Memoria Técnica SITES

Víctor Manuel Arroyo Martín
Álvaro Beltrán Camacho
Sergio Cabezas González de Lara
Pedro Gallego López

Doble Grado de Ingeniería Informática y Matemáticas

Universidad de Granada
Escuela Técnica Superior de Ingeniería Informática y Telecomunicaciones

7 de febrero de 2021

Índice

1. Introducción

En una ciudad llena de accidentes geográficos como es Granada, los miradores se pueden encontrar en cada esquina y rincón de ésta con vistas espectaculares y sitios para relajarse en cualquier momento del día. De ahí nace Sites, una aplicación para descubrir Granada y sobre todo para descubrir los miradores de la ciudad.

Sites es una aplicación enfocada a los numerosos miradores de la ciudad de Granada, así como en las zonas en las que se divide la ciudad y monumentos simbólicos.
En esta memoria describiremos cómo hemos implementado un prototipo de la aplicación con las funcionalidades más importantes y básicas para dar una idea de a dónde puede llegar la aplicación con más tiempo y recursos. En primer lugar vamos a describir qué hace este prototipo y qué tecnologías tiene implementadas para la interacción con el usuario y el entorno, tras ello describiremos cómo la hemos implementado y daremos ejemplos de uso e interfaz que hemos añadido.

2. Descripción de la Aplicación

La aplicación se centra sobre todo en guiar al usuario a distintos miradores a través de un mapa o con el uso de la cámara.
Tiene incluidos un número de miradores y ofrecerá al usuario visitarlos. Está enfocada para que cualquier persona pueda, de forma intuitiva, elegir un mirador en concreto, de forma aleatoria, según el mapa o en función de la orientación en la que se quiera dirigir.

Para facilitar esto, la aplicación tiene varias funcionalidades como una lista de todos los miradores disponibles separados por las zonas en las que se encuentran. Para elegir por la dirección y la distancia se usará el modo visión en el que se abre la cámara y apuntando con el móvil en la dirección que deseemos se mostrarán los miradores que se encuentren ordenados en una lista según la distancia a ellos. Además, para elegir un mirador de forma aleatoria, bastará con agitar el móvil cuando nos encontremos en el menú principal de la aplicación.

Una vez elegido el mirador se nos abrirá su página de información, tenemos la opción de que la aplicación nos dirija a él pulsando un botón de cómo llegar. Hecho esto, se abrirá el modo visión con una flecha en la pantalla. Esta flecha guiará al usuario hasta su destino. Si en cualquier momento de su trayecto el usuario pasa cerca de un monumento y se encuentra con el modo visión abierto, la aplicación mostrará un botón de aviso. Pulsando en él, se accederá a una página de información detallada del mismo.

Otra utilidad del modo visión aparece cuando el usuario se encuentra en un mirador. Cuando la aplicación detecta esto por la ubicación, se activará una funcionalidad nueva para mostrarle al usuario qué zonas de Granada está viendo desde el mirador al apuntarlas con la cámara. En concreto aparece una etiqueta sobre la vista de la cámara con el nombre de la zona. Pulsando en la etiqueta se podrá acceder a la información de la zona que se está observando.

Para facilitar también la orientación del usuario y el cómo llegar, la app dispone de un mapa con su propia localización dada por una chincheta roja y la localización de los miradores puesta en chinchetas azules. Al pulsar en las chinchetas se abrirá una ventana, en las azules con el nombre y una pequeña descripción del mirador y en la roja se dará la ubicación actual del usuario. Si se pulsa la ventada de una de estas chinchetas de miradores se podrá ver información detallada de éste e ir hacia él en el modo visión.

Además para obtener información de los miradores podremos en cualquier momento acceder a la lista de miradores. También en el modo visión podremos obtener información de miradores y zonas con gestos y multitouch. Esto último lo explicaremos a continuación de forma más detallada.

La aplicación también contiene una ayuda donde se explica de forma detallada qué hace cada parte del menú principal y también incluye una sección de gestos donde se explica cómo hacer multitouch (dibujando un rayo) y los gestos del móvil en la aplicación (agitar y hacer un infinito con el móvil) y qué hacen éstos.

Por último, la aplicación tiene implementado un asistente (llamado SitesBot) por voz o texto que ayudará al usuario y le dará información de los miradores y zonas que hay en la aplicación.

2.1. Tecnologías usadas

2.1.1. GPS

Nuestra aplicación se basa en gran parte en el uso de este sensor. Cada cierto tiempo necesitamos medir la posición del usuario tanto para actualizar la chincheta roja en el mapa como para mostrar en el modo cámara una información u otra, esto es, si estamos en un mirador mostramos sólo las zonas a las que se está apuntando y si no, se muestra una lista de miradores a los que se apunta, así que el uso de este sensor es esencial en la app.

