Programació multimèdia

Joan Climent Balaguer, Samir Kanaan

Programació multimèdia i dispositius mòbils

Índex

Introducció

Una de les característiques més notables dels dispositius mòbils actuals és la seva capacitat per reproduir continguts multimèdia de tot tipus (imatge, àudio, vídeo, animacions, etc.). Alhora, també són capaços d'obtenir tots aquests tipus d'informació multimèdia del seu entorn (fer fotografies, gravar àudio i vídeo, etc.). A més, aquests dispositius poden capturar una altra informació, com ara la seva posició geogràfica, la seva orientació, etc.

Totes aquestes característiques fan molt atractius els dispositius mòbils, i per aquest motiu els usuaris esperen que les aplicacions dels dispositius en facin ús per tal d'enriquir o fer més fàcil la seva utilització.

En aquesta unitat aprendreu a afegir les característiques multimèdia més importants a les vostres aplicacions, tant per reproduir continguts com per recollir informació multimèdia amb els dispositius Android.

En l'apartat "Programació multimèdia" veureu com es pot fer que les aplicacions mòbils reprodueixin imatges, àudio, animacions i vídeo fent servir els formats més populars i adients per a aquesta tasca. També veureu com fer que les vostres aplicacions obtinguin la posició geogràfica del dispositiu mitjançant GPS i la mostrin en un mapa. Finalment, s'explica com dibuixar formes geomètriques per permetre que les aplicacions generin dinàmicament els gràfics que elssiguin necessaris.

En l'apartat "Objectes multimèdia" veureu l'altra banda de les característiques multimèdia, és a dir, com es pot enregistrar informació multimèdia a les aplicacions mòbils. Concretament, es veurà com fer i desar fotografies amb la càmera del mòbil, com gravar àudio i com gravar vídeos. Tot això també implica saber accedir als continguts emmagatzemats a la memòria del dispositiu, per la qual cosa també hi ha una explicació de com llegir els continguts multimèdia desats a la targeta de memòria.

Per seguir els continguts d'aquest mòdul és convenient anar fent les activitats i els exercicis d'autoavaluació. Tot i que les unitats formatives tenen un contingut important des del punt de vista conceptual, sempre s'ha procurat donar-los un enfocament pràctic en les activitats proposades.

Resultats d'aprenentatge

1. En finalitzar aquesta unitat, l'alumne/a:

    - Analitza entorns de desenvolupament multimèdia.
    - Reconeix les classes que permeten la captura, processament i emmagatzematge de dades multimèdia.
    - Utilitza classes per a la conversió de dades multimèdia d'un format a un altre.
    - Utilitza classes per construir processadors per a la transformació de les fonts de dades multimèdia.
    - Utilitza classes per controlar esdeveniments, tipus de mèdia i excepcions, entre d'altres.
    - Utilitza classes per a la creació i el control d'animacions.
    - Utilitza classes per construir reproductors de continguts multimèdia.
    - Depura i documenta els programes desenvolupats.

1. Programació multimèdia

Gairebé qualsevol aplicació actual per a dispositius mòbils que vulgui ser atractiva per als usuaris, aprofitar el maquinari i donar la màxima informació amb una pantalla reduïda, ha d'incorporar informació en un o més sentits per a l'usuari: imatges, vídeo, so, etc. Els dispositius mòbils també porten un conjunt de sensors (d'orientació, de posició geogràfica, etc.) que faciliten molt la interacció dels usuaris i les aplicacions.

Per construir aplicacions mòbils valuoses cal saber emprar tots aquests elements, i a Android no és gaire complicat fer-ho.

1.1 Visualització d'imatges

Existeixen diferents tècniques disponibles per visualitzar imatges a les aplicacions Android. Començarem amb la tècnica més senzilla, per visualitzar una imatge fixa, i a continuació veurem com canviar la imatge dinàmicament, és a dir, com a resposta a una acció de l'usuari. Finalment, anirem més enllà i visualitzarem tot un conjunt d'imatges fent servir una galeria d'imatges, que ens permetrà navegar entre totes les imatges d'una col·lecció.

1.1.1 Imatges estàtiques

Per tal de fer servir imatges a les vostres aplicacions, cal que primer les afegiu com a recursos de l'aplicació. Però abans és convenient preparar-les per tal que després no us donin cap mena de problemes. Concretament, el que cal fer és el següent:

- Es recomana fer servir imatges en format PNG.
- El nom dels fitxers d'imatge només pot contenir lletres minúscules i números, sense espais, signes, lletres majúscules ni amb accent, ce trencada o d'altres caràcters no anglesos.
- Quan feu servir moltes imatges a una aplicació cal fixar-se bé en quina és la seva resolució (punts d'ample i d'alt). En general, és convenient que totes les imatges tinguin una mida semblant.

El format PNG

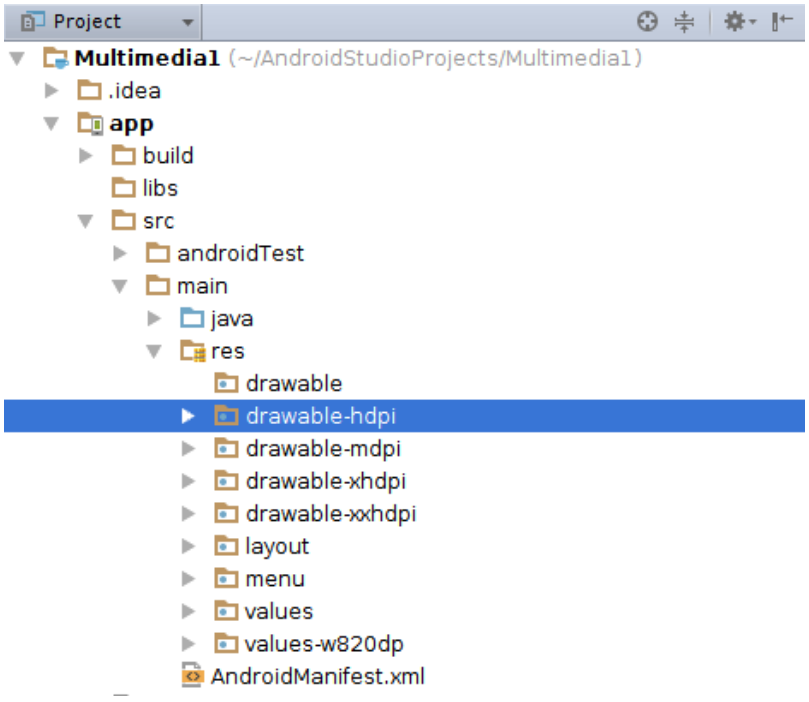
El format Portable Network Graphics (PNG) és un format d'imatge lliure (sense patents) i sense pèrdua de qualitat que, a més, permet transparència. Per aquestes raons és el format d'imatge recomanat per a les aplicacions Android, enfront d'altres formats populars com ara el JPEG.

Podeu descarregar el codi corresponent a aquesta activitat en l'annex anomenat "Imatges estàtiques" de la secció "Annexos".

Un cop preparada la imatge cal incorporar-la a la vostra aplicació. En primer lloc heu de crear un nou projecte Android (podeu anomenar-lo **multimedia1** i deixar

la resta per defecte). Tot seguit, canvieu la vista del projecte d'*Android* a *Project* i desplegueu la carpeta del vostre projecte. Accediu tot seguit a la subcarpeta */app/src/main/res* (*res* ve de *resources*, 'recursos', és a dir, les dades i fitxers addicionals com ara imatges o sons d'una aplicació) i, dins seu, a la subcarpeta *drawable-hdpi* (vegeu la figura 1.1).

FIGURA 1.1. Subcarpeta //drawable-hdpi// a la vista //project// de la nostra aplicació



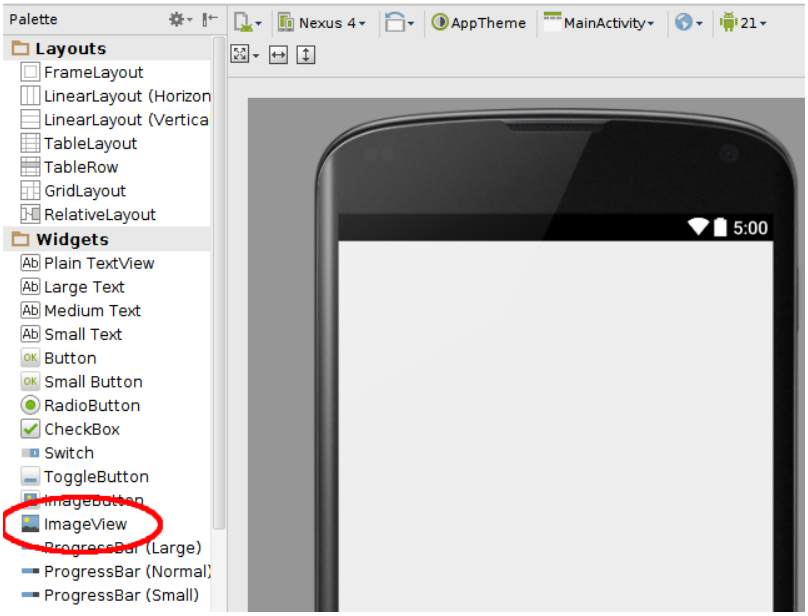
Quan hàgiu trobat aquesta subcarpeta, heu d'arrossegar-hi la imatge que voleu visualitzar per tal que passi a formar part de l'aplicació. Us apareixerà un diàleg per modificar el nom de la imatge i la seva ruta; cal que accepteu.

Resolució d'imatges

Com podeu observar, hi ha diverses subcarpetes el nom de les quals comença amb *drawable* ('*dibuixable*'). Per què? Una aplicació Android es pot executar en molts models diferents de dispositius, amb pantalles de mides i resolucions molt diferents. Quan creem les imatges per a les nostres aplicacions és recomanable crear-les per a resolucions diferents: molt altes, altes, mitjanes i baixes, per a dispositius amb aquestes resolucions. D'aquesta manera es garanteix que la visualització serà òptima en tots els casos. Precisament això representen les subcarpetes: imatges de baixa resolució (**ldpi**, *low dots per inch*, els punts per polsada de la imatge), mitjana (**mdpi**, *medium dots per inch*), alta (**hdpi**, *high dots per inch*), extra alta (**xhdpi**, *extra high dots per inch*) i extra extra alta (**xxhdpi**, *extra extra high dots per inch*). De moment només farem servir l'alta resolució, per no allargar els exemples.

D'aquesta manera, la imatge ja està incorporada a la vostra aplicació, però això no implica que sigui visible. La forma més senzilla de visualitzar una imatge en una aplicació Android és mitjançant el control `ImageView`, que es troba a la secció *widgets* de la paleta de l'editor del ***layout*** (obriu la carpeta de l'aplicació i aneu a **/res/layout/activity_main.xml**). A la figura 1.2 es pot veure aquest control.

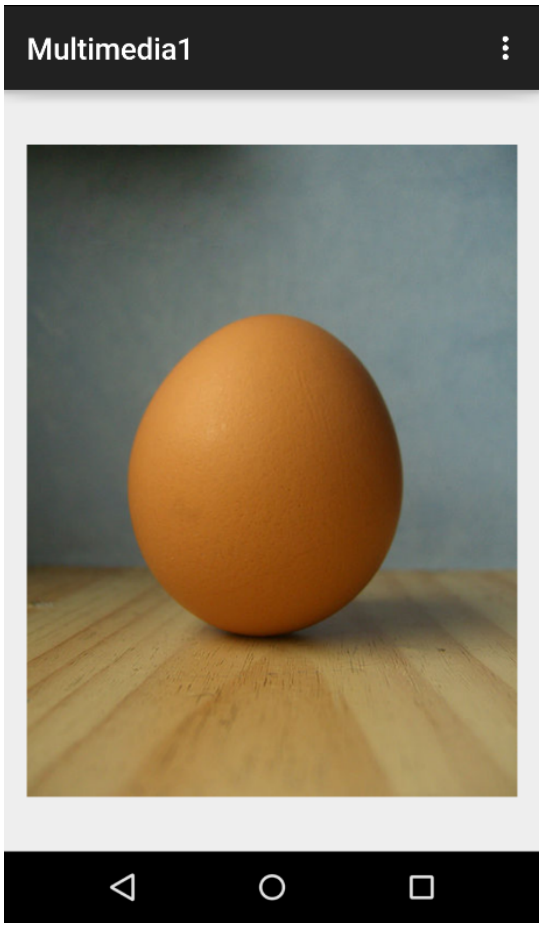
FIGURA 1.2. Localització del control "ImageView" a l'editor de layout



Afegiu l'`ImageView` a la vostra aplicació a la cantonada superior esquerra, l'`ImageView` no ocuparà tota la pantalla, per tal que ho faci haurem de fer clic als botons *set layout_width to match_parent* i *set layout_height to match_parent* o canvieu a l'XML les propietats `android:layout_width` i `android:layout_height` a `match_parent`. Fent doble-clic sobre l'`ImageView` apareixerà un diàleg amb la propietat *src*, seleccioneu la imatge que heu copiat abans des del directory `Drawable` de la pestanya `Project`.

Un cop fet això ja podeu executar el programa i gaudir de la vostra primera imatge, com es mostra a la figura 1.3.

FIGURA 1.3. Aplicació que visualitza una imatge



Descripció textual de les imatges

Si feu clic a la imatge o us posicioneu a `ImageView` de l'XML, veureu que s'hi mostra un avís (la icona d'una bombeta groga). Si feu clic a l'avís, veureu que aquest ens diu que falta la propietat `contentDescription` a la imatge. Aquesta propietat és una descripció textual de les imatges, necessària, per exemple, per a persones amb discapacitats visuals que fan servir assistents de veu.

Per acabar d'ajustar l'aplicació, cal que afegiu una cadena explicant el que hi ha a la imatge (al fitxer **strings.xml**), amb el nom `descripcio_imatge`, per exemple, i que torneu al **main.xml** i afegiu la línia del `contentDescription` (la darrera línia del següent codi) de manera que la part de l'**ImageView** quedi de la següent manera:

```
1 <ImageView
2         android:layout_width="match_parent"
3         android:layout_height="match_parent"
4         android:id="@+id/imageView"
5         android:layout_alignParentTop="true"
6         android:layout_alignParentLeft="true"
7         android:layout_alignParentStart="true"
8         android:src="@drawable/ou"
9         android:contentDescription="@string/descripcio_imatge"/>
```

1.1.2 Selecció dinàmica d'imatges

Acabeu de veure com mostrar una imatge seleccionant el fitxer d'imatge amb l'editor de *layouts* d'Android. Aquesta és la manera més senzilla i directa de mostrar una imatge a l'usuari. Però com es pot canviar aquesta imatge dinàmicament, és a dir, en resposta a una acció de l'usuari?

Podeu descarregar el codi corresponent a aquesta activitat en l'annex anomenat "Selecció dinàmica d'imatges" de la secció "Annexos".

En aquest exemple modificareu l'aplicació anterior per tal que la imatge visualitzada canviï quan l'usuari la toqui. Per tal que la nova aplicació tingui un nom diferent, aneu al fitxer strings.xml i canvieu la cadena `app_name`.

A continuació afegiu una nova imatge a la carpeta */res/drawable-hdpi*. De moment, si executeu l'aplicació només veureu la imatge inicial. Com podeu fer que el programa reaccioni i mostri una altra imatge quan la toqueu?

