Integración con la base de datos SQLite

La aplicación necesita guardar una serie de datos de forma persistente, no volátil. Estos datos son la información relevante a cada profesor (identificador, nombre, apellidos, despacho, correo electrónico y departamento entre otros). De entre todas las formas que ofrece Android para almacenar información se ha elegido SQLite dada su idoneidad para el propósito necesitado, pues es un sistema de bases de datos que no necesita configuración ni servidor y tiene una biblioteca de un tamaño relativamente pequeño (sobre los 275 KiB).

El acceso al gestor de bases de datos SQLite se describe en [19] y consiste en utilizar una clase que herede de `SQLiteOpenHelper` pero para esta aplicación concreta se ha adoptado un enfoque diferente, utilizando de base el contenido expuesto en [16]. Este enfoque consiste en utilizar una clase `AccesoBD` que controla los accesos, peticiones y modificaciones a la base de datos, y en la que uno de sus atributos es un objeto de la clase que deriva de `SQLiteOpenHelper` (a través de este atributo es como se realizarán realmente las

conexiones a la base de datos). Así, se obtiene un mayor grado de modularidad y centralización al pasar todas las peticiones a través de este objeto intermediario. Se muestra a continuación este punto de vista. No se muestran todos los métodos en el diagrama UML, sólo los más significativos.

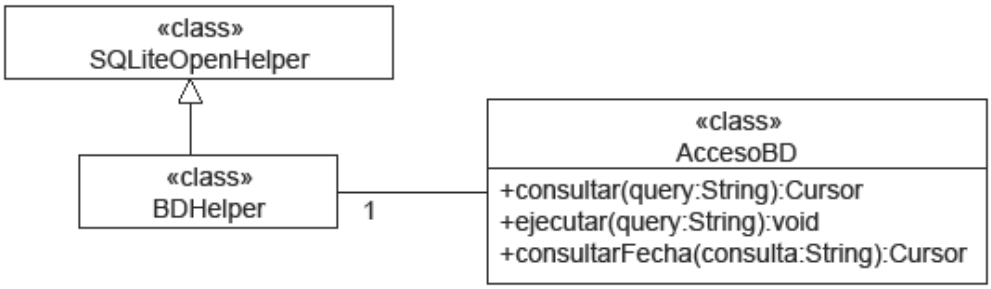


Figura 51: Enfoque SQLite

El esquema presentado es importante porque se necesita extraer, añadir o consultar información en ciertos momentos del funcionamiento de la aplicación:

- En la fase de conexión, en el caso de que haya que crear desde cero la base de datos, se necesitará añadir a la base de datos la información que envíe el servidor, lo que se traduce en ejecutar una serie de instrucciones CREATE e INSERT. Si sólo hay que actualizar la base de datos, igualmente se necesita consultar las fechas de última modificación por ser el valor que determina si hay que actualizarla o no.
- En la fase de selección de ubicación, hay que extraer toda la información necesaria sobre el lugar que elija el usuario, como puede ser el teléfono o el piso en el que se sitúa.
- En la fase de ruta, se realizan consultas cuando el usuario escanea un código QR para determinar en qué punto exacto está (planta y nodo concretos) para así poder dibujar el camino a seguir además de dibujar el gráfico que corresponda si hay que subir o bajar una planta.

El problema está en que estas tres fases se realizan dentro de actividades diferentes de la aplicación, lo que impide un punto de acceso único a la base de datos. Para solventar este aspecto se utiliza la clase `AccesoBD`, cuyas funciones serán recoger las peticiones para enviárselas a la base de datos y recoger los resultados que éstas produzcan para enviárselos de nuevo a quien hizo la petición. Así, se pueden realizar dos tipos de peticiones a la base de datos: para añadir información o para extraerla. Sea cual sea el tipo de petición, el constructor de `AccesoBD` será el mismo: necesita un objeto `Context` porque al crearse un objeto `AccesoBD` éste internamente creará un objeto `BDHelper` que es el que usará para hacer las peticiones y recoger sus resultados; este objeto necesita obligatoriamente un `Context` para poder ser creado. Mediante este punto de vista, cada clase puede crear distintos objetos de acceso pero la base de datos accedida es la misma en todas las actividades.

Gracias a la necesidad de este objeto `Context` se puede crear un `AccesoBD` en las `Activity` de la aplicación. El primer paso sería crear el objeto dentro de la actividad:

```
AccesoBD accesoBD;
accesoBD = new AccesoBD(this);
```

Después de crear el punto de acceso, hay que realizar un paso previo para poder operar con la base de datos, consistente en abrirla en modo lectura o en modo escritura (se ejecutará uno de los dos, no ambos):

```
accesoBD.abrirEscritura();
accesoBD.abrirLectura();
```

Ya está todo dispuesto para poder operar con la base de datos. Ya que la clase `SQLiteDatabase` (que también se usa en `AccesoBD`) ofrece métodos para ejecutar órdenes CREATE o INSERT y otros métodos para ejecutar órdenes SELECT, aquí se implementarán métodos que los encapsulen. De esta forma al llamar a alguno de estos métodos se ejecutarán los métodos `rawQuery` o `execSQL` por debajo:

```
Cursor c = null;
c = accesoBD.consultar("SELECT apellidos,nombre FROM Profesores");

linea = "INSERT INTO Profesores (id,apellidos,nombre,despacho,correo,
        telefono,planta,departamento,mapa,nodo)
        VALUES (1, Alcaraz Espín', 'Juan José', 14,
       'juan.alcaraz@upct.es', 968325973, 1, 'TIC', 1, 14)";
accesoBD.ejecutar(linea);
```

Por último, cuando se haya terminado de trabajar con la base de datos habrá que cerrar el acceso a ella:

```
accesoBD.cerrar();
```

Es importante constar que muchos de estos métodos pueden lanzar una `SQLException`, por lo que llevan la cláusula `throws` en sus cabeceras y tendrán que ejecutarse dentro de un bloque `try-catch`. Se presenta a continuación el contenido de `AccesoBD` para una mejor comprensión.

```
public class AccesoBD extends Activity {

      private SQLiteDatabase db;
      private BDHelper dbHelper;

      public AccesoBD(Context contexto) {
            dbHelper = new BDHelper(contexto);
      }

      public void abrirEscritura() throws SQLException {
            db = dbHelper.getWritableDatabase();
      }

      public void abrirLectura() throws SQLException {
            db = dbHelper.getReadableDatabase();
      }

      public void cerrar() {
            dbHelper.close();
      }

      public Cursor consultar(String query) throws SQLException {
            return db.rawQuery(query, null);
      }

      public void ejecutar(String query) {
            db.execSQL(query);
```

```
        }


//      //Un método que nos diga si existe la tabla que nos pasan
        public boolean existeTabla(String tableName) {
                if (tableName == null || db == null || !db.isOpen()) {
                    return false;
                }

                Cursor cursor = db.rawQuery("SELECT COUNT(*) FROM
                    sqlite_master WHERE type = ? AND name = ?", new
                    String[] {"table", tableName});

                if (!cursor.moveToFirst()) {
                    return false;
                }

                int count = cursor.getInt(0);
                cursor.close();
                return count > 0;
        }


        public Cursor consultarFecha(String consulta) {
              Cursor c = db.rawQuery(consulta, null);
              return c;
        }


}       //Cierra clase
```

4.2.5. Dibujado de las rutas

Cuando se calcula una ruta hay que mostrarla de alguna forma. Se ha optado por hacerlo de forma gráfica. En un primer momento se apostó por gráficos de tamaño fijo, pero se descartó por la gran diversidad de teléfonos Android existentes en el mercado con diferentes tamaños de pantalla que provocarían que la ruta no se mostrara de forma clara en pantallas de distinto tamaño al esperado. Es necesario que la aplicación se pueda adaptar a distintos tamaños, lo que implica que el tamaño de los gráficos mostrados dependerá del teléfono en que se ejecute. El tamaño en píxeles del ancho y alto de la pantalla del teléfono se obtiene con el código que se muestra.

```
int anchoPantalla, altoPantalla;
DisplayMetrics dm;
dm = new DisplayMetrics();
getWindowManager().getDefaultDisplay().getMetrics(dm);
anchoPantalla = dm.widthPixels;
altoPantalla = dm.heightPixels;
```

Se realizará una división del tamaño de la pantalla en forma de matriz. El ancho y alto de cada fila y columna dependerá de la ruta creada: a mayor número de nodos más estrechas serán las filas y columnas, y por tanto más pequeños serán los puntos dibujados. También, para mayor comprensión del usuario, los nodos intermedios (los dibujados en color blanco) se dibujaran a un tercio del tamaño de los nodos inicial y final. Se puede ver una comparación en las figuras.

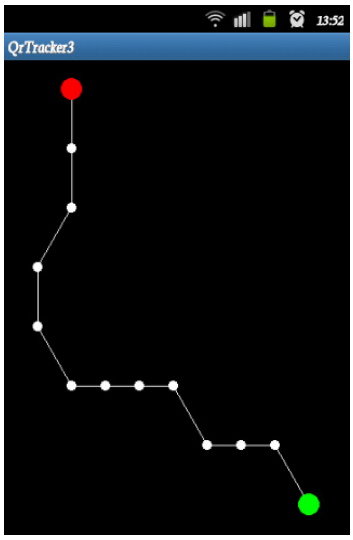


Figura 53: Muchos nodos – círculos pequeños

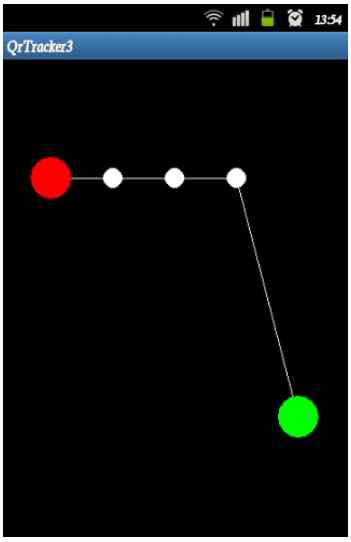


Figura 52: Pocos nodos – círculos grandes

Como los gráficos dependen de la ruta elegida, se hace un procesado sobre ella para generar el gráfico. Lo primero será obtener las dimensiones de la pantalla de la forma explicada antes. Una vez obtenidas, se recorren los nodos de la ruta buscando el máximo y el mínimo valor de la coordenada *x* y de la coordenada *y*. Estos son los valores que se necesitan para calcular el tamaño de la matriz. Mediante su diferencia se podrá saber el número de filas y de columnas que contiene la matriz (que no tiene por qué ser cuadrada). Después de obtener el número de filas y el número de columnas, ambos números se incrementarán en 2, con el objetivo de añadir columnas a izquierda y derecha de la pantalla y filas arriba y abajo; de esta manera se evitará que el gráfico aparezca pegado a los bordes de la pantalla.

Cuando ya se tiene el tamaño correcto de la matriz, se obtienen los incrementos de ancho y de alto:

```
incrementoAncho = (int)(anchoPantalla/columnasPintar);
incrementoAlto = (int)(altoPantalla/filasPintar);
```

Como podría pasar que las celdas no sean cuadradas sino rectangulares, se obtiene el lado mayor, y que usaremos como radio de los círculos a dibujar (para que el tamaño del círculo no exceda del tamaño de la celda):

```
if(incrementoAncho < incrementoAlto)
      radio = incrementoAncho;
else if(incrementoAlto < incrementoAncho)
      radio = incrementoAlto;
else
      radio = incrementoAncho;
```

Anteriormente se han obtenido los máximos y mínimos de las coordenadas, pero éstos son valores absolutos. Estos valores no sirven para hacer los gráficos porque se necesitan valores relativos. Para este propósito se dispone de la clase `Centro`, que contiene los atributos *x* e *y*, las coordenadas traducidas a valor relativo de cada nodo de la ruta. Así, la tarea consiste en

crear un array de objetos `Centro` e irlo rellenando con la traducción de las coordenadas de los nodos que componen la ruta.

Se dibujará un círculo en el interior de cada casilla de la matriz, pero se necesitan las coordenadas de cada uno de esos puntos de la pantalla. Se obtendrán mediante el array de objetos `Centro`: se sabe el incremento de ancho y de alto, y se sabe el número de columna y de fila por lo que se puede localizar con exactitud el píxel exacto en el que habrá que pintar. Combinando estas operaciones con la clase `Path` definida en [21] se dibuja la ruta.

Pero no basta con dibujar el `Path` sin más, porque sólo se obtendría una línea quebrada cuando lo que se desea es dibujar un círculo en cada vértice. Para esta tarea se vuelve a recorrer el array que contiene los centros con las coordenadas traducidas, y se utilizan las clases `Paint` y `Canvas` definidas en [22] y [23] para dibujar los círculos. El radio que usarán se ha establecido anteriormente, pero ese radio es para los nodos inicial o final, los nodos intermedios se dibujarán utilizando la tercera parte de ese radio. El nodo inicial se pintará de color verde y el nodo final de color rojo, o en el caso de que sea una escalera, se dibujará un gráfico distinto que indicará si subir o bajar la escalera (la aplicación implementa la lógica necesaria para decidir cuál dibujar y qué tamaño debe tener ese dibujo). Se muestra un ejemplo en las siguientes figuras.

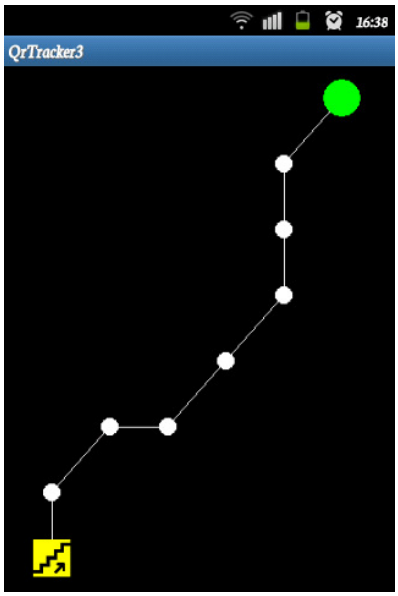


Figura 55: Subir escaleras

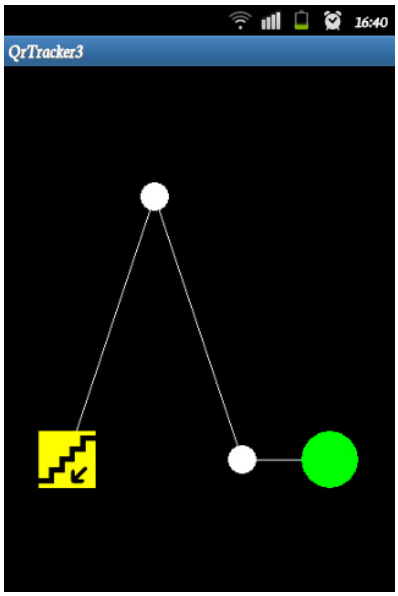


Figura 54: Bajar escaleras

Capítulo 5 – Conclusiones y trabajos futuros

A lo largo de los distintos capítulos de la presente memoria se ha presentado el diseño de una aplicación para *smartphones* Android cuyo objetivo es dotar al usuario de una solución mediante la cual pueda ubicarse dentro de un entorno *indoor*, sirviéndose de una serie de códigos QR dispuestos para tal fin.

Entre las principales bondades de esta aplicación se puede indicar:

- Es muy dinámica, porque se adapta en todo momento a la situación en que se encuentre el usuario.
- Como lo único variable que hay en la aplicación es la disposición del mapa físico, con sólo cambiar la definición de los mapas y la base de datos se puede adaptar la aplicación para cualquier entorno.
- Está desarrollada para Android, que es actualmente el sistema operativo móvil más difundido y con mayor penetración en el mercado (750 millones de dispositivos en todo el mundo, como se especifica en [24]).
- Se ha buscado un diseño muy sencillo y simple con el objetivo de maximizar la experiencia del usuario.

Hay otra característica de la aplicación, que se trata por separado de las demás debido a la relevancia tan importante que tiene en este caso. Esta característica no es otra que la escalabilidad: el diseño realizado brinda la posibilidad de desarrollar la solución por fases. Esta aproximación ha permitido una implementación mucho más asequible puesto que hace posible poder implementar una parte del entorno y hacer las pruebas pertinentes de depuración y funcionamiento antes de emprender la siguiente, garantizándose así que todo trabajará correctamente de forma conjunta cuando se haya terminado el desarrollo.

No obstante también se debe mencionar una limitación de esta aplicación: a la hora de mostrar la ruta en la pantalla del teléfono móvil presupone que el usuario está mirando a una dirección concreta ya preestablecida (en el presente caso se trata del acceso a la biblioteca del Cuartel de Antigones de la Universidad Politécnica de Cartagena). Esta limitación es una de las posibles líneas de futuro de la aplicación, de las cuales se hablará más adelante.

En cuanto a las conclusiones personales, ha resultado muy gratificante desarrollar esta aplicación debido al gran interés que tienen actualmente las aplicaciones móviles. Este desarrollo, al ser el más grande que he abordado hasta la flecha, me ha permitido adquirir unas habilidades para el diseño y programación no utilizadas anteriormente y que han resultado claves para poder llevar el proyecto a buen puerto, así como la capacidad para buscar información y para la investigación. También me ha permitido profundizar conocimientos sobre la algoritmia, el diseño de casos de uso y el estudio de las ventajas e inconvenientes de las opciones disponibles para desarrollar una solución. Ha sido muy alentador haber sido capaz de salvar todos los problemas y obstáculos que han ido surgiendo por el camino. Por último, aunque no menos importante, se debe marcar que ha sido muy entretenido poder ejercer la afición y el gusto por la programación y al mismo tiempo desarrollar un proyecto fin de carrera.

Aunque la aplicación presentada sea totalmente funcional, esto no impide que pueda ser mejorada con más posibilidades. La primera ampliación que se le debería implementar es, por supuesto, eliminar la limitación que obliga al usuario a mirar a un lugar concreto porque la ruta se genera desde ese punto de vista; una posible solución es que el gráfico gire en la pantalla del teléfono móvil en función del movimiento del usuario. Aunque esta es la mejora más prioritaria no es la única; otras posibles mejoras serían las que se mencionan a continuación:

- Al hilo de la limitación antes mencionada, utilizar los acelerómetros del teléfono (si los tiene) para que se dé cuenta de los movimientos del usuario para girar el gráfico en pantalla de la manera conveniente.
- Dibujar las rutas sobre un mapa de la ubicación: esta mejora consiste en dibujar en la pantalla del teléfono el mapa real de la ubicación en la que se encuentre el usuario, y sobre él dibujar la ruta.
- Utilizar indicaciones de voz.
- Conectar periódicamente con el servidor para consultar si hay cambios y actualizar la base de datos. Así se evitará la necesidad de la aplicación de conectarse inmediatamente nada más ejecutarla.
- Reproducir un sonido cuando el usuario vaya siguiendo su ruta y se aleje de una ruta preestablecida como la ideal.