2.1.2. Acelerómetro

Hemos hecho uso de este sensor mediante los gestos del móvil. Hemos añadido dos gestos, uno que se puede hacer desde el menú principal al agitar el móvil simulando el lanzamiento de unos dados y muestra un mirador aleatorio de los disponibles en la app con la opción de ir a él. Esto ayuda a la decisión de elegir un mirador.
El otro gesto añadido a la aplicación se puede realizar en el modo visión dibujando un infinito con el teléfono. Lo que hace este gesto es mostrar información de la zona en la que se encuentra el usuario, haciendo así que pueda obtenerla en cualquier momento de su trayecto con sólo hacer el gesto.

2.1.3. Orientación

Usamos un sensor denominado orientación que nos aporta el ángulo en el que se encuentra el dispositivo mediante los sensores acelerómetro y geomagnético. Se para los gestos anteriormente descritos de la app pero también tiene uso en el modo visión cuando nos estamos dirigiendo a un mirador pues funciona de brújula y gracias a ella podemos orientar la flecha hacia donde se encuentra el mirador en todo momento. Además, se usa para identificar el campo de visión del usuario y poder saber qué está viendo.

2.1.4. Proximidad

Lo utilizamos en el mapa. Si desplazamos la mano por encima de la pantalla, se recargará la ubicación y se centrará el mapa en la posición del usuario.

2.1.5. Luminosidad

También en el modo mapa, si la luminosidad es pobre, el mapa pasará a modo oscuro.

2.1.6. Multitouch

Dibujando un rayo cuando nos encontramos en un mirador, podemos obtener la pantalla de información del mirador en el que nos encontramos.
El éxito al dibujarlo se mide con un umbral que lo hemos establecido para que no tenga que ser totalmente igual al registrado en la aplicación pero para que tampoco acepte cualquier dibujo.

2.2. Diseño

Al entrar a la aplicación lo primero que aparece es una pantalla de carga con el logo de la misma y tras él, un menú principal con las opciones que ofrece la aplicación. El menú se ha dispuesto de forma que las tres actividades principales (visión, mapa y lista de miradores) destaquen sobre las dos más secundarias (ayuda y asistente). Vamos a explicar el diseño que se ha seguido para cada actividad de la aplicación.

2.2.1. Diagrama de flujo

A continuación mostramos un diagrama de flujo con el que se explica de forma visual las conexiones entre cada actividad y cómo se acceden:

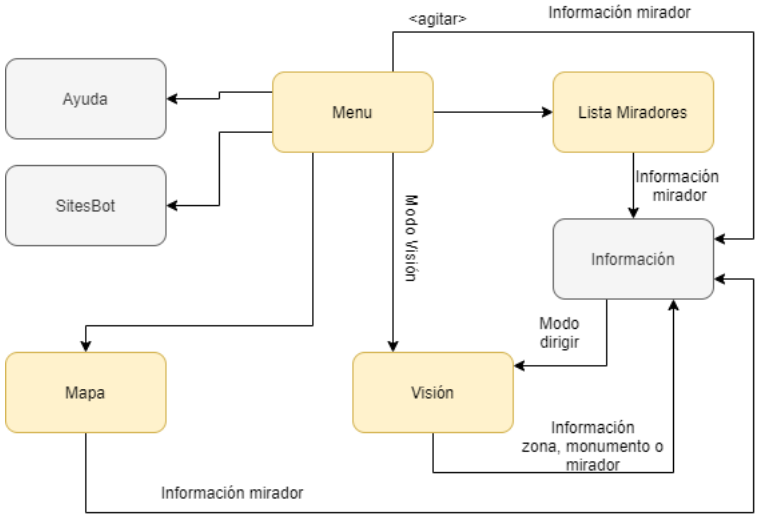

Figura 1: Diagrama de Flujo

2.2.2. Visión

El layout de Visión varía según el lugar en el que nos encontremos: si fuera o dentro de un mirador, pero en ambas la cámara está activa y el botón flotante de aviso de monumento se muestra en el mismo sitio, a la derecha de la pantalla.

Si nos encontramos fuera de un mirador, aparece una lista de los miradores a los que estamos apuntando en la parte inferior de la pantalla junto a una flecha para volver atrás. Los miradores que aparecen en la lista se pueden pulsar. También hay un texto en la parte superior de la pantalla indicando en qué zona de Granada nos encontramos.

En cambio, cuando nos encontramos en un mirador, el diseño cambia: ahora no aparece la lista inferior sino sólo el botón de volver atrás. Además, aparece una tarjeta flotante que muestra la zona más cercana a la que apuntamos con la cámara. Esta tarjeta se puede pulsar también.

Hay un último diseño en el modo Visión para cuando se dirige al usuario a un mirador en concreto. Es igual que cuando no nos encontramos en un mirador con la diferencia de que hay una flecha en el inferior de la pantalla indicando la dirección del mirador al que nos dirigimos. También en la lista de miradores inferior, se ancla en verde el mirador al que vamos.