Obriu el codi del programa (fitxer **MainActivity.java**) per tal d'afegir el codi de resposta al clic a la imatge. En primer lloc, afegiu la modificació tal com segueix a la declaració de la classe per tal que pugui detectar clics:

```
1 public class MainActivity extends ActionBarActivity implements View.
      OnClickListener {
```

Heu d'afegir el següent a les importacions que fa Java per tal que pugui treballar amb `OnClickListener` i poder accedir a l'`ImageView`:

```
1 import android.view.View;
2 import android.widget.ImageView;
```

El primer que heu de fer és associar el clic damunt la imatge amb una resposta per part de l'aplicació afegint el següent codi al mètode `onCreate()`:

```
1 ImageView visorImatge =(ImageView) findViewById(R.id.imageView);
2         visorImatge.setOnClickListener(this);
```

A continuació cal crear el mètode de resposta al clic, que el que farà és accedir a l'**ImageView** i canviar el recurs al qual està associat:

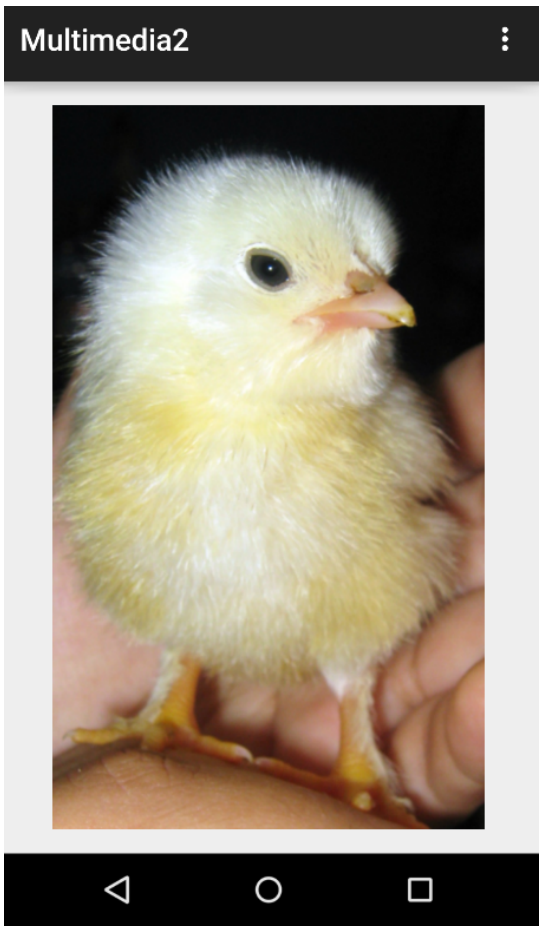
```
1 @Override
2 public void onClick(View v) {
3   // Canviem la imatge assignant-li l'altre recurs
4   ImageView visorImatge = (ImageView) findViewById(R.id.imageView);
5   visorImatge.setImageResource(R.drawable.pollastre);
6 }
```

Com podeu veure, per cada recurs que afegim al projecte es crea un identificador (en realitat, un enter) que ens permet identificar els recursos i treballar-hi fàcilment. En aquest cas és `R.drawable.pollet`.

Amb les cadenes de text i els controls del *layout* passa el mateix. Podeu mirar (però no canviar) els fitxers **R.java** que es troben a la carpeta */app/build/source/r/*, accessible des de la vista *project* de l'explorador del projecte.

Si proveu d'executar el programa, comprovareu com al començament es veu la imatge inicial, però quan la toqueu canvia per la segona imatge, com es veu a la figura 1.4.

FIGURA 1.4. Aplicació amb la imatge canviada



1.2 Animacions

Un aspecte fonamental per donar una aparença més dinàmica a les aplicacions amb fort contingut multimèdia són les animacions, que pel que fa al desenvolupament per a Android podem classificar en dos tipus: animacions per interpolació (*tween*) i animacions per fotogrames (*frames*). Veureu exemples de tots dos tipus.

Per cert, quan hagueu experimentat amb els dos tipus d'animació veureu que és possible combinar-los per crear efectes encara més especials.

1.2.1 Animacions per interpolació

En primer lloc definirem què vol dir això d'interpolació, per tal d'entendre com funcionen aquests tipus d'animacions.

De manera intuïtiva, podem definir **interpolació** com l'acció de trobar els valors que té una funció entre dos punts coneguts. Nosaltres fem interpolacions quan, per exemple, unim dos punts amb una línia, o tres punts amb una corba.

Tween en anglès vol dir el mateix que *between*, és a dir, entre (entre dos punts, per exemple). S'anomenen animacions *tween* perquè troben el que s'ha de fer entre dos punts determinats pel programador.

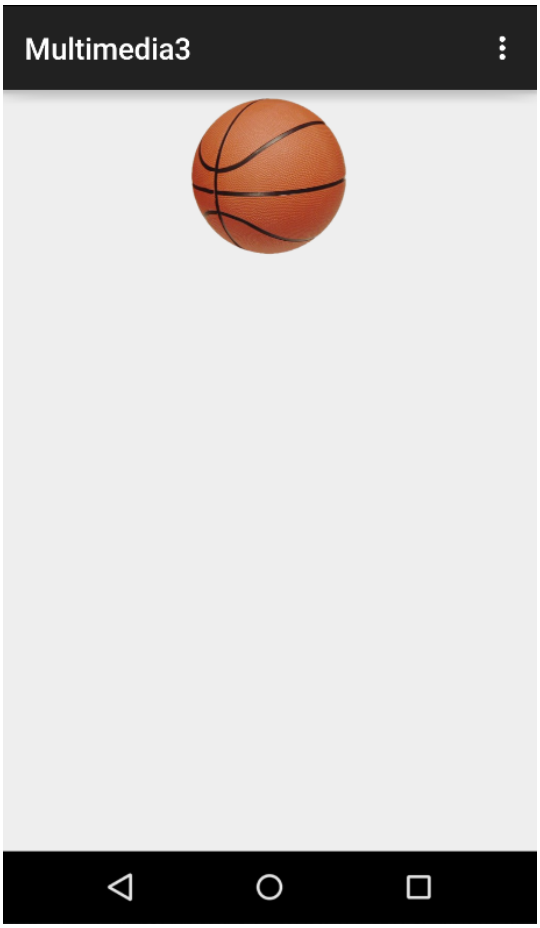
Per crear una animació per interpolació només cal que definiu l'estat inicial d'una imatge (de fet, es pot aplicar a qualsevol altra vista, com ara un text) i l'estat final desitjat, i quant ha de trigar la imatge en passar de l'estat inicial al final, i l'interpolador ja fa la resta. Els paràmetres que podeu canviar de la nostra imatge són l'**escala**, la **posició** i la **rotació**. És a dir, podem fer que la imatge vagi canviat de mida, que es vagi movent o que vagi girant. O que es produeixi més d'un efecte alhora, com veureu de seguida.

També es pot canviar la **transparència** de la imatge, cosa que permet fer que les imatges apareguin o desapareguin de forma gradual.

Podeu descarregar el codi corresponent a aquesta activitat en l'annex anomenat "Animacions per interpolació" de la secció "Annexos".

Per tal de poder experimentar amb les animacions, cal crear un nou projecte Android (l'anomenarem **Multimedia3**), afegint-li una imatge i un **ImageView** que ens la mostri. Us recomanem que feu servir una imatge en format **PNG** amb un motiu clar (una pilota, un cotxe, un animal...) i amb el fons transparent, de manera que no es vegi més que el motiu principal. En general serà més fàcil si es tracta d'un dibuix que no pas d'una foto. Centreu la imatge a la part superior del *layout*, com es veu a la figura 1.5.

Figura 1.5. Aplicació d'animacions amb la imatge inicial



Per tal d'apreciar millor les animacions, fareu que s'iniciin quan fem clic a la imatge. Cal afegir un mètode com el següent a la classe `MainActivity`, que omplireu més endavant amb les instruccions escaients:

```
1 public void onClick(View v){
2
3 }
```

I a continuació, heu de cercar la propietat de la imatge *onClick*, i escriure-hi **onClick** per tal que en fer clic a la imatge es cridi el mètode que acabem d'escriure. Això genera un atribut a la definició de l'element al **layout.xml** que fa que quan l'usuari cliqui l'**ImageView** es cridi el mètode `onClick()`:

```
1  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools" android:layout_width="
           match_parent"
3      android:layout_height="match_parent"  tools:context=".MainActivity">
4
5      <ImageView
6          android:id="@+id/imageView1"
7          android:layout_width="match_parent"
8          android:layout_height="116dp"
9          android:onClick="onClick"
10         android:src="@drawable/basketball"/>
11 </RelativeLayout>
```

Farem quatre animacions diferents:

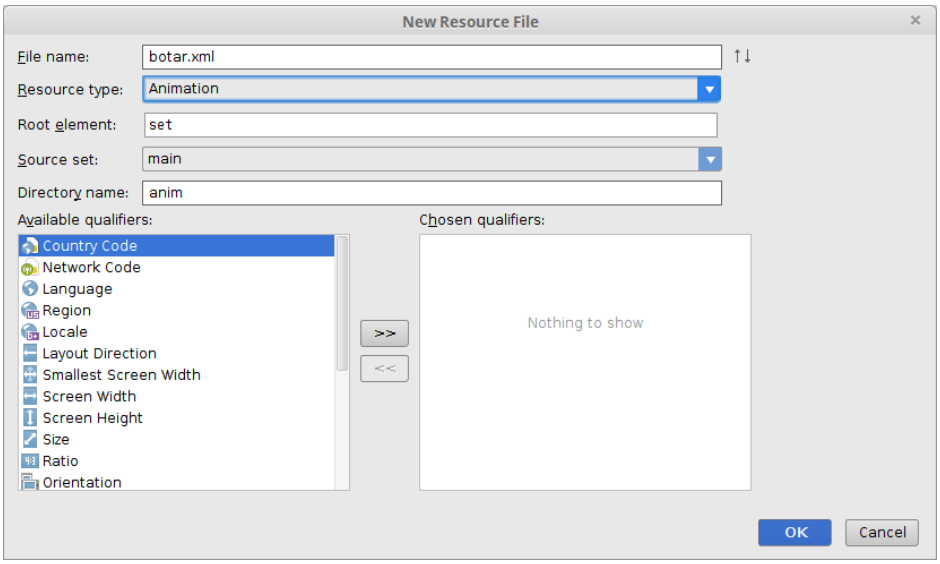
- Fer botar la pilota.
- Màgia: una pilota que apareix.
- Fer rodar la pilota.
- Fer venir de lluny la pilota.

Fer botar la pilota

La primera animació que fareu consisteix a fer botar la pilota. Per afegir una animació a la vostra aplicació, en primer lloc cal crear una nova carpeta anomenada *anim/* dins de la carpeta *res/* (pitjant el botó dret sobre *res/* i *New/Android resouce directory* i escollint de tipus *anim*). A continuació cal afegir un nou fitxer XML que contindrà les especificacions de l'animació. Pitgeu el botó dret sobre la carpeta *res/* i aneu a *New/New resource file*. Anomeneu-lo, per exemple, **botar.xml**. El tipus de recurs ha de ser **Animation**, i el **Root Element** pot ser ***set***, com es pot veure a la figura 1.6. De seguida veurem què volen dir aquestes opcions, de moment és suficient saber que són els diferents tipus d'animacions disponibles.

Recordeu: els noms de tots els fitxers (imatges, XML, etc.) que afegiu als vostres projectes han de ser en minúscules i sense espais ni signes (només el guió baix).

FIGURA 1.6. Afegint un fitxer d'animació



El contingut del fitxer **botar.xml** ha de ser el següent:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
  <translate
  android:interpolator="@android:anim/accelerate_interpolator"
  android:repeatCount="infinite"
  android:repeatMode="reverse"
  android:fromYDelta="0%p"
  android:toYDelta="80%p"
  android:duration="1000"/>
</set>
```

Veiem ràpidament què volen dir aquestes línies:

- `set` permet definir una animació composta de més d'una transformació. En aquest cas no s'aplica, però d'aquesta manera es pot ampliar fàcilment més endavant.
- `xmlns:android` defineix l'espai de noms **android** per tal que les línies següents puguin fer-lo servir com a prefix per a la resta d'opcions.
- `translate` indica que comença una transformació de posició, volem moure la imatge.
- `interpolator` és la funció que fareu servir per calcular els valors intermedis. Si és `linear` vol dir que els valors canvien a ritme constant, i si és `accelerate` vol dir que els valors canvien cada cop més ràpid. En aquest cas és el que ens interessa, perquè la pilota caigui cada cop més ràpid.
- `repeatCount`: defineix quantes vegades s'ha de repetir l'efecte. En aquest cas, volem que es repeteixi per sempre.
- `repeatMode` té dos possibles valors: `restart` faria que en caure la pilota aquesta tornés directament al començament, mentre que `reverse` faria que la pilota pugués fer el recorregut invers al que ha fet per caure.

- `fromYDelta`: estableix en quin punt (de la ***y***, és a dir, a la coordenada vertical) comença l'animació. Es pot posar un valor absolut en píxels ("140") , un valor relatiu a la vista ("40%") o al seu **parent**, en el *layout* de l'aplicació, indicant un valor tipus "50%p".
- `toYDelta`: estableix en quin punt de la coordenada vertical acaba l'animació. El format és el mateix que per `fromYDelta`.
- `duration`: estableix la durada de l'animació en mil·lisegons.

Per a les animacions de tipus `translate` també hi ha les opcions `fromXDelta` i `toXDelta`, equivalents als de la `Y` que acabem de veure.

Per tal que l'animació es visualitzi, cal carregar-la, associar-la a la imatge i posar-la en marxa. Aneu al mètode `onClick()` que heu escrit abans i afegiu el següent:

```
1 ImageView imatge = (ImageView)findViewById(R.id.imageView1);
2 Animation animacioPilota = AnimationUtils.loadAnimation(this, R.anim.botar);
3 imatge.startAnimation(animacioPilota);
```

Fixeu-vos que s'ha creat un identificador per a l'animació amb el nom del fitxer on l'heu definida.

Màgia: una pilota que apareix

Sovint fareu servir animacions per fer aparèixer imatges i d'altres elements de forma gradual, per tal de donar un aspecte més interessant a la vostra aplicació. Fer-ho és molt senzill, només cal que creeu un altre fitxer anomenat **res/anim/apareixer.xml** i que li doneu el següent contingut:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <set xmlns:android="http://schemas.android.com/apk/res/android">
3   <alpha
4     android:fromAlpha="0"
5     android:toAlpha="1"
6     android:duration="2000"
7   />
8 </set>
```

L'estructura és molt semblant a la del fitxer d'animació anterior, però en aquest cas ens trobem alguns elements nous:

- `alpha` es refereix al grau de transparència d'una imatge, o sigui que les animacions d'aquest tipus el que faran és canviar la transparència amb el temps.
- `fromAlpha` és el valor inicial de transparència; 0 vol dir transparent del tot, és a dir, invisible.
- `toAlpha` és el valor final de transparència; 1 vol dir opac del tot.

Una aplicació Android pot tenir més d'un fitxer d'animacions i seleccionar qualsevol d'ells.

Per tal que la vostra aplicació faci servir aquesta nova animació, cal editar el mètode `onClick()` i fer que l'ordre `loadAnimation` carregui `R.anim.apareixer`,

que és la nova animació. Proveu d'executar l'aplicació per notar l'efecte. Com veieu, en aquest cas l'animació no es repeteix si no torneu a clicar a la imatge.

Fer rodar la pilota

Si el que voleu veure és la pilota rodant d'una banda a una altra, cal que afegiu un nou fitxer d'animació anomenat **rodar.xml** amb el següent contingut:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
  <rotate
    android:interpolator="@android:anim/linear_interpolator"
    android:repeatCount="infinite"
    android:repeatMode="reverse"
    android:fromDegrees="0"
    android:toDegrees="360"
    android:pivotX="50%"
    android:pivotY="50%"
    android:duration="4000"/>

  <translate
    android:interpolator="@android:anim/linear_interpolator"
    android:repeatCount="infinite"
    android:repeatMode="reverse"
    android:fromXDelta="-50%p"
    android:toXDelta="50%p"
    android:duration="4000"/>
</set>
```

La novetat, en aquest cas, és que hi ha dues transformacions: la pilota gira (**rotate**) però també es desplaça. Per això cal escriure dues entrades. A la secció `rotate` hi ha algunes opcions noves:

L'ordre en el qual s'escriuen les diferents transformacions en un fitxer d'animacions és fonamental, perquè és l'ordre en el qual s'apliquen. Proveu a canviar d'ordre el `rotate` i el `translate` i comprovareu la diferència.

- `fromDegrees`: en quin angle (mesurat en graus) comença la rotació.
- `toDegrees`: en quin angle acaba la rotació. En aquest exemple acaba als 360° per tal que faci la volta completa. Si voleu que doni més voltes podeu posar un valor més gran que 360, per exemple 720 perquè doni dues voltes cada cop.
- `pivotX` i `pivotY`: indiquen el punt de la imatge respecte al qual aquesta girarà: no és el mateix girar respecte del centre, com en aquest cas, que girar respecte d'un costat.

Per cert, recordeu d'editar l'`onClick()` perquè carregui aquesta animació i no cap altra.

Fer venir de lluny la pilota

Finalment, creareu una animació que mostri la imatge venint de lluny i alhora girant. Aquesta animació quedaria molt maca amb la imatge d'una portada de diari, com a les pel·lícules antigues on s'acostumava a fer aquest efecte quan sortia alguna notícia als diaris.

Afegiu un altre fitxer d'animació que podeu anomenar **venir.xml** i escriviu el següent codi:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
  <rotate
    android:interpolator="@android:anim/linear_interpolator"
    android:duration="500"
    android:repeatCount="8"
    android:pivotX="40%"
    android:pivotY="40%"
    android:fromDegrees="0"
    android:toDegrees="360"/>

  <scale
    android:interpolator="@android:anim/linear_interpolator"
    android:repeatCount="0"
    android:duration="4000"
    android:pivotX="50%"
    android:pivotY="50%"
    android:fromXScale="0"
    android:toXScale="1"
    android:fromYScale="0"
    android:toYScale="1"/>
</set>
```

Reproducció d'animacions

Des de la versió d'Android Honeycomb (3.x) es pot afegir una opció a l'etiqueta `set` per tal de reproduir les operacions d'animació una darrera de l'altra i no totes alhora: `<set android:ordering = ''sequentially''>`.