Bibliografía

[1] – 75% de penetración en el mercado en el cuarto aniversario de Android - https://www.idc.com/getdoc.jsp?containerId=prUS23771812

[2] – Presentación de Google TV - http://www.google.com/tv/features.html

[3] – Ordenador portátil con sistema operativo Android - http://h41104.www4.hp.com/promociones/compaq_airlife/home.html

[4] – Cámara fotográfica con Android - http://www.samsung.com/es/consumer/mobile-phone/galaxy-camera

[5] – Presentación de Ekahau - http://www.ekahau.com/products/real-time-location-system/overview.html

[6] – Códigos QR - http://en.wikipedia.org/wiki/QR_code

[7] – Diseño de aplicaciones Android - http://developer.android.com/design/get-started/principles.html

[8] – Presentación de SQLite - http://www.sqlite.org/about.html

[9] – Novedades de SQLite - http://www.sqlite.org/different.html

[10] - El gran libro de Android. Jesús Tomás Gironés. Editorial Marcombo, 2011

[11] – Patrón MVC en SmallTalk - http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html

[12] – Patrón MVC - http://es.wikipedia.org/wiki/Modelo_Vista_Controlador

[13] – Documentación oficial de Java - http://docs.oracle.com/javase/1.4.2/docs/api/

[14] – Introducción al algoritmo A* - http://www.policyalmanac.org/games/aStarTutorial.htm

[15] – Estudio detallado sobre A* - http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

[16] – Acceso a base de datos SQLite mediante controlador intermedio - http://www.vogella.com/articles/AndroidSQLite/article.html

[17] – Clase BufferedReader- http://ulibgcj.sourceforge.net/javadoc/java/io/BufferedReader.html

[18] – Clase Collections - http://docs.oracle.com/javase/1.4.2/docs/api/java/util/Collections.html

[19] – Almacenamiento en bases de datos en Android - http://developer.android.com/guide/topics/data/data-storage.html#db

[20] – Aplicación QR Droid - http://qrdroid.com/android-developers.php

[21] – Clase Path - http://developer.android.com/reference/android/graphics/Path.html

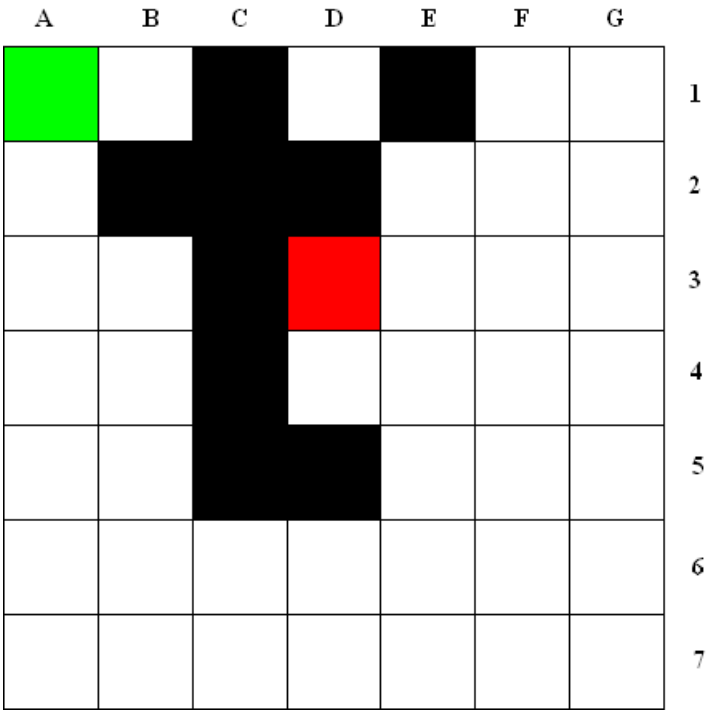
[22] – Clase Canvas - http://developer.android.com/reference/android/graphics/Canvas.html

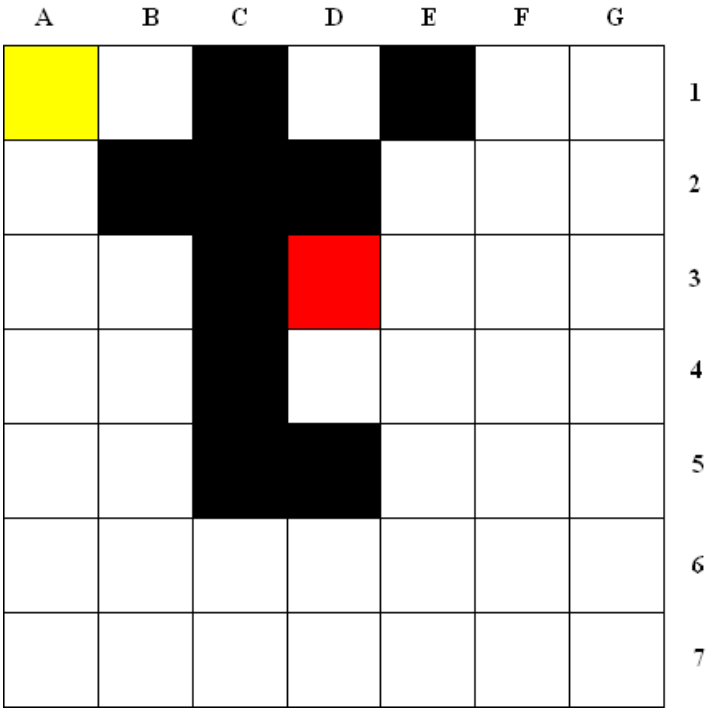
[23] – Clase Paint - http://developer.android.com/reference/android/graphics/Paint.html

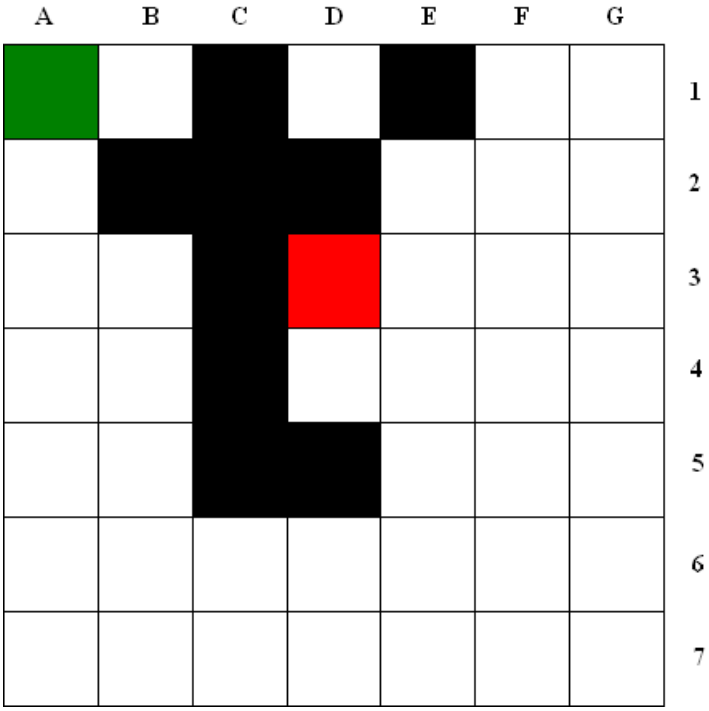
[24] – 750 millones de dispositivos Android en el mercado - http://googleblog.blogspot.com.es/2013/03/update-from-ceo.html

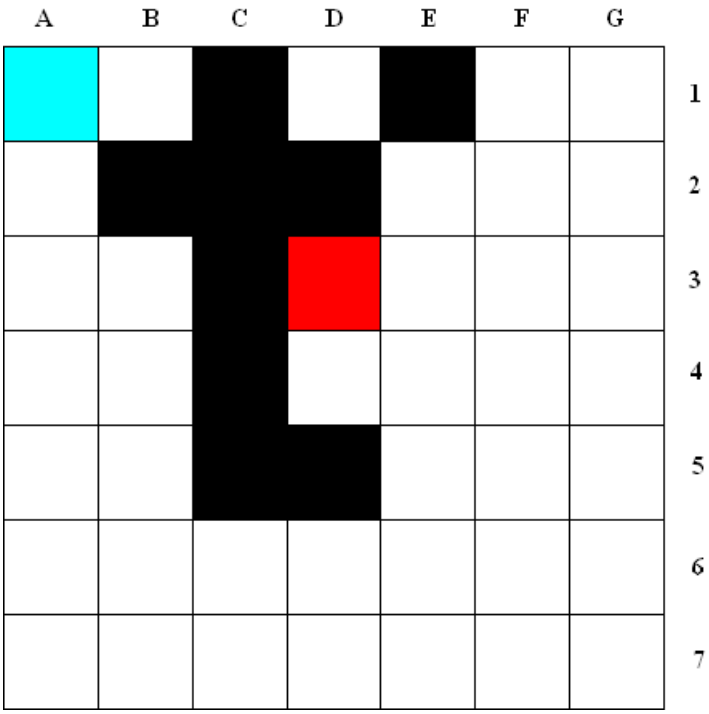
Anexo I – Ejemplo paso a paso del algoritmo A*

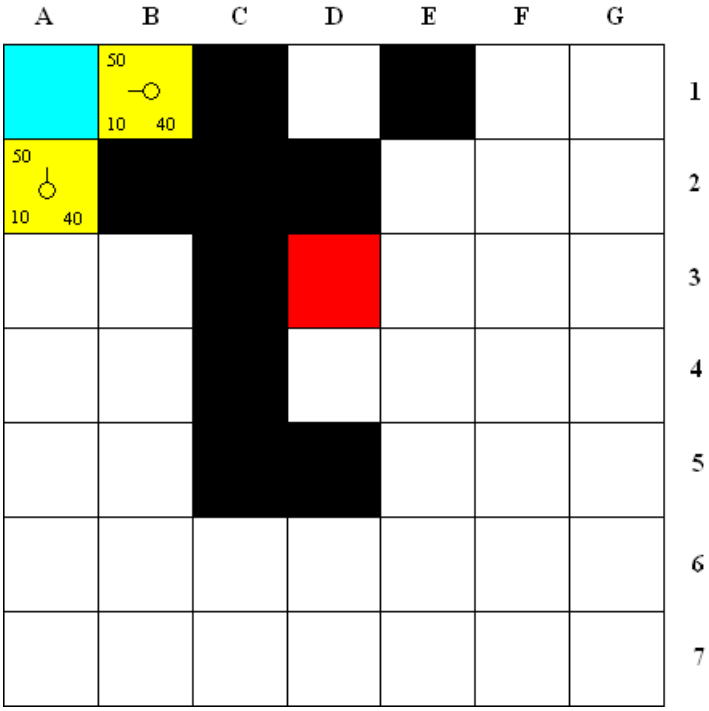
A continuación se muestra un ejemplo detallado paso a paso del funcionamiento del algoritmo de *pathfinding*. El nodo inicial se muestra de color verde, el final de color rojo, los nodos en lista abierta se marcan en amarillo, los nodos en lista cerrada en verde oscuro y la ruta se marca en naranja.