Los Layouts quedan de la siguiente forma :

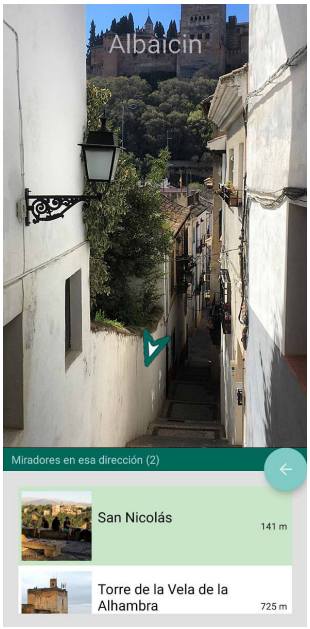

Figura 2: Visión dirigiéndose a un mirador

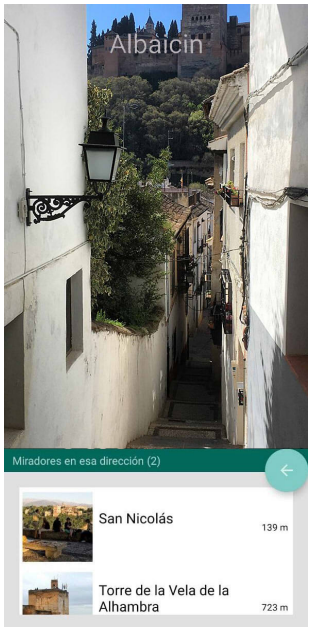

Figura 3: Visión fuera de mirador

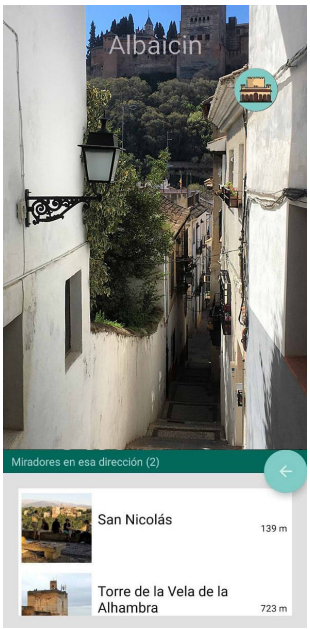

Figura 4: Visión cerca de un monumento

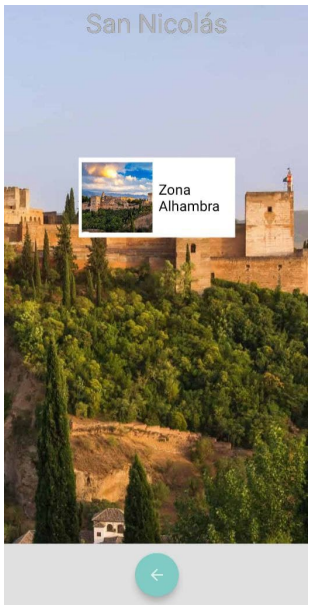

Figura 5: Visión en un mirador

2.2.3. Mapa

En este modo, el layout es sencillo, se trata de un mapa de Google Maps donde hemos añadido las ya mencionadas chinchetas para marcar miradores y la ubicación actual. Al pulsar una de ellas, aparece una ventana con la imagen del mirador en cuestión y una muy breve descripción junto a una indicación de que si pulsamos esta ventana nos lleva a la pantalla de información del mirador.

Figura 6: Mapa

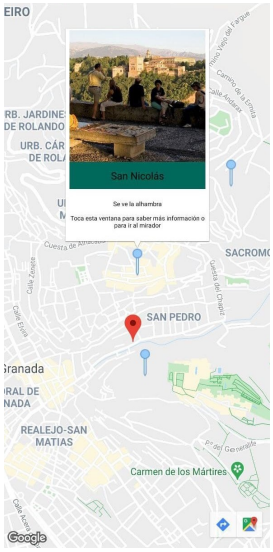

Figura 7: Mapa chinchéta Mirador

2.2.4. Lista de miradores

Aquí se muestran en forma de lista los miradores disponibles en la app. Están separados por colores según la zona en la que se encuentran y todos se pueden pulsar para acceder a la pantalla de información del mismo.

Figura 8: Lista Miradores

2.2.5. Pantalla de información

Se usa para miradores, zonas y monumentos. Se muestra una foto del lugar y a continuación mostramos un texto de información del sitio. También está incluido en la pantalla un botón de volver atrás, para poder salir de la pantalla de información. En el caso de que la pantalla que mostremos corresponda a un mirador, habrá un botón más, correspondiente a la opción de como llegar al mirador. Pulsando en él se abrirá el modo visión con la flecha.

