UNIVERSIDAD DE GRANADA

Nuevos Paradigmas de Interacción

grado en ingeniería informática

Práctica Sensores

Memoria Técnica

Autores

Vladislav Nikolov Vasilev

José María Sánchez Guerrero

Fernando Vallecillos Ruiz

Rama

Computación y Sistemas Inteligentes

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Curso 2019-2020

Índice

1. Introducción

En este proyecto vamos a explicar nuestra *Natural User Interface (NUI)* que hará que nuestra visita a la Alhambra sea más dinámica y productiva. El proyecto constaría de varios dispositivos, como pueden ser principalmente unas gafas de realidad aumentada, un dispositivo *Leap Motion* y un micrófono integrados en las gafas, y un *smartphone* que utilizaremos tanto para manejar el sistema gracias a los múltiples sensores que incorpora como para controlar la aplicación.

En esta primera versión del proyecto vamos a encargarnos de la interfaz por sensores, es decir, dejaremos a un lado el *Leap Motion* y el controlador por voz. Vamos a encargarnos de la realidad aumentada y el resto de sensores para manejar el sistema.

2. Interfaz por sensores

Como no disponemos de unas gafas con estas características, simularemos su funcionamiento en la propia aplicación, utilizando un visor VR con una imagen 3D del Patio de los Arrayanes y un lector de código QR. La idea es que estas dos características fuesen una sola, en la cámara de las gafas. Por otro lado, llevaremos el *smartphone* en la mano para controlar el contenido mostrado en las gafas, y además, mostrará una maqueta de la Alhambra que nos servirá como mapa.

2.1. Descripción del visor VR

Por defecto, la aplicación lanza el **MainActivity** [4.1], inicializando la aplicación y mostrando el **ViewARFragment** [4.5] (nuestro visor VR). El visor muestra una imagen de 360º, en la cual, cuando estemos mirando una estructura en concreto (ya sea un edificio, puertas, columnas, jarrones, etc.), se nos marcará indicando que podremos interaccionar con ella. Si movemos ligeramente el móvil hacia abajo cuando una de ellas esté resaltada, iniciamos la **InfoActivity** [4.7], que es una pequeña página que muestra información extra sobre la estructura, como si fuese una enciclopedia. En nuestra aplicación esto puede suponer un problema, ya que el visor lo tenemos en el móvil y al realizar el movimiento podemos deseleccionar lo que teníamos resaltado; sin embargo, como esto estaría integrado en las gafas, cuando movamos el móvil, las gafas (que es donde vemos el objeto) no se verán afectadas.

Una vez en el **InfoActivity**, para volver al visor VR, realizaremos un movimiento horizontal con nuestro dispositivo. En nuestra aplicación, todo esto aparece en el dispositivo, pero en la versión definitiva, toda la información se mostraría en las gafas.

2.2. Descripción del lector QR

En el menú desplegable, también tenemos la opción de *Camera*, que nos lleva al **CameraFragment** [4.3]. Este implementa el lector de códigos QR, el cuál nos servirá a la hora de entrar a un edificio. Habrá códigos QR en cada uno y se obtendrá un plano del edificio ya que las señales GPS pueden fallar o simplemente que el edificio tenga varias plantas y necesitemos ubicarnos mejor. Cuando leamos un código se iniciará un **BlueprintsActivity**, que muestra la información anterior. Al igual que antes, para volver a la cámara, realizaremos un movimiento horizontal con nuestro dispositivo.

2.3. Descripción del mapa

Por último, en el menú desplegable tenemos un apartado llamado *Navigation*, el cual inicia el **NavigationFragment** [4.4] que nos muestra un maquetado 3D de toda la Alhambra con regiones clickables señaladas para saber en que lugar nos encontramos o si queremos ir a algún otro lugar. Esta parte será la única que se vea en el *smartphone*. Tendremos implementados en él, el siguiente multitouch:

- **Click simple**. Servirá para seleccionar y deseleccionar las regiones clickables.
- **Click simple arrastable**. Nos permite desplazarnos por el mapa.
- **Doble click**. Hacemos zoom en el mapa un valor predeterminado.
- **Pellizcar**. Ampliamos y alejamos la zona del mapa, dependiendo de hacia donde pellizquemos.
- **Click con dos dedos y arrastre**. Entramos en el modo 3D del mapa. En él nos desplazaremos como acabamos de ver.
- **Click con dos dedos y rotación**. Estando en el modo 3D, esto nos servirá para rotar la vista alrededor de la maqueta.