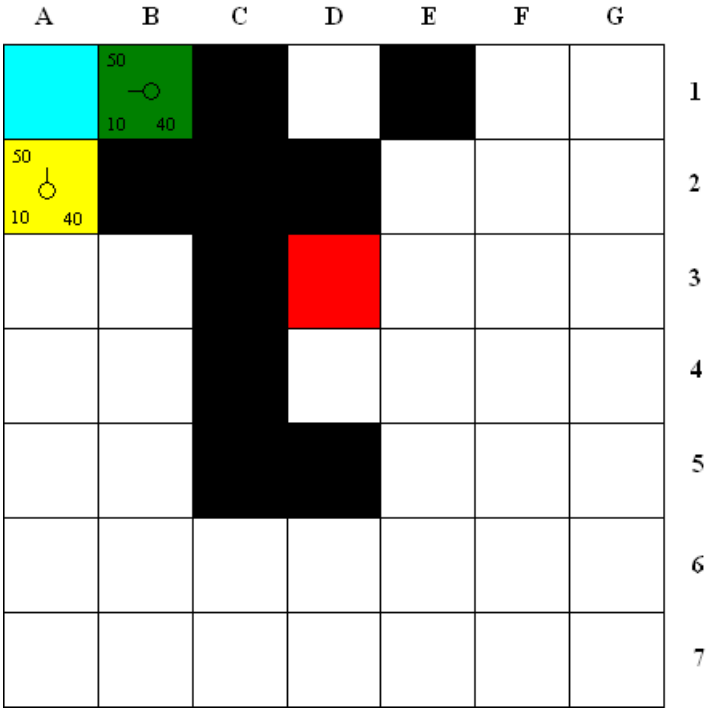


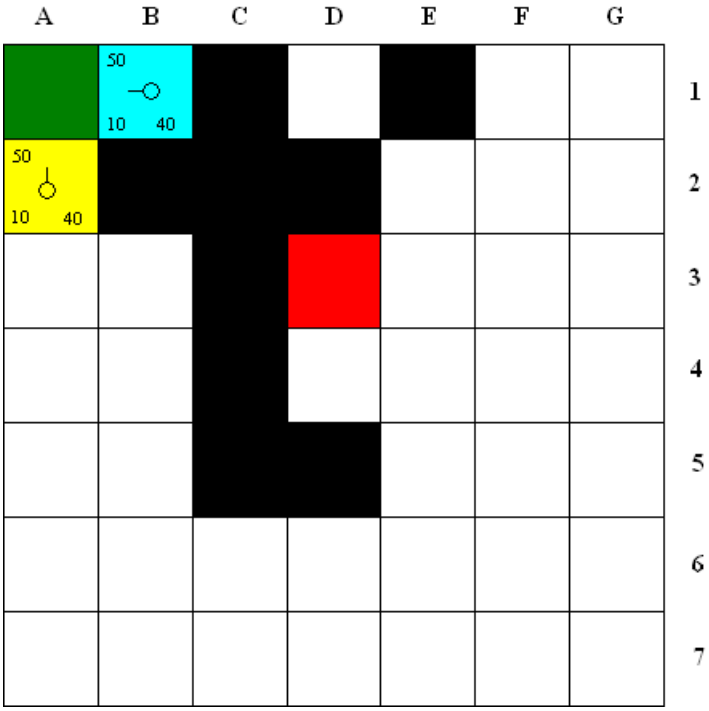


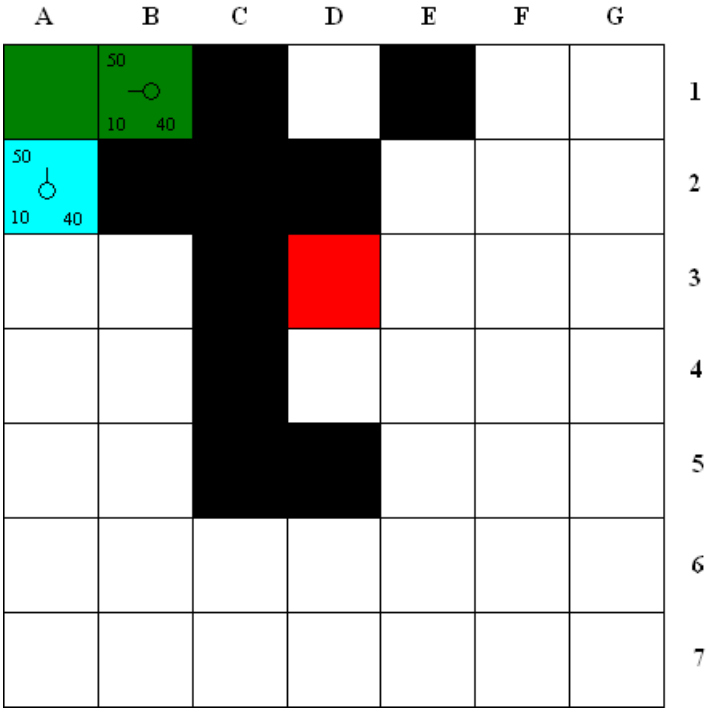

A
B
C
D
E
F
G
1
2
3
4
5
6
7

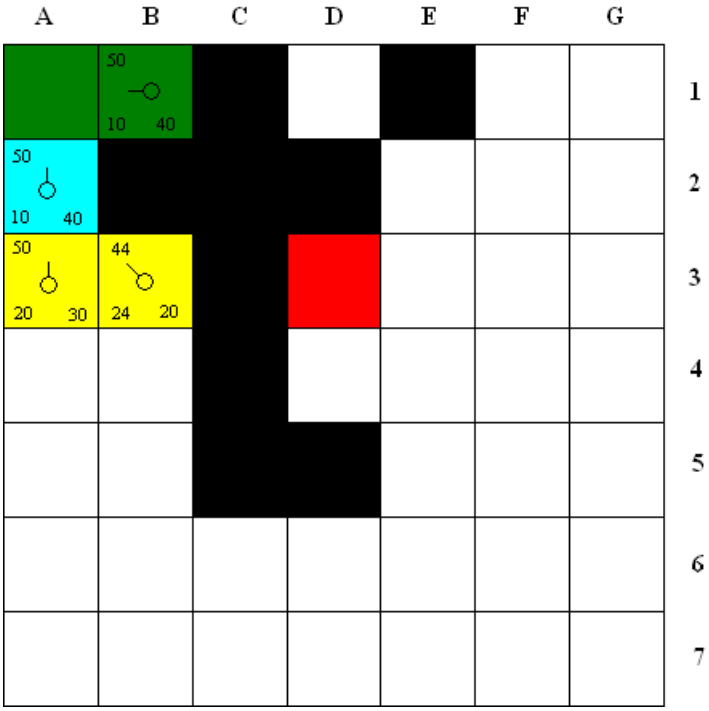

A
B
C
D
E
F
G
1
2
3
4
5
6
7

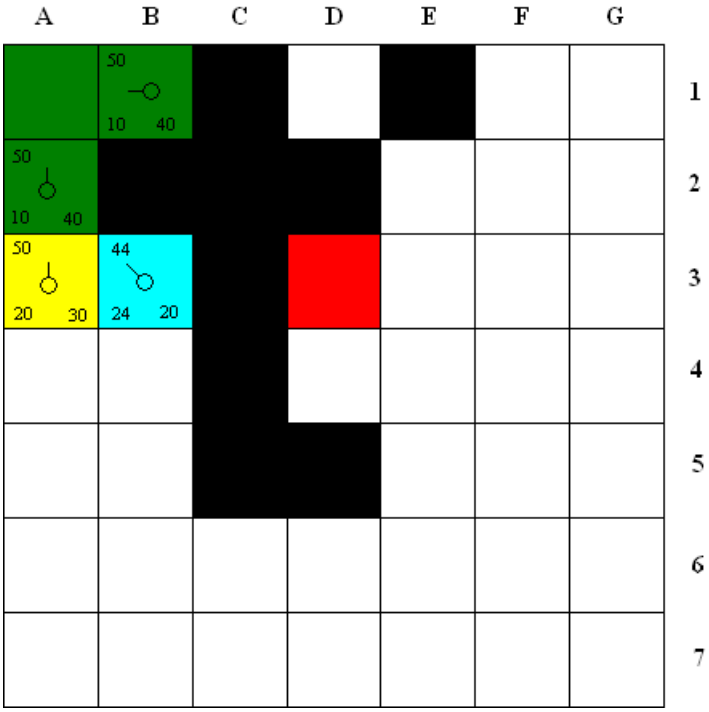

A
B
C
D
E
F
G
50
10
40
50
10
40
1
2
3
4
5
6
7

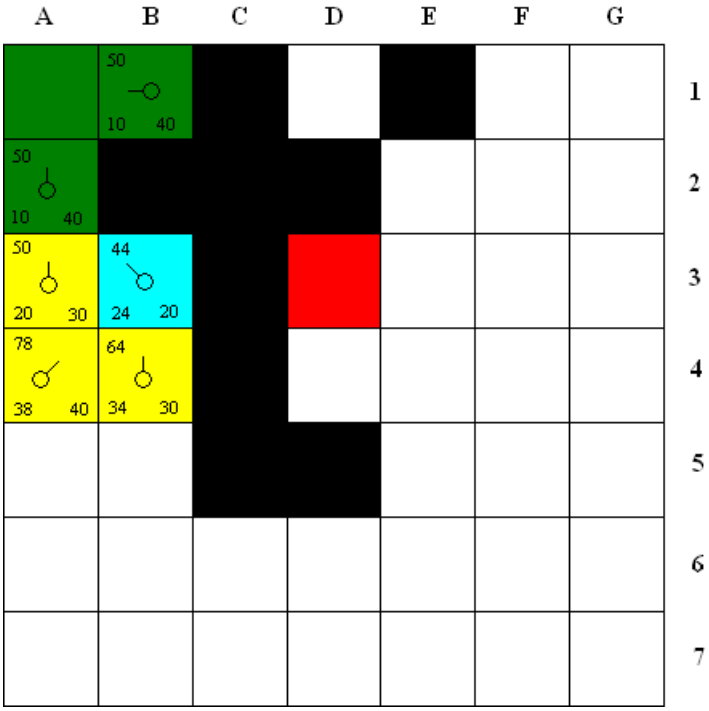

A
B
C
D
E
F
G
50
10
40
50
10
40
1
2
3
4
5
6
7

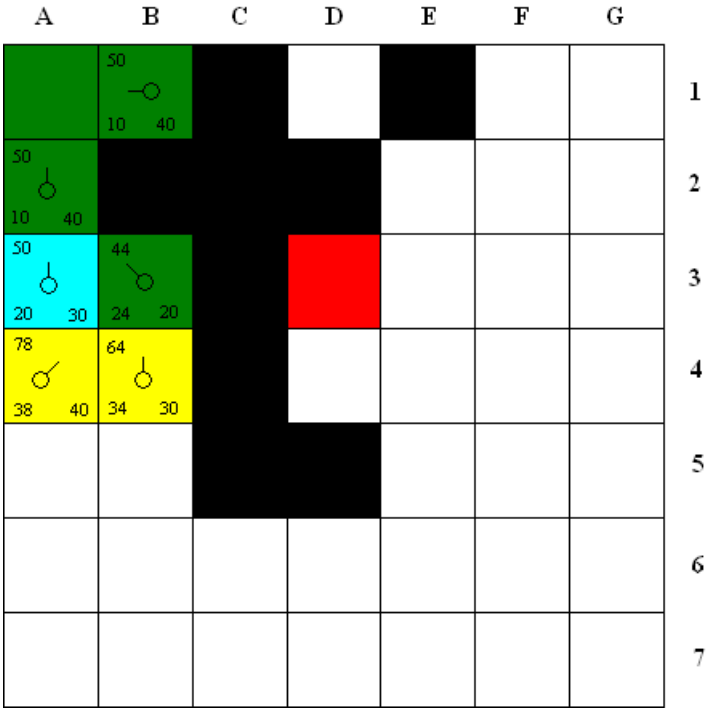

A
B
C
D
E
F
G
50
10
40
50
10
40
1
2
3
4
5
6
7

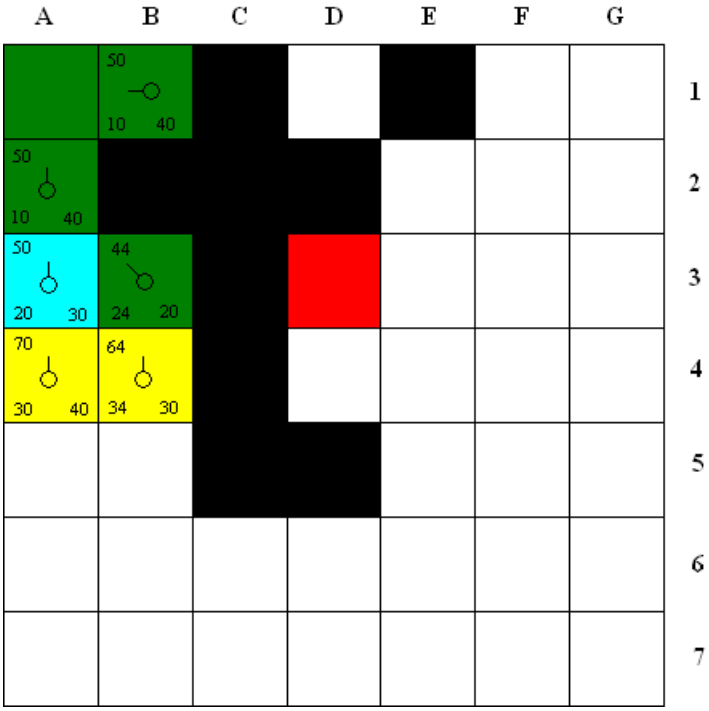

A
B
C
D
E
F
G
50
10
40
50
10
40
1
2
3
4
5
6
7

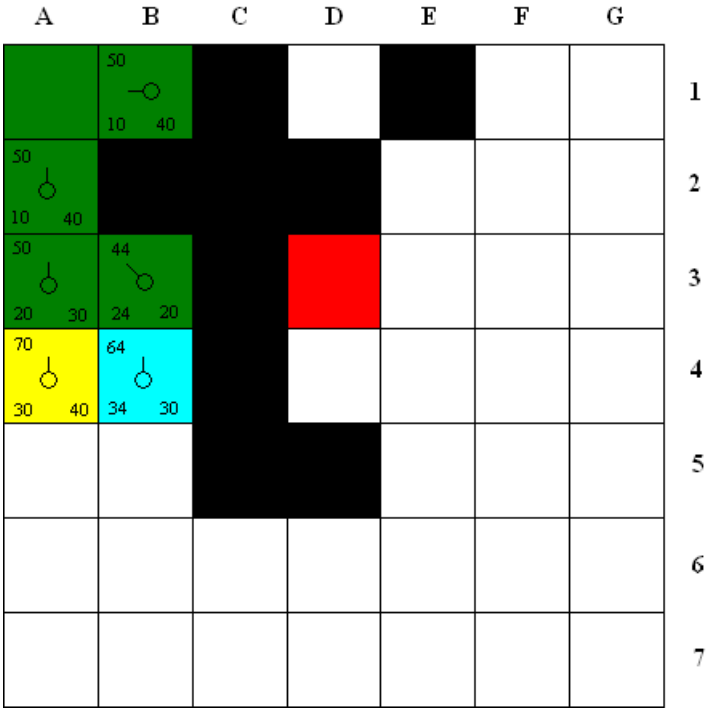

A B C D E F G
50
10 40
50
10 40
50
20 30
44
24 20
1 2 3 4 5 6 7

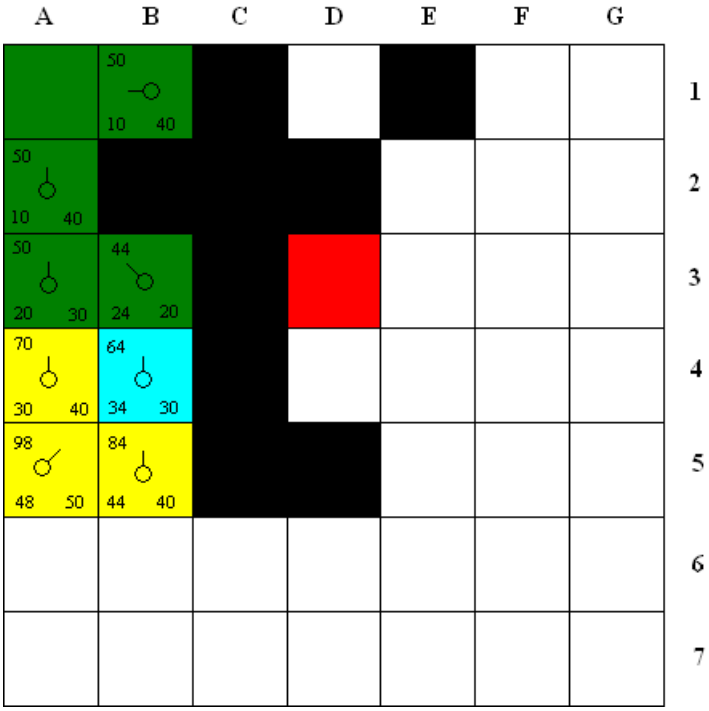

A B C D E F G
50
10 40
50
10 40
50
20 30
44
24 20
1 2 3 4 5 6 7

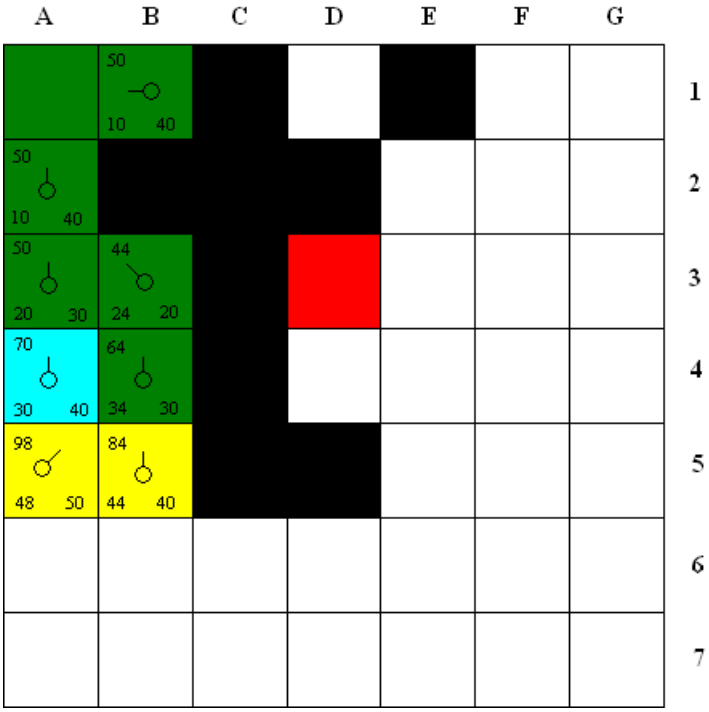

A B C D E F G
50
10 40
50
10 40
50
20 30
44
24 20
78
38 40
64
34 30
1 2 3 4 5 6 7

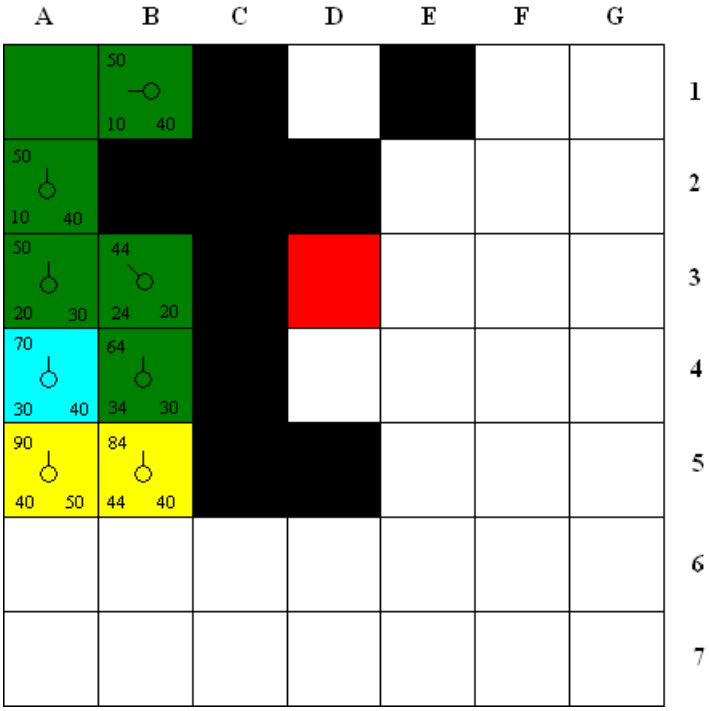

A B C D E F G
50
10 40
50
10 40
50
20 30
44
24 20
78
38 40
64
34 30
1 2 3 4 5 6 7

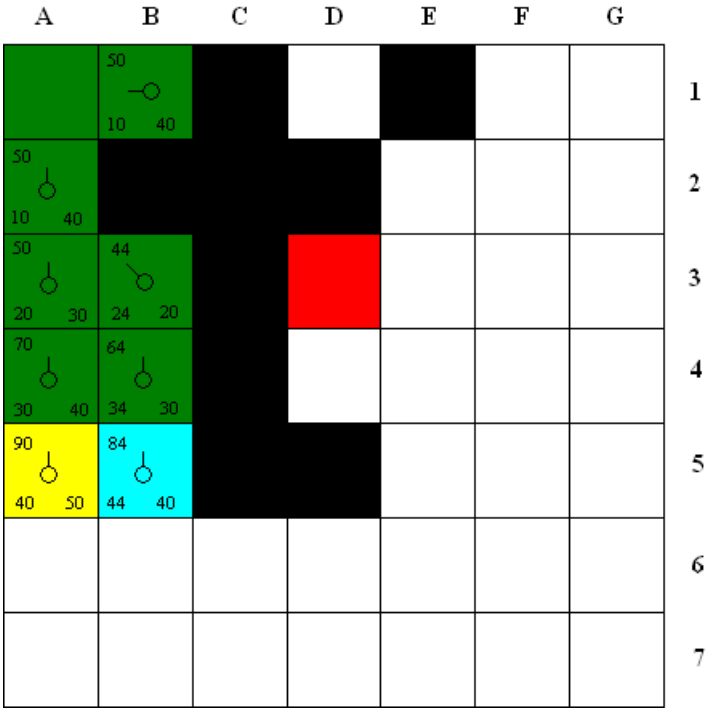

A B C D E F G
50
10 40
50
10 40
50
20 30
44
24 20
70
30 40
64
34 30
1 2 3 4 5 6 7

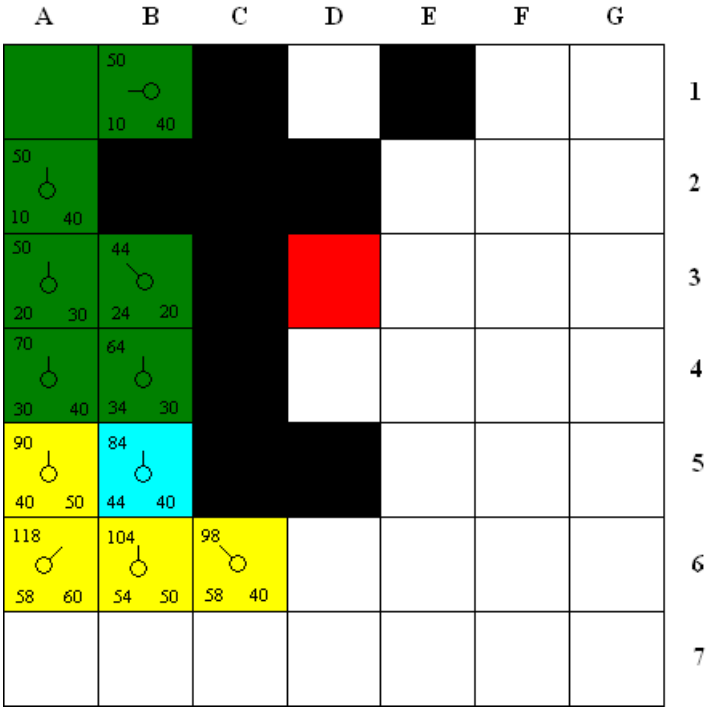

A B C D E F G
50
10 40
50
10 40
50
20 30
44
24 20
70
30 40
64
34 30
1 2 3 4 5 6 7

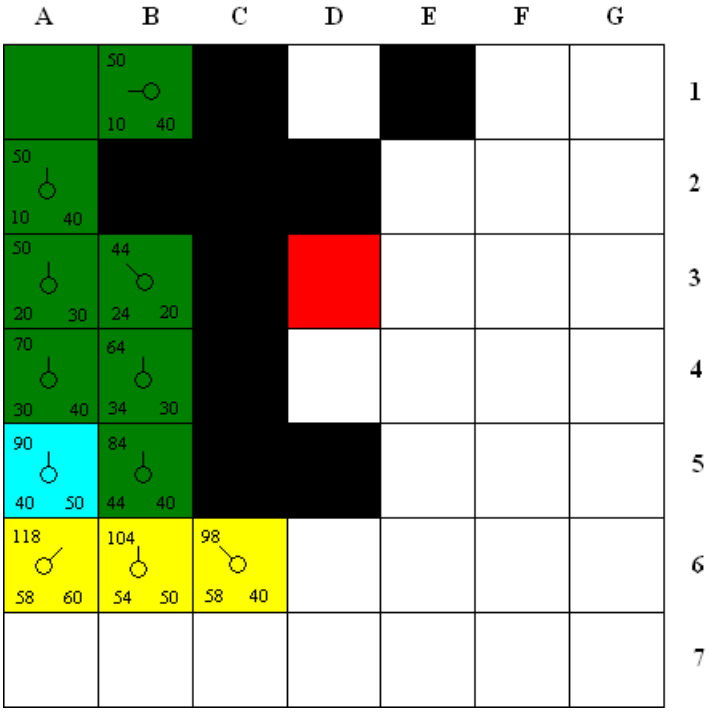

A B C D E F G
50
10 40
50
10 40
50
20 30
44
24 20
70
30 40
64
34 30
98
48 50
84
44 40
1
2
3
4
5
6
7

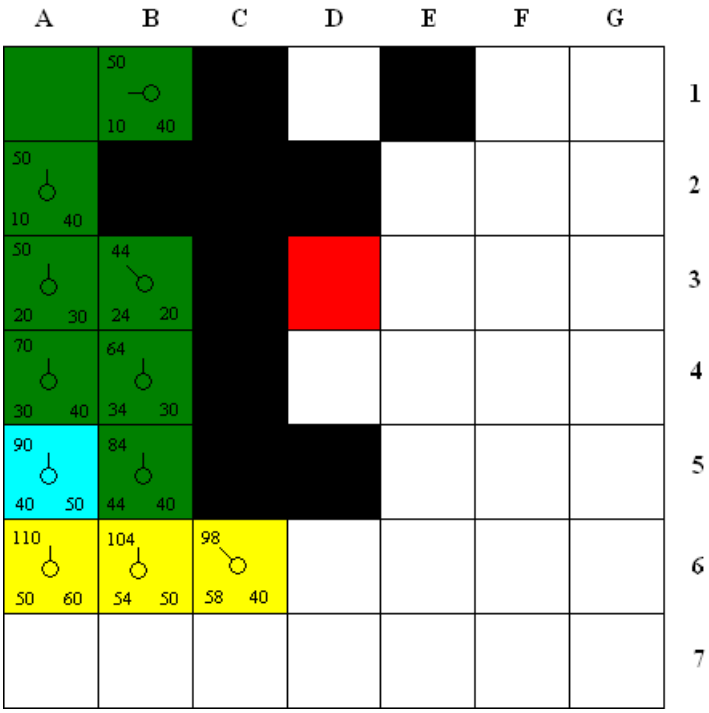

A B C D E F G
50
10 40
50
10 40
50
20 30
44
24 20
70
30 40
64
34 30
98
48 50
84
44 40
1
2
3
4
5
6
7

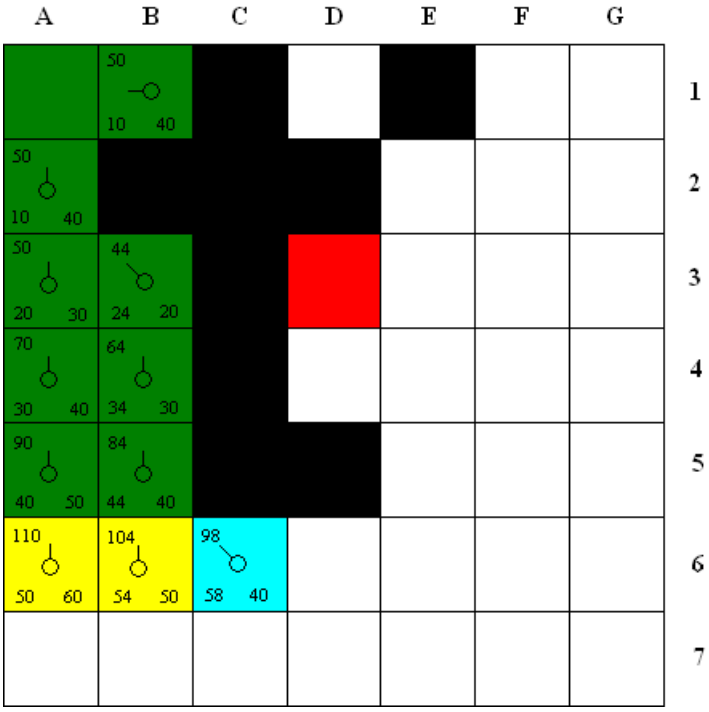

A B C D E F G
50
10 40
50
10 40
50
20 30
44
24 20
70
30 40
64
34 30
90
40 50
84
44 40
1 2 3 4 5 6 7

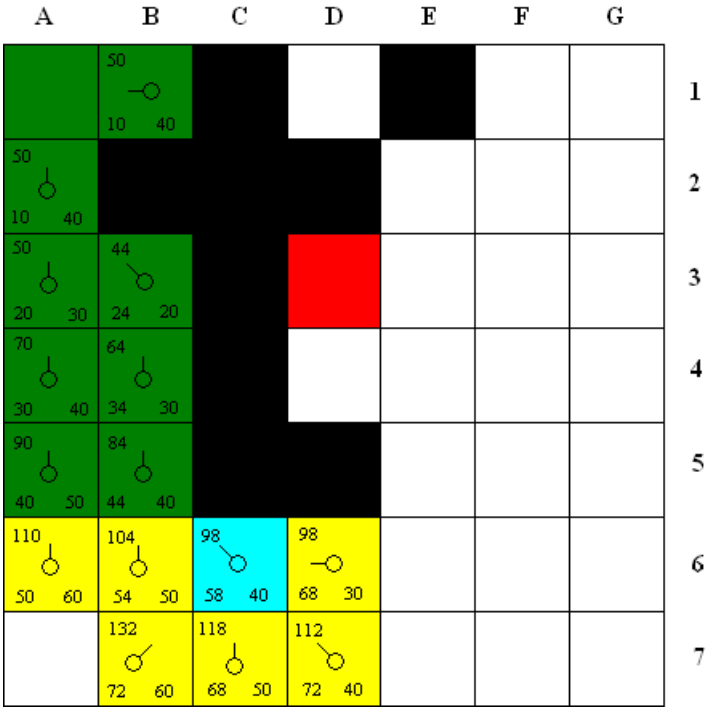

A B C D E F G
50
10 40
50
10 40
50
20 30
44
24 20
70
30 40
64
34 30
90
40 50
84
44 40
1 2 3 4 5 6 7

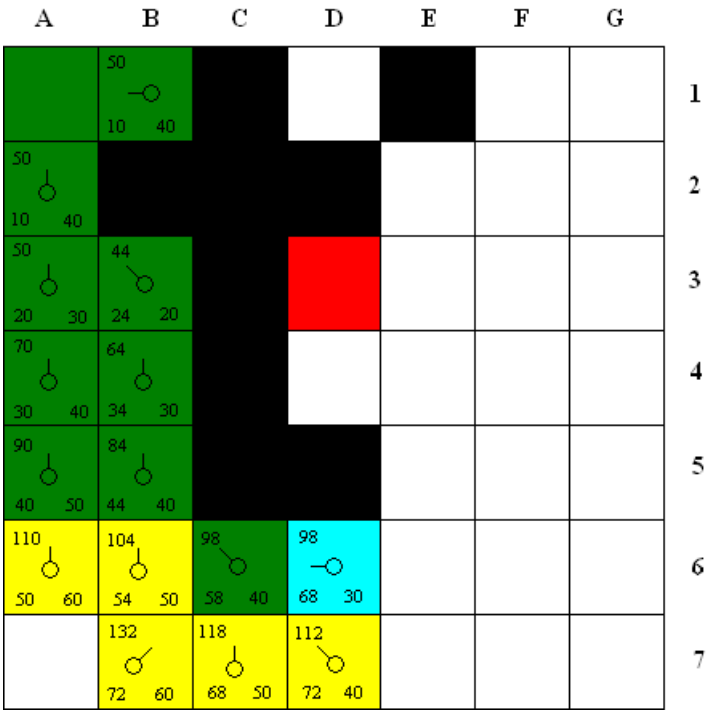

A B C D E F G
50 10 40
50 10 40
50 20 30
44 24 20
70 30 40
64 34 30
90 40 50
84 44 40
118 58 60
104 54 50
98 58 40
1 2 3 4 5 6 7

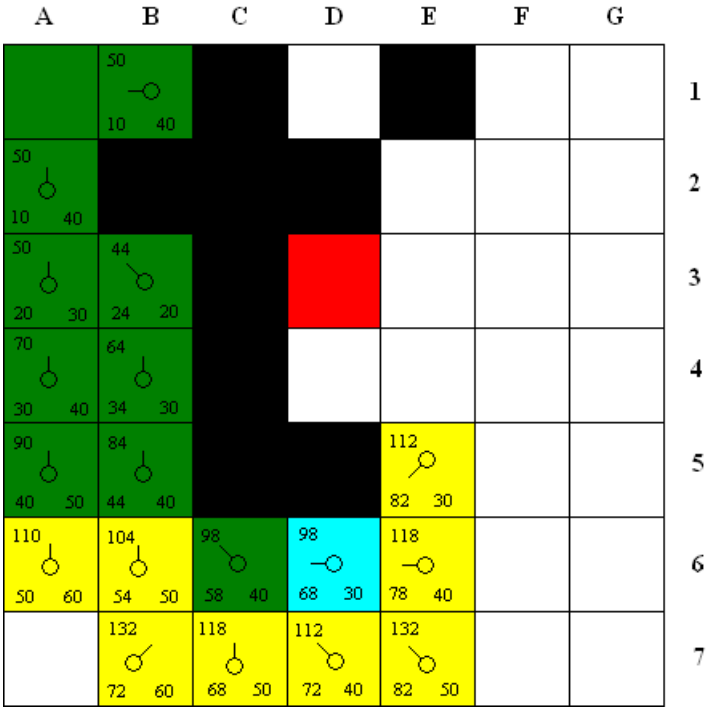