Figura 9: Info Mirador

Figura 10: Info Monumento

2.2.6. Ayuda

Aquí se muestra una lista de tarjeta con los nombres e iconos de cada actividad del menú principal para hacer la búsqueda más intuitiva. Al pulsar en una de ellas se muestra la información correspondiente de esa actividad, para qué sirve y qué hace. También hemos añadido una con la información de los gestos.

Figura 11: Ayuda

3. Implementación

La implementación de la aplicación se hará en Android Studio usando el lenguaje de programación Kotlin.

Primero vamos a dar una idea visual de la implementación de la aplicación mediante el diagrama de clases, tras ello, pasaremos a explicar cada actividad y sus principales funciones y variables de clase.

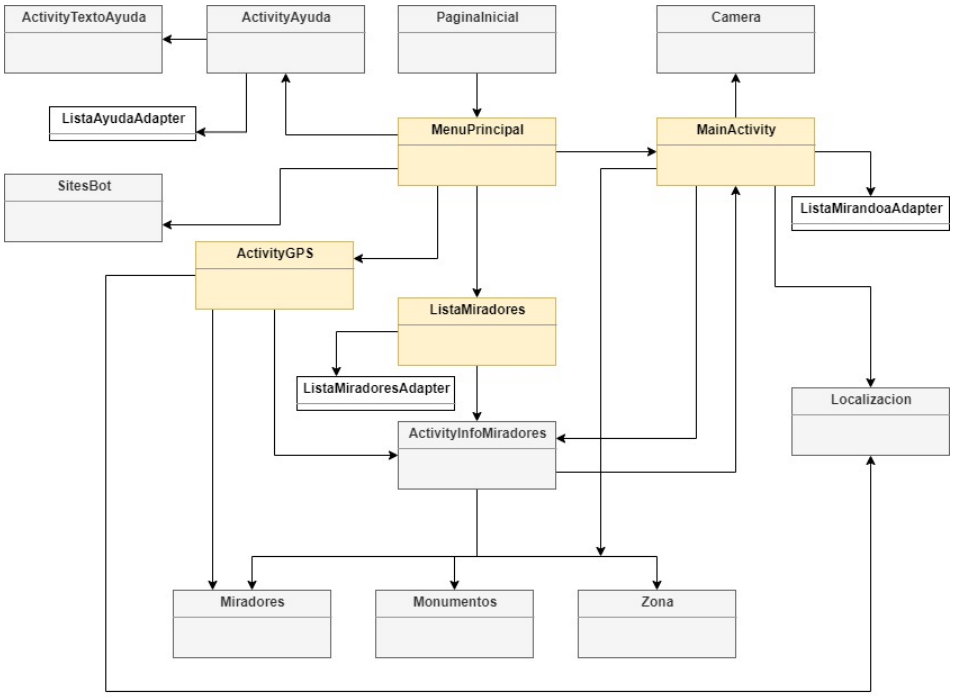

Figura 12: Diagrama Clases

3.1. Librerías usadas

- Bibliotecas para crear la acción del gesto en la pantalla. Son todas las bibliotecas que tienen como nombre Gesture:

```
import android.gesture.Gesture
import android.gesture.GestureLibraries
import android.gesture.GestureLibrary
import android.gesture.GestureOverlayView
```

- Bibliotecas para poder manipular el mapa de Google Maps:

```
import com.google.android.gms.maps.*
import com.google.android.gms.maps.model.BitmapDescriptorFactory
import com.google.android.gms.maps.model.LatLng
import com.google.android.gms.maps.model.MapStyleOptions
import com.google.android.gms.maps.model.MarkerOptions
```

- Bibliotecas para poder controlar la cámara:

```
import android.hardware.camera2.*
import android.graphics.SurfaceTexture
```

- Biblioteca Picasso, para modificar imágenes en tiempo de ejecución.

```
import com.squareup.picasso.Picasso
```

3.2. Clases

3.2.1. MenuPrincipal

El archivo MenuPrincipal.kt controla el menú de la aplicación y contiene una clase con su mismo nombre. Tiene varios atributos relacionados con la gestión de los gestos Android, una variable auxiliar `entrando` para evitar que la actividad que inician los gestos se inicie dos veces y otra más `accAnt` para la precisión de los gestos. Cambiamos la implementación de los métodos `finish` y `onPause` para evitar también el problema del inicio de dos actividades y en `onCreate` establecemos el layout. Pasemos ahora a explicar los principales métodos de la clase:

- **onResume**: En este método se establecen los ClickListener para cada tarjeta del menú principal y se inicia la correspondiente actividad cuando se pulsa sobre alguno de ellos. Para eso, hace falta establecer una variable botón que será la que use el listener para comprobar si se ha pulsado sobre él llamando a `setOnClickListener`.
- **setOnClickListener**: Método sobreescrito para gestionar el gesto de agitar el móvil. Si el gesto supera un cierto umbral, para que no se reconozca el gesto haciendo cualquier otro con el móvil, se entra a la pantalla de información de un mirador seleccionado de forma aleatoria. Para ello, se inicia con `Intent` la actividad correspondiente: `ActivityInfoMiradores`.