Como podemos ver, utilizamos bastantes sensores multitouch, pero no hay ninguno que requiera del acelerómetro, giroscopio, etc., por lo que no tendremos colisiones entre éste y los mencionados anteriormente.

3. Diagrama de clases

A continuación se puede ver el diagrama de clases. No se han incluido atributos ni métodos porque se van a describir más adelante.

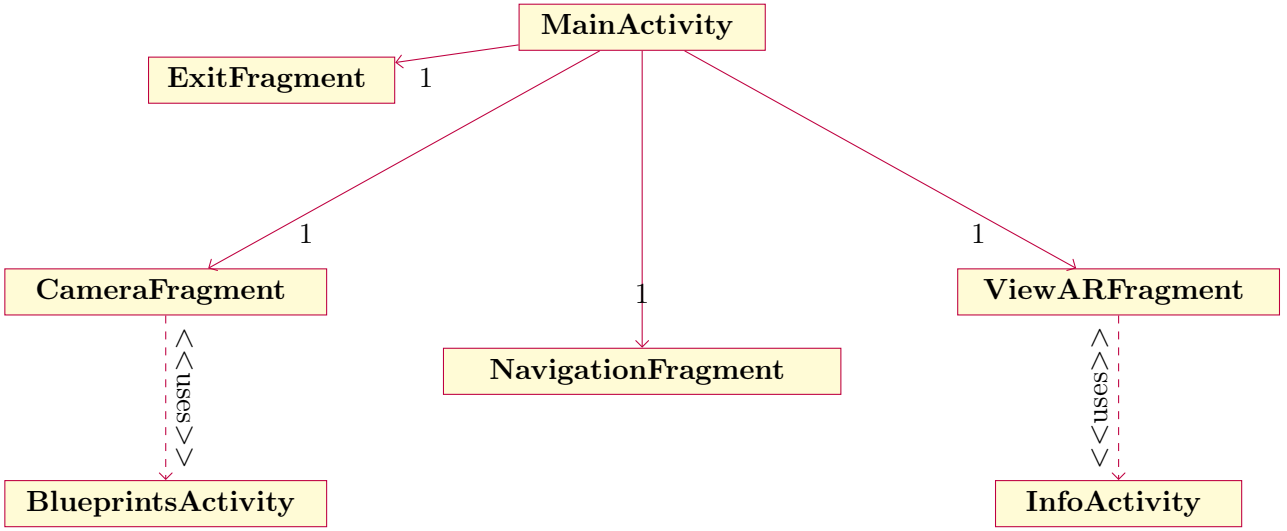

Figura 1: Diagrama de clases simplificado.

4. Descripción de las clases

Una vez visto el diagrama de clases, vamos a proceder a comentar brevemente qué es lo que hace cada clase, sus atributos, y sus métodos.

4.1. MainActivity

Es la actividad principal, encargada de construir la barra de actividades que permite acceder a los *fragments* y de solicitar los permisos de cámara y localización necesarios para ejecutar la aplicación.

4.1.1. Atributos

Los atributos que tiene esta clase son los siguientes:

- *mAppBarConfiguration*: Esta variable contiene la barra de navegación, con las distintas opciones que ofrece (navegación, VR, etc.).
- ***PERMISSION_ALL***: Código de permiso usado cuando se piden los permisos. Su valor da igual, pero tiene que ser mayor o igual a 0, según la documentación de Android [1]. En este caso, se le ha asignado el valor 1.
- ***PERMISSIONS***: Array con los permisos a pedir (localización y cámara).