A B C D E F G
50 10 40
50 10 40
50 20 30
44 24 20
70 30 40
64 34 30
90 40 50
84 44 40
118 58 60
104 54 50
98 58 40
1 2 3 4 5 6 7

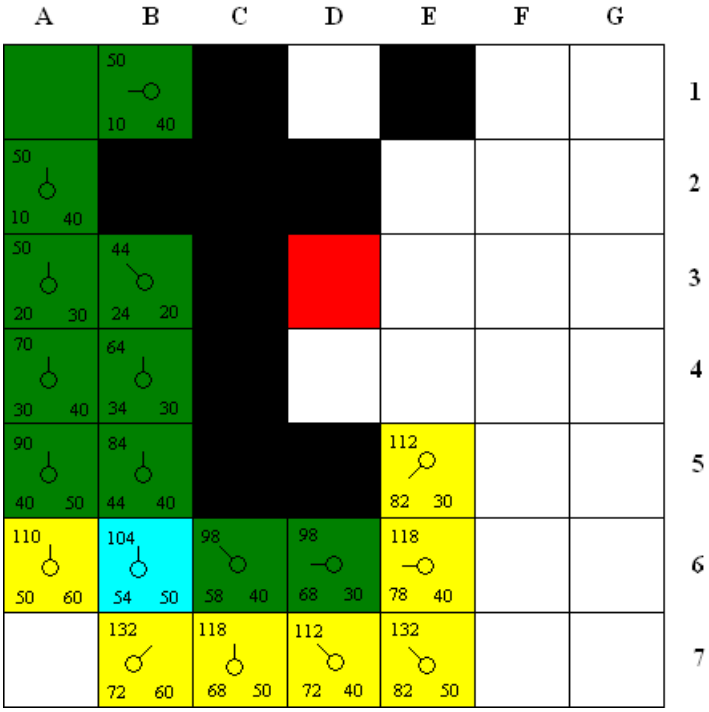

A B C D E F G
1
2
3
4
5
6
7
50 10 40
50 10 40
50 20 30
44 24 20
70 30 40
64 34 30
90 40 50
84 44 40
110 50 60
104 54 50
98 58 40

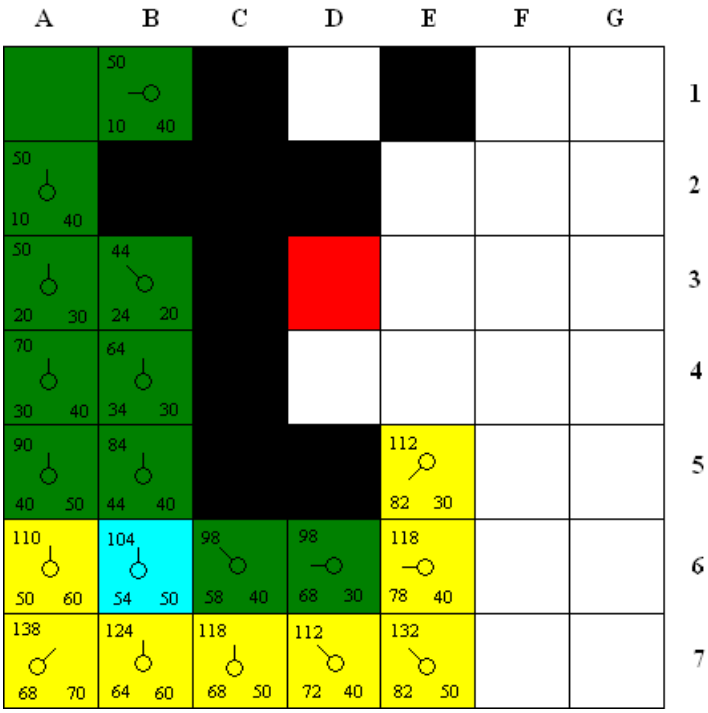

A B C D E F G
1
2
3
4
5
6
7
50 10 40
50 10 40
50 20 30
44 24 20
70 30 40
64 34 30
90 40 50
84 44 40
110 50 60
104 54 50
98 58 40

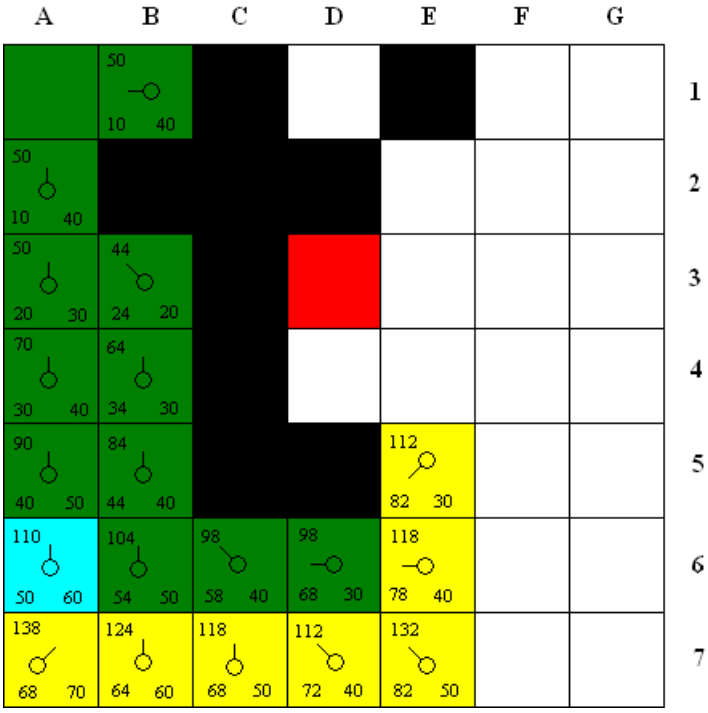

A B C D E F G
50 10 40
50 10 40
50 20 30
44 24 20
70 30 40
64 34 30
90 40 50
84 44 40
110 50 60
104 54 50
98 58 40
98 68 30
132 72 60
118 68 50
112 72 40
1 2 3 4 5 6 7

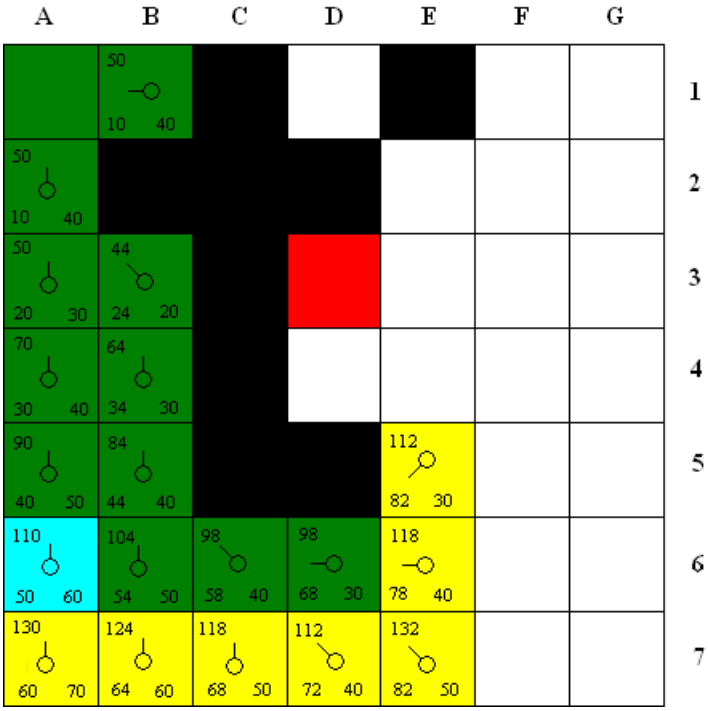

A B C D E F G
50 10 40
50 10 40
50 20 30
44 24 20
70 30 40
64 34 30
90 40 50
84 44 40
110 50 60
104 54 50
98 58 40
98 68 30
132 72 60
118 68 50
112 72 40
1 2 3 4 5 6 7

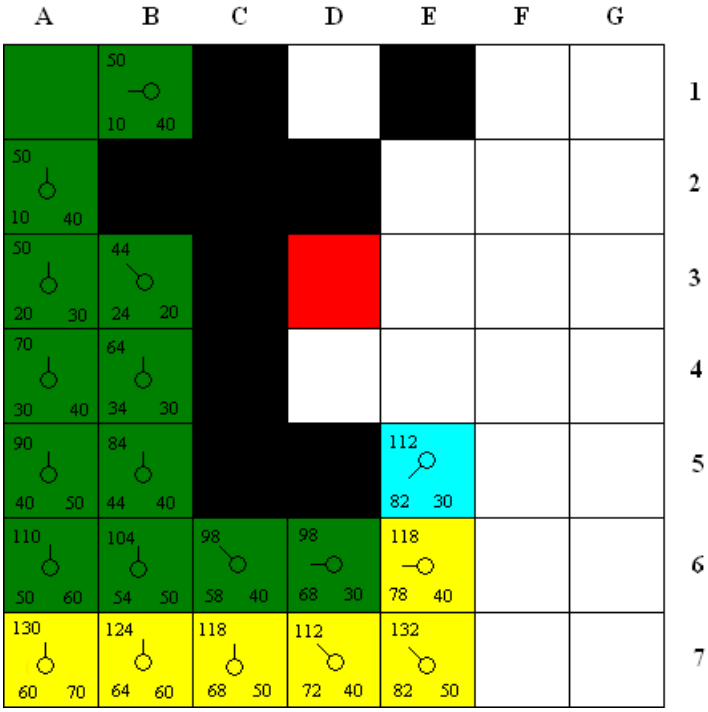

A B C D E F G
1
50 10 40
2
50 10 40
3
50 20 30
44 24 20
4
70 30 40
64 34 30
5
90 40 50
84 44 40
112 82 30
6
110 50 60
104 54 50
98 58 40
98 68 30
118 78 40
7
132 72 60
118 68 50
112 72 40
132 82 50


A B C D E F G
1
50 10 40
2
50 10 40
3
50 20 30
44 24 20
4
70 30 40
64 34 30
5
90 40 50
84 44 40
112 82 30
6
110 50 60
104 54 50
98 58 40
98 68 30
118 78 40
7
132 72 60
118 68 50
112 72 40
132 82 50


A B C D E F G
1
2
3
4
5
6
7
50 10 40
50 10 40
50 20 30
44 24 20
70 30 40
64 34 30
90 40 50
84 44 40
112 82 30
110 50 60
104 54 50
98 58 40
98 68 30
118 78 40
138 68 70
124 64 60
118 68 50
112 72 40
132 82 50

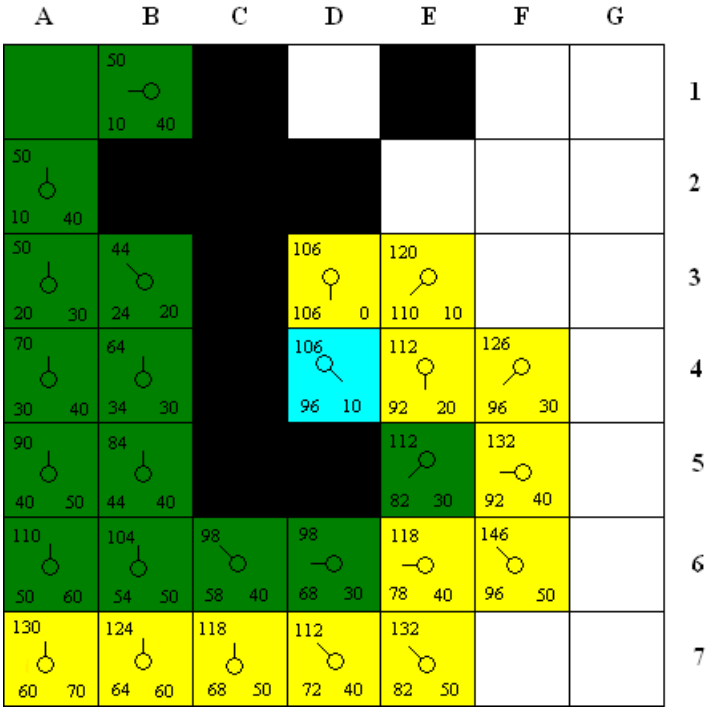

A B C D E F G
1
2
3
4
5
6
7
50 10 40
50 10 40
50 20 30
44 24 20
70 30 40
64 34 30
90 40 50
84 44 40
112 82 30
110 50 60
104 54 50
98 58 40
98 68 30
118 78 40
138 68 70
124 64 60
118 68 50
112 72 40
132 82 50

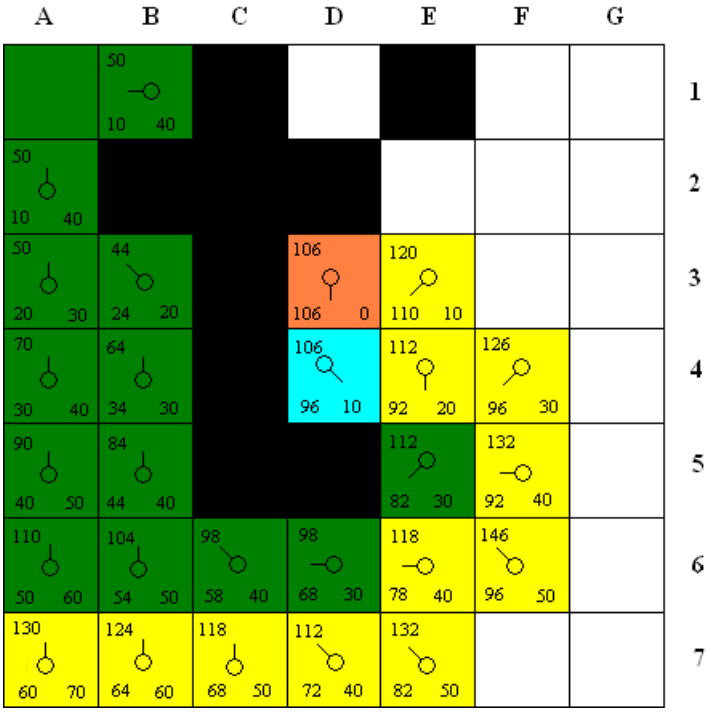

A B C D E F G
1
2
3
4
5
6
7
50 10 40
50 10 40
50 20 30
44 24 20
70 30 40
64 34 30
90 40 50
84 44 40
112 82 30
110 50 60
104 54 50
98 58 40
98 68 30
118 78 40
130 60 70
124 64 60
118 68 50
112 72 40
132 82 50