Esta clase extiende a `SensorEventListener` para gestionar los gestos, sobreescribiendo los métodos necesarios para ello.

3.2.2. MainActivity

MainActivity es la clase que representa la funcionalidad principal del programa, denominada visión. Esta clase implementa SensorEventListener y GestureOverlayView.OnGesturePerformedListener. La clase contiene los siguientes atributos:

- `gLibrary:GestureLibrary`. Se usa para que no se pulsen otras cosas mientras se hace el gesto.
- `cam:Camera`. Instancia de la clase Camera.
- `currentDegree`. Refleja los grados actuales de la brujula.
- `sensoreventlistener:SensorEventListener`. Referencia la clase actual
- `mSensorManager: SensorManager`. Para el uso de los sensores.
- `gyroscopeSensor:Sensor`. Almacena la instancia al sensor giroscopio.
- `accelerometer:Sensor`. Almacena la instancia al sensor acelerómetro.
- `MIN_TIME: Long`. Variable que almacena el tiempo de recarga de la ubicación.

- `locationManager:LocationManager`. Para el uso de la Localización.
- `longit, latit`. Posición actual del usuario.
- `latlonActualizadas:Boolean`. Para saber si se encuentra actualizada la posición.
- `enMirador:Boolean`. Aporta la información necesaria para saber si nos encontramos en un Mirador.
- `miradorElegido:String,posicionElegido:Int`. Mirador al que nos dirigimos.
- `degreeAnt:Float`. Grados anteriores a la medición actual.
- `gyroscopeAnt:Array<Float>`. Para calcular la diferencia en el movimiento del infinito entre el actual y el anterior.
- `compl:Array<Boolean>`. Para comprobar cada uno de los submovimientos que intervienen en el infinito.
- `mirador_destino:Int`. Posición en la matriz de miradores del mirador al que nos dirigimos.
- `locationListener:LocationListener`. Listener encargado de obtener la actualización de ubicación, en su OnSensorChange. Además de actualizar la ubicación también llama a `CalcularZona()`.
- `imview`. Foto de la brújula.

La clase contiene los siguientes métodos:

- **onCreate**. En este método se realizan las comprobaciones de permisos. Después de esto, inicializamos la cámara con `cam.onCreate()`, tras ello llamamos a `initData()` y a `gestureSetup()` para iniciar el listener de los gestos. Por último, se comprueba si al iniciar la actividad se le ha pasado el parámetro `POS` a través del intent. Si esto a ocurrido, significa que estamos en modo dirigir a y por tanto se llama a `ComoLlegar()`.
- **iniciarLocalizacion()**. Esta función se va a encargar de lanzar el listener de la localización. Dentro de ella comprobamos si tenemos permiso y si el gps como tal está activado. Después de esta función tendremos un listener de ubicación activo que recibirá actualizaciones cada `MIN_TIME` ms
- **gestureSetup()**. Primero se accede a la información del gesto, en este caso un rayo, en la carpeta que hemos creado llamada raw. En caso de encontrarse el archivo, se inicia el listener con gOverlay.addOnGesturePerformedListener. Además, se fija la visibilidad del gesto a invisible.
- **onGesturePerformed**. Método que es llamado cuando se realiza el gesto. Si nos encontramos en un mirador, empieza la actividad con la información de ese mirador. Si no nos encontramos en un mirador, aparece un toast en la pantalla indicando que no nos encontramos en un mirador.
- **onRequestPermissionsResult**. Función encargada de controlar los permisos de cámara y localización.
- **onResume**. Reabre la cámara y el listener del rayo. Además, reinicializa el atributo `compl` a false en todas sus posiciones y lanza listener de los sensores de orientación, giroscopio y acelerómetro.