Ara, al `rotate` hem canviat el punt de pivot per tal que la imatge giri una mica més i no només doni voltes sobre ella mateixa.

Pel que fa a la transformació d'escala (`scale`), els seus paràmetres són força evidents després d'haver treballat amb les altres transformacions. Bàsicament cal indicar l'escala inicial i final en X i Y.

Podem associar una mateixa animació a diversos visors alhora, per tal de donar un estil uniforme a la nostra aplicació.

1.2.2 Animacions per a fotogrames

Fotogrames a alta velocitat

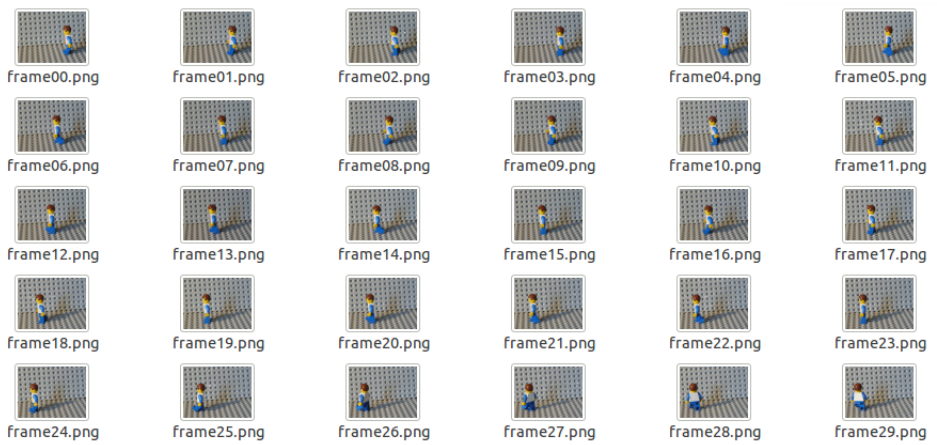
Cadascuna de les imatges de la seqüència rep el nom de fotograma, i al cap i a la fi el cine, la televisió i els videojocs, el que fan és mostrar fotogrames a alta velocitat (com a mínim, 25 per segon) per tal de donar-nos la impressió que estem veient moviment continu.

Podeu descarregar el codi corresponent a aquesta activitat en l'annex anomenat "Animacions per fotogrames" de la secció "Annexos".

Sovint, les transformacions que ens proporciona Android no ens serveixen per aconseguir l'animació que necessitem a la nostra aplicació i és necessari generar-les amb programes específics d'animació. En aquest punt tenim diferents possibilitats, com ara generar un vídeo o també, si l'animació no és massa llarga, podem generar un conjunt d'imatges amb els diferents passos de l'animació, de manera que passant ràpidament les imatges l'usuari tingui la sensació de presenciar moviment.

Comenceu un nou projecte Android (en l'exemple l'anomenarem Multimedia4). Trobeu un conjunt d'imatges que pugueu fer servir com a animació, com per exemple el que es mostra a la figura 1.7.

Figura 1.7. Conjunt d'imatges per crear una animació per fotogrames

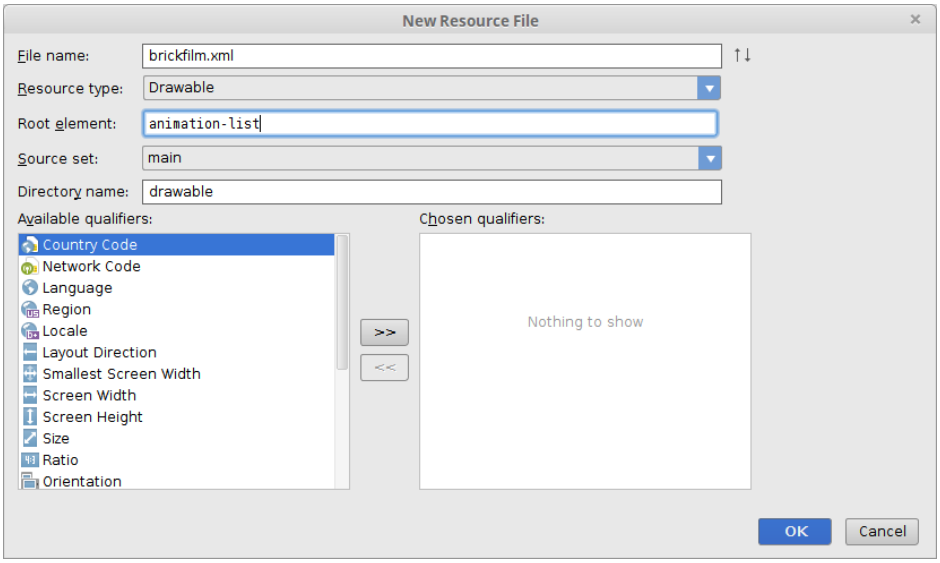


Amb vuit o deu imatges és suficient per apreciar l'efecte, no cal que feu servir moltes més imatges.

Tot i que l'efecte és més evident si les imatges formen part d'una seqüència, també podeu fer servir imatges independents.

En primer lloc, assegureu-vos que els fitxers d'imatge tenen un nom adient per afegir-los als projectes: noms en minúscula, sense espais i sense signes que no siguin guions baixos. A continuació, arrossegueu les imatges a la carpeta *res/drawable*. Tot seguit, afegiu-hi un nou fitxer XML de tipus (**Resource Type**) `drawable` i escriviu a **Root Element**: `animation-list`, que és una llista d'animació. Tal com mostra la figura 1.8, doneu-li un nom vàlid al fitxer i continueu.

Figura 1.8. Creació d'una llista d'animació



A continuació, editeu el fitxer XML per tal que quedi de la següent manera (en comptes de "frameXX", haureu d'escriure el nom de les imatges que heu afegit prèviament, sense l'extensió):

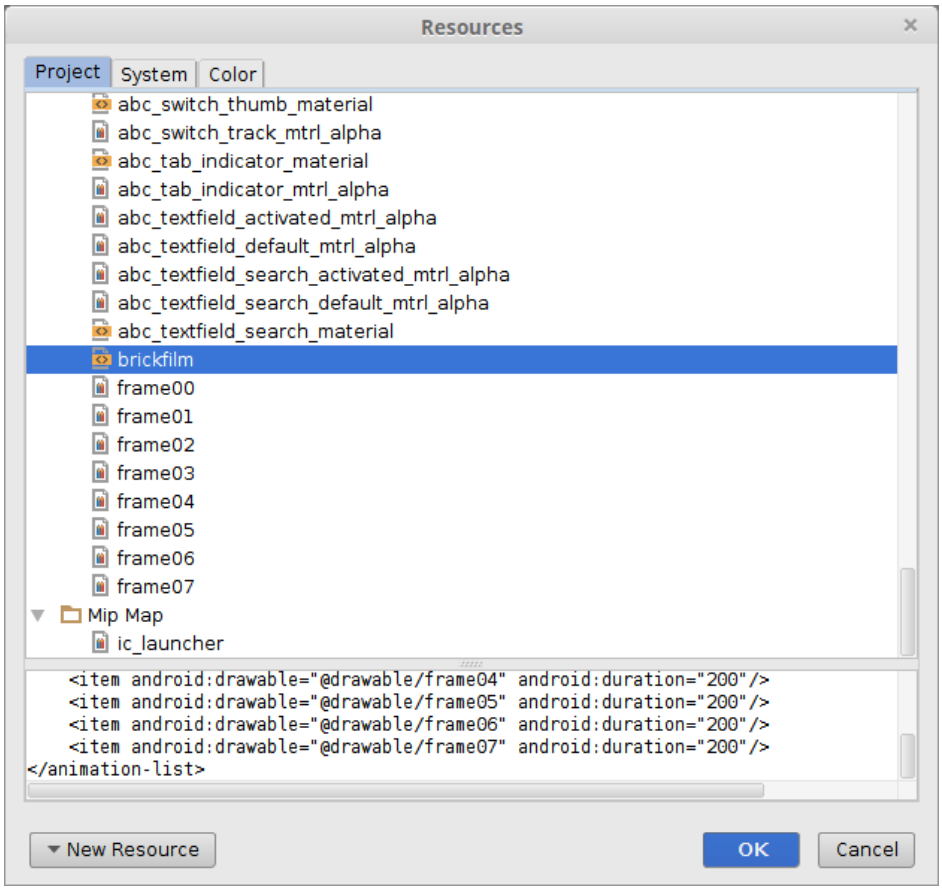
```
1 <?xml version="1.0"encoding="utf-8"?>
2 <animation-list xmlns:android="http://schemas.android.com/apk/res/android">
3   <item android:drawable="@drawable/frame00" android:duration="200"/>
4   <item android:drawable="@drawable/frame01" android:duration="200"/>
5   <item android:drawable="@drawable/frame02" android:duration="200"/>
```

```
  <item android:drawable="@drawable/frame03" android:duration="200"/>
  <item android:drawable="@drawable/frame04" android:duration="200"/>
  <item android:drawable="@drawable/frame05" android:duration="200"/>
  <item android:drawable="@drawable/frame06" android:duration="200"/>
  <item android:drawable="@drawable/frame07" android:duration="200"/>
</animation_list>
```

Amb aquest fitxer es defineix quina és la seqüència d'imatges que formaran l'animació i quina és la durada de cada fotograma, en mil·lisegons.

Tot seguit cal crear un **ImageView** al ***layout*** i, a la propietat *src*, associar-li l'animació que tot just heu creat (en aquest exemple s'anomena **brickfilm**), com es mostra a la figura 1.9.

FIGURA 1.9. Seleccionar l'animació per a fotogrames

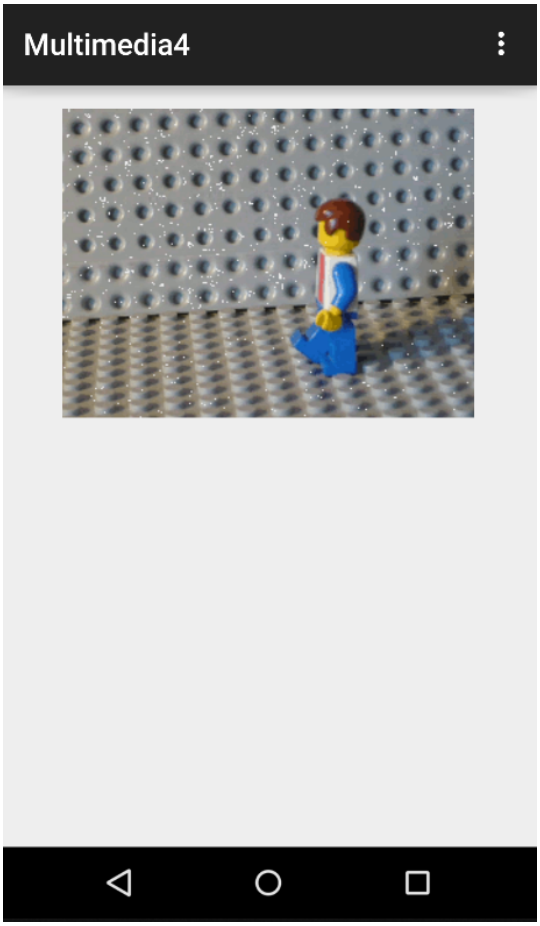


Si proveu d'executar l'aplicació només veureu el primer fotograma; cal dir-li a l'animació que s'executi. Per fer-ho, aneu al MainActivity.java i afegiu les següents instruccions al final del mètode `onCreate()`:

```
ImageView imatge = (ImageView) findViewById(R.id.imageView);
AnimationDrawable animacio = (AnimationDrawable) imatge.getDrawable();
animacio.start();
```

Comproveu que s'han afegit els *imports* i executeu l'aplicació. Bàsicament el que heu fet amb aquestes instruccions és accedir a la imatge, obtenir el que està dibuixant amb `getDrawable()` (en aquest cas, l'animació, perquè així li ho heu indicat), i dir-li a aquesta animació que comenci a executar-se. A la figura 1.10 podeu veure una captura de l'aplicació en funcionament.

FIGURA 1.10. Visualització de l'animació per a fotogrames



Com veieu, l'animació es repeteix de manera indefinida. Hi ha l'opció de dir-li que s'aturi en acabar, si voleu provar-la aneu al fitxer que defineix l'animació (**brickfilm.xml** en aquest exemple) i afegiu la següent opció a l'etiqueta `animation-list` per tal que quedi:

```
1 <animation_list xmlns:android="http://schemas.android.com/apk/res/android"
      android:oneshot="true">
```

Les animacions per fotogrames es poden aplicar tant a la *view* que estem fent servir com al seu fons (*background*). Penseu que, com que les imatges en format PNG poden tenir transparència, seria possible tenir una imatge de fons i una animació al damunt, o fins i tot una animació a sobre de l'altra. Això sí, cal ser una mica artista perquè estèticament això quedi bé!

Les animacions es poden aplicar a botons i d'altres controls. Això s'acostuma a fer servir a jocs i aplicacions semblants, per donar-los un aspecte més lúdic i dinàmic.

1.3 So i música

Tot seguit veureu com afegir arxius de so o música a les vostres aplicacions i com reproduir-los de manera més o menys sofisticada. També es poden reproduir els fitxers d'àudio que hi ha emmagatzemats al dispositiu, o fins i tot accedir a àudio *online*.

Els formats d'àudio suportats per Android són, principalment: mp3, ogg, flac i wav.

1.3.1 Reproducció bàsica

Si només voleu que la vostra aplicació reprodueixi una pista d'àudio quan s'engega, els passos a seguir són senzills.

En primer lloc s'ha d'afegir un arxiu d'àudio amb un format compatible amb Android i un nom de fitxer compatible amb Java. Cal crear una nova carpeta a *res/* que es digui *raw/* i arrossegar el fitxer d'àudio a aquesta carpeta.

Per escoltar àudio no cal afegir cap control al *layout*, creareu dinàmicament un **MediaPlayer** que és l'objecte encarregat de reproduir àudio i vídeo. Per això només cal afegir les següents línies al final de l'`onCreate()` de l'*activity* principal (`nompista` és el nom del fitxer d'àudio que hem afegit, sense l'extensió):

```
1 MediaPlayer so = MediaPlayer.create(this, R.raw.nompista);
2 so.start();
```

Podeu comprovar que en obrir l'aplicació es reprodueix l'arxiu de so. Això sí, sense opció a aturar-lo ni res de semblant. Hi ha molt a millorar!

1.3.2 Controls de reproducció

Tot i que és possible controlar el MediaPlayer amb botons i d'altres elements estàndard, hi ha un *widget* específic per a aquesta tasca que anomenat **MediaController**. Aquest tipus de control es pot afegir des de l'editor del *layout*, però és més senzill afegir-lo des del codi directament.

Podeu descarregar el codi corresponent a aquesta activitat en l'annex anomenat "Controls de reproducció de so" de la secció "Annexos".

Primerament, afegiu a l'activitat el següent:

```
1 public class MainActivity extends ActionBarActivity implements MediaController.
      MediaPlayerControl {
```

Amb aquest codi indiqueu que la vostra activitat durà a terme les funcions de control de so que ofereix MediaController. Cal afegir un ***import*** amb **android.widget.MediaController**. Aleshores, l'Android Studio indicarà un error a l'activitat, dient que ha de definir els mètodes abstractes de `MediaPlayerControl`. La manera més senzilla de resoldre-ho és situar-se sobre l'error i clicar a l'opció *Implement methods* de la bombeta d'error (o prémer `Alt + Enter`); aleshores afegirà automàticament deu mètodes (*canPause* fins *start*), que anireu fent servir per afegir funcionalitat a la vostra aplicació.