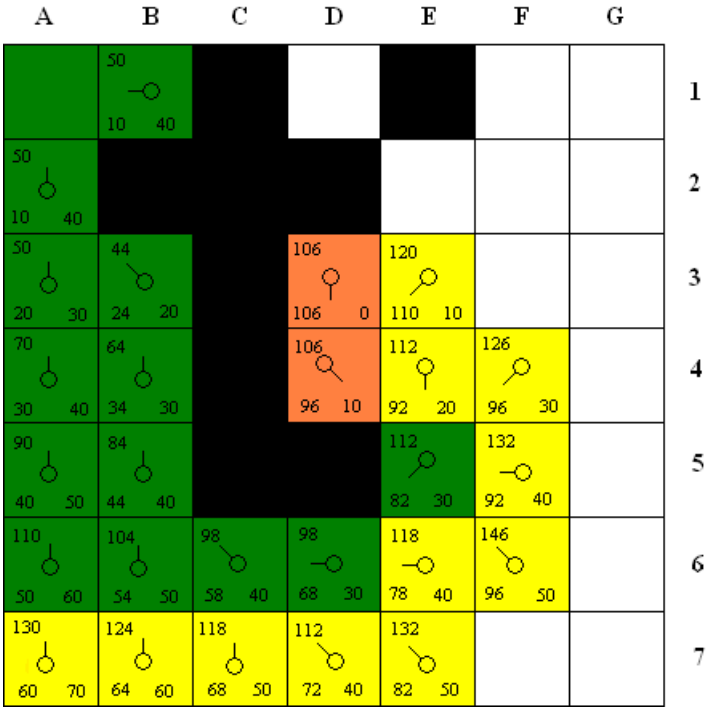

A B C D E F G
1
2
3
4
5
6
7
50 10 40
50 10 40
50 20 30
44 24 20
70 30 40
64 34 30
90 40 50
84 44 40
112 82 30
110 50 60
104 54 50
98 58 40
98 68 30
118 78 40
130 60 70
124 64 60
118 68 50
112 72 40
132 82 50

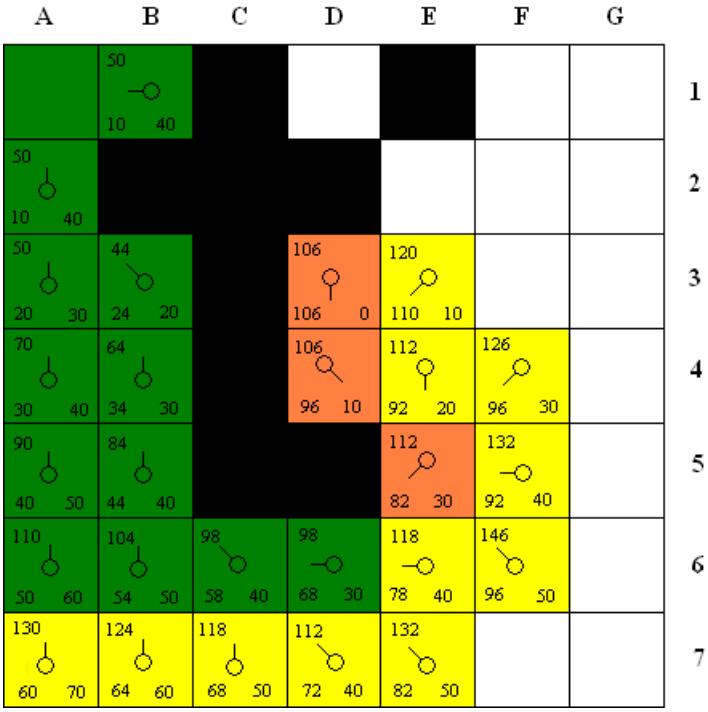
A	B	C	D	E	F	G	
	50 10 40						1
50 10 40							2
50 20 30	44 24 20						3
70 30 40	64 34 30		106 96 10	112 92 20	126 96 30		4
90 40 50	84 44 40			112 82 30	132 92 40		5
110 50 60	104 54 50	98 58 40	98 68 30	118 78 40	146 96 50		6
130 60 70	124 64 60	118 68 50	112 72 40	132 82 50			7

A	B	C	D	E	F	G	
	50 10 40						1
50 10 40							2
50 20 30	44 24 20						3
70 30 40	64 34 30		106 96 10	112 92 20	126 96 30		4
90 40 50	84 44 40			112 82 30	132 92 40		5
110 50 60	104 54 50	98 58 40	98 68 30	118 78 40	146 96 50		6
130 60 70	124 64 60	118 68 50	112 72 40	132 82 50			7


A B C D E F G
1
50 10 40
2
50 10 40
3
50 20 30
44 24 20
106 106 0
120 110 10
4
70 30 40
64 34 30
106 96 10
112 92 20
126 96 30
5
90 40 50
84 44 40
112 82 30
132 92 40
6
110 50 60
104 54 50
98 58 40
98 68 30
118 78 40
146 96 50
7
130 60 70
124 64 60
118 68 50
112 72 40
132 82 50

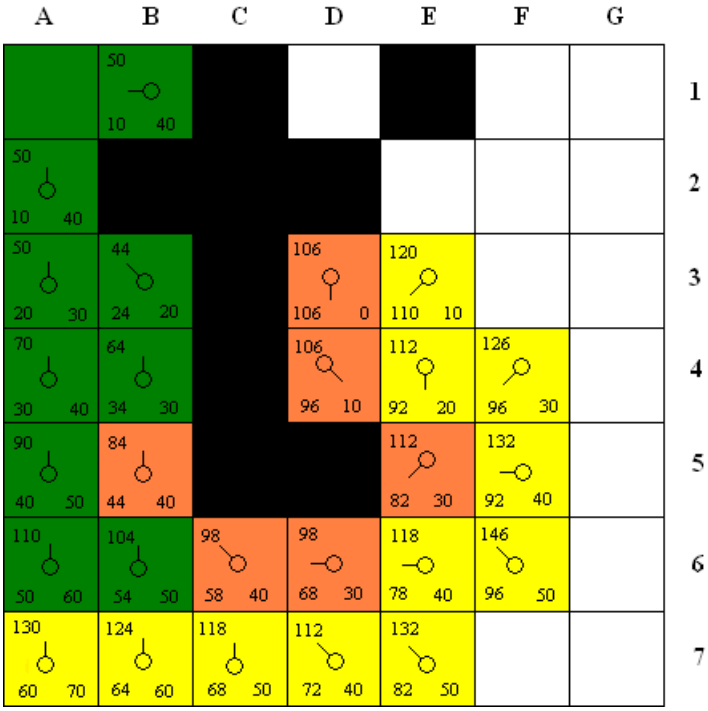

A B C D E F G
1
50 10 40
2
50 10 40
3
50 20 30
44 24 20
106 106 0
120 110 10
4
70 30 40
64 34 30
106 96 10
112 92 20
126 96 30
5
90 40 50
84 44 40
112 82 30
132 92 40
6
110 50 60
104 54 50
98 58 40
98 68 30
118 78 40
146 96 50
7
130 60 70
124 64 60
118 68 50
112 72 40
132 82 50

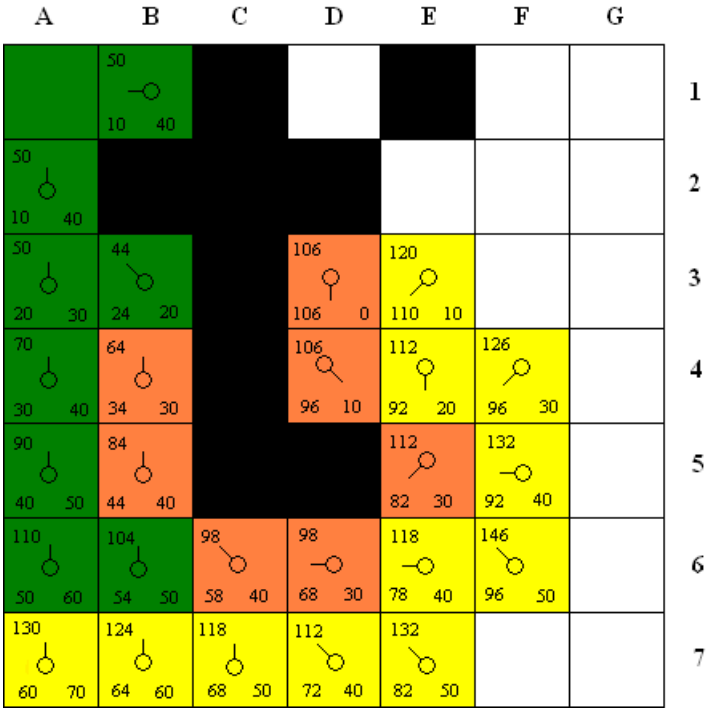

A B C D E F G
1
50 10 40
2
50 10 40
3
50 20 30
44 24 20
106 106 0
120 110 10
4
70 30 40
64 34 30
106 96 10
112 92 20
126 96 30
5
90 40 50
84 44 40
112 82 30
132 92 40
6
110 50 60
104 54 50
98 58 40
98 68 30
118 78 40
146 96 50
7
130 60 70
124 64 60
118 68 50
112 72 40
132 82 50

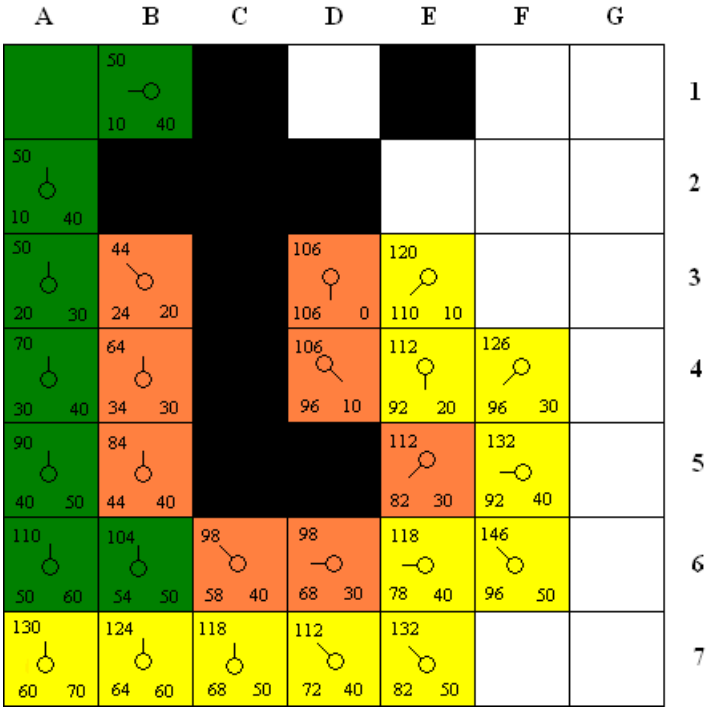

A B C D E F G
1
50 10 40
2
50 10 40
3
50 20 30
44 24 20
106 106 0
120 110 10
4
70 30 40
64 34 30
106 96 10
112 92 20
126 96 30
5
90 40 50
84 44 40
112 82 30
132 92 40
6
110 50 60
104 54 50
98 58 40
98 68 30
118 78 40
146 96 50
7
130 60 70
124 64 60
118 68 50
112 72 40
132 82 50

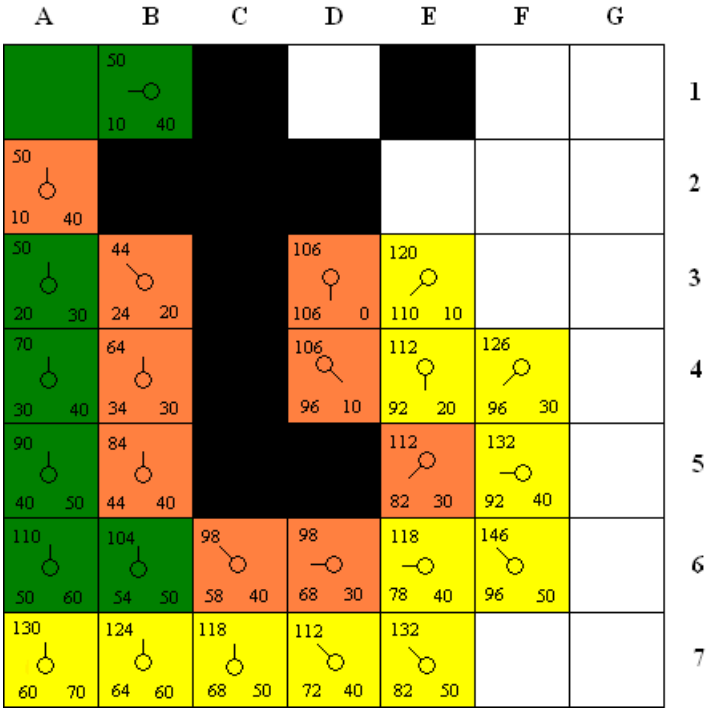
A	B	C	D	E	F	G	
	50 10 40						1
50 10 40							2
50 20 30	44 24 20		106 106 0	120 110 10			3
70 30 40	64 34 30		106 96 10	112 92 20	126 96 30		4
90 40 50	84 44 40			112 82 30	132 92 40		5
110 50 60	104 54 50	98 58 40	98 68 30	118 78 40	146 96 50		6
130 60 70	124 64 60	118 68 50	112 72 40	132 82 50			7

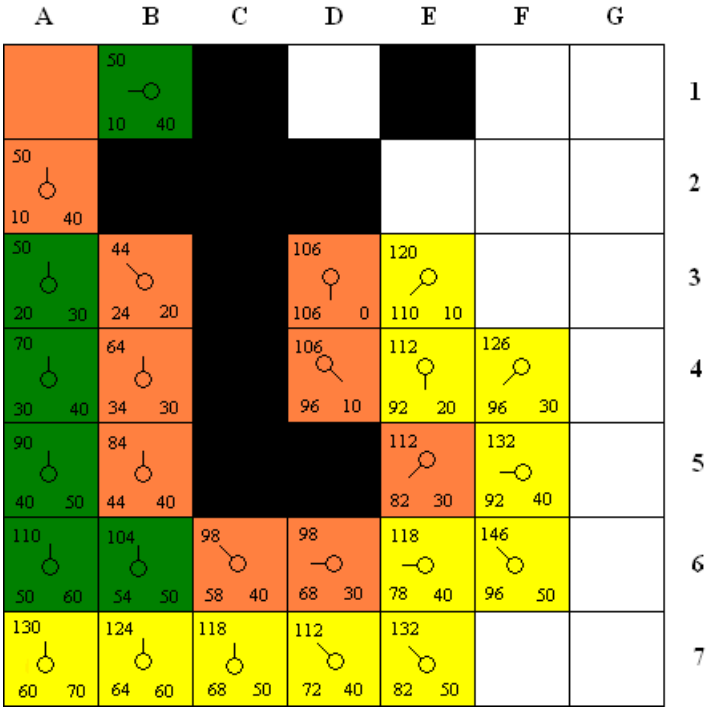
A	B	C	D	E	F	G	
	50 10 40						1
50 10 40							2
50 20 30	44 24 20		106 106 0	120 110 10			3
70 30 40	64 34 30		106 96 10	112 92 20	126 96 30		4
90 40 50	84 44 40			112 82 30	132 92 40		5
110 50 60	104 54 50	98 58 40	98 68 30	118 78 40	146 96 50		6
130 60 70	124 64 60	118 68 50	112 72 40	132 82 50			7


A B C D E F G
1
2
3
4
5
6
7


A B C D E F G
1
2
3
4
5
6
7


A B C D E F G
1
2
3
4
5
6
7


A B C D E F G
1
2
3
4
5
6
7


A B C D E F G
1 2 3 4 5 6 7
50 10 40
50 10 40
50 20 30
44 24 20
106 106 0
120 110 10
70 30 40
64 34 30
106 96 10
112 92 20
126 96 30
90 40 50
84 44 40
112 82 30
132 92 40
110 50 60
104 54 50
98 58 40
98 68 30
118 78 40
146 96 50
130 60 70
124 64 60
118 68 50
112 72 40
132 82 50