4.1.2. Métodos

El siguiente método comprueba si se tienen los permisos necesarios (según los que se hayan especificado en ***PERMISSIONS***). Si están todos, devuelve **true**. En caso contrario, devuelve **false**.

```
1 private boolean hasPermisions() { ... }
```

El siguiente método inicializa la barra de navegación y solicita los permisos necesarios, si se tiene que hacer. Sobreescribe al método de la superclase. El parámetro serviría para restaurar información, de ser necesario.

```
1 protected void onCreate(Bundle savedInstanceState) { ... }
```

El siguiente método se llama al crear el menú (la barra de navegación, en este caso) la primera vez. Devuelve siempre **true**.

```
1 public boolean onCreateOptionsMenu(Menu menu) { return true; }
```

El siguiente método se llama cuando se despliega la barra de navegación, y devuelve **true** si se ha podido desplegar correctamente, y **false** en caso contrario. Sobreescribe al método de la superclase.

```
public boolean onSupportNavigateUp() { ... }
```

4.2. ExitFragment

Fragment que pregunta al usuario si quiere salir o no de la aplicación y, en caso afirmativo, la cierra.

4.2.1. Atributos

Esta clase no dispone de atributos, ya que no necesita guardar ningún tipo de información.

4.2.2. Métodos

El siguiente método es llamado cuando se crea la vista. Crea un cuadro de diálogo donde pregunta al usuario si quiere o no salir de la aplicación. En caso afirmativo, la cierra. En caso contrario, no hace nada y cierra el diálogo. Los atributos pasados son los típicos a la hora de crear una vista de un *fragment* [2].

```
public View onCreateView(@NonNull LayoutInflater inflater,
                         ViewGroup container,
                         Bundle savedInstanceState) { ... }
```

4.3. CameraFragment

Fragment que permite acceder al sensor de la cámara para poder realizar la lectura de códigos QR. Una vez leídos, se encarga también de pasarlos a **BlueprintActivity** para que éste los procese. Para la detección y lectura de los códigos QR se utiliza la biblioteca *BarcodeDetector*.

4.3.1. Atributos

La clase dispone de los siguientes atributos:

- *surfaceView*: Superficie donde se mostrará la cámara.
- *cameraSource*: Se encarga de recibir la información de la cámara y de mandársela al lector de códigos QR.
- *barcodeDetector*: Detector de códigos QR que se va a utilizar.

- *lecturaQR*: Contiene la información de lo que ha leído el lector de QR en formato String. Se utilizará para determinar que información mostrar en función del valor que tenga guardado.

4.3.2. Métodos

El siguiente método se llama al crear la vista donde se mostrará lo que vaya viendo la cámara. Se usa un **SurfaceView** para mostrar la imagen de manera más rápida. Los parámetros son los mismos que en el caso de **ExitFragment**. Para obtener más información sobre estos, se puede consultar la misma referencia que en ese caso.

```
public View onCreateView(@NonNull LayoutInflater inflater,
                         ViewGroup container,
                         Bundle savedInstanceState) { ... }
```

El siguiente método se llama cuando se pausa la actividad del *fragment*. Llama al método de la superclase.

```
public void onPause() { ... }
```

El siguiente método se llama cuando se reanuda la actividad del *fragment*. Llama al método de la superclase y a *activateCameraReader()* para activar la cámara.

```
public void onResume() { ... }
```

El siguiente método se llama cuando se destruye la actividad. Llama al método de la superclase.

```
public void onDestroy() { ... }
```

El siguiente método es llamado cuando se reanuda la actividad. Activa la cámara, establece cómo se tiene que manejar la información que le llegue a ésta utilizando para ello las variables *barcodeDetector* y *cameraSource* y crea una vista donde se podrá ver la imagen que esté procesando la cámara, utilizando para ello *surfaceView*. Establece además que cuando se detecte un código QR se pare la cámara, dejando de recibir información nueva, y que se lance una **BlueprintsActivity** para determinar qué mostrar en función del primer valor leído. Después de este procesamiento, se limpia la lista de QR leídos.

```
public void activateCameraReader() { ... }
```

4.4. NavigationFragment

Fragment que permite acceder a un mapa de la Alhambra y navegar por él, destacando algunos de los edificios y permitiendo interactuar con ellos, tal y como se ha descrito en la sección 2.3. Para poder trabajar con mapas, se utiliza la API de GoogleMaps.