- **onPause**. Cierra la cámara y establece a false todas las posiciones del atributo `compl`.
- **anguloLatLon**. Dadas dos latitudes y dos longitudes, calcula el ángulo que hay entre ellas.
- **onSensorChanged**. Este es el método que detecta los cambios en los sensores activados. En el método se diferencian dos funcionalidades, una para actualizar la brújula y otra para detectar el gesto del infinito.
  - **Brújula**. Si la diferencia en el ángulo de giro es suficiente (medido mediante un threshold) y estamos en el modo de dirigirnos a un mirador, se actualiza la brújula mediante una animación y se comprueba si ya hemos llegado al lugar de destino. Si no estamos en modo dirigirnos a un mirador, pero en cambio si ha habido una diferencia en el ángulo de giro, llamamos al método `mirandoHacia()` para actualizar lo que vemos.
  - **Gesto infinito**. Para implementar este gesto. Descomponemos el gesto del infinito en cuatro gestos más simples, dos de movimiento y dos de rotación, intercalados entre ellos. Comprobaremos los cambios producidos usando el acelerómetro y el giroscopio. Para aceptar que se ha producido un gesto simple, además de producirse, tenemos que comprobar que se han producido los anteriores. Esto lo vamos comprobando, ya que vamos poniendo a true los elementos del atributo `compl`, al ir realizando los gestos simples. Cuando el último elemento esté a true, es porque todos los gestos simples se han producido, y por tanto han dado lugar al gesto complejo.
- **initData**. Inicializa el sensor Manager y la foto de la brújula.
- **getDistanceFromLatLonInKm**. Dadas dos longitudes y dos latitudes, calcula la distancia en kilómetros de estos dos puntos.
- **deg2rad**. Pasa de grados a radianes.
- **comoLlegar**. Inicializa el atributo `mirador_destino` al mirador al que nos dirigimos.
- **destinoAlcanzado**. Establece el valor del atributo `mirador_destino` a -1.
- **mirandoHacia**. Calcula los puntos que son visibles desde nuestra posición. Para ello, obtenemos los datos de nuestra brújula y nuestra localización. Con nuestra localización y un punto de interés dado calculamos el ángulo que forman nuestras coordenadas (lat, lon). Con estas coordenadas se cálcula el ángulo con respecto al ecuador que está dado en el espacio $[-\pi,\pi]$ en sentido antihorario y esto contrasta con el formato de los ángulos que nos da la brújula (0 a 360 en sentido horario y desfasado $\pi/2$ respecto al ángulo de nuestras coordenadas). Para poder trabajar con los ángulos realizamos una serie de transformaciones a espacios equivalentes. Una vez realizado estas transformaciones, restando ambos ángulos sabremos cuando el dispositivo está apuntando al lugar de interés si el ángulo es nulo. Hay que establecer un intervalo de visión desde el cual el punto de interés es visible (si apuntamos en un ángulo dentro del intervalo definido, se nos mostrará); para esto hay que tener en cuenta la distancia a la que estamos de él, de forma que cuanto más lejos esté el punto, más pequeño será el intervalo. Un aspecto importante a tratar es en la discontinuidad dad entre π y $-\pi$, que hay que tenerla en cuanta para el buen uso de la función.

  Estos puntos detectados pueden ser miradores, zonas o monumentos. Cuando nos encontramos ante un monumento hacemos visible un botón que empieza la actividad de información del monumento encontrado. Además, diferenciando si nos encontramos en un

mirador o no y si estamos en modo dirigir a un mirador o no. Se añaden los miradores y las zonas a los ListView indicados mediante la clase ListaMirandoaAdapter, las instancias de estas listas son siempre clickeables comenzando la actividad de información del mirador o de la zona. Finalmente, se activan y desactivan de distinta forma botones, ListViews y Textviews dependiento del modo, como se aprecia visualmente en la sección 2.2.2 .

- **calcularZona**. En este método comprobamos en que zona estamos, una vez se sabe la zona se actualiza el texto de la actividad referente a esta información. Además, se inicializa el atributo `enMirador` verificando si nos encontramos a menos de 20m de las posiciones predefinidas para cada mirador.
- **calcularZonaIndex**. Devolvemos el índice de la zona en la que nos encontramos (-1 si está no está en una zona reconocible).
- **finish**. Hemos Reescrito la función finish (función que es llamada cuando cerramos la actividad) añadiendo una instrucción para eliminar el registro de los listener creados en esta actividad.

3.2.3. Camera

Esta clase controlará la interacción de nuestra aplicación con la cámara de nuestro dispositivo. La vamos a usar concretamente para obtener la vista de la cámara trasera para usarla en MainActivity.

Para ello se ha adaptado el código obtenido de Android Camera2 API Example [2] para el uso de la cámara.

Las adaptaciones han consistido en eliminar la parte de tomar fotos y de almacenamiento, de imágenes, y quedarnos únicamente con la vista de la cámara, pues estas funcionalidades no se necesitan en nuestra aplicación.

Además hemos ajustado la resolución de la vista de cámara para que la imagen no se distorsione dependiendo del la pantalla del dispositivo. Para ello creamos una nueva variable `imageDimension` y con la función `getOutputSizes` obtenemos el tamaño de la pantalla para ajustar la vista de cámara a ella sin distorsionar la imagen.