Anexo II – Código fuente de la aplicación

```
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileReader;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;

/*
 * Esto es el servidor de QRTracker.
 * El programa del móvil se conectará a este servidor
 */

public class Server {

	public static final String CREAR_BD = "CLIENTE_CREAR";
	public static final String ACTUALIZAR_BD = "CLIENTE_ACTUALIZAR";
	public static final String NO_ACTUAR = "CLIENTE_NO_ACTUAR";

	public static void main(String args[]) {

		try {

			ServerSocket socket = null;
			BufferedReader br = null;
			FileReader fr = null;
			File fichero = null;
			Socket cliente = null;
			OutputStream salida = null;
			DataOutputStream salidaDatos = null;
			InputStream entrada = null;
			DataInputStream entradaDatos = null;
			String linea = "";
			int numeroLineas = 0;

			long fechaServidorCreate = 0;
			long fechaServidorInsert = 0;

			//Abro socket
			//System.out.println("Abro socket");
			socket = new ServerSocket(4000);

			while(true) {
				//Aceptamos cliente y creamos todos los flujos de
				//entrada/salida
				cliente = socket.accept();
				entrada = cliente.getInputStream();
				entradaDatos = new DataInputStream(entrada);
				salida = cliente.getOutputStream();
				salidaDatos = new DataOutputStream(salida);

				//Obtenemos fechas de ficheros
				fichero = new File("./instruccionesCreate.txt");
				fechaServidorCreate = fichero.lastModified();
				fichero = new File("./instruccionesInsert.txt");
				fechaServidorInsert = fichero.lastModified();

				//Enviamos las fechas
				salidaDatos.writeLong(fechaServidorCreate);
				salidaDatos.writeLong(fechaServidorInsert);
```

```
//Después de enviar la fecha, el servidor se queda a la
//espera de que el cliente le comunique qué hacer
linea = entradaDatos.readUTF();

//Ahora decide qué hacer según lo que haya recibido
if(linea.equals(CREAR_BD)) {
	//La base de datos del cliente está vacía, hay que
	//rellenarla.

	//Le decimos al cliente el número de mensajes
	//(líneas) que vamos a mandarle de
	//instruccionesCreate
	br = new BufferedReader(new FileReader(new
		File("./instruccionesCreate.txt")));
	numeroLineas = 0;
	while(br.readLine() != null)
		numeroLineas++;
	salidaDatos.writeInt(numeroLineas);

	//Abrimos el fichero y lo dejamos listo para su
	//lectura
	fichero = new File("./instruccionesCreate.txt");
	fr = new FileReader(fichero);
	br = new BufferedReader(fr);

	/*
	 * Como sabemos el número de líneas del fichero,
	 * podemos realizar el
	 * envío del fichero línea a línea con un bucle for
	 */

	for(int i = 1; i <= numeroLineas; i++) {
		linea = br.readLine();
	salidaDatos.writeUTF(linea);
	}

	//Hemos enviado el fichero de create, ahora hay que
	//enviar el fichero de insert

	//Abrimos fichero y contamos cuántas líneas se van
	a enviar
	br = new BufferedReader(new FileReader(new
		File("./instruccionesInsert.txt")));
	numeroLineas = 0;
	while(br.readLine() != null)
		numeroLineas++;
	salidaDatos.writeInt(numeroLineas);

	//Como sabemos el número de líneas que contiene el
	//fichero, usamos un bucle for para enviarlo
	br = new BufferedReader(new FileReader(new
		File("./instruccionesInsert.txt")));
	for(int i = 1; i <= numeroLineas; i++) {
		linea = br.readLine();
		salidaDatos.writeUTF(linea);
	}

	//Cerramos todos los flujos
	cliente.close();
	entrada.close();
	entradaDatos.close();
	salida.close();
	salidaDatos.close();
	fr.close();
	br.close();

}
```

```
                        else if(linea.equals(ACTUALIZAR_BD)) {
                                //Abrimos fichero y lo barremos para contar las
                                //líneas
                                br = new BufferedReader(new FileReader(new
                                        File("./instruccionesInsert.txt")));
                                numeroLineas = 0;
                                while(br.readLine() != null)
                                        numeroLineas++;
                                salidaDatos.writeInt(numeroLineas);

                                //Como sabemos el número de líneas que contiene el
                                //fichero, usamos un bucle for para enviarlo
                                br = new BufferedReader(new FileReader(new
                                        File("./instruccionesInsert.txt")));
                                for(int i = 1; i <= numeroLineas; i++) {
                                        linea = br.readLine();
                                        salidaDatos.writeUTF(linea);
                                }

                                //Cerramos todos los flujos
                                cliente.close();
                                entrada.close();
                                entradaDatos.close();
                                salida.close();
                                salidaDatos.close();
                                br.close();
                        }

                        else if(linea.equals(NO_ACTUAR)) {
                                continue;
                        }

                }


        }       //cierra try

        catch(Exception e) {
                e.printStackTrace();
        }
    }

}       //Cierra clase
```

```
package pfc.algoritmo;

import java.io.BufferedReader;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;

import android.app.Activity;
import android.content.Intent;
import android.database.Cursor;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Toast;

/*
 * Esta actividad es la primera del programa.
 * Muestra el mensaje de bienvenida y se conectará
 * al servidor cuando se pulse el botón
 */

public class Conexion extends Activity {

	Socket socket;
	InputStream entrada;
	DataInputStream entradaDatos;
	OutputStream salida;
	DataOutputStream salidaDatos;
	File ficheroCreate, ficheroInsert;
	FileOutputStream salidaFicheroCreate, salidaFicheroInsert;
	FileInputStream entradaFicheroCreate, entradaFicheroInsert;
	AccesoBD accesoBD;

	static final String CREAR_BD = "CLIENTE_CREAR";
	static final String ACTUALIZAR_BD = "CLIENTE_ACTUALIZAR";
	static final String NO_ACTUAR = "CLIENTE_NO_ACTUAR";

	int numeroLineasCreate = 0, numeroLineasInsert = 0;
	String linea = "";
	long fechaServidorCreate = 0, fechaServidorInsert = 0;
	long fechaClienteCreate = 0, fechaClienteInsert = 0;
	boolean existeTabla;
	boolean existeFichero = false;
	Cursor c = null;

	    /** Called when the activity is first created. */
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.vista_conexion);

		//Creamos un objeto que nos de acceso a la base de datos
		accesoBD = new AccesoBD(this);
		//Abrimos el acceso a la base de datos: modo escritura
		accesoBD.abrirEscritura();

		final Button btnConectar = (Button)findViewById(R.id.BtnConectar);
```

```
btnConectar.setOnClickListener(new OnClickListener() {

/*
 * Al pulsar el botón se conectará al servidor y se descargará
 * o actualizará la base de datos. Después preguntará a
 * qué profesor queremos elegir
 */
	public void onClick(View v) {

		try {

			/*
			* Tres posibles caminos:
			* 1 - No hay base de datos
			* 2 - Hay base de datos pero hay que actualizarla
			* 3 - Hay base de datos y está actualizada
			*/

			//Creamos el canal de comunicaciones y abrimos
			//todos los flujos de entrada/salida
			//TODO Dirección IP a la que conectarse
			socket = new Socket("192.168.0.102", 4000);
			salida = socket.getOutputStream();
			salidaDatos = new DataOutputStream(salida);
			entrada = socket.getInputStream();
			entradaDatos = new DataInputStream(entrada);

			//Recibimos fechas del servidor
			fechaServidorCreate = entradaDatos.readLong();
			fechaServidorInsert = entradaDatos.readLong();

			/*
			 * Un fichero contendrá los CREATE, otro fichero
			 * contendrá los INSERT. Los dejamos listos para su
			 * escritura
			 */
			ficheroCreate = new File(getExternalFilesDir(null)
				+ "/instruccionesCreate.dat");
			salidaFicheroCreate = new
				FileOutputStream(ficheroCreate);
			ficheroInsert = new File(getExternalFilesDir(null)
				+ "/instruccionesInsert.dat");
			salidaFicheroInsert = new
				FileOutputStream(ficheroInsert);

			//Averiguamos si la tabla Profesores existe en la
			//base de datos
			existeTabla = accesoBD.existeTabla("Profesores");

			//Primera opción: No existe la tabla Profesores en
			//la base datos. Hay que empezar de cero
			if(existeTabla == false) {
				Log.d("DEBUG", "\"Profesores\" no existe");
				//Notifica al servidor que hay que rellenar
				//la base de datos
				salidaDatos.writeUTF(CREAR_BD);

				//El servidor nos dice el número de mensajes
				//(líneas) que nos va a enviar de
				//instruccionesCreate
				numeroLineasCreate = entradaDatos.readInt();

				/*
				 * Como sabemos el número de líneas del
				 * fichero, podemos realizar la
				 * recepción del fichero línea a línea con
				 * un bucle for
```

```
         */

        //Vamos recibiendo y volcando en el fichero
        for(int i = 1;i <= numeroLineasCreate;i++) {
                linea = entradaDatos.readUTF();
                salidaFicheroCreate.write((linea +
                        "\n").getBytes());
        }

        //Recibimos el número de líneas que contiene
        //instruccionesInsert
        numeroLineasInsert = entradaDatos.readInt();

        //Recibimos el fichero de insert línea a
        //línea
        for(int i = 1;i <= numeroLineasInsert;i++) {
                linea = entradaDatos.readUTF();
                salidaFicheroInsert.write((linea +
                        "\n").getBytes());
        }

        //Insertamos las fechas de modificación en
        //la base de datos
        accesoBD.ejecutar("INSERT INTO Fechas
                (fechaCreate,fechaInsert) VALUES ("
                + ficheroCreate.lastModified() + ",
                " + ficheroInsert.lastModified() +
                ") ");

        //Una vez tenemos los datos necesarios,
        //barremos los ficheros para crear la base
        //de datos y meterle información
        BufferedReader br = new BufferedReader(new
                FileReader(new
                File(getExternalFilesDir(null) +
                "/instruccionesCreate.dat")));

        for(int i = 1;i <= numeroLineasCreate;i++) {
                linea = br.readLine();
                accesoBD.ejecutar(linea);
        }


        br = new BufferedReader(new FileReader(new
                File(getExternalFilesDir(null) +
                "/instruccionesInsert.dat")));

        for(int i = 1;i <= numeroLineasInsert;i++) {
                linea = br.readLine();
                accesoBD.ejecutar(linea);
        }

        //Cerramos todos los flujos
        socket.close();
        salida.close();
        salidaDatos.close();
        entrada.close();
        entradaDatos.close();
        salidaFicheroCreate.close();
        salidaFicheroInsert.close();
        accesoBD.cerrar();

}       //Cierra if(existeTabla == false)

else {  //La tabla de profesores sí existe
        Log.d("DEBUG", "\"Profesores\" sí existe");
```

```
/*
 * Obtenemos las últimas fechas de
 * instruccionesCreate e instruccionesInsert
 * Para ello hacemos una consulta a la tabla
 * de Fechas. Nos movemos a la primera
 * posición del Cursor devuelto y, sabiendo
 * que el primer campo es fechaCreate y
 * el segundo es fechaInsert, los
 * identificamos como 0 y como 1
 * respectivamente.
 */

Cursor c = accesoBD.consultarFecha("SELECT
        fechaCreate,fechaInsert FROM
        Fechas");
c.moveToFirst();
fechaClienteCreate = c.getLong(0);
fechaClienteInsert = c.getLong(1);

/*
 * Comparamos fechas, y en base a eso
 * actuaremos de una u otra forma.
 * La fecha más reciente será aquella cuyo
 * lastModified() sea mayor (cuanto más
 * reciente, más segundos habrán
 * transcurrido desde el 01/01/1970).
 *
 * Posibles situaciones:
 * 1 - La fecha de instruccionesCreate
 * recibida es más reciente que la que
 * tenemos: se recibe el nuevo
 * instruccionesCreate y el nuevo
 * instruccionesInsert.
 * 2 - La fecha de instruccionesCreate
 * recibida es más antigua que la que
 * tenemos: se desdobla en dos casos más:
 * 2a -La fecha de InstruccionesInsert es
 * más reciente: recibimos el
 * nuevo instruccionesInsert
 * 2b -La fecha de instruccionesInsert es
 * más antigua: no hacemos nada
 */

//Caso 1
if(fechaServidorCreate>fechaClienteCreate) {
        //Partimos de cero: CREAR_BD
        //Hay que vaciar la base de datos y
        //rellenarla de nuevo

Log.d("DEBUG",
        "fechaServidorCreate >
        fechaClienteCreate");

//Hay que borrar la tabla Profesores
//porque al barrer el fichero recibido se
//volverá a crear
accesoBD.ejecutar("DROP TABLE
        Profesores");

//La tabla de fechas únicamente se vuelve a
//poner a cero
accesoBD.ejecutar("UPDATE Fechas SET
        fechaCreate=0,fechaInsert=0");

//Si existen los ficheros, los eliminamos
if(ficheroCreate.exists())
        existeFichero =
```

```
                ficheroCreate.delete();
if(existeFichero == false)
        throw new
                ExcepcionFichero("Problema al
                crear ficheroCreate");
if(ficheroInsert.exists())
        existeFichero =
                ficheroInsert.delete();
if(existeFichero == false)
        throw new
                ExcepcionFichero("Problema al
                crear ficheroInsert");

//Creamos de nuevo los ficheros y sus
//flujos de salida

ficheroCreate = new
        File(getExternalFilesDir(null) +
        "/instruccionesCreate.dat");
salidaFicheroCreate = new
        FileOutputStream(ficheroCreate);
ficheroInsert = new
        File(getExternalFilesDir(null) +
        "/instruccionesInsert.dat");
salidaFicheroInsert = new
        FileOutputStream(ficheroInsert);

/*
 * Iniciamos la transferencia de datos:
 * notificamos al servidor.
 * (En realidad estamos haciendo lo mismo
 * que si no existiera la tabla
 * Profesores).
 */

salidaDatos.writeUTF(CREAR_BD);

//El servidor nos dice el número de mensajes
//(líneas) que nos va a enviar de
//instruccionesCreate
numeroLineasCreate = entradaDatos.readInt();

/*
 * Como sabemos el número de líneas
 * del fichero, podemos realizar la
 * recepción del fichero línea a
 * línea con un bucle for
 */

//Vamos recibiendo y volcando en el fichero
for(int i = 1;i <= numeroLineasCreate;i++) {
        linea = entradaDatos.readUTF();
        salidaFicheroCreate.write((linea +
                "\n").getBytes());
}

//Recibimos el número de líneas que contiene
//instruccionesInsert
numeroLineasInsert = entradaDatos.readInt();
//Recibimos el fichero de insert línea a
//línea
for(int i = 1;i <= numeroLineasInsert;i++) {
        linea = entradaDatos.readUTF();
        salidaFicheroInsert.write((linea +
                "\n").getBytes());
}
```

```
            //Insertamos las fechas de modificación en
            //la base de datos
            accesoBD.ejecutar("UPDATE Fechas SET
                    fechaCreate=" +
                    ficheroCreate.lastModified());
                    accesoBD.ejecutar("UPDATE Fechas SET
                    fechaInsert=" +
                    ficheroInsert.lastModified());

            /*
             * Una vez tenemos toda la información
             * necesaria, barremos los ficheros
             * para crear y rellenar la tabla
             */

            BufferedReader br = new BufferedReader(new
                    FileReader(new
                    File(getExternalFilesDir(null) +
                    "/instruccionesCreate.dat")));

            for(int i = 1;i <= numeroLineasCreate;i++) {
                    linea = br.readLine();
                    accesoBD.ejecutar(linea);
            }

            br = new BufferedReader(new FileReader(new
                    File(getExternalFilesDir(null) +
                    "/instruccionesInsert.dat")));

            for(int i = 1;i <= numeroLineasInsert;i++) {
                    linea = br.readLine();
                    accesoBD.ejecutar(linea);
            }

            //Cerramos todos los flujos
            socket.close();
            salida.close();
            salidaDatos.close();
            entrada.close();
            entradaDatos.close();
            salidaFicheroCreate.close();
            salidaFicheroInsert.close();
            accesoBD.cerrar();

    }       //Cierra el caso 1

    //Caso 2a
    else if(fechaServidorInsert >
    fechaClienteInsert) {
            //Obtenemos sólo el nuevo
            //instruccionesInsert
            Log.d("DEBUG", "fechaServidorInsert>
                    fechaClienteInsert");

            //Notificamos al servidor
            salidaDatos.writeUTF(ACTUALIZAR_BD);

            //Recibimos el número de líneas del
            //fichero
            numeroLineasInsert = entradaDatos.readInt();

            //Creamos el fichero de nuevo
            ficheroInsert = new
                    File(getExternalFilesDir(null) +
                    "/instruccionesInsert.dat");
            salidaFicheroInsert = new
                    FileOutputStream(ficheroInsert);
```

```
                                //Recibimos el fichero línea a línea
                                for(int i = 1;i <= numeroLineasInsert;i++) {
                                        linea = entradaDatos.readUTF();
                                        salidaFicheroInsert.write((linea
                                                + "\n").getBytes());
                                }

                                //Actualizamos la fecha de
                                //instruccionesInsert
                                accesoBD.ejecutar("UPDATE Fechas SET
                                        fechaInsert=" +
                                        ficheroInsert.lastModified());

                                //Eliminamos la tabla de profesores
                                accesoBD.ejecutar("DELETE FROM Profesores");
                                //Barremos el fichero y vamos ejecutando
                                //línea a línea
                                BufferedReader br = new BufferedReader(new
                                        FileReader(new
                                        File(getExternalFilesDir(null) +
                                        "/instruccionesInsert.dat")));


                                for(int i = 1;i <= numeroLineasInsert;i++) {
                                        linea = br.readLine();
                                        accesoBD.ejecutar(linea);
                                }

                                //Cerramos todos los flujos
                                socket.close();
                                salida.close();
                                salidaDatos.close();
                                entrada.close();
                                entradaDatos.close();
                                salidaFicheroCreate.close();
                                salidaFicheroInsert.close();
                                accesoBD.cerrar();

                        }       //Cierra el caso 2a

                        //Caso 2b
                        else {
                                //No hacer nada
                                Log.d("DEBUG", "No hacer nada");

                                salidaDatos.writeUTF(NO_ACTUAR);
                        }       //Cierra el caso 2b

                }       //Cierra el else: "Profesores" existe

                //Llamamos a Selección para que el usuario elija el
                //profesor
                Intent intent = new Intent(Conexion.this,
                        Seleccion.class);
                startActivity(intent);
                finish();

        }       //Cierra el try

        catch(Exception e) {
                (Toast.makeText(getApplicationContext(), e.getMessage(),
                Toast.LENGTH_LONG)).show();
        }       //Cierra catch

}       //Cierra onClick de btnConectar
```

```
        });    //Cierra onClickListener de btnConectar

    }  //Cierra main

    @Override
    protected void onResume() {
      //datasource.open();
      super.onResume();
    }

    @Override
    protected void onPause() {
      //datasource.close();
      super.onPause();
    }

}       //Cierra clase
```

```
package pfc.algoritmo;
import java.util.StringTokenizer;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.AutoCompleteTextView;
import android.widget.Toast;

/*
 * Esta actividad nos muestra el cuadro de texto
 * con autocompletado para elegir un profesor
 */

public class Seleccion extends Activity {

	/*
	 * Declaramos aquí estas variables para poder usarlas
	 * cuando capturemos los eventos generados
	 */
	Cursor c = null;
	Intent intent = null;
	Bundle bundle = null;
	StringTokenizer tokenizer = null;
	String nombre = null, apellidos = null;
	AccesoBD accesoBD;
	boolean llamadaPermitida;

	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.vista_seleccion);

		final AutoCompleteTextView lista =
			(AutoCompleteTextView)findViewById(R.id.lstProfesores);

		String[] datos = null;//Apellidos, Nombre

		ArrayAdapter<String> adaptador = null;

		try {

			//Obtenemos acceso a la base de datos
			accesoBD = new AccesoBD(this);

			//Obtenemos la base de datos
			accesoBD.abrirLectura();

			//Obtenemos un Cursor con el nombre y el apellido
			//de todos y cada uno de los profesores
			c = accesoBD.consultar("SELECT apellidos,nombre FROM Profesores");

			//Disponemos todo lo necesario para crear la lista desplegable
			c.moveToFirst();
			datos = new String[c.getCount()];
			for(int i = 0; i < datos.length; i++) {
				datos[i] = "" + c.getString(0) + ", " +
				c.getString(1);
				c.moveToNext();
			}
```

```
adaptador = new ArrayAdapter<String>(this,
        android.R.layout.simple_dropdown_item_1line, datos);

//Insertamos la información desplegable en la spinner
adaptador.setDropDownViewResource(android.R.layout.
        simple_spinner_dropdown_item);
lista.setAdapter(adaptador);

//Preparamos la lista desplegable para que esté atenta a //los
//eventos que se puedan generar
lista.setOnItemClickListener(new OnItemClickListener() {

        public void onItemClick(AdapterView<?> parent, View
        view, int position, long id) {
                //Usamos un StringTokenizer para dividir la String
                //por la coma
                tokenizer = new StringTokenizer(parent.
                        getItemAtPosition(position).
                        toString(), ",");
                apellidos = tokenizer.nextToken();
                nombre = tokenizer.nextToken().trim();
                Log.i("DEBUG", "Nombre: " + nombre);
                Log.i("DEBUG", "Apellidos: " + apellidos);

                //Crearemos un tipo de diálogo u otro en función
                //de si se puede llamar o no al sitio

                c = accesoBD.consultar("SELECT telefono FROM
                        Profesores WHERE nombre = '" + nombre + "'
                        AND apellidos = '" + apellidos + "'");

                //Si el teléfono es 0 no tenemos opción de llamar
                //por teléfono
                c.moveToFirst();
                if(c.getInt(0) == 0)
                        llamadaPermitida = false;
                else
                        llamadaPermitida = true;

                //Creamos el cuadro de diálogo que nos hace la
                //pregunta

                AlertDialog.Builder builder = new
                        AlertDialog.Builder(Seleccion.this);
                builder.setTitle("" + nombre + " " +
                        apellidos);
                builder.setMessage("¿Qué quieres hacer?");
                builder.setCancelable(true);

                //TODO Esto sólo se debería producir si se nos
                //permite llamar
                if(llamadaPermitida) {
                        builder.setPositiveButton("Llamar al
                        despacho", new DialogInterface.
                        OnClickListener() {

                                //Hacemos todo lo necesario si el
                                //usuario quiere llamar
                                public void onClick(DialogInterface
                                dialog, int which) {
                                        Log.i("TRAZA", "SELECT
                                        telefono FROM Profesores
                                        WHERE nombre='" + nombre +"'
                                        AND apellidos ='" + apellidos
                                        + "'");

                                        c = accesoBD.consultar(
```

```
                                "SELECT telefono FROM
                                Profesores WHERE
                                nombre='" + nombre +"'
                                AND apellidos ='" +
                                apellidos + "'");
                        c.moveToFirst();
                        Log.i("TRAZA", "" +
                                c.getColumnCount() +
                                " " + c.getInt(0));

                        intent = new
                          Intent(Intent.ACTION_CALL);
                        intent.setData(Uri.parse("tel
                                :" + c.getInt(0)));
                        startActivity(intent);

                        //Cancelamos el diálogo
                        dialog.cancel();
                }
        });
}       //if(llamadaPermitida)


builder.setNegativeButton("Ir en persona",
new DialogInterface.OnClickListener() {

        //Si el usuario quiere ir al despacho,
        //lanzaremos un cuadro de diálogo
        //avisándole para que se prepare para leer
        //un QR

        public void onClick(DialogInterface dialog,
        int which) {
                AlertDialog.Builder builderQR =
                        New AlertDialog.
                        Builder(Seleccion.this);
                builderQR.setTitle("AVISO");
                builderQR.setMessage(
                        "Busca un código QR");

                builderQR.setCancelable(false);

                builderQR.setNeutralButton("OK",
                new DialogInterface.
                OnClickListener() {

                        public void
                        onClick(DialogInterface
                        dialog, int which) {

                                /*
                                Llama   a la
                                actividad que tiene el
                                lector QR y el
                                algoritmo
                                */
                                //Tiene que pasarle
                                //el destino
                                //buscado: mapa y
                                //nodo
                                intent = new
                                        Intent(
                                        Seleccion.this
                                        ,RecogeQR.clas
                                        s);
                                bundle = new
                                        Bundle();
```

```
                                                        c = accesoBD.
                                                                consultar("
                                                                SELECT
                                                                mapa,nodo FROM
                                                                Profesores
                                                                WHERE
                                                                nombre='" +
                                                                nombre +"' AND
                                                                apellidos ='"
                                                                + apellidos +
                                                                "'");

                                                        c.moveToFirst();

                                                        bundle.putInt(
                                                                "MAPA",
                                                                c.getInt(0));

                                                        bundle. putInt(
                                                                "NODO",
                                                                c.getInt(1));

                                                        intent.putExtras(
                                                                bundle);
                                                        intent.
                                                                setFlags(
                                                                Intent.
                                                                FLAG_ACTIVITY_
                                                                CLEAR_TOP);

                                                        startActivity(intent);

                                                        //Cancelamos el
                                                        //diálogo

                                                        dialog.cancel();
                                                }
                                                });
                                                builderQR.create().show();
                                        }
                                });
                                builder.create().show();
                        }

                });

        }

        catch(Exception e) {
                (Toast.makeText(getApplicationContext(), e.getMessage(),
                        Toast.LENGTH_LONG)).show();
        }
    }
}       //Cierra la clase
```

```
package pfc.algoritmo;

import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.Collections;
import java.util.Vector;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.database.Cursor;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Paint.Style;
import android.graphics.Path;
import android.os.Bundle;
import android.util.DisplayMetrics;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.widget.ImageView;
import android.widget.Toast;

public class RecogeQR extends Activity {

	/*
	 * Declaramos estas variables fuera para que sean
	 * visibles en los dos métodos
	 */

	private static final String LLAMAR_LECTOR = "la.droid.qr.scan";
	private static final String QR_RESULT = "la.droid.qr.result";

	private int mapaDestino, idNodoDestino;
	private Intent qrDroid = null;
	private Nodo nodoDestino = null;

	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);

		try {

			mapaDestino = getIntent().getExtras().getInt("MAPA");
			idNodoDestino = getIntent().getExtras().getInt("NODO");
			nodoDestino = new Nodo(idNodoDestino);
			nodoDestino.setMapa(mapaDestino);

			qrDroid = new Intent(LLAMAR_LECTOR);
			qrDroid.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
			startActivityForResult(qrDroid,0);

			//El QR contiene un simple número, el ID del profesor buscado
		}
		catch(Exception e) {
			(Toast.makeText(getApplicationContext(), "Algo ha pasado en
			onCreate: " + e.getMessage(), Toast.LENGTH_LONG)).show();
		}

	}

	public void onActivityResult(int requestCode, int resultCode, Intent data) {
```

```
setContentView(R.layout.vista_recogida);

Nodo nodoOrigen = null;
boolean dibujar = true;
int idNodo, idProfesor;
int mapaOrigen, idNodoOrigen;
int idMapaOrigen, idMapaDestino;
int maximos[], minimos[];
int filasPintar, columnasPintar;
int anchoPantalla, altoPantalla;
int incrementoAncho, incrementoAlto;
int inicioX, inicioY;
int posicionX, posicionY;
int radio;
Nodo[] nodosEnlace = null;
String resultado = null, linea = null;
String[] cadenas = null, numeros = null, coordenadas = null,
       enlaces = null, mapas = null;
InputStream inputStream = null;
InputStreamReader inputStreamReader = null;
BufferedReader br = null;
AccesoBD accesoBD = null;
Cursor c = null;
Camino[] listaRutas = null;
Vector<Nodo> rutaEscogida = null;
DisplayMetrics dm;
Centro listaCentros[] = null;
Path path = null;

//Estas variables son para manejar los gráficos
Canvas canvas = null;
Bitmap bitmap = null;
Bitmap escalera = null;
Paint paint = null;

ImageView img = (ImageView)findViewById(R.id.imagenRuta);

//Obtenemos la información de la ubicación de inicio
resultado = data.getExtras().getString(QR_RESULT);

/*
Ahora el QR sólo contiene un número, el ID del profesor buscado.
Mediante ese ID, se consultará en la base de datos para obtener
toda la información que necesitemos.
*/

//Obtenemos el ID del profesor buscado
idProfesor = Integer.parseInt(resultado);

//Conectamos con la base de datos
accesoBD = new AccesoBD(this);
accesoBD.abrirLectura();

//Obtenemos la información necesaria
c = accesoBD.consultar("SELECT mapa,nodo FROM Profesores WHERE id
       = " + idProfesor);
c.moveToFirst();
mapaOrigen = c.getInt(0);
idNodoOrigen = c.getInt(1);

//Como ya no necesitamos más información de la base de datos,
//cerramos el acceso
accesoBD.cerrar();

//Creamos el nodo origen
nodoOrigen = new Nodo(idNodoOrigen);
nodoOrigen.setMapa(mapaOrigen);