També cal anar al fitxer **main.xml** i assignar-li al **RelativeLayout** un ID, per exemple *vista_controls_so*, com podeu veure a continuació:

```
1 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2     xmlns:tools="http://schemas.android.com/tools" android:layout_width="
          match_parent"
3     android:layout_height="match_parent" android:paddingLeft="@dimen/
          activity_horizontal_margin"
```

```
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin" android:id="@+id/
        vista_controls_so" tools:context=".MainActivity">
```

Tot seguit, afegiu els següents atributs a la classe de l'activitat:

```
MediaPlayer so;
MediaController controls;
```

Tot seguit, aneu a l'`onCreate()`, elimineu les línies que vau afegir per a la primera versió del programa de reproducció de so i afegiu-hi les següents línies:

```
so= MediaPlayer.create(this, R.raw.bluestutorial);
controls = new MediaController(this);
controls.setMediaPlayer(this);
controls.setAnchorView(findViewById(R.id.vista_controls_so));
```

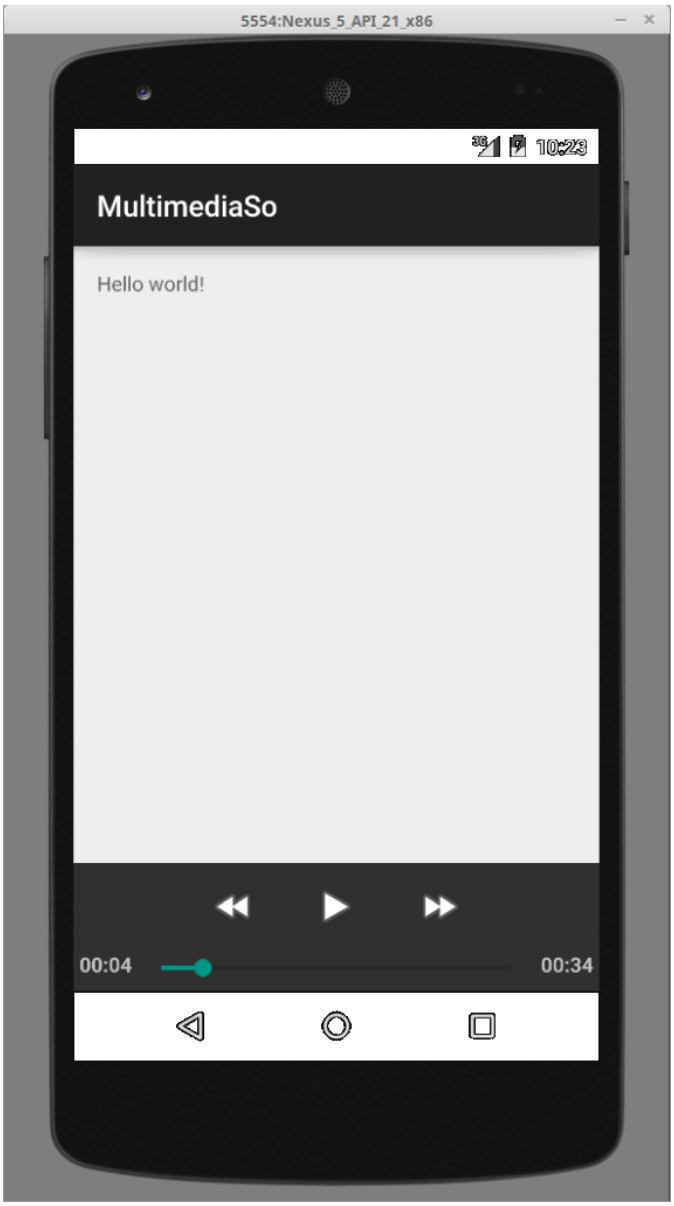
Amb aquestes instruccions, el que esteu fent és crear el *MediaPlayer* per reproduir la pista d'àudio, i després crear el *MediaController* per disposar d'uns controls per a l'àudio. A continuació associeu el codi del *MediaController* a l'activitat actual, i finalment definiu a quina vista de l'aplicació s'ha de dibuixar el *MediaController*.

Que encara no s'hi veu res? Això és perquè, per defecte, el *MediaController* s'amaga automàticament. Cal que l'obligueu a mostrar-se quan toqueu la pantalla, afegint el següent mètode a la vostra activitat:

```
@Override
public boolean onTouchEvent(MotionEvent event) {
  controls.show();
  return false;
}
```

Si al `show()` li passeu un nombre enter *n*, els controls d'àudio romandran visibles durant *n* segons; si li passeu un zero, els controls no s'amagaran. Ara sí, si executeu l'aplicació ja s'haurien de veure els controls d'àudio com es mostren a la figura 1.11.

FIGURA 1.11. Controls d'àudio del Mediaplayer



De moment, aquests controls no fan gaire cosa perquè heu d'omplir els seus mètodes amb codi que faci el que vosaltres voleu. Comenceu pel més bàsic: fer que el botó de reproduir (*play*) engegui la música. Per això cal anar al mètode *start* (creat automàticament per implementar `MediaController.MediaPlayerControl`) i afegir:

```
1 so.start();
```

Amb la qual cosa li dieu al reproductor d'àudio que comenci a reproduir la pista d'àudio. També s'ha d'anar al mètode `canPause()` i fer que torni `true` per tal que el botó de reproduir sigui actiu. Amb això ja s'hauria d'escoltar quan toqueu el *play*. Comproveu-ho.

Per tal que la interacció entre els controls i el reproductor d'àudio sigui més completa, cal anar omplint els mètodes que s'han generat per tal que quedin com es veu a continuació:

```
@Override
public int getCurrentPosition() {
  // Torna la posició actual a la pista d'àudio
  return so.getCurrentPosition();
}

@Override
public int getDuration() {
  // Torna la durada de la pista d'àudio
  return so.getDuration();
}

@Override
public boolean isPlaying() {
  // Torna cert quan s'està reproduint àudio
  return so.isPlaying();
}

@Override
public void pause() {
  // Quan l'usuari toca el botó de pausa
  so.pause();
}

@Override
public void seekTo(int pos) {
  // Per anar a una posició de la pista
  so.seekTo(pos);
}
```

La durada de les pistes multimèdia i les posicions es mesuren en mil·lisegons.

La resta de mètodes es poden deixar tal com estan ara mateix. Amb això ja tindreu un reproductor d'àudio amb funcions de reproducció, pausa i cerca d'una posició.

1.4 Vídeo

Reproduir vídeos als dispositius Android és molt senzill, com veureu en aquest apartat. Per començar, afegireu els vídeos com a recursos a la vostra aplicació d'exemple. També es poden reproduir vídeos d'Internet o emmagatzemats al dispositiu.

Els formats de vídeo més àmpliament suportats pels dispositius Android són l'MPEG-4 (.mp4) i el 3GPP (.3gp).

Per tal de visualitzar un vídeo a les vostres aplicacions, cal que afegiu el següent element a la nostra aplicació: dins de la pestanya, a l'editor gràfic de *layouts Images & Media*, afegiu un **VideoView**, que serà la pantalla amb la qual es veurà el vídeo.

Podeu descarregar el codi corresponent a aquesta activitat en l'annex anomenat "Reproducció de vídeo" de la secció "Annexos".

Per afegir un vídeo com a recurs, heu de crear una carpeta anomenada *raw/* dins de la carpeta *res/* i arrossegar-hi un vídeo amb el format adequat i amb un nom compatible amb Java, com és habitual. Al vostre exemple, aquest vídeo s'anomenarà **castell.3gp**.

A continuació cal anar al codi de l'activitat i afegir-hi el següent:

```
VideoView visor = (VideoView)findViewById(R.id.videoView1);
Uri video = Uri.parse("android.resource://" +
  getPackageName() + "/"
  + R.raw.castell);
```

```
5 visor.setVideoURI(video);
6 visor.setMediaController(new MediaController(this));
7 visor.start();
```

Fixeu-vos que el **VideoView** no accepta directament un identificador de recurs com a vídeo, per aquesta raó cal accedir-hi amb l'expressió que forma una adreça URI (*Universal Resource Identifier*, identificador universal de recursos) accedint als recursos empaquetats a l'aplicació que esteu construint.

A continuació s'associa aquest recurs al `VideoView`, es crea un `MediaController` per tal de tenir controls de reproducció i, finalment, es fa que el vídeo comenci. És clar que per a què tot això funcioni cal afegir els *imports* adients.

Malauradament, la **reproducció de vídeo** no sempre funciona a l'emulador d'Android, raó per la qual és possible que només escolteu el so del vídeo. En aquest cas us caldrà passar l'aplicació a un dispositiu real per poder visualitzar-lo.

1.5 Geolocalització

GPS significa *Global Positioning System*, és a dir, sistema de posicionament global, i és un sistema que permet conèixer la situació pròpia (longitud i latitud) mitjançant senyals enviats per satèl·lits.

Una de les característiques més interessants dels dispositius mòbils és la seva capacitat de detectar la seva posició geogràfica per tal de donar a l'usuari un seguit d'informació, com ara mostrar la seva situació en un mapa, llistar els recursos més propers (restaurants, benzineres...), registrar el seu recorregut mentre marxa per la muntanya, etc. El mètode més conegut de **geolocalització**, és a dir, d'obtenció de la posició geogràfica, és el sistema **GPS**.

Hi ha altres mètodes de geolocalització, com ara detectar les xarxes Wi-Fi presents i deduir a partir de les xarxes identificades quina és la posició geogràfica. Aquest mètode no és tan precís com el GPS, però d'altra banda pot funcionar sota terra o dins d'edificis, on el senyal GPS generalment no arriba.

En aquest apartat veureu com obtenir les coordenades GPS i després les fareu servir per visualitzar la vostra posició en un mapa.

1.5.1 Coordenades GPS

La posició de l'usuari del dispositiu és un aspecte més de la seva privacitat, per això cal que l'aplicació demani permís per accedir al GPS.

Primerament cal donar permís a la vostra aplicació per accedir al sensor GPS. Per fer-ho, després de crear una nova aplicació Android, editeu el fitxer **AndroidManifest.xml** i afegiu la següent línia just abans de l'etiqueta `<application>`:

Podeu descarregar el codi corresponent a aquesta activitat en l'annex anomenat "Coordenades GPS" de la secció "Annexos".

```
1 <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

Per tal que la vostra activitat pugui rebre les actualitzacions de la posició del dispositiu, cal que implementi `LocationListener`, aleshores editeu la capçalera de l'activitat perquè quedi de la següent manera:

```
public class MainActivity extends ActionBarActivity implements LocationListener
    {
```

Afegiu l'*import* **android.location.LocationListener** si no ho ha fet ja l'Android Studio. Veureu que ens marca com a error el nom de la classe perquè no implementa els mètodes del `LocationListener`; al missatge que apareix, feu clic a "Implement methods" per tal que els hi afegeixi. Concretament, els mètodes que hem d'implementar són:

- `onLocationChanged`: és cridat quan la localització GPS canvia, típicament perquè el dispositiu es desplaça a una altra posició. Serveix per actualitzar la posició a l'aplicació.
- `onProviderDisabled`: es fa la crida quan l'usuari ha deshabilitat el servei GPS.
- `onProviderEnabled`: al contrari que l'anterior, és cridat quan l'usuari ha habilitat el servei.
- `onStatusChanged`: és cridat quan el servei canvia d'estat, d'activat a desactivat o viceversa. Ens serveix per saber quan el *provider* és incapaç d'obtenir la informació de posició o si la recupera després d'un temps d'inactivitat. El paràmetre `status` ens informa d'aquest fet, pren tres valors: `OUT_OF_SERVICE`, `TEMPORARILY_UNAVAILABLE` i `AVAILABLE`.

Abans de completar els mètodes anteriors cal activar el sistema de localització afegint el següent codi a l'`onCreate()` de la vostra activitat:

```
LocationManager gestorLoc = (LocationManager)getSystemService(Context.
    LOCATION_SERVICE);
gestorLoc.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1000, 1, this);
```

Com sempre, caldrà afegir els *imports* que us digui l'Android Studio. Amb aquestes instruccions s'accedeix al servei GPS i es demana que les actualitzacions de posició s'enviïn a aquesta activitat. Els arguments numèrics són, respectivament, el temps aproximat (en mil·lisegons) entre actualitzacions i la distància mínima (en metres) que ha de canviar la posició per rebre una actualització. A una aplicació real caldrà posar-hi valors adients.

El més important és el mètode `onLocationChanged()`, que és cridat cada cop que la posició del dispositiu canvia. Editeu-lo per tal que quedi com segueix:

```
@Override
    public void onLocationChanged(Location location) {

        String text = "Posició actual:\n" +
                "Latitud = " + location.getLatitude() + "\n" +
                "Longitud = " + location.getLongitude();

        Toast.makeText(getApplicationContext(), text,
```

```
                Toast.LENGTH_LONG).show();

    }
```

Les *toast* (torrades) són petits missatges temporals que podem fer servir per mostrar informació a l'usuari.

El mètode rep la posició actual i el que fa és formar un text amb la latitud i longitud, que són els paràmetres que defineixen la posició geogràfica.

També omplireu els mètodes següents per tal que l'aplicació notifiqui a l'usuari quan la cobertura GPS es perd o es recupera:

```
@Override
    public void onProviderDisabled(String provider) {
        Toast.makeText(getApplicationContext(),
                "GPS desactivat per l'usuari",
                Toast.LENGTH_LONG ).show();
    }

    @Override
    public void onProviderEnabled(String provider) {
        Toast.makeText(getApplicationContext(),
                "GPS habilitat per l'usuari",
                Toast.LENGTH_LONG).show();
    }

    @Override
    public void onStatusChanged(String provider, int status, Bundle extras) {

        String missatge = "";
        switch (status) {
            case LocationProvider.OUT_OF_SERVICE:
                missatge = "GPS status: Out of service";
                break;
            case LocationProvider.TEMPORARILY_UNAVAILABLE:
                missatge = "GPS status: Temporarily unavailable";
                break;
            case LocationProvider.AVAILABLE:
                missatge = "GPS status: Available";
                break;
        }

        Toast.makeText(getApplicationContext(),
                missatge,
                Toast.LENGTH_LONG).show();
    }