4.4.1. Atributos

Los atributos de la clase son los siguientes:

- *googleApiClient*: Cliente de la API de Google.
- ***MIN_CAMBIO_DISTANCIA_PARA_UPDATES***: Mínima distancia que se debe haber recorrido para que se realice una actualización del marcador de posición actual. Se ha establecido que sean 10 metros.
- ***MIN_TIEMPO_ENTRE_UPDATES***: Mínimo tiempo que debe transcurrir para que se actualice el marcador de la posición actual. El tiempo viene dado en milisegundos. Se ha establecido que sean 5000 milisegundos, es decir, unos 5 segundos.
- *nMap*: Mapa de Google que se mostrará sobre el que se crearán las zonas destacadas y se mostrará la posición del dispositivo.
- *image*: Imagen que se mostrará al pulsar sobre una zona destacada.
- *locListener*: *Listener* que se usará para actualizar la posición actual del dispositivo en el mapa.

4.4.2. Métodos

El siguiente método se lanza al crear la vista. Establece una conexión con la API de Google para poder utilizar sus servicios más adelante. Además, carga la *ImageView* donde se mostrarán las imágenes de las zonas destacadas al pulsar sobre ellas, poniéndola invisible de momento.

```
1 public View onCreateView(@NonNull LayoutInflater inflater,
2                          ViewGroup container,
3                          Bundle savedInstanceState) { ... }
```

El siguiente método se lanza cuando el mapa está listo. Recibe como parámetro el mapa de Google, el cuál se asigna a *nMap*. Establece un estilo personalizado que utilizará el mapa, además de ponerle unos límites para que esté más o menos centrado en la Alhambra y para que no se pueda salir de éste, y establece un zoom inicial, el cuál será el máximo *zoom out* que se pueda hacer. Además, crea las zonas destacadas sobre el mapa, las pone visibles y establece qué imagen se debe mostrar al pulsar sobre cada una de ellas. Finalmente, establece cómo se debe realizar la actualización de la posición actual, utilizando para ello *locListener* y los atributos de distancia mínima y tiempo mínimo.

```
1 public void onMapReady(GoogleMap googleMap) { ... }
```

El siguiente método se lanza cuando se establece la conexión con la API de Google. No se hace nada.

```
1 public void onConnected(@Nullable Bundle bundle) { }
```

El siguiente método se lanza cuando se suspende la conexión con la API de Google. No se hace nada.

```
public void onConnectionSuspended(int i) { }
```

El siguiente método se lanza cuando falla la conexión con la API de Google. No se hace nada.

```
public void onConnectionFailed(@NonNull ConnectionResult connectionResult) { }
```

4.5. ViewARFragment

Fragment que simula la visión en realidad aumentada del interior de los Palacios Nazaríes. Simula que detecta zonas interesantes y las destaca. Cuando se realiza un gesto con el dispositivo, tal y como se ha indicado en la sección 2.1, muestra más información sobre el elemento destacado, utilizando para ello la clase **InfoActivity**.

4.5.1. Atributos

Los atributos de los que dispone la clase son los siguiente:

- *imagen*: Imagen a mostrar en cada momento.
- *mVRPanoramaView*: VR asociado a la imagen correspondiente.
- *sensorManager*: Gestor de sensores. Permite acceder a los distintos sensores, aunque nos interesa el acelerómetro.
- *mHeadRotation*: Rotación de la cabeza en grados.
- *lastUpdate*: Último instante en el que se ha actualizado el sensor.
- *last_y*: Último valor de Y registrado con el acelerómetro.
- ***SHAKE_THRESHOLD***: Umbral del acelerómetro. Se reconoce el gesto del *shake* cuando la aceleración del acelerómetro supera este umbral, el cuál es de 200.
- ***UPDATE_TIME_THRESHOLD***: Tiempo mínimo entre actualizaciones del acelerómetro en milisegundos. Es decir, no se van a tener en cuenta aquellas actualizaciones que se hagan con un margen de tiempo demasiado pequeño. Se ha establecido que sean 250 milisegundos.