```

```
try {
        //Ahora necesitamos saber las coordenadas
        //tanto del nodo origen como del nodo destino
        switch(nodoOrigen.getMapa()) {
                case 0: inputStream = getResources().
                                openRawResource(R.raw.planta0);
                        break;
                case 1: inputStream = getResources().
                                openRawResource(R.raw.planta1);
                        break;
                case 2: inputStream = getResources().
                                openRawResource(R.raw.planta2);
                        break;
        }       //Cierra switch

        //Vamos barriendo el fichero buscando el id del nodo origen
        //para obtener sus coordenadas
        inputStreamReader = new InputStreamReader(inputStream);
        br = new BufferedReader(inputStreamReader);

        //La primera línea no nos interesa
        linea = br.readLine();

        while((linea = br.readLine()) != null) {
                //Separamos la lista de adyacencia de las
                //coordenadas
                cadenas = linea.split("-");

                //Obtenemos el nodo y sus coordenadas
                numeros = cadenas[0].split(" ");
                coordenadas = cadenas[1].split(" ");
                //El primer número es el identificador de nodo
                idNodo = Integer.parseInt(numeros[0]);

                if(idNodo == nodoOrigen.getId()) {

                        nodoOrigen.setX(Integer.
                                parseInt(coordenadas[0]));

                        nodoOrigen.setY(Integer.
                                parseInt(coordenadas[1]));

                        //Como ya tenemos toda la información
                        //necesaria no necesitamos seguir
                        //recorriendo el fichero
                        break;
                }

        }       //Cierra while

        //Repetimos el proceso para el nodo destino

        switch(nodoDestino.getMapa()) {
                case 0: inputStream = getResources().
                                openRawResource(R.raw.planta0);
                        break;
                case 1: inputStream = getResources().
                                openRawResource(R.raw.planta1);
                        break;
                case 2: inputStream = getResources().
                                openRawResource(R.raw.planta2);
                        break;
        }       //Cierra switch

        inputStreamReader = new InputStreamReader(inputStream);
        br = new BufferedReader(inputStreamReader);
```

```
linea = br.readLine();

while((linea = br.readLine()) != null) {
        cadenas = linea.split("-");
        numeros = cadenas[0].split(" ");
        coordenadas = cadenas[1].split(" ");
        idNodo = Integer.parseInt(numeros[0]);

        if(idNodo == nodoDestino.getId()) {

                nodoDestino.setX(Integer.
                        parseInt(coordenadas[0]));
                nodoDestino.setY(Integer.
                        parseInt(coordenadas[1]));
                //Como ya tenemos toda la información
                //necesaria no necesitamos seguir
                //recorriendo el fichero
                break;
        }

}       //Cierra while

/*
 * Ya tenemos toda la información necesaria tanto del nodo
 * origen como del nodo destino. Ahora hay que decidir qué
 * hacer en función de si los dos nodos están en el mismo
 * mapa o no...
 */

//Si el origen y el destino están en el mismo mapa
if(nodoOrigen.getMapa() == nodoDestino.getMapa()) {
        //No necesitamos buscar los nodos enlace
        if((nodoOrigen.getX() == nodoDestino.getX()) &&
        nodoOrigen.getY() == nodoDestino.getY()) {
                dibujar = false;
        }
        else {
                switch(nodoOrigen.getMapa()) {
                        case 0: inputStream = getResources().
                                        openRawResource(
                                        R.raw.planta0);
                                break;
                        case 1: inputStream = getResources().
                                        openRawResource(
                                        R.raw.planta1);
                                break;
                        case 2: inputStream = getResources().
                                        openRawResource(
                                        R.raw.planta2);
                                break;
                }

                inputStreamReader = new
                        InputStreamReader(inputStream);
                br = new BufferedReader(inputStreamReader);
                rutaEscogida = buscarRuta(nodoOrigen,
                        nodoDestino, br);
        }
}
else {
        //No están en el mismo mapa -> buscar enlaces
        //Necesitamos saber los enlaces del mapa origen
        //También las coordenadas de esos enlaces

        inputStream = getResources().openRawResource(
                R.raw.supermapa);
        inputStreamReader =
```

```
        new InputStreamReader(inputStream);
br = new BufferedReader(inputStreamReader);

/*
Hay que buscar la línea del fichero cuyos mapas
origen y destino sean los mismos que los que
tenemos
*/

while((linea = br.readLine()) != null) {

        cadenas = linea.split("-");
        mapas = cadenas[0].split(" ");

        idMapaOrigen = Integer.parseInt(mapas[0]);
        idMapaDestino = Integer.parseInt(mapas[1]);

        if(idMapaOrigen == mapaOrigen &&
        idMapaDestino == mapaDestino) {
                //Aquí obtenemos los nodos enlace
                //pertinentes
                enlaces = cadenas[1].split(" ");
                nodosEnlace = new
                        Nodo[enlaces.length];

                for(int k = 0;k <
                nodosEnlace.length;k++) {
                        nodosEnlace[k] = new
                                Nodo(Integer.parseInt(
                                enlaces[k]));

                        nodosEnlace[k].setMapa(
                                mapaOrigen);
                }
        }

}       //Cierra while

//Buscamos las coordenadas de los nodos enlace

switch(nodosEnlace[0].getMapa()) {
        //Como todos los nodos están en el mismo
        //mapa, basta con obtener el mapa de uno de
        //ellos
        case 0: inputStream = getResources().
                        openRawResource(
                        R.raw.planta0);
                break;
        case 1: inputStream = getResources().
                        openRawResource(
                        R.raw.planta1);
                break;

        case 2: inputStream = getResources().
                        openRawResource(
                        R.raw.planta2);
                break;
}

inputStreamReader = new
        InputStreamReader(inputStream);
br = new BufferedReader(inputStreamReader);

//La primera línea no nos interesa
br.readLine();
//Barremos el fichero buscando las coordenadas de
//cada nodo enlace
```

```
while((linea = br.readLine()) != null) {
        cadenas = linea.split("-");
        numeros = cadenas[0].split(" ");

        for(int n = 0;n < nodosEnlace.length; n++) {
                if(Integer.parseInt(numeros[0]) ==
                nodosEnlace[n].getId()) {
                        coordenadas =
                                cadenas[1].split(" ");

                        nodosEnlace[n].setX(Integer.
                                parseInt(
                                coordenadas[0]));
                        nodosEnlace[n].setY(Integer.
                                parseInt(
                                coordenadas[1]));
                }
        }

}       //Cierra while

//Ya tenemos las coordenadas de los nodos enlace
//Crear todas las rutas posibles y elegir la más
//corta
//Nos creamos todas las posibles rutas
listaRutas = new Camino[nodosEnlace.length];

try {
        for(int i = 0;i < nodosEnlace.length;i++) {
                switch(nodosEnlace[i].getMapa()) {
                        case 0: inputStream =
                                     getResources().
                                     openRawResource(
                                     R.raw.planta0);
                                break;
                        case 1: inputStream =
                                     getResources().
                                     openRawResource(
                                     R.raw.planta1);
                                break;
                        case 2: inputStream =
                                     getResources().
                                     openRawResource(
                                     R.raw.planta2);
                                break;
                }

        inputStreamReader = new
                InputStreamReader(inputStream);
        br = new BufferedReader(
                inputStreamReader);
        listaRutas[i] = new Camino(
                                buscarRuta(
                                nodoOrigen,
                                nodosEnlace[i],br));
        }
}
catch(Exception e) {

}

//Recorremos el array de objetos Camino buscando la
//ruta más corta
rutaEscogida = listaRutas[0].getRuta();
for(int j = 1; j < listaRutas.length; j++) {
        if(listaRutas[j].getRuta().size() <
        rutaEscogida.size()) {
```

```
                        rutaEscogida =
                                listaRutas[j].getRuta();
                }
        }

}       //Cierra else

//TODO Este código se ejecutará siempre (mientras la flag
//nos lo permita)
if(dibujar) {
        //Una vez tenemos la ruta, habrá que añadirle el
        //nodo de inicio también
        rutaEscogida.add(nodoOrigen);

        //Ahora RecogeQR procesa la ruta para generar el
        //gráfico
        //Tenemos que recorrer la ruta elegida buscando el
        //máximo valor de x e y
        /*
        Para el caso de los despachos de Antigones, x
        valdrá como máximo 28, e y valdrá como máximo 8
        */
        maximos = new int[2];
        minimos = new int[2];
        maximos[0] = 0;
        maximos[1] = 0;
        minimos[0] = 100;
        minimos[1] = 100;
        //maximos[0] almacenará el mayor valor de la
        //coordenada x, maximos[1] hace lo mismo con la
        //coordenada y
        //El array minimos hace lo mismo pero con los
        //minimos
        for(int i = 0; i < rutaEscogida.size(); i++) {
                if(rutaEscogida.elementAt(i).getX() >
                maximos[0])
                        maximos[0] =
                                rutaEscogida.
                                elementAt(i).getX();
                if(rutaEscogida.elementAt(i).getY() >
                maximos[1])
                        maximos[1] =
                                rutaEscogida.
                                elementAt(i).getY();
                if(rutaEscogida.elementAt(i).getX() <
                minimos[0])
                        minimos[0] =
                                rutaEscogida.
                                elementAt(i).getX();
                if(rutaEscogida.elementAt(i).getY() <
                minimos[1])
                        minimos[1] =
                                rutaEscogida.
                                        elementAt(i).getY();
        }

        filasPintar = maximos[0] - minimos[0] + 1;
        columnasPintar = maximos[1] - minimos[1] + 1;

        //Obtenemos las dimensiones de la pantalla
        dm = new DisplayMetrics();

        getWindowManager().getDefaultDisplay().
                getMetrics(dm);
        anchoPantalla = dm.widthPixels;
        altoPantalla = dm.heightPixels;
```

```
//Realizamos la división de filas y columnas

incrementoAncho =
        (int)(anchoPantalla/columnasPintar);
incrementoAlto =
        (int)(altoPantalla/filasPintar);

//Calculamos el radio de los círculos a pintar
if(incrementoAncho < incrementoAlto)
        radio = incrementoAncho;
else if(incrementoAlto < incrementoAncho)
        radio = incrementoAlto;
else
        radio = incrementoAncho;

radio = radio/3;

//Nos creamos la lista de puntos que usaremos como
//centro para dibujar los círculos
//hay que traducir las coordenadas absolutas a
//relativas
//La x e y que contiene el array minimos son el
//origen de nuestras celdas
listaCentros = new Centro[rutaEscogida.size()];

for(int i = 0; i < listaCentros.length; i++) {
        listaCentros[i] = new
                Centro(rutaEscogida.elementAt(i).
                getX() - minimos[0] + 1,
                rutaEscogida.elementAt(i).getY() -
                minimos[1] + 1);
}

//Preparamos los objetos para dibujar
bitmap = Bitmap.createBitmap(anchoPantalla,
        altoPantalla, Bitmap.Config.ARGB_8888);
canvas = new Canvas(bitmap);
paint = new Paint();
path = new Path();

canvas.drawColor(Color.BLACK);
paint.setStrokeWidth(0);
paint.setStyle(Style.STROKE);
paint.setColor(Color.LTGRAY);

//Creamos el dibujo de la ruta
inicioX = (listaCentros[0].getY() -
        1)*incrementoAncho - 1;
inicioY = (listaCentros[0].getX() -
        1)*incrementoAlto - 1;
posicionX = inicioX + incrementoAncho/2;
posicionY = inicioY + incrementoAlto/2;
path.moveTo(posicionX, posicionY);

for(int i = 1; i < listaCentros.length; i++) {
        inicioX = (listaCentros[i].getY() -
                1)*incrementoAncho - 1;
        inicioY = (listaCentros[i].getX() -
                1)*incrementoAlto - 1;
        posicionX = inicioX + incrementoAncho/2;
        posicionY = inicioY + incrementoAlto/2;
        path.lineTo(posicionX, posicionY);
}

//Dibujamos la ruta
paint.setColor(Color.WHITE);
canvas.drawPath(path, paint);
```

```
//Dibujamos un círculo en cada punto, más pequeño
//que origen y destino
paint.setStyle(Style.FILL);
for(int i = 0; i < listaCentros.length; i++) {
	inicioX = (listaCentros[i].getY() -
		1)*incrementoAncho - 1;
	inicioY = (listaCentros[i].getX() -
		1)*incrementoAlto - 1;
	posicionX = inicioX + incrementoAncho/2;
	posicionY = inicioY + incrementoAlto/2;
	canvas.drawCircle(posicionX, posicionY,
		radio/2, paint);
}

//Pintamos el primer círculo de verde
inicioX = (listaCentros[listaCentros.length -
	1].getY() - 1)*incrementoAncho - 1;
inicioY = (listaCentros[listaCentros.length -
	1].getX() - 1)*incrementoAlto - 1;
posicionX = inicioX + incrementoAncho/2;
posicionY = inicioY + incrementoAlto/2;
paint.setColor(Color.GREEN);
canvas.drawCircle(posicionX, posicionY, radio,
	paint);

//TODO Pintamos el último círculo de rojo o
//dibujamos una escalera. Dependerá de si el mapa
//origen y el nodo destino son el mismo o no
if(mapaOrigen == mapaDestino) {
	//Si el destino está en este mapa dibujamos
	//el punto rojo
	inicioX = (listaCentros[0].getY() -
		1)*incrementoAncho - 1;
	inicioY = (listaCentros[0].getX() -
		1)*incrementoAlto - 1;
	posicionX = inicioX + incrementoAncho/2;
	posicionY = inicioY + incrementoAlto/2;
	paint.setColor(Color.RED);
	canvas.drawCircle(posicionX, posicionY,
		radio, paint);
}

else if(mapaOrigen != mapaDestino) {
	//Si el destino está en otro mapa, hay que
	//dibujar una escalera
	//Compruebo en qué mapa estoy
	if(rutaEscogida.elementAt(0).getMapa() <
	mapaDestino) {
		//Hay que subir
		escalera = BitmapFactory.
				decodeResource
				(getResources(),
				R.drawable.upstairs);
	}
	else {
		//Hay que bajar
		escalera = BitmapFactory.
				decodeResource
				(getResources(),
				R.drawable.downstairs)
				;
	}
	//Escalamos el gráfico
	escalera = Bitmap.createScaledBitmap(
		escalera, radio*2, radio*2, true);
```

```
                                        //Dibujamos la escalera
                                        inicioX = (listaCentros[0].getY() -
                                                1)*incrementoAncho - 1;
                                        inicioY = (listaCentros[0].getX() -
                                                1)*incrementoAlto - 1;
                                        posicionX = inicioX + incrementoAncho/2;
                                        posicionY = inicioY + incrementoAlto/2;
                                        canvas.drawBitmap(escalera, posicionX -
                                                escalera.getWidth()/2, posicionY -
                                                escalera.getHeight()/2, null);
                                }

                                img.setImageBitmap(bitmap);
                        }

                        else {
                                AlertDialog.Builder builder = new
                                        AlertDialog.Builder(RecogeQR.this);
                                builder.setMessage("Ya estás en tu destino");
                                builder.setCancelable(false);
                                builder.setPositiveButton("OK", new
                                DialogInterface.OnClickListener() {
                                        //Hacemos todo lo necesario si el usuario
                                        //quiere llamar
                                        public void onClick(DialogInterface dialog,
                                        int which) {
                                                //Cancelamos el diálogo
                                                Intent intent = new
                                                        Intent(RecogeQR.this,
                                                        Seleccion.class);

                                                intent.setFlags(
                                                        Intent.
                                                        FLAG_ACTIVITY_CLEAR_TOP);
                                                startActivity(intent);
                                                dialog.cancel();
                                        }
                                });
                                builder.create().show();
                        }

                }       //Cierra try
                catch(Exception e) {
                        (Toast.makeText(getApplicationContext(), "Algo ha pasado en
                        onActivityResult: " + e.getMessage(),
                        Toast.LENGTH_LONG)).show();
                }
                finally {
                        try {
                                inputStream.close();
                                inputStreamReader.close();
                                br.close();
                        }
                        catch(Exception e) {
                                (Toast.makeText(getApplicationContext(), "Algo ha
                                pasado al cerrar flujos: " + e.getMessage(),
                                Toast.LENGTH_LONG)).show();
                        }
                }

        }

public static Vector<Nodo> buscarRuta(Nodo nodoOrigen, Nodo nodoDestino,
BufferedReader br) {

        //Al llamar al constructor de mapaNodos se establecen todas las
        //adyacencias
```

```
ListaNodos mapaNodos = new ListaNodos(br);
ListaEnlazada listaAbierta = new ListaEnlazada();
//Declaramos los incrementos como constantes para que nunca cambien su
//valor
final int incrementoOrtogonal = 10;
final int incrementoDiagonal = 14;
int g = 0, h = 0, id = 0;
int gActual = 0, gPrima = 0;
boolean encontradoFinal = false;

//El nodo inicial y final nos los dará el programa
Nodo nodoInicial = mapaNodos.buscar(nodoOrigen.getId());
Nodo nodoFinal = mapaNodos.buscar(nodoDestino.getId());
int idNodoFinal = nodoFinal.getId();

Nodo actual = null;
Vector<Nodo> adyacentes = null;
Vector<Nodo> ruta = new Vector<Nodo>(4,4);
Nodo aux = null;


//Primer paso del algoritmo
listaAbierta.add(nodoInicial);

//Comienza el algoritmo
while(!encontradoFinal) {
	//Buscamos el nodo con la F más baja, lo marcamos como
	////procesado (equivalente a mover a lista cerrada) y //obtenemos
	//sus adyacentes
	actual = listaAbierta.poll();
	actual.setProcesado();
	adyacentes = actual.getAdyacentes();

	//Empezamos a evaluar todos sus nodos adyacentes
	for(int i = 0; i < adyacentes.size(); i++) {
		if(adyacentes.elementAt(i).getProcesado()) {
			//Si está procesado, avanzo al siguiente
		}
		else {
			//Nodo no procesado, comprobamos si está en lista
			//abierta


			if(listaAbierta.contieneId(adyacentes.elementAt(i).
			getId())) {
				//Está en lista abierta, hay que comprobar
				//el mejor camino
				gActual = actual.getG();

				//Hay que averiguar si el camino del nodo
				//actual a este es ortogonal o diagonal
				if(actual.getX() ==
				adyacentes.elementAt(i).getX() ||
				actual.getY() ==
				adyacentes.elementAt(i).getY()) {
					//Movimiento ortogonal
					gPrima = gActual +
						incrementoOrtogonal;
				}
				else {
					//Movimiento diagonal
					gPrima = gActual +
						incrementoDiagonal;
				}

				//Comprobamos si este camino es mejor
				if(gPrima<adyacentes.elementAt(i).getG()) {
```

```
                //Este camino es mejor, cambiamos
                //el padre y reordenamos lista
                //abierta
                id =
                    adyacentes.elementAt(i).getId();
                aux = listaAbierta.buscarNodo(id);

                //Cambiamos el padre de este nodo
                //por el actual
                aux.setPadre(actual);

                //Actualizamos el coste G
                aux.setG(gPrima);

                //Reordenamos lista abierta

                Collections.sort(listaAbierta);

            }
            else {

            }

        }
        else {
            //No está en lista abierta: calcular
            //costes, marcar padre e insertar en lista
            //abierta

            //Calculamos el coste h y lo marcamos
            h = (Math.abs(nodoFinal.getX() -
                    adyacentes.elementAt(i).getX()) +
                    Math.abs(nodoFinal.getY() -
                    adyacentes.elementAt(i).getY()))*10;
            adyacentes.elementAt(i).setH(h);

            //Calculamos el coste g y lo marcamos
            if( (actual.getX() ==
            adyacentes.elementAt(i).getX()) ||
            (actual.getY() ==
            adyacentes.elementAt(i).getY()) ) {
                    //Movimiento ortogonal
                    g = actual.getG() +
                            incrementoOrtogonal;

                    adyacentes.elementAt(i).setG(g);
            }
            else {
                    //Movimiento diagonal
                    g = actual.getG() +
                            incrementoDiagonal;

                    adyacentes.elementAt(i).setG(g);
            }

            //Marcamos el nodo actual como el padre de
            //este nodo
            adyacentes.elementAt(i).setPadre(actual);

            //Insertamos el nodo en la lista abierta

            listaAbierta.add(adyacentes.elementAt(i));

            //Reordenamos la lista abierta
            Collections.sort(listaAbierta);

        }
```

```
                        }

                }
                //Comprobamos si hemos alcanzado el nodo final
                if(listaAbierta.contieneId(idNodoFinal)) {
                        encontradoFinal = true;
                }

        }       //Cierra el while

        //Vamos avanzando de padre en padre creando el camino. Está al revés
        aux = listaAbierta.buscarNodo(idNodoFinal);
        while(aux.getPadre() != null) {
                ruta.add(aux);
                aux = aux.getPadre();
        }

        //Hemos terminado, vaciamos todo
        adyacentes.clear();
        listaAbierta.clear();

        //Devolvemos la ruta obtenida
        return ruta;


    }

    public boolean onCreateOptionsMenu(Menu menu) {
        //Alternativa 1
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.menu_nuevoprofesor, menu);
        return true;
    }

    public boolean onOptionsItemSelected(MenuItem item) {
            Intent intent = new Intent(RecogeQR.this, RecogeQR.class);
            Bundle bundle = new Bundle();
            bundle.putInt("MAPA", mapaDestino);
            bundle.putInt("NODO", idNodoDestino);
            intent.putExtras(bundle);
            intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(intent);
            return super.onOptionsItemSelected(item);
    }

}
```

```
package pfc.algoritmo;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class BDHelper extends SQLiteOpenHelper {

	private static final String DB_NAME = "DB";
	private static final int DB_VERSION = 1;

	String fechasBD = "CREATE TABLE Fechas (fechaCreate LONG, fechaInsert LONG)";

	public BDHelper(Context contexto) {
		super(contexto, DB_NAME, null, DB_VERSION);
		// TODO Auto-generated constructor stub
	}

	@Override
	public void onCreate(SQLiteDatabase db) {
		// TODO Auto-generated method stub
		db.execSQL(fechasBD);
	}

	@Override
	public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
		// TODO Auto-generated method stub

	}

}
```

```
package pfc.algoritmo;

import android.app.Activity;
import android.content.Context;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;

public class AccesoBD extends Activity {

	private SQLiteDatabase db;
	private BDHelper dbHelper;

	public AccesoBD(Context contexto) {
		dbHelper = new BDHelper(contexto);
	}

	public void abrirEscritura() throws SQLException {
		db = dbHelper.getWritableDatabase();
	}

	public void abrirLectura() throws SQLException {
		db = dbHelper.getReadableDatabase();
	}

	public void cerrar() {
		dbHelper.close();
	}

	public Cursor consultar(String query) throws SQLException {
		return db.rawQuery(query, null);
	}

	public void ejecutar(String query) {
		db.execSQL(query);
	}

	//Un método que nos diga si existe la tabla que nos pasan
	 public boolean existeTabla(String tableName) {
		if (tableName == null || db == null || !db.isOpen()) {
		    return false;
		}

		Cursor cursor = db.rawQuery("SELECT COUNT(*) FROM sqlite_master
		       WHERE type = ? AND name = ?", new String[] {"table", tableName});
		if (!cursor.moveToFirst()) {
		    return false;
		}

		int count = cursor.getInt(0);
		cursor.close();
		return count > 0;
	 }

	 public Cursor consultarFecha(String consulta) {
		Cursor c = db.rawQuery(consulta, null);
		return c;
	 }

}	//Cierra clase
```

```
package pfc.algoritmo;
import java.io.BufferedReader;
import java.io.IOException;
import java.util.Vector;

public class ListaNodos {

	private Nodo[] listaNodos = null;
	private Vector<Nodo> adyacentes = null;
	String array[] = null, numeros[] = null, coordenadas[] = null;
	String linea;
	int numNodos = 0, numLineas = 0;
	int id = 0, x = 0, y = 0;
	Nodo nodoAux = null, nodoBuscado = null;

	public ListaNodos(BufferedReader br) {

		try {
			br.mark(3072);
			// Abrimos fichero y obtenemos el número de líneas que contiene
			while (br.readLine() != null) {
				numLineas++;
			}

			//Abrimos fichero de nuevo. Creamos un array de tantos nodos
			//como indique el fichero
			br.reset();
			linea = br.readLine();
			numNodos = Integer.parseInt(linea);
			listaNodos = new Nodo[numNodos];

			// Rellenamos el array de nodos
			for (int i = 1; i < numLineas; i++) {
				array = br.readLine().split("-");
				numeros = array[0].split(" ");
				coordenadas = array[1].split(" ");
				x = Integer.parseInt(coordenadas[0]);
				y = Integer.parseInt(coordenadas[1]);
				listaNodos[i - 1] = new
					Nodo(Integer.parseInt(numeros[0]), x, y);
			}

			// Están todos los nodos creados, luego a la hora de crear la
			//lista de nodos adyacentes sólo habrá que buscar en el array
			br.reset();
			// Necesitamos avanzar a la segunda línea del fichero
			array = br.readLine().split("");
			// Recorremos y vamos añadiendo los adyacentes
			for (int i = 1; i < numLineas; i++) {
				array = br.readLine().split("-");
				numeros = array[0].split(" ");
				//Obtenemos el nodo cuyo índice es el primer //número de la
				//línea
				id = Integer.parseInt(numeros[0]);
				nodoAux = buscar(id);
				for (int j = 1; j < numeros.length; j++) {
					nodoBuscado = buscar(Integer.parseInt(numeros[j]));
					nodoAux.insertarAdyacente(nodoBuscado);
				}
			}

		} catch (IOException e) {
			e.printStackTrace();
		}

	}	//Constructor
```

```
	public boolean existe(int id) {
		boolean existe = false;
		int cuenta = 0;
		if (listaNodos[0] != null) { // Lista no vacía
			//Hay que averiguar cuantos elementos no nulos hay, para no
			//salirnos. Cuando este bucle termine sabremos cuántos elementos
			//recorrer
			for (int i = 0; i < listaNodos.length; i++) {
				if (listaNodos[i] == null) {
					cuenta = i;
					break;
				}
			}

			for (int i = 0; i < cuenta; i++) {
				//Cuando sabemos el número de elementos, podemos buscar sin
				//miedo
				if (listaNodos[i].getId() == id) {
					existe = true;
					break;
				}
			}
		}
		return existe;

	}

	public Nodo buscar(int id) {
		Nodo aux = null;
		for (int i = 0; i < listaNodos.length; i++) {
			if (listaNodos[i].getId() == id) {
				aux = listaNodos[i];
				break;
			}
		}
		return aux;
	}

	public void mostrar() {
		for(int i = 0; i < listaNodos.length; i++) {
			System.out.println("Nodo " + listaNodos[i].getId());
			adyacentes = listaNodos[i].getAdyacentes();
			System.out.print("\tAdyacentes: ");
			for(int j = 0; j < adyacentes.size(); j++) {
				if(adyacentes.elementAt(j).getId() != 0) {

					System.out.print(adyacentes.
						elementAt(j).getId() + " ");
				}
			}
			System.out.println();
			System.out.println("\tCoordenadas: " + listaNodos[i].getX()
				+ " " + listaNodos[i].getY());
		}
	}

}	//Cierra la clase
```

```
package pfc.algoritmo;
import java.util.LinkedList;

@SuppressWarnings("serial")
public class ListaEnlazada extends LinkedList<Nodo> {

	public ListaEnlazada() {
		super();
	}

	public void mostrar() {
		for(Nodo n : this) {
			System.out.print(n.getId() + " ");
		}
		System.out.println();
	}

	public void mostrarNoProcesados() {
		for(Nodo n : this) {
			if(!n.getProcesado())
				System.out.println(n.getId());
		}
	}

	//Este método nos dice si la lista contiene el id pasado
	public boolean contieneId(int id) {
		for(Nodo n : this) {
			if(n.getId() == id) {
				return true;
			}
		}
		return false;
	}

	public Nodo buscarNodo(int id) {
		Nodo nodo = null;
		for(Nodo n : this) {
			if(n.getId() == id) {
				nodo = n;
				break;
			}
		}
		return nodo;
	}

}
```

```
package pfc.algoritmo;

import java.util.Vector;

//Podemos utilizar Collections.sort() para ordenar la lista abierta.
//Como la lista enlazada se compone de objetos Nodo, tenemos que
//implementar la interfaz Comparable<Nodo>

public class Nodo implements Comparable<Nodo> {

	//Variables de instancia
	int id,f,g,h,x,y,mapa;
	boolean procesado;
	Vector<Nodo> adyacentes;
	Nodo padre;

	//Constructor
	public Nodo(int id, int x, int y) {
		this.id = id;
		procesado = false;
		adyacentes = new Vector<Nodo>(4,4);
		padre = null;
		this.x = x;
		this.y = y;
		f = 0;
		g = 0;
		h = 0;
	}

	public Nodo(int id) {
		this.id = id;
		adyacentes = new Vector<Nodo>(4,4);
	}

	//Métodos get/set

	public int getId() {
		return id;
	}

	public int getX() {
		return x;
	}

	public int getY() {
		return y;
	}

	public void setX(int x) {
		this.x = x;
	}

	public void setY(int y) {
		this.y = y;
	}

	public int getMapa() {
		return mapa;
	}

	public void setMapa(int mapa) {
		this.mapa = mapa;
	}

	public void setProcesado() {
		procesado = true;
	}
```

```
	public boolean getProcesado() {
		return procesado;
	}

	public int getF() {
		return f;
	}

	public int getG() {
		return g;
	}

	public void setG(int g) {	//En cuanto cambia g o h se actualiza f
		this.g = g;
		f = g + h;
	}

	public int getH() {
		return h;
	}

	public void setH(int h) {	//En cuanto cambia g o h se actualiza f
		this.h = h;
		f = h + g;
	}

	public Nodo getPadre() {
		return padre;
	}

	public void setPadre(Nodo n) {
		padre = n;
	}

	public Vector<Nodo> getAdyacentes() {
		return adyacentes;
	}


	public void insertarAdyacente(Nodo nodo) {
		adyacentes.add(nodo);
	}

	public void mostrarAdyacentes() {
		System.out.print("Adyacentes: ");
		for(int i = 0; i < adyacentes.size(); i++) {
			if(adyacentes.elementAt(i).getId() != 0) {
				System.out.print(adyacentes.elementAt(i).getId() + " ");
			}
		}
		System.out.println();
	}

	public void mostrarCoordenadas() {
		System.out.println("Coordenadas: " + x + " " + y);
	}

	public int compareTo(Nodo arg0) {
		if(this.f == arg0.getF())
			return 0;
		else if(this.f > arg0.getF())
			return 1;
		else
			return -1;
	}
}
```

```
package pfc.algoritmo;

import java.util.Vector;

public class Camino {

	//Este objeto modela las rutas que nos devuelve el algoritmo
	Vector<Nodo> ruta;

	public Camino(Vector<Nodo> camino) {
		ruta = camino;
	}

	public int getLongitud() {
		return ruta.size();
	}

	public Vector<Nodo> getRuta() {
		return ruta;
	}

}
```

```
package pfc.algoritmo;

public class Centro {

	//Variables de instancia
	int x, y;

	public Centro(int coordx, int coordy) {
		x = coordx;
		y = coordy;
	}

	public int getX() {
		return x;
	}

	public int getY() {
		return y;
	}

}
```

```
package pfc.algoritmo;

public class ExcepcionFichero extends Exception {

	private static final long serialVersionUID = 1L;

	public ExcepcionFichero(String msg) {
		super(msg);
	}

}
```