}
```

Ara bé, com podeu provar aquesta aplicació? La primera opció i la més realista és executar-la en un dispositiu real, però si això no és possible, teniu dues opcions que no són massa realistes però que us permetran de provar-la.

En primer lloc, executeu l'aplicació a l'emulador. Amb l'aplicació en marxa, obriu una finestra del terminal i escriviu el següent:

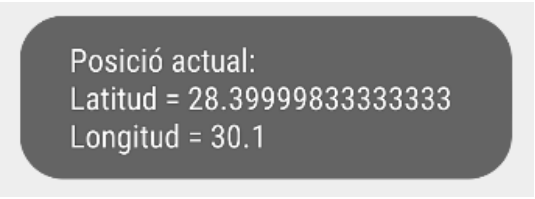
```
telnet localhost 5554
geo fix 30.1 28.4
```

Amb això, el que feu és connectar-vos a l'emulador i enviar-li la longitud i la latitud. El número 5554 és el port per defecte pel qual us podeu connectar a l'emulador. El port específic que fa servir el vostre emulador es mostra a la seva barra d'aplicació, mireu per exemple el 5554 a la figura 1.19. Si mireu la pantalla de l'emulador,

veureu com apareix la informació que li heu enviat, semblant a la *toast* de la figura 1.12. Compte, que desapareix en uns segons!

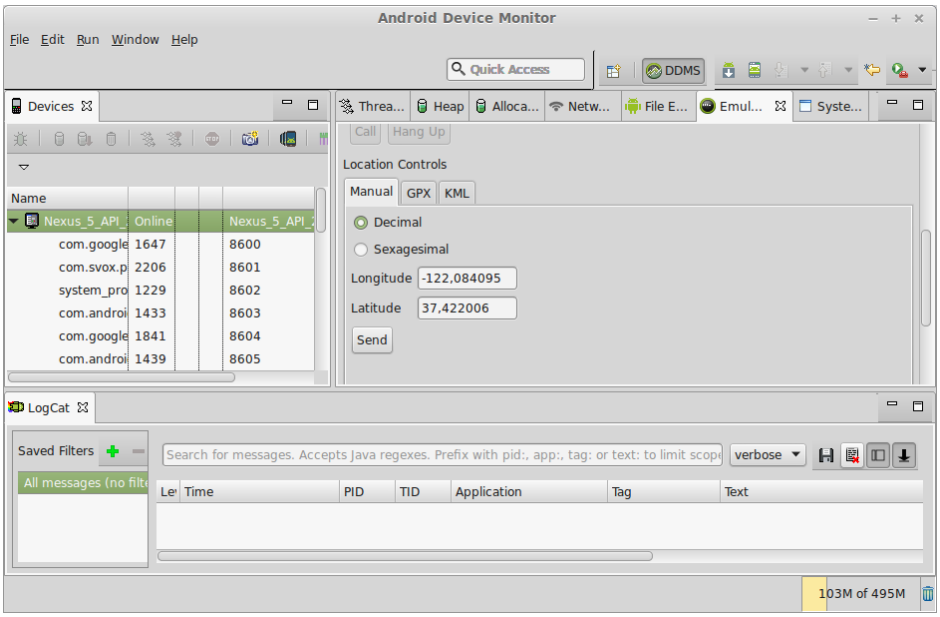
La **latitud** és la distància (angular) d'un punt a l'equador, és a dir, a l'eix sud-nord. La **longitud** és la distància (angular) d'un punt al meridià de Greenwich, és a dir, a l'eix est-oest.

FIGURA 1.12. Toast amb les coordenades GPS



L'altra opció, només disponible si feu servir l'entorn Android Studio, és anar a *Tools/Android/Android Device Monitor*, concretament a la pestanya *Emulator Control*, que conté diferents eines per controlar l'emulador: per simular una trucada telefònica, la recepció d'un SMS... Si continueu baixant dins d'aquesta finestra hi trobareu una eina anomenada *Location Controls* que permet enviar la posició a l'emulador (pestanya **GPS**), i que es mostra a la figura 1.13.

FIGURA 1.13. Eina per enviar coordenades GPS

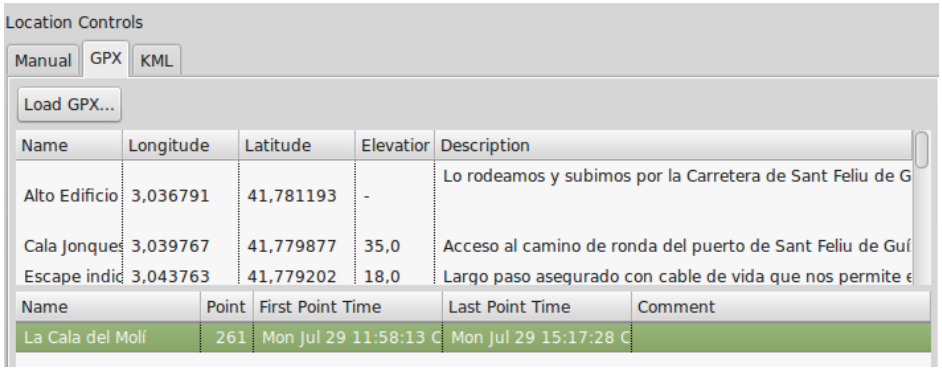


Si introduïu una coordenada i premeu el botó *Send* l'enviareu a l'emulador. Però aquesta eina us ofereix una manera molt millor d'introduir coordenades GPS: fer servir fitxers **GPX**.

Els fitxers GPX (GPS eXchange format) són fitxers XML que contenen recorreguts expressats com a seqüències de coordenades GPS, amb un temps d'arribada associat a cada punt.

Si obriu la pestanya *GPX* podreu carregar un fitxer en aquest format i enviar-lo a l'emulador amb el botó *Play*. També hi ha un control anomenat *Speed* que us permet enviar les coordenades més ràpidament si voleu, sobretot perquè sovint aquests fitxers s'han enregistrat caminant o amb bicicleta i, per tant, es mouen lentament pel mapa. A la figura 1.14 en podeu veure un exemple.

FIGURA 1.14. Càrrega d'una ruta GPX



D'aquesta manera podeu veure com es van actualitzant les coordenades a l'aplicació, fent el recorregut del mapa.

L'altra opció, KML (*Keyhole Markup Language*), és equivalent a GPX amb la diferència que és el format que fa servir Google Earth.

Podeu descarregar gratis fitxers GPX o KML al web http://es.wikiloc.com.

1.5.2 Visualització de mapes

Tot just heu vist com llegir la posició geogràfica del dispositiu (i de l'usuari, segurament!) mitjançant el servei GPS, però només l'heu pogut fer servir per mostrar les coordenades per pantalla amb una simple *toast*. La vostra aplicació seria més atractiva si en comptes d'escriure uns números per pantalla dibuixés el mapa amb la situació actual i l'anés actualitzant a mesura que us moveu. Això és el que fareu en aquest apartat.

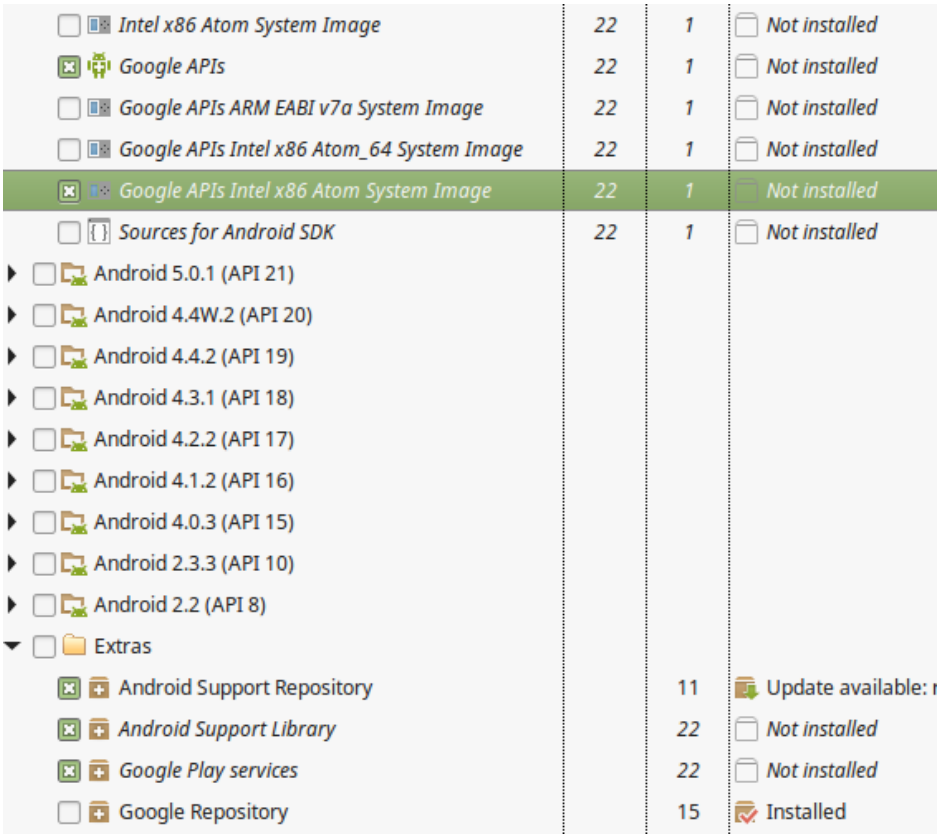
Com probablement ja sabeu, la font més important de mapes i imatges per satèl·lit és **Google Maps/Earth**, i atès que Google ha estat la creadora d'Android ho ha posat força fàcil perquè feu servir els seus mapes a les aplicacions Android.

Per poder-ho fer, cal configurar el vostre projecte per tal que faci servir les **Google API** i els **Google Play services**, és a dir, el conjunt de biblioteques i serveis per accedir a les aplicacions i recursos de Google.

En primer lloc configurarem un emulador per treballar amb les google APIs. Haurem d'anar a l'Android SDK Manager i descarregar la darrera versió de:

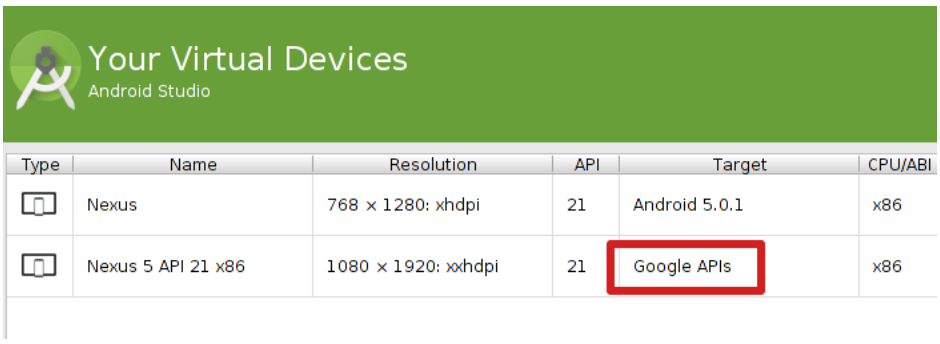
- **Google APIs Intel x86 Atom System Image** de la darrera versió de l'SDK.
- **Google APIs** de la darrera versió de l'SDK.
- **Google Play services**, que trobareu a la secció *extras*.

Figura 1.15. Android SDK Manager amb Google API instal·lat



Per tal d'executar un projecte que faci servir les **Google APIs** necessitem disposar d'un AVD amb les llibreries i aplicacions necessàries. Així, comproveu que el *target* del vostre emulador sigui "Google APIs", com podeu observar a la figura 1.16.

Figura 1.16. Emulador amb Google APIs

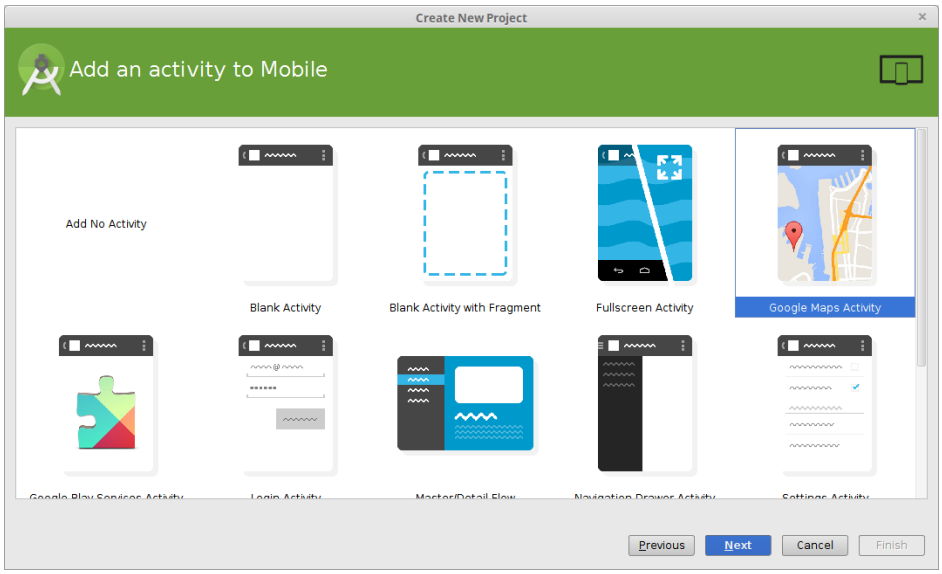


Ara que ja ho tenim tot llest per poder crear el projecte, introduïu la següent configuració:

- *Application name*: **MultimediaMaps**
- *Company domain*: **cat.xtec.ioc**
- *Minium SDK*: **API 10: Android 2.3.3 (Gingerbread)**

A la selecció d'activitats escolliu **Google Maps Activity** (figura 1.17) i deixeu els paràmetres per defecte.

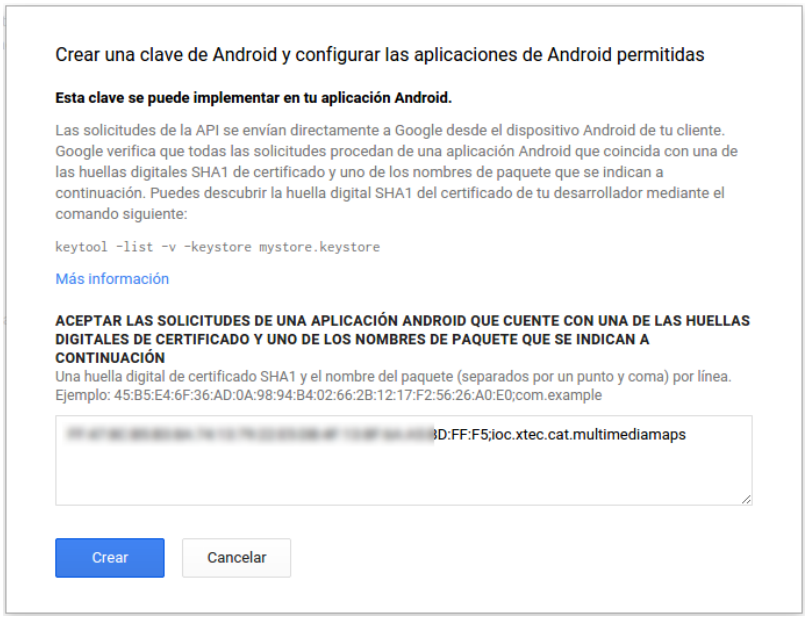
FIGURA 1.17. Selecció de l'activitat



Al directori */res/values* trobarem un fitxer anomenat `google_maps_api.xml` amb un contingut similar al següent:

```
<resources>
    <!--
    TODO: Before you run your application, you need a Google Maps API key.

    To get one, follow this link, follow the directions and press "Create" at
        the end:

https://console.developers.google.com/flows/enableapi?apiid=
    maps_android_backend&keyType=CLIENT_SIDE_ANDROID&r=
    FF:47:8C:B5:B3:8A:XX:13:79:XX:E5:DB:4F:XX:8F:6A:A5:BD:FF:F5%3Bioc.xtec.cat
    .multimediamaps

    You can also add your credentials to an existing key, using this line:
    FF:47:8C:B5:B3:8A:XX:13:79:XX:E5:DB:4F:XX:8F:6A:A5:BD:FF:F5;ioc.xtec.cat.
        multimediamaps

    Once you have your key (it starts with "AIza"), replace the "
        google_maps_key"
    string in this file.
    -->
    <string name="google_maps_key" translatable="false" templateMergeStrategy="
        preserve">
        YOUR_KEY_HERE
    </string>
</resources>
```

FIGURA 1.18. Creació de la clau



Accediu a la URL que se us proposa des d'un navegador web i, després de crear un nou projecte, se us permetrà crear una clau per poder fer servir l'API de Google Maps (com podeu veure a la figura 1.18). Per afegir més aplicacions lligades a una mateixa clau hem d'afegir línies amb l'SHA1, un ";" i el *package name* de l'aplicació. Una vegada generada, copieu el valor del camp **CLAVE DE LA API** i enganxeu-lo a l'XML com a contingut de l'etiqueta `string` (substituirem el text *YOUR_KEY_HERE*).

Amb aquests passos ja podrem executar l'emulador per comprovar el funcionament del mapa, però abans observem quines són les modificacions que ha fet l'Android Studio per tal que tot funcioni. Si obriu el fitxer **AndroidManifest.xml** podreu observar el següent contingut:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="ioc.xtec.cat.multimediamaps" >
4
5      <uses-permission android:name="android.permission.INTERNET" />
6      <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
7      <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" /
           >
8      <uses-permission android:name="com.google.android.providers.gsf.permission.
           READ_GSERVICES" />
9      <!--
10  The ACCESS_COARSE/FINE_LOCATION permissions are not required to use
11         Google Maps Android API v2, but are recommended.
12     -->
13     <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" /
           >
14     <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
15
16     <application
17         android:allowBackup="true"
18         android:icon="@mipmap/ic_launcher"
19         android:label="@string/app_name"
20         android:theme="@style/AppTheme" >
21         <meta-data
22             android:name="com.google.android.gms.version"
23             android:value="@integer/google_play_services_version" />
```

```
24          <meta-data
25              android:name="com.google.android.maps.v2.API_KEY"
26              android:value="@string/google_maps_key" />
27
28          <activity
29              android:name=".MapsActivity"
30              android:label="@string/title_activity_maps" >
31              <intent-filter>
32                  <action android:name="android.intent.action.MAIN" />
33
34                  <category android:name="android.intent.category.LAUNCHER" />
35              </intent-filter>
36          </activity>
37      </application>
38
39  </manifest>
```

Podem observar que afegeix els següents permisos:

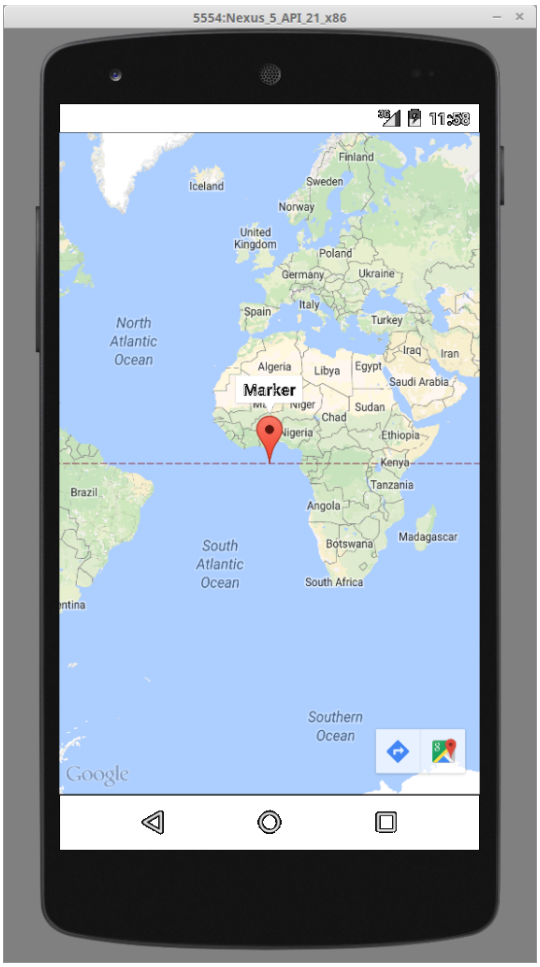
- `android.permission.INTERNET`: per accedir a Internet, necessari per descarregar els mapes.
- `android.permission.ACCESS_NETWORK_STATE`: per detectar canvis en la xarxa.
- `android.permission.WRITE_EXTERNAL_STORAGE`: permet escriure a l'emmagatzemament del telèfon, en aquest cas és necessari per descarregar fitxers per generar la memòria cau dels mapes.
- `com.google.android.providers.gsf.permission.READ_GSERVICES`: permet a l'API accedir als serveis de Google.
- `android.permission.ACCESS_COARSE_LOCATION`: accés a la posició aproximada mitjançant la xarxa: wifi i torres de telefonia.
- `android.permission.ACCESS_FINE_LOCATION`: accés a la posició més acurada, fent servir a més el GPS per obtenir-la.