3.2.4. Adapters

Los adapters son usados para modificar la estética de las listas que tenemos en las demás actividades. Estas listas se implementan mediante un ListView. Para todas las clases adapter se ha usado la estructura definida por un tutorial para usar ListView [1] .Tenemos un adapter para cada lista con su propia estética:

- **ListaAyudaAdapter.**
- **ListaMiradoresAdapter.**
- **ListaMirandoaAdapter.**

Todas estas clases heredan de `BaseAdapter`. En la clase **ListaMirandoaAdapter** hemos realizado modificaciones con respecto al tutorial, pues se obtiene por parámetro miradores o zonas y se diferencian entre ambas opciones. Además, cabe la posibilidad mediante una variable booleana también pasada por parámetro de que el primer mirador que se pasa se pueda resaltar en verde claro.

3.2.5. ActivityGPS

Una de las clases principales y más complejas del código de la aplicación, extiende a `OnMapReadyCallback` que es un callback de interfaz para cuando el mapa está listo para usarse. Tenemos los siguientes atributos:

- **MIN_TIME**: Establece el tiempo en el que se actualiza la localización.
- **local**: De tipo Localizacion servirá como listener de la localización.
- **locationManager**: locationManager del tipo con su mismo nombre, da acceso al servicio del sistema de localización para obtener las actualizaciones periódicas y puntuales necesarias de localización.
- **lat**: Variable donde se guardará la latitud actual.
- **lon**: Longitud actual.
- **context**: Contexto usado para llamar a varias funciones. Lo establecemos al contexto actual, `this`.

El método `onRequestPermissionsResult` comprueba que se tengan los permisos de acceso a la localización del dispositivo. Veamos los métodos más importantes de esta clase:

- **onCreate**: Establece el layout de la actividad y comprueba que los permisos necesarios están garantizados. En caso de que no, los pide y si están ya concedidos, llama a la función `iniciarLocalizacion` para iniciar la localización. Por último muestra un toast para hacer una indicación de funcionalidad al usuario.
- **iniciarLocalizacion**: Inicializa los atributos `locationManager` y `local` y lanza el proveedor GPS. Tras ello, comprueba los permisos y establece el proveedor de localización.
- **onResume**: En este método hay que comprobar de nuevo los permisos e iniciar la localización. Se establecen e inicializan los sensores de tipo proximidad. En la función `onSensorChanged` que se llama cuando el sensor se activa (en este caso cuando hay algo cerca del dispositivo) se comprueban los permisos y si están concedidos, se recarga la ubicación haciendo uso de `requestLocationUpdates`. Por último se establece como listener el sensor de giroscopio ya nombrado.
- **onCreateView**: En esta función se crea el fragment para añadir el mapa a la aplicación.
- **onMapReady**: Función sobreescrita de `OnMapReadyCallback`, se le pasa una instancia de googleMaps. Lo primero que hace es establecer un nuevo sensor, esta vez de luminosidad para cambiar el mapa a modo nocturno si la luz está por debajo de un umbral y de nuevo se registra este sensor como listener. Con la instancia de googleMaps lo primero que se hace es mover la cámara a la posición actual con cierto zoom y se añade un marcador en ella, el marcador rojo.
  Ahora se pasa a añadir los marcadores de cada mirador con `addMarker` donde se añade la descripción y título correspondiente a cada uno haciendo uso de la variable de tipo Miradores. Tras ello, se añade la información que aparecerá en la ventana al pulsar en la chincheta con `getInfoContents` usando el layout de la ventana establece los textos e imagen que de nuevo están en la variable de tipo Mirador con un bucle pasando por todos los marcadores. Por último, establecemos el listener para pulsar sobre un marcador `setOnInfoWindowClickListener` que inicia la actividad de información del mirador sobre el que se ha pulsado con un intent de `ActivityInfoMiradores`.

3.2.6. Localizacion

Esta es una clase auxiliar a la clase `ActivityGPS` donde se generan distintos mensajes como cuando la localización está activa o no disponible en Log.

3.2.7. ListaMiradores

Mediante la clase ListaMiradoresAdapter se genera la lista de miradores ordenados por zonas y con un color de fondo característico a la zona que pertenece. Cuando hacemos click sobre una instancia de la lista creamos una instancia a la actividad de información del mirador seleccionado. Además, en esta clase se añade un botón con la función de volver a atrás.

3.2.8. ActivityInfoMiradores

Establece la ya mencionada página de información de miradores, así como de zonas o monumentos según el argumento que se le pase al llamar el intent.
El layout de los tres es parecido: se establece una imagen arriba y abajo un texto con la información referente al mirador/monumento/zona en cuestión y en los tres hay un botón con una flecha que sirve para volver atrás. Esto lo hace llamando a `finish` y terminando la actividad de forma que vuelve a donde estuviera antes.
El único cambio está si es un mirador. En este caso hay un botón que inicia la actividad correspondiente a dirigirse a y muestra un Toast indicando que se siga la dirección de la fecha que hay en esa actividad.

3.2.9. Ayuda

Esta clase maneja la pantalla de ayuda de la aplicación. Lo que se hace dentro de esta clase, se hace todo en el método `onCreate`.
Aquí se añade una lista de items en los que se puede pulsar para ir a la actividad de texto correspondiente `ActivityTextoAyuda`. Se hace con los Adapters ya explicados anteriormente y con un `listView` llamado `listaAyuda`.

3.2.10. ActivityTextoAyuda

En esta pequeña clase se implementa todo en la función `onCreate`, guardamos los textos correspondientes a la ayuda y según la elección que haya hecho el usuario en la pantalla anteriorontenida con la variable `position` se muestra un texto u otro. Con `nombreElegido` obtenemos el string del nombre de la opción de ayuda elegida para mostrarla como título en el layout.

3.2.11. PaginaInicial

Esta clase es la que se inicia primero en la aplicación. Se usa como añadido estético para enseñar el logo de la aplicación. Tras 2 segundos se pasa al MenuPrincipal.

3.3. Objetos

3.3.1. Zona

Se trata de un objeto que tiene como atributos de instancia a todas las zonas de Granada definidas en la aplicación. Estos atributos son arrays constantes de puntos cuyo identificador es el nombre de la zona, y donde cada punto contiene dos valores, latitud y longitud.

Concretamente cada uno de estos arrays contiene 4 puntos. Estos 4 puntos forman un cuadrilátero sobre el mapa de la ciudad de Granada que delimita cada zona. Al calcular las zonas

se ha tenido en cuenta que no haya ningún punto dentro de la ciudad de Granada de forma que quede entre dos zonas sin estar incluido en ninguna de ellas.

Todos estos atributos estarán incluidos en un nuevo atributo constante array denominado `arrayZonas`, y además creamos otro atributo `arrayNombres` con los nombres de las zonas, de forma que se haga corresponder en la posición i de cada array, el nombre de la zona y su atributo. Esto se hará para poder iterar sobre las zonas en las otras clases.

Además, hemos definido, para poder ofrecer información de cada zona, tres atributos constantes array más. Concretamente `descripcion`, `info` e `image`, que aportarán pequeñas descripciones, información completa e imagen de cada zona respectivamente. Estos arrays estarán ordenados de acuerdo a la ordenación dada para el vector de nombres y de zonas.

Para comprobar si nos encontramos dentro de una zona hemos definido los siguientes métodos:

- **onSegment.** Dados 3 puntos comprueba si uno de ellos está en el segmento formado por los otros dos.
- **orientation.** Dados 3 puntos devuelve el sentido de la orientación del triángulo formado por ellos (horario o antihorario) o si los 3 puntos son colineales.
- **getIndex.** Devuelve el índice de una determinada zona en los arrays descritos anteriormente a partir del nombre de la zona.
- **doIntersect.** Comprueba si dados dos segmentos (cada uno dado por una pareja de puntos del plano) se intersecan.
- **isInside.** Dado un punto y un polígono (vector de n puntos), comprueba si el punto está dentro del polígono. Usaremos como polígonos los cuadriláteros que definen las zonas.
- **puntoMedio.** Dado un índice para obtener un elemento del vector de zonas, devolveremos el punto de intersección de las diagonales del cuadrilátero correspondiente a la zona.

3.3.2. Miradores

Al igual que Zonas se trata de un objeto que tiene como atributos de instancia a todos los miradores de Granada definidos en la aplicación. Cada atributo es un punto cuyo identificador es el nombre del mirador, y que tiene dos valores latitud y longitud.

También se vuelven a tener los mismos atributos que en zonas, pero esta vez aplicando el caso para miradores. En concreto para iterar se tienen `arrayNombres` y `arraySitios`. Y para la información de los miradores `descripcion`, `info` e `image`.

Para el caso de miradores tenemos además, un nuevo array, denominado `zonas`, que contiene la zona en la que se encuentra cada mirador.

Además como métodos tenemos:

- **getIndex.** Devuelve el índice de un determinado mirador en los arrays descritos anteriormente a partir del nombre del mirador.
- **getCount.** Devuelve el número de miradores que tenemos incluidos en la aplicación.

3.3.3. Monumentos

Al igual que Zonas y Miradores se trata de un objeto que tiene como atributos de instancia a todos los monumentos definidos en la aplicación. Como en Miradores cada atributo es un punto cuyo identificador es el nombre del monumento, y que tiene dos valores latitud y longitud.

Los demás atributos y los métodos son los mismos que para el objeto Miradores, ya que en ambos casos estamos tratando con puntos del plano de Granada que se sitúan en una determinada zona, tienen imágenes e información.

Referencias

[1] Howard. *ListView Tutorial en kotlin*. https://www.raywenderlich.com/155-android-listview-tutorial-with-kotlin. 2020.

[2] Jiankai. *Android Camera2 API Example*. https://github.com/Jiankai-Sun/Android-Camera2-API-Example. 2019.