4.5.2. Métodos

El siguiente método se llama cuando se crea la vista. Se carga el gesetor de sensores, la imagen inicial del VR y se obtiene la rotación inicial de la cabeza.

```
public View onCreateView(@NonNull LayoutInflater inflater,
                         ViewGroup container,
                         Bundle savedInstanceState) { ... }
```

El siguiente método se llama cuando se pausa la actividad. Se pasa la renderización del VR y se detiene el registro de los sensores.

```
public void onPause() { ... }
```

El siguiente método se llama cuando se reanuda la actividad. La renderización del VR continua y se comienza a registrar la actividad del acelerómetro.

```
public void onResume() { ... }
```

El siguiente método se llama cuando se destruye la actividad. Apaga la renderización del VR.

```
public void onDestroy() { ... }
```

El siguiente método carga la imagen actual en el VR. En caso de que se produzca algún error, se muestra información de depuración.

```
private void loadPhotoSphere() { ... }
```

El siguiente método se llama si se cambia la precisión de los sensores. Se le debe hacer un *override* ya que la clase implementa la interfaz **SensorEventListener**. No se realiza ninguna acción.

```
public void onAccuracyChanged(Sensor sensor, int accuracy) { ... }
```

El siguiente método se llama cuando se produce algún cambio en alguno de los sensores. Recibe como parámetro un evento provocado por algún sensor. Si el evento es provocado por el acelerómetro, ha pasado tiempo suficiente desde la última actualización y se ha superado la aceleración umbral, se lanza un *intent* con la clase **InfoActivity** para mostrar información sobre las regiones destacadas y se actualizan los valores de *lastUpdate* y *last_y*. En cualquier caso, se produzca el evento que se produzca, se actualiza la rotación de la cabeza y se mira si se tiene que actualizar la imagen que se está mostrando llamando para ello a *updateReticule()*.

```
public void onSensorChanged(SensorEvent sensorEvent) { ... }
```

El siguiente método actualiza la imagen que se está mostrando en el VR en función de la rotación de la cabeza que se ha obtenido en *onSensorChanged()*.

```
private void updateReticule() { ... }
```

4.6. BlueprintsActivity

Actividad que se encarga de recibir las lecturas QR de **CameraFragment** y de procesarlas, mostrando la información correspondiente a la lectura.

4.6.1. Atributos

Los atributos presentes en esta clase son los siguientes:

- *tituloPlano*: **TextView** del título del plano que se va a mostrar en la vista.
- *plantaPlano*: **TextView** con la información sobre la planta o localización.
- *imagenPlano*: **ImageView** con la imagen que se mostrará. En este caso, es una imagen de un plano con o sin una localización marcada.
- *sensorManager*: Gestor de sensores. Permite acceder a los distintos sensores, aunque nos interesa el acelerómetro.
- *senAccelerometer*: Acelerómetro.
- *lastUpdate*: Último instante en el que se ha actualizado el sensor.
- *last_x*: Último valor de X registrado con el acelerómetro.
- ***SHAKE_THRESHOLD***: Umbral del acelerómetro. Se reconoce el gesto del *shake* cuando la aceleración del acelerómetro supera este umbral, el cuál es de 200.
- ***UPDATE_TIME_THRESHOLD***: Tiempo mínimo entre actualizaciones del acelerómetro en milisegundos. Es decir, no se van a tener en cuenta aquellas actualizaciones que se hagan con un margen de tiempo demasiado pequeño. Se ha establecido que sean 200 milisegundos.