I a més, afegeix dues etiquetes `meta-data` amb la informació sobre la versió dels *Google Play Services* i la clau que hem afegit al fitxer *google_maps_api.xml*.

El *layout* de l'aplicació constarà d'un fragment que serà manipulat des del fitxer MapsActivity.java. Si ens fixem amb el codi de l'activitat, podem veure com acaba fent una crida a `setUpMap()` que afegeix un marcador al mapa en la coordenada (0,0) amb el títol de "*Marker*".

Executeu el projecte i visualitzareu un mapa del món amb un marcador en la coordenada geogràfica (0,0), és a dir, a la latitud i longitud 0. Si feu clic sobre la marca us sortirà el text "*Marker*" tal com podeu veure a la figura 1.19.

FIGURA 1.19. Google Maps a l'emulador



Per tal d'afegir controls de *zoom* a la vostra aplicació de manera que pugueu apropar-vos des de l'emulador d'una manera còmoda cal que modifiqueu el mètode `setUpMapIfNeeded()` per tal que quedi de la següent manera:

```
1  private void setUpMapIfNeeded() {
2          // Do a null check to confirm that we have not already instantiated the
                map.
3          if (mMap == null) {
4              // Try to obtain the map from the SupportMapFragment.
5              mMap = ((SupportMapFragment) getSupportFragmentManager().
                   findFragmentById(R.id.map))
6                       .getMap();
7              // Check if we were successful in obtaining the map.
8              if (mMap != null) {
9                  setUpMap();
10                 UiSettings settings = mMap.getUiSettings();
11                 settings.setZoomControlsEnabled(true);
12             }
13         }
14     }
```

És a dir, afegim les línies `UiSettings settings = mMap.getUiSettings();` i `settings.setZoomControlsEnabled(true);` per obtenir la configuració del mapa i modificar el valor de `ZoomControlsEnabled` per posar-lo a `true`.

Podreu trobar més mètodes per configurar el vostre mapa mitjançant la variable `mMap`; per exemple, si voleu canviar el tipus de mapa podeu accedir

al mètode `mMap.setMapType(int type)`, que rep com a argument el tipus de mapa: `GoogleMap.MAP_TYPE_NORMAL`, `GoogleMap.MAP_TYPE_HYBRID`, `GoogleMap.MAP_TYPE_SATELLITE`, `GoogleMap.MAP_TYPE_TERRAIN` i `GoogleMap.MAP_TYPE_NONE`. Proveu els diferents valors i observeu els canvis que es produeixen en la representació.

1.5.3 Seguiment de la vostra posició

Si volem que el mapa s'actualitzi amb la nostra posició haurem d'implementar **LocationListener** i, cada cop que es produeixi un canvi d'aquests, actualitzar el mapa.

Així, feu que la classe **MapsActivity** implementi *LocationListener* i recordeu afegir els mètodes que Android Studio ens exigeix.

```
1  public class MapsActivity extends FragmentActivity implements LocationListener
       {
2  ...
3      @Override
4      public void onLocationChanged(Location location) {
5
6      }
7
8      @Override
9      public void onStatusChanged(String provider, int status, Bundle extras) {
10
11     }
12
13     @Override
14     public void onProviderEnabled(String provider) {
15
16     }
17
18     @Override
19     public void onProviderDisabled(String provider) {
20
21     }
22   ...
23   }
```

Des dels mètodes `onStatusChanged`, `onProviderEnabled` i `onProviderDisabled` podríem informar sobre els canvis que es produeixin al servei GPS; ara, però, ens centrarem en el mètode `onLocationChanged`, on haurem de:

1. Obtenir la posició a partir del paràmetre *location*.
2. Crear la càmera (àrea del mapa que es visualitza en un moment determinat) amb la posició i un nivell de zoom.
3. Desplaçar la càmera al punt proposat.

`LatLng` emmagatzema un parell de coordenades, la latitud i la longitud.

Així, afegirem el següent codi a `onLocationChanged`:

```
@Override
    public void onLocationChanged(Location location) {
        // Creem un LatLng a partir de la posició
        LatLng posicio = new LatLng(location.getLatitude(),location.
            getLongitude());
        // Afegim la càmera amb el punt generat abans i un nivell de zoom
        CameraUpdate camera = CameraUpdateFactory.newLatLngZoom(posicio, 19);
        // Desplacem la càmera al nou punt
        mMap.moveCamera(camera);
    }
```

Una vegada hem obtingut la posició, fem la crida al mètode `newLatLngZomm(LatLng latLng, float zoom)` de `CameraUpdateFactory` i que rep dos paràmetres: el primer determinarà el punt on es situarà la càmera i el segon especificarà el nivell de *zoom*, que haurà de prendre una valor entre 2.0f i 21.0f. Finalment, actualitzem la càmera del mapa.

Per tal que el *listener* comenci a controlar els canvis de posició, recordeu afegir a l'`onCreate` el següent codi:

```
LocationManager gestorLoc = (LocationManager)getSystemService(Context.
    LOCATION_SERVICE);
gestorLoc.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1000, 1, this);
```

Si proveu d'enviar unes coordenades GPS a l'emulador, comprovareu com canvia la vista. Lògicament, com més *zoom* hi apliquem, més evident serà el moviment. Per veure l'efecte és recomanable que carregueu un fitxer GPX amb el DDMS (Tools/Android/Android Device Monitor), i que gaudiu de la passejada!

1.5.4 Afegint marques al mapa

Sovint les aplicacions que mostren mapes afegeixen marques per assenyalar a l'usuari els punts que li siguin interessants, com ara els restaurants més propers, les parades de metro o les ciutats que ha visitat. En aquest apartat veureu com afegir aquests senyals (icones) al vostre mapa, i com situar-los on vosaltres voleu. Per fer-ho continuareu amb l'aplicació anterior.

Per anar afegint les marques a mesura que va canviant la posició haurem de modificar el mètode `onLocationChanged`. El procediment serà el següent:

1. Creem un element del tipus `MarkerOptions`.
2. Afegim les propietats desitjades: Títol (*title*), posició (*position*), descripció (*snippet*) i icona (*icon*).
3. Afegim el marcador al mapa.
4. Si hem de manipular el marcador posteriorment, ens podem guardar l'objecte *Marker* que retorna la funció `addMarker`; una possible utilitat de guardar-lo és fer la crida a `showInfoWindow()` per fer que es mostri la informació del marcador sense haver de fer clic sobre ell.

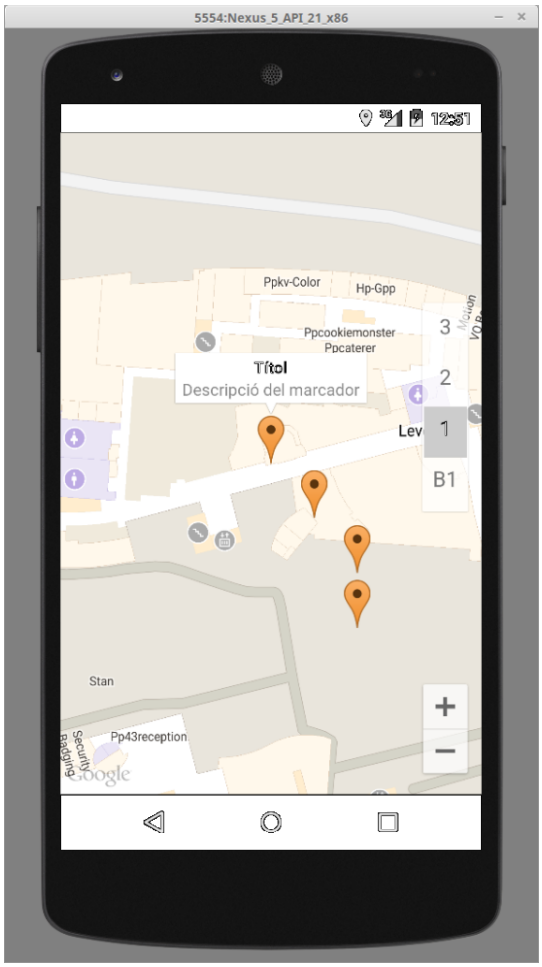
El codi del mètode `onLocationChanged` quedarà de la següent manera:

```
@Override
    public void onLocationChanged(Location location) {

        // Creem un LatLng a partir de la posició
        LatLng posicio = new LatLng(location.getLatitude(), location.
            getLongitude());
        // Afegim la càmera amb el punt generat abans i un nivell de zoom
        CameraUpdate camera = CameraUpdateFactory.newLatLngZoom(posicio, 19);
        // Desplacem la càmera al nou punt
        mMap.moveCamera(camera);

        // Creem i afegim el marcador
        MarkerOptions opcions = new MarkerOptions();
        opcions.title("Títol");
        opcions.position(posicio);
        opcions.snippet("Descripció del marcador");
        opcions.icon(BitmapDescriptorFactory.defaultMarker(
            BitmapDescriptorFactory.HUE_ORANGE));
        // Aquí l'afegim al mapa i ens el guardem a una variable (ja es veu la
            icona al mapa però cal fer-hi clic per veure la informació)
        Marker marca = mMap.addMarker(opcions);
        // Fem que es mostri la informació sense haver-hi de fer clic
        marca.showInfoWindow();
    }
```

Podeu descarregar el codi corresponent a aquesta activitat a l'annex anomenat "Google Maps" de la secció "Annexos".

Si executem l'aplicació i anem enviant diferents posicions fent servir l'*Android Device Manager* veurem un resultat similar al que es mostra en la figura 1.20.

Figura 1.20. Google Maps amb marcadors



1.6 Dibuixar

Sovint es necessita que les aplicacions puguin dibuixar qualsevol element a qualsevol posició de la pantalla. L'exemple més clar serien les aplicacions de dibuix, però n'hi ha moltes altres que també ho fan servir: per mostrar gràfiques i esquemes, per a jocs, per mostrar marques especials...

A més, moltes vegades no podeu fer servir imatges predefinides perquè necessiteu canviar-ne la mida, la forma o el color.

Tot seguit veureu com es poden dibuixar formes geomètriques tot controlant els seus paràmetres (mida, posició, color...) segons les indicacions de l'usuari. Concretament, quan l'usuari toqui en una posició de la pantalla, s'hi dibuixarà una figura. Per donar-li més varietat, l'aplicació anirà dibuixant formes diferents cada cop que toqueu la pantalla.

Podeu descarregar el codi corresponent a aquesta activitat en l'annex anomenat "Dibuixar" de la secció "Annexos".

Comenceu per crear un nou projecte Android, anomenat per exemple ***Dibuixar***, que tingui el *layout* per defecte. En aquest exemple no fareu servir aquest *layout*, sinó que el generareu i treballareu amb ell des del codi Java. Per això ara cal que obriu l'activitat principal i escriviu el següent codi, eliminant les instruccions que calgui:

```
1  package ioc.xtec.cat.dibuixar;
2
3  import android.graphics.Color;
4  import android.os.Bundle;
5  import android.support.v7.app.ActionBarActivity;
6
7
8  public class MainActivity extends ActionBarActivity {
9
10     @Override
11     protected void onCreate(Bundle savedInstanceState) {
12         super.onCreate(savedInstanceState);
13
14         Llenc llenc = new Llenc(this);
15         llenc.setBackgroundColor(Color.BLACK);
16         setContentView(llenc);
17     }
18 }
```

De moment hi ha dues idees clau: estem creant un objecte de classe `Llenc` (pels llenços de pintar) i fem servir aquest objecte com a vista de la nostra aplicació en comptes del *layout* per defecte.

En informàtica s'anomena **llenç** (*canvas* en anglès) a qualsevol superfície en la qual es dibuixa.

Què és aquesta classe `Llenc`? Doncs l'heu de definir vosaltres. Per fer-ho, creeu una nova classe fent clic al botó dret sobre l'espai de noms i seleccionant */New/Java Class* i farem que aquesta nova classe derivi de la classe `View` afegint `public class Llenc extends View {`; concretament, escolliu `View (android.view)`. Amb això l'Android Studio us indica un error, que heu de resoldre posant-vos a sobre de la línia i triant l'opció "Create constructor matching super" i seguidament "View(context:Context)". Un cop fet això, us hauria de quedar un codi com el següent:

```
package ioc.xtec.cat.dibuixar;

import android.content.Context;
import android.view.View;

public class Llenc extends View {
    public Llenc(Context context) {
        super(context);
    }
}
```

Què vol dir que la classe `Llenc` derivi de **View**? Doncs que es tracta d'una vista igual que les que feu servir a les vostres aplicacions (com ara els TextViews, els botons, els ImageView, els MapView, o qualsevol altre). Això implica que vosaltres teniu el dret de dibuixar el que vulgueu en aquest element. I com que hem dit que els continguts de la nostra aplicació es mostren com un objecte de classe `Llenc`, doncs ja teniu la manera de dibuixar el que vulgueu.

Per començar, cal crear alguns atributs de la vostra classe i indicar que els objectes `Llenc` es poden tocar (amb `setFocusable()`):

```
public class Llenc extends View {

    Paint pintar = new Paint();

    int x = 100;
    int y = 100;
    int cont = 0;

    public Llenc(Context context) {
        super(context);
        setFocusable(true);
    }
  }
```

L'objecte de classe `Paint` és una mena de pinzell amb el qual controlareu el color i altres paràmetres del que dibuixareu; amb la **x** i la **y** controlareu les coordenades on ha tocat l'usuari, i finalment el comptador el fareu servir per alternar entre diferents motius de dibuix.

Tot seguit, volem detectar quan l'usuari toca l'aplicació, la qual cosa es fa fàcilment amb el mètode `onTouchEvent()` que es mostra a continuació. El que fa és mirar si es tracta d'un *event* del tipus *tocar la pantalla* (**ACTION_DOWN**) i aleshores prendre les coordenades on s'ha tocat:

```
@Override
    public boolean onTouchEvent(MotionEvent event) {
        int eventaction = event.getAction();

        if (eventaction == MotionEvent.ACTION_DOWN) {
            x = (int) event.getX();
            y = (int) event.getY();
            invalidate();
        }
        return true;
    }
```

L'ordre `invalidate()` és molt important, perquè li diu al llenç que els seus continguts s'han de tornar a dibuixar; això provoca que es cridi el mètode `onDraw()`, que és on fareu els dibuixos.

El `Canvas` funciona com una mena de full de paper en blanc al qual s'hi poden afegir diferents tipus de figures; per exemple, amb `drawRect()` podeu dibuixar rectangles, amb `drawCircle()` cercles, etc. En general, es poden dibuixar formes senzilles, com ara òvals, punts, línies, o també text. Com és habitual, les coordenades del llenç comencen al cantó superior esquerre.

En aquest cas alternareu entre diferents motius senzills, dibuixats a les coordenades x i y en les que hagi tocat l'usuari, i amb l'atribut `cont` controlareu quin motiu toca dibuixar cada cop. Recordeu que el pinzell `pintar` és el que us permet triar el color. La resta d'operacions de dibuix queden prou clares amb el seu nom:

```
@Override
    public void onDraw(Canvas canvas) {
        switch (cont) {
            case 0:
                pintar.setColor(Color.CYAN);
                canvas.drawRect(x – 20, y – 20, x + 20, y + 10, pintar);
                pintar.setColor(Color.YELLOW);
                canvas.drawRect(x – 20, y, x + 20, y + 20, pintar);
                break;

            case 1:
                pintar.setColor(Color.RED);
                canvas.drawCircle(x, y, 40, pintar);
                break;

            case 2:
                pintar.setColor(Color.GREEN);
                pintar.setTextSize(40);
                canvas.drawText("HOLA", x, y, pintar);
                break;

            case 3:
                pintar.setColor(Color.MAGENTA);
                canvas.drawOval(new RectF(x – 40, y – 20, x + 40, y + 20),
                    pintar);
                break;
        }
        cont = (cont + 1) % 4;
    }
```

A la figura 1.21 podeu veure l'aplicació en funcionament:

FIGURA 1.21. Aplicació que dibuixa amb un llenç

Amb aquestes tècniques podeu enriquir la vostra aplicació afegint botons per triar les eines de dibuix, les mides i els colors, i fent que s'emmagatzemin els elements dibuixats perquè es visualitzin tots a la pantalla, no només l'últim on hem tocat.