Fichero vista_conexion.xml

```
<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <Button
        android:id="@+id/BtnConectar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:text="@string/btnConectar" >
    </Button>

    <TextView
        android:id="@+id/etiqueta"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_above="@+id/BtnConectar"
        android:layout_alignParentLeft="true"
        android:text="@string/presentacion"
        android:layout_marginBottom="39dp" />

</RelativeLayout>
```

Fichero vista_seleccion.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/txtLocalizacion"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:text="@string/Seleccionar"
        android:textAppearance="?android:attr/textAppearanceMedium" >
    </TextView>

    <AutoCompleteTextView
        android:id="@+id/lstProfesores"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/txtLocalizacion"
        android:layout_centerHorizontal="true"
        android:completionThreshold="1" />

</RelativeLayout>
```

Fichero vista_recogida.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

        <ImageView
         android:id="@+id/imagenRuta"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:contentDescription="@string/mostrarRuta" />

</RelativeLayout>
```

Fichero menu_nuevoprofesor.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu
    xmlns:android="http://schemas.android.com/apk/res/android" >

    <item
        android:id="@+id/nuevoprofesor"
        android:title="@string/btnQR"
       android:icon="@drawable/magnifying_glass">
    </item>

</menu>
```

Fichero strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="hello">Hello World, ClienteActivity!</string>
    <string name="app_name">QrTracker3</string>
    <string name="btnConectar">Conectar</string>
    <string name="btnBorrar">Borrar</string>
    <string name="txtInicial">Texto inicial</string>
    <string name="presentacion">QrTracker v3\n\n\nBienvenido a Qr Tracker.\nEste
       programa te ayudará a llegar a tu destino.\nPulsa el botón
        Conectar. </string>
    <string name="btnLlamar">Llamar</string>
    <string name="btnDespacho">Ir en persona</string>
    <string name="btnOK">OK</string>
    <string name="Seleccionar">Seleccionar profesor o ubicación</string>
    <string name="btnQR">Escanear QR</string>
    <string name ="mostrarRuta">No se puede mostrar la imagen</string>
</resources>
```

Fichero AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="pfc.algoritmo"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="10" />

    <uses-permission
        android:name="android.permission.INTERNET">
   </uses-permission>
   <uses-permission
        android:name="android.permission.WRITE_EXTERNAL_STORAGE">
   </uses-permission>
   <uses-permission
        android:name="android.permission.CALL_PHONE">
   </uses-permission>
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".Conexion"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".Seleccion" >
        </activity>

        <activity
            android:name=".RecogeQR" >
        </activity>

        <activity
            android:name=".Muestra" >
        </activity>
    </application>

</manifest>
```