4.6.2. Métodos

El siguiente método se llama cuando se crea la actividad. Determina qué información debe mostrar en función de lo que indique la variable *lecturaQr* de la clase **CameraFragment**. Además de eso, inicializa el gestor de sensores y la detección del acelerómetro.

```
protected void onCreate(Bundle savedInstanceState) { ... }
```

El siguiente método se llama para mostrar la vista que contiene el título, el plano y la información sobre el plano. Recibe como entrada el nombre del archivo con la imagen del plano correspondiente, el texto del título y el texto de información.

```
private void showImageTextInfo(String fileName, String title, String
    information) { ... }
```

El siguiente método se llama cuando se produce algún cambio en alguno de los sensores. Recibe como parámetro un evento provocado por algún sensor. Si el evento es provocado por el acelerómetro, ha pasado tiempo suficiente desde la última actualización y se ha superado la aceleración umbral, se finaliza la tarea actual para volver a la que estábamos anteriormente y se actualizan los valores de *lastUpdate* y *last_x*.

```
public void onSensorChanged(SensorEvent sensorEvent) { ... }
```

El siguiente método se llama si se cambia la precisión de los sensores. Se le debe hacer un *override* ya que la clase implementa la interfaz **SensorEventListener**. No se realiza ninguna acción.

```
public void onAccuracyChanged(Sensor sensor, int accuracy) { ... }
```

4.7. InfoActivity

Actividad que muestra información sobre la zona que ha sido destacada en **ViewARFragment**. Procesa la información que tiene ésta y muestra una u otra información.

4.7.1. Atributos

Esta clase tiene los siguientes atributos:

- *tituloInfo*: **TextView** del título de lo que se quiere mostrar.
- *textInfo*: **TextView** con la información de lo que se quiere mostrar.
- *imageView*: **ImageView** con la imagen que se mostrará de alguna de las regiones destacadas.
- *sensorManager*: Gestor de sensores. Permite acceder a los distintos sensores, aunque nos interesa el acelerómetro.
- *senAccelerometer*: Acelerómetro.
- *lastUpdate*: Último instante en el que se ha actualizado el sensor.
- *last_x*: Último valor de X registrado con el acelerómetro.
- ***SHAKE_THRESHOLD***: Umbral del acelerómetro. Se reconoce el gesto del *shake* cuando la aceleración del acelerómetro supera este umbral, el cuál es de 200.
- ***UPDATE_TIME_THRESHOLD***: Tiempo mínimo entre actualizaciones del acelerómetro en milisegundos. Es decir, no se van a tener en cuenta aquellas actualizaciones que se hagan con un margen de tiempo demasiado pequeño. Se ha establecido que sean 200 milisegundos.

4.7.2. Métodos

El siguiente método se llama cuando se crea la actividad. Inicializa el gestor de sensores y la detección del acelerómetro y muestra la vista correspondiente en función del atributo *imagen* de la clase **ViewARFragment**.

```
protected void onCreate(Bundle savedInstanceState) { ... }
```

El siguiente método se llama cuando se produce algún cambio en alguno de los sensores. Recibe como parámetro un evento provocado por algún sensor. Si el evento es provocado por el acelerómetro, ha pasado tiempo suficiente desde la última actualización y se ha superado la aceleración umbral, se finaliza la tarea actual para volver a la que estábamos anteriormente y se actualizan los valores de *lastUpdate* y *last_x*.

```
public void onSensorChanged(SensorEvent sensorEvent) { ... }
```

El siguiente método se llama cuando se pausa la actividad. Se le indica al gestor de sensores que deje recibir información.

```
public void onPause() { ... }
```

El siguiente método se llama cuando se reanuda la actividad. Se vuelve a registrar el acelerómetro.

```
public void onResume() { ... }
```

El siguiente método se llama cuando se destruye la actividad. Llama al método de la superclase.

```
public void onDestroy() { ... }
```

El siguiente método se llama si se cambia la precisión de los sensores. Se le debe hacer un *override* ya que la clase implementa la interfaz **SensorEventListener**. No se realiza ninguna acción.

```
public void onAccuracyChanged(Sensor sensor, int accuracy) { ... }
```

Referencias

[1] Permisos en Android
https://developer.android.com/reference/androidx/core/app/ActivityCompat.html#requestPermissions(android.app.Activity,%20java.lang.String%5B%5D,%20int)

[2] *Fragment* en Android
https://developer.android.com/reference/android/app/Fragment#onCreateView(android.view.LayoutInflater,%20android.view.ViewGroup,%20android.os.Bundle)