2. Objectes multimèdia

Un dels grans avantatges dels dispositius mòbils és que l'usuari té al seu abast una càmera de fotos i vídeo i una gravadora d'àudio en qualsevol moment, cosa que ha disparat el nombre de captura d'imatges i àudio per part dels usuaris, així com les aplicacions que permeten publicar instantàniament aquesta informació, compartir-la amb d'altres usuaris o emmagatzemar-la per donar-li ús posteriorment. Per treure profit d'un dispositiu mòbil cal saber, doncs, accedir a aquests accessoris, engegar-los i accedir als continguts gravats per reproduir-los o fer-ne un altre ús.

També és molt important saber accedir als continguts emmagatzemats al dispositiu per posar-los a l'abast de l'usuari, com ara mitjançant una galeria d'imatges, i distingir les diferents possibilitats d'emmagatzemament que ens ofereixen els dispositius mòbils (memòria interna, targetes de memòria externa...).

Finalment, al món multimèdia hi ha nombrosos formats per desar imatges, àudio i vídeo, i cal conèixer els més importants i saber controlar les aplicacions perquè generin els continguts multimèdia en diferents formats i amb diferents qualitats en funció del seu ús previst.

Com veureu, de tot això s'encarreguen un conjunt d'objectes (des del punt de vista de la programació orientada a objectes) que faciliten molt el desenvolupament d'aquest tipus d'aplicacions.

2.1 Ús de la càmera

Un dels accessoris o objectes multimèdia més importants als dispositius mòbils és la càmera fotogràfica. Més enllà de fer fotos amb les aplicacions estàndard, pot ser molt interessant que les vostres aplicacions permetin fer fotos directament i fer-les servir, com per exemple per fer fotos alhora que enregistreu una ruta amb GPS, o per fer un receptari de cuina amb la recepta i la foto del plat, o qualsevol altra aplicació que pugueu pensar.

Hi ha dos procediments per utilitzar la càmera a les nostres aplicacions: engegar l'aplicació estàndard de la càmera o accedir directament al dispositiu.

2.1.1 Accés a l'aplicació estàndard de la càmera de fotos

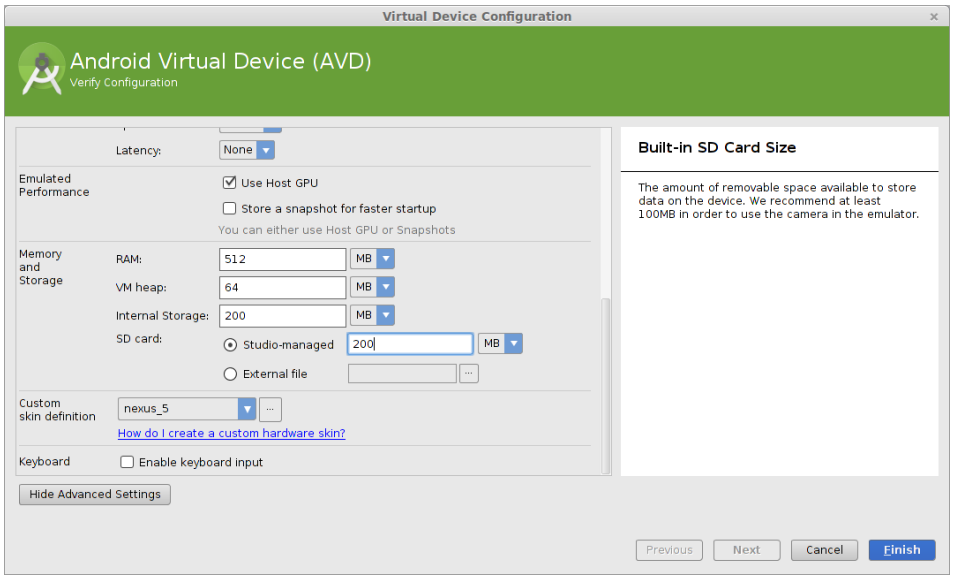
Podeu descarregar el codi corresponent a aquesta activitat en l'annex anomenat "Aplicació càmera de fotos" de la secció "Annexos".

La manera més senzilla de fer una foto des d'una aplicació Android és engegant l'aplicació de càmera de fotos que tingui instal·lada el dispositiu. Lògicament, en

contrapartida, la vostra aplicació tindrà menys control sobre el procés i el resultat de l'operació, però en molts casos només us caldrà fer una foto i emmagatzemar-la.

Abans de continuar, tingueu present que en general les fotos i d'altres captures s'emmagatzemen a la memòria externa (generalment, la targeta SD) del dispositiu. De fet, al principi l'aplicació estàndard per fer servir la càmera de fotos no funcionava si no hi havia una targeta SD al dispositiu. Per comprovar que el vostre emulador tingui configurada una targeta SD, heu d'obrir *Tools/Android/AVD Manager*, clicar a la icona d'edició i comprovar que tingui algun valor al camp **SD Card** (heu de mostrar les opcions avançades), com per exemple 200 (que són megabytes de capacitat de la targeta virtual), com es veu a la figura 2.1.

FIGURA 2.1. Emulador Android amb targeta SD de 200 MB



Hem d'afegir el permís `<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />` al nostre AndroidManifest.xml. Aquest permís ens permet llegir i escriure a la memòria externa del dispositiu.

El *layout* per a aquesta aplicació és molt senzill: només es necessita un botó per indicar que es vol fer la foto i un **ImageView** per visualitzar la foto que s'ha fet. Tot seguit podeu veure el contingut del *layout*, en el qual l'única propietat destacable és la propietat `onClick` del botó que té de valor `fesFoto`, que serà el mètode responsable d'engegar l'aplicació de la càmera:

Els recursos *mipmap* són utilitzats per definir les icones de l'aplicació. En aquest cas, Google ens recomana usar aquests recursos per damunt dels *drawable*, ja que ens permeten treballar amb imatges de diferent resolució de la que realment té el dispositiu. Per exemple, en un dispositiu *xhdpi*, fent ús d'un *mipmap* aquest podria utilitzar una imatge de major resolució (per exemple una imatge *xxhdpi*) per la icona del llançador d'aplicacions.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="fill_parent"
4      android:layout_height="fill_parent"
5      android:gravity="bottom"
6      android:orientation="vertical" >
7
8      <ImageView
9          android:id="@+id/imageView1"
10         android:layout_width="match_parent"
11         android:layout_height="match_parent"
12         android:layout_weight="5"
13         android:src="@mipmap/ic_launcher" />
```

```
14
15      <Button
16          android:id="@+id/button1"
17          android:layout_width="wrap_content"
18          android:layout_height="wrap_content"
19          android:onClick="fesFoto"
20          android:text="Foto" />
21
22  </LinearLayout>
```

Per iniciar una altra aplicació a Android cal crear un **Intent** i començar una nova activitat amb `startActivityForResult()`, per tal que puguem recollir el resultat de l'activitat mitjançant el mètode `onActivityResult()`. Per saber quina és l'activitat que ha retornat el resultat cal assignar-li un número qualsevol que la identifiqui, que en el codi següent hem associat a la variable APP_CAMERA, creada amb la finalitat d'emagatzemar-lo:

```
1   // Número que identifica l'activitat de l'aplicació de fotos
2       private static final int APP_CAMERA = 0;
3
4       // Identificador de la imatge que crearà l'aplicació de fotos
5       private Uri identificadorImatge;
6
7       public void fesFoto(View view) {
8           // Es crea l'intent per l'aplicació de fotos
9           Intent intent = new Intent("android.media.action.IMAGE_CAPTURE");
10          // Es crea un nou fitxer a l'emmagatzematge extern i se li passa a l'
                intent
11          File foto = new File(Environment.getExternalStorageDirectory(), "Foto.
                jpg");
12          intent.putExtra(MediaStore.EXTRA_OUTPUT, Uri.fromFile(foto));
13          // Es desa l'identificador de la imatge per recuperar-la quan estigui
                feta
14          identificadorImatge = Uri.fromFile(foto);
15          // S'engega l'activitat
16          startActivityForResult(intent, APP_CAMERA);
17      }
```

Abans d'iniciar l'aplicació estàndard de captura d'imatges es crea un nou fitxer a l'emmagatzemament extern (típicament, la targeta de memòria SD) i se'l passa a l'*intent* per tal que sàpiga on ha de desar la imatge que prengui. Finalment, s'executa l'activitat, que iniciarà l'aplicació de càmera fotogràfica per defecte (definida per l'usuari de cada dispositiu).

El segon pas, un cop feta la foto, és recollir el resultat de l'activitat que heu iniciat. En aquest cas només es vol mostrar la imatge que s'ha fet, però es podrien dur a terme altres operacions. També es mostrarà un ***Toast*** amb el nom amb el qual s'ha desat la imatge, si és el cas.

Totes aquestes operacions es porten a terme dins del mètode `onActivityResult()`, que s'executa cada cop que finalitza alguna activitat iniciada per la vostra aplicació. Com que una aplicació pot executar moltes activitats, cal fer servir un codi per distingir-les; en aquest cas és el paràmetre `requestCode` que coincideix amb el valor `APP_CAMERA` que hem passat a l'activitat en engegar-la. Distingir quina activitat ha finalitzat és l'objectiu del `switch` que veureu al codi d'exemple.

Tanmateix, l'activitat pot haver funcionat bé o pot haver fallat per diferents causes; per aquest motiu hi ha l'*if*, que comprova que el resultat (paràmetre `resultCode`) sigui `RESULT_OK`. L'altre valor possible per al `resultCode` d'una activitat és `RESULT_CANCELED` quan l'aplicació falla però tenim l'opció de definir altres codis personalitzats.

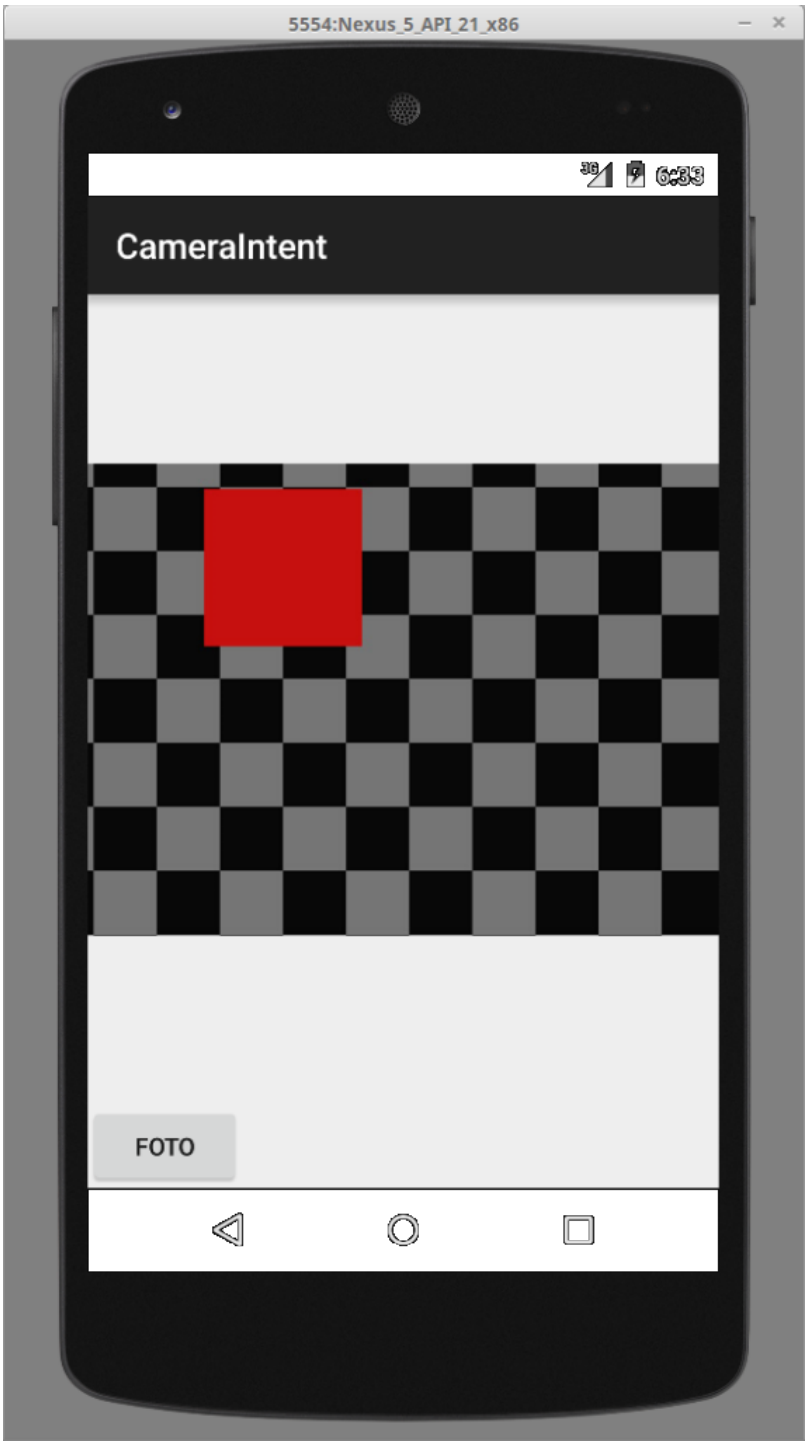
Un cop comprovat això, el codi accedeix a la imatge mitjançant el `ContentResolver`, que dóna accés als continguts del dispositiu, en aquest cas la foto que ha fet la càmera. Com que la càrrega de la imatge pot fallar, s'ha de fer dins d'un `try... catch` per tal de tractar les possibles excepcions. A més de la imatge, es mostra un *toast* amb el nom del fitxer on s'ha guardat la imatge o bé un missatge d'error si la càrrega falla.

```
@Override
    public void onActivityResult(int requestCode, int resultCode, Intent data)
        {
        // Primer cridem al mètode d'Activity perquè faci la seva tasca
        super.onActivityResult(requestCode, resultCode, data);
        switch (requestCode) {
            case APP_CAMERA:
                if (resultCode == Activity.RESULT_OK) {
                    /* El ContentResolver dóna accés als continguts
                       (la imatge emmagatzemada en aquest cas)*/
                    ContentResolver contRes = getContentResolver();
                    // Cal indicar que el contingut del fitxer ha canviat
                    contRes.notifyChange(identificadorImatge, null);
                    /* Accedeix a l'ImageView i hi carrega la foto que ha fet
                        la
                       càmera */
                    ImageView imageView = (ImageView) findViewById(R.id.
                        imageView1);
                    Bitmap bitmap;
                    /* Com que la càrrega de la imatge pot fallar, cal tractar
                       les possibles excepcions*/
                    try {

                        bitmap = android.provider.MediaStore.Images.Media
                                .getBitmap(contRes, identificadorImatge);

                        /* Reduïm la imatge per no tenir problemes de
                            visualització.
                           Calculem l'alçada per mantenir la proporció amb una
                               amplada de 1080 píxels*/
                        int alt = (int) (bitmap.getHeight() * 1080 / bitmap.
                            getWidth());
                        Bitmap reduit = Bitmap.createScaledBitmap(bitmap, 1080,
                             alt, true);

                        imageView.setImageBitmap(reduit);

                    } catch (Exception e) {
                        Toast.makeText(this, "No es pot carregar la imatge" +
                                        identificadorImatge.toString(),
                                Toast.LENGTH_SHORT).show();
                    }
                }
        }
    }
}
```

La carpeta */storage/sdcard/* del dispositiu Android correspon a la targeta de memòria SD.

Ara ja podeu executar l'aplicació i veure el seu funcionament: en clicar el botó s'engega l'aplicació per defecte per fer fotos, i aleshores la foto s'emmagatzema al dispositiu. A la figura 2.2 teniu una captura de l'aplicació quan es mostra la foto realitzada.

FIGURA 2.2. Aplicació que fa fotos mitjançant l'aplicació estàndard



2.1.2 Accés directe a la càmera de fotos

De vegades les aplicacions requereixen un control més estret sobre el procés de captura d'imatges fotogràfiques, com per exemple obtenir-ne una previsualització en la mateixa aplicació, emmagatzemar les fotos directament, o fins i tot convertir-les a un determinat format. Per tal de veure com es pot dur a terme tot això, la següent aplicació accedirà directament a la càmera, integrarà la previsualització al seu *layout* principal i convertirà els píxels capturats al format desitjat per l'usuari, ja sigui JPEG o PNG, abans de desar la imatge.

Com que aquesta aplicació fa servir directament la càmera, que és un dispositiu important pel que fa a la privacitat de l'usuari (ningú no vol que una aplicació faci fotos sense el seu permís), en primer lloc cal donar permís a l'aplicació perquè pugui accedir a la càmera i perquè pugui llegir i escriure a la memòria externa. Creeu un nou projecte d'Android (anomenat **Camera**, per exemple) i obriu el seu fitxer **AndroidManifest.xml** per afegir el següent tot just abans de l'etiqueta `application`:

```
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

Aquesta aplicació s'ha d'executar a un emulador amb targeta SD instal·lada per poder desar les fotos.

Podeu descarregar el codi corresponent a aquesta activitat en l'annex anomenat "Accés directe a la càmera de fotos" de la secció "Annexos".

Començareu per fer una aplicació que desi les fotos en format JPEG, i més endavant afegirem un botó per desar-les en format PNG. El *layout* és molt senzill, amb una àrea gran de previsualització i, a sota, un botó per fer la foto i un **ImageView** petit per veure la foto que s'ha fet.

La previsualització es fa mitjançant un control nou anomenat `SurfaceView`, dissenyat per a aquesta mena de tasques, i que podeu veure al codi del *layout*. Pel que fa a les seves propietats, és molt semblant a un `ImageView`. El botó té associa el mètode `onClickFoto()` a la seva propietat `onClick`.

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

    <SurfaceView
        android:id="@+id/surfaceview"
        android:layout_width="fill_parent"
        android:layout_height="396dp"
        android:layout_weight="0.88" />

    <LinearLayout
        android:id="@+id/linearLayout1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="0.88" >

        <Button
            android:id="@+id/button1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onClickFoto"
            android:text="FOTO" />

        <ImageView
            android:id="@+id/miniatura"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:scaleType="fitCenter"
            android:src="@mipmap/ic_launcher" />

    </LinearLayout>

</LinearLayout>
```

Fet això, ja es pot començar a editar el codi de l'activitat de l'aplicació. En primer lloc, perquè l'aplicació sigui capaç de mostrar la previsualització de la càmera pel **SurfaceView**, la seva activitat principal ha d'implementar la interfície **SurfaceHolder.Callback**. Per fer-ho, heu de canviar la declaració de la vostra activitat tal com segueix:

```
1 public class MainActivity extends ActionBarActivity  implements SurfaceHolder.
      Callback {
```

Afegiu els *imports* que siguin necessaris. Veureu que l'Android Studio assenyala l'error `CameraActivity` degut a que, per tal que l'activitat implementi `SurfaceHolder.Callback`, ha d'incloure els mètodes `surfaceChanged()`, `surfaceCreated()` i `surfaceDestroyed()`. Feu clic sobre la línia que conté l'error i us apareixerà a l'esquerra un requadre amb diverses opcions disponibles, i seleccioneu l'opció *Implement methods*, amb la qual cosa s'afegiran els mètodes que faltaven:

```
1  @Override
2      public void surfaceCreated(SurfaceHolder holder) {
3
4      }
5
6      @Override
7      public void surfaceChanged(SurfaceHolder holder, int format, int width, int
           height) {
8
9      }
10
11     @Override
12     public void surfaceDestroyed(SurfaceHolder holder) {
13
14     }
```

La funció d'aquests mètodes i els seus paràmetres és donar suport a la previsualització de la càmera de fotos. Són cridats automàticament quan s'executa l'aplicació. Més concretament:

- `surfaceCreated()` és cridat quan es crea la superfície de previsualització (és a dir, quan es crea l'activitat de l'aplicació) i rep l'activitat a la qual pertany (que és un ***SurfaceHolder***, és a dir, que conté una superfície de previsualització). El que ha de fer és activar la càmera perquè comenci a rebre imatges.
- `surfaceChanged()` és cridat quan la superfície de previsualització canvia de mida o format, per adaptar la previsualització a la seva mida real. A més del *SurfaceHolder*, rep el format de píxel i l'amplada i alçada de la superfície de previsualització en píxels. En aquest mètode s'ha de dir a la càmera la mida de previsualització que es necessita. Aquest mètode és cridat quan s'inicia l'aplicació, tot seguit de `surfaceCreated()`.
- `surfaceDestroyed()`és cridat quan la superfície de previsualització s'ha de destruir, i el que ha de fer en aquest cas és desconnectar la càmera.

Abans d'omplir aquests mètodes, s'ha de començar pel principi i afegir un atribut de tipus ***Camera*** per accedir a la càmera de fotos i inicialitzar el ***SurfaceView*** i les

crides d'actualització de la vista prèvia. Editeu el començament de l'activitat del vostre projecte per afegir l'atribut i dur a terme la inicialització:

```
// Aquest atribut permet accedir a la càmera de fotos
    Camera camera;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Accedeix al SurfaceView i d'aquest al SurfaceHolder per activar les
        // actualitzacions de la vista prèvia (addCallback)
        SurfaceView surfaceView = (SurfaceView) findViewById(R.id.surfaceview);
        SurfaceHolder surfaceHolder = surfaceView.getHolder();
        surfaceHolder.addCallback(this);
    }
```

No us oblideu d'afegir els ***imports*** necessaris si no ho ha fet automàticament l'Android Studio. Fet això, quan s'engegui l'aplicació es cridaran els tres mètodes de `SurfaceHolder.Callback` que heu creat abans. En primer lloc cal que editeu el `surfaceCreated()` per fer que quan es creï la superfície de previsualització s'activi la càmera:

```
@Override
    public void surfaceCreated(SurfaceHolder holder) {
        camera = Camera.open();
    }
```

Per tal que la superfície de previsualització i la càmera es posin d'acord pel que fa a la mida de les imatges de la vista prèvia, s'ha d'editar el `surfaceChanged()` per ajustar l'orientació del ***SurfaceView*** i la càmera; fixeu-vos que hem reanomenat els paràmetres perquè siguin més entenedors; vosaltres també haureu de reanomenar-los. Un cop fet això, s'activa la vista prèvia. Com que aquesta operació pot fallar, s'ha d'executar dins d'un `try ... catch` per capturar les possibles excepcions.

```
@Override
    public void surfaceChanged(SurfaceHolder holder, int format, int ample, int
         alt) {
        try {
            ficarOrientacioCamera(this, Camera.CameraInfo.CAMERA_FACING_BACK,
                camera);

            camera.setPreviewDisplay(holder);
        } catch (Exception e) {
            Toast.makeText(this, "Error accedint a la càmera, causa:" + e.
                toString(),
                    Toast.LENGTH_LONG).show();
            e.printStackTrace();
        }
        camera.startPreview();
    }
```

El mètode `ficarOrientacioCamera` ens permetrà calcular i aplicar l'orientació que tindrà la previsualització. Així, si tenim el mòbil en disposició vertical (mode *portrait*), la previsualització s'haurà de voltejar 90° ja que inicialment estarà preparada per rebre la informació en disposició horitzontal (mode *landscape*, 0°). Calculem els graus de diferència i els apliquem amb el mètode `setDisplayOrientation`. Tot seguit podeu veure el codi del mètode:

```
public static void ficarOrientacioCamera(Activity activity, int cameraId,
    android.hardware.Camera camera) {
        Camera.CameraInfo info = new android.hardware.Camera.CameraInfo();
        Camera.getCameraInfo(cameraId, info);
        int rotacio = activity.getWindowManager().getDefaultDisplay().
            getRotation();
        int graus = 0;

        switch (rotacio) {
            case Surface.ROTATION_0:
                graus = 0;
                break;
            case Surface.ROTATION_90:
                graus = 90;
                break;
            case Surface.ROTATION_180:
                graus = 180;
                break;
            case Surface.ROTATION_270:
                graus = 270;
                break;
        }

        int result;
        if (info.facing == Camera.CameraInfo.CAMERA_FACING_FRONT) {
            result = (info.orientation + graus) % 360;
            result = (360 - result) % 360;  // compensar l'efecte mirall
        } else {  // camera posterior
            result = (info.orientation - graus + 360) % 360;
        }
        camera.setDisplayOrientation(result);
    }
```

L'últim punt important és aturar la vista prèvia i alliberar la càmera quan es tanqui l'aplicació. Això s'ha de fer al mètode `surfaceDestroyed()`:

```
@Override
    public void surfaceDestroyed(SurfaceHolder holder) {
        camera.stopPreview();
        camera.release();
    }
```

Ja podeu provar d'executar l'aplicació, però encara no cliqueu el botó de *Foto* ja que encara no hem escrit el mètode `onClickFoto()` al qual està associat, i l'aplicació fallaria. El que ha de fer aquest mètode és molt senzill: només ha de dir-li a la càmera que faci la foto i desar-la on calgui. Però això, que és fàcil de dir, és una mica més complicat de fer en realitat perquè la càmera no fa les fotos immediatament, sinó que es pren el seu temps (ha d'enfocar, disparar i enviar els píxels de la imatge).

Per aquest motiu, l'ordre enviada a la càmera perquè faci la foto, `takePicture()`, s'executa immediatament i no retorna la foto (la definició del mètode és `void`), sinó que aquesta es rep més endavant en un objecte de tipus `PictureCallback` que hem de crear prèviament perquè rebi la foto i, en aquest cas, la desi.

Aquest comportament és el que s'anomena una **crida asíncrona**, en la qual els resultats es reben més tard, en un altre lloc. Aquestes crides són freqüents quan accedim a dispositius i xarxes.

Més concretament, `takePicture()` rep tres paràmetres que són tres objectes amb les següents finalitats:

- `Camera.ShutterCallback shutter`: l'objecte que s'ha de cridar just quan la càmera fa la foto. Per defecte aquest objecte fa el so del "clic"; si li asssigneu *null* aleshores no farà aquest clic, i si hi afegiu el vostre codi podeu redefinir el seu comportament.
- `Camera.PictureCallback raw`: rep la foto que ha fet la càmera sense comprimir. Pot ser útil per emmagatzemar la foto en altres formats que no siguin JPEG, com ara en PNG. També pot ser *null*.
- `Camera.PictureCallback jpeg`: rep la foto comprimida en format JPEG, per emmagatzemar-la o fer-ne qualsevol altra operació. També pot ser *null*.

En aquest exemple treballareu amb els paràmetres `shutter` i `jpeg`, perquè el paràmetre `raw` és més complicat de fer servir. Creareu un *shutter* buit per tal de gestionar el clic i veure quin és el seu format.

Comenceu amb la primera versió de l' `onClickFoto()` per anar veient pas per pas com afegir les diferents funcionalitats:

```
1  public void onClickFoto(View v) {
2        camera.takePicture(null, null, null);
3      }
```

Si executeu la vostra aplicació amb aquesta versió del mètode, veureu que quan cliqueu el botó *Foto* l'aplicació queda com aturada. Això és perquè la càmera atura la previsualització quan fa una foto, i aleshores, tot i que fa la foto, ni es sent el clic, ni es recull la foto, ni es continua amb la vista prèvia.

Canvieu el contingut de l'`onClickFoto()` per sentir el clic i veure un *toast* per comprovar el que està passant. Fixeu-vos que a la crida `camera.takePicture()` canvia el primer argument per tal que es faci servir el `ShutterCallback`:

```
1  public void onClickFoto(View v) {
2         // Amb aquest objecte s'activa el clic en fer la foto i es mostra
3         // un toast per comprovar que s'ha activat
4         Camera.ShutterCallback shutterCB = new Camera.ShutterCallback() {
5             @Override
6             public void onShutter() {
7                 Toast.makeText(getApplicationContext(), "S'ha disparat!",
8                         Toast.LENGTH_LONG).show();
9             }
10        };
11
12       camera.takePicture(shutterCB, null, null);
13    }
```

Si proveu d'executar l'aplicació i cliqueu el botó, sentireu el clic i veureu el *toast*, però res més. Encara queda feina per fer.

La manera més senzilla d'emmagatzemar una foto és amb el `PictureCallback jpeg`, que dóna els píxels de la imatge i amb unes quantes instruccions es pot desar.

A més, amb aquest objecte mostrareu la foto que s'ha pres al petit *ImageView* que hi ha a la vostra aplicació; i finalment també fareu servir aquest nou objecte per tornar a activar la vista prèvia de la càmera. El següent codi s'ha d'afegir a l'`onClickFoto()`, després del `ShutterCallback` que heu escrit abans:

```
Camera.PictureCallback jpegCB = new Camera.PictureCallback() {
            // Aquest mètode és cridat quan la foto està feta
            @Override
            public void onPictureTaken(byte[] data, Camera cam) {
  // A "data" arriben les dades de la foto
                if (data != null) {
                    // Converteix els píxels en bitmap
                    Bitmap bm = BitmapFactory.decodeByteArray(data, 0, data.
                        length);
                    // Mostra la imatge a l'ImageView de l'aplicació
                    ImageView miniatura = (ImageView) findViewById(R.id.
                        miniatura);
                    miniatura.setImageBitmap(bm);

                    //Crea el directori fotos en cas de no existir
                    File directori = new File(Environment.
                        getExternalStorageDirectory().toString() + "/fotos/");
                    if (!directori.exists()) {
                        directori.mkdir();
                    }

                    /* Desa la imatge en format JPG */
                    try {
                        // Genera un nom únic per la imatge
                        String nomFitx = directori.getAbsolutePath() + "/" +
                            UUID.randomUUID().toString() + "-foto.jpg";

                        // Generem la sortida a partir del nom del fitxer
                        FileOutputStream fos = new FileOutputStream(nomFitx);
                        // Comprimeix la imatge com a JPG
                        bm.compress(Bitmap.CompressFormat.JPEG, 85, fos);
                        // Mostra el nom del fitxer on s'ha desat
                        Toast.makeText(getApplicationContext(), nomFitx, Toast.
                            LENGTH_SHORT).show();
                        // Alliberem l'OutputStream
                        fos.flush();
                        fos.close();

                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
                // Torna a activar la vista prèvia de la càmera
                camera.startPreview();
            }
        };
```

Si ens fixem amb el codi del mètode `onPictureTaken` podem diferenciar els següents blocs de codi:

```
// Converteix els píxels en bitmap
                    Bitmap bm = BitmapFactory.decodeByteArray(data, 0, data.
                        length);
                    // Mostra la imatge a l'ImageView de l'aplicació
                    ImageView miniatura = (ImageView) findViewById(R.id.
                        miniatura);
                    miniatura.setImageBitmap(bm);
```

Amb aquest codi obtenim el `Bitmap` a partir de l'argument `data` del mètode, i posteriorment l'assignem al nostre `ImageView`. Amb aquestes instruccions ja

tindrem la representació en pantalla de la imatge i disposarem del `Bitmap` per després guardar-lo a l'emmagatzemament extern.

```
//Crea el directori fotos en cas de no existir
                    File directori = new File(Environment.
                        getExternalStorageDirectory().toString() + "/fotos/");
                    if (!directori.exists()) {
                        directori.mkdir();
                    }
```

Hem d'escollir a quin directori guardarem la imatge, amb les línies anteriors estem seleccionant el directori "fotos" que es trobarà a l'emmagatzemament extern per defecte (habitualment */storage/sdcard/*).

El condicional comprovarà si no existeix aquest directori i, en cas així sigui, el crearà amb la comanda `mkdir`.

```
/* Desa la imatge en format JPG */
                    try {
                        // Genera un nom únic per la imatge
                        String nomFitx = directori.getAbsolutePath() + "/" +
                            UUID.randomUUID().toString() + "_foto.jpg";

                        // Generem la sortida a partir del nom del fitxer
                        FileOutputStream fos = new FileOutputStream(nomFitx);
                        // Comprimeix la imatge com a JPG
                        bm.compress(Bitmap.CompressFormat.JPEG, 85, fos);
                        // Mostra el nom del fitxer on s'ha desat
                        Toast.makeText(getApplicationContext(), nomFitx, Toast.
                            LENGTH_SHORT).show();
                        // Alliberem l'OutputStream
                        fos.flush();
                        fos.close();

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

El mètode `Bitmap.compress()` rep el format amb el qual es vol desar la imatge, la seva qualitat i el fitxer on es desarà. El paràmetre de la qualitat controla si la imatge s'emmagatzema amb una mida mínima i una qualitat mínima (si qualitat = 0), o amb una mida màxima