21. Orientación del dispositivo

Android permite dos orientaciones de la pantalla: retrato (*portrait*) y apaisado (*landscape*). Como sabes, el fichero `activity_main.xml` dentro de `/res/layout/` especifica la interfaz de usuario de la actividad en modo retrato. Si creas una carpeta alternativa de nombre `/res/layout-land/` con una nueva versión de `activity_main.xml`, este fichero especificará la interfaz de la actividad en modo apaisado. Android cargará automáticamente el fichero adecuado dependiendo de la orientación.

Android recrea las actividades, es decir, las destruye y vuelve a crear, cuando tiene lugar un cambio de orientación, cuando se conecta un teclado o una pantalla externa, etc. Al volver a ejecutarse el método `onCreate()` es cuando se infla el fichero alternativo si es que se ha suministrado.

Para evitar que la actividad se recree después de una o varias de estas circunstancias tenemos que indicarlas en el atributo `android:configChanges` del elemento `activity` del fichero de manifiesto, concatenadas mediante el símbolo `|`. Por ejemplo, esto es lo que hay que hacer para evitar que la actividad se recree después de una rotación, evitando que se actualice la interfaz:

```
<activity name= ".MainActivity" android:configChanges="orientation|screenSize"/>
```

Para completar nuestro proyecto vamos a añadir el siguiente fichero de diseño para el modo apaisado:

`/res/values/layout-land/activity_main.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <RadioButton
        android:id="@+id/f2"
        android:layout_width="@dimen/button_side"
        android:layout_height="@dimen/button_side"
        android:layout_centerHorizontal="true"/>
    <RadioButton
        android:id="@+id/f1"
        android:layout_width="@dimen/button_side"
        android:layout_height="@dimen/button_side"
        android:layout_toLeftOf="@id/f2"/>
    ...
    <RadioButton
        android:id="@+id/f33"
        android:layout_width="@dimen/button_side"
        android:layout_height="@dimen/button_side"
        android:layout_alignBaseline="@id/f31"
        android:layout_toRightOf="@id/f32"/>
</RelativeLayout>
```

Como ves la única diferencia entre este archivo y el del modo retrato es el tamaño de los botones. Aquí utilizamos un recurso de dimensión definido en el archivo `dimens.xml`, de tal forma que quepa el tablero completo en una pantalla de tamaño normal:

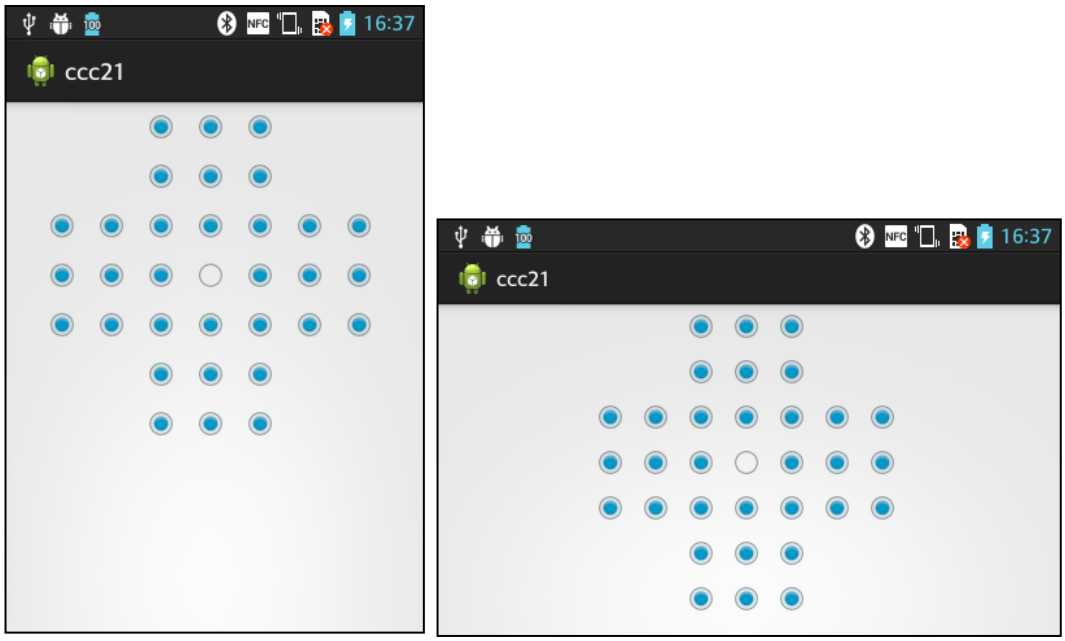
/res/values/dimens.xml

```
<resources>

    <!-- Default screen margins, per the Android Design guidelines. -->
    <dimen name="activity_horizontal_margin">16dp</dimen>
    <dimen name="activity_vertical_margin">16dp</dimen>
    <dimen name="button_side">35dp</dimen>

</resources>
```

A continuación puedes comparar el aspecto de la interfaz en modo retrato y apaisado en un teléfono LG OPTIMUS L5 E610:



Cuando una actividad se recrea, por ejemplo, después de un cambio de orientación, su estado actual puede perderse. Para asegurarnos de la conservación del estado, debemos sobrecargar los métodos `onPause()` u `onSaveInstanceState()`:

- Puedes sobrecargar `onPause()` para guardar los datos de tu actividad en una base de datos o en un fichero. La llamada al método `onPause()` está garantizada antes de la destrucción de la actividad, así que es el momento en el que debemos guardar datos necesarios para las siguientes ejecuciones de la aplicación, como un correo en borrador.
- Para evitar el uso de una base de datos, también puedes sobrecargar `onSaveInstanceState()`, que viene con un objeto `Bundle` como argumento y también se llama cuando una actividad está a punto de ser abortada. Sin embargo, al revés que `onPause()`, este método puede no ser llamado cuando la actividad se destruye. Por ejemplo, este método nunca se llama cuando el usuario pulsa el botón Back.

El objeto `Bundle` nos permite almacenar datos en pares clave-valor. Por ejemplo, el siguiente método almacena el miembro `grid` de `Game` en la cadena `GRID` mediante una llamada a `putString()`:

```
public void onSaveInstanceState (Bundle outState) {
    outState.putString("GRID", game.gridToString());
    super.onSaveInstanceState(outState);
}
```

Para ello hemos añadido el método `gridToString()` a la clase `Game`, que transforma el array `grid` en una cadena de caracteres:

```
public String gridToString (){
    String str = "";
    for (int i=0; i<SIZE; i++)
        for (int j=0; j<SIZE; j++)
            str += grid[i][j];
    return str;
}
```

Cuando la actividad se recrea, primero se ejecuta `onCreate()`, luego `onStart()` y luego `onRestoreInstanceState()` (repasa la unidad 19), al que se le pasa el objeto de tipo `Bundle` salvado por el método `onSaveInstanceState()`. Esto nos permite recuperar el estado del tablero guardado en la cadena `GRID`:

```
public void onRestoreInstanceState (Bundle savedInstanceState){
    super.onRestoreInstanceState(savedInstanceState);
    String grid = savedInstanceState.getString("GRID");
    game.stringToGrid(grid);
    setFigureFromGrid();
}
```

En este método utilizamos el método `stringToGrid()` añadido para este propósito a la clase `Game`:

```
public void stringToGrid (String str){
    for (int i=0, cont=0; i<SIZE; i++)
        for (int j=0; j<SIZE; j++)
            grid[i][j] = str.charAt(cont++)-'0';
}
```

De forma parecida podemos almacenar mas información como, por ejemplo, el estado del juego o alguno de los miembros necesarios en la implementación de la lógica. Puedes encontrar el código completo de `MainActivity` en el fichero del proyecto asociado a esta unidad: `CCC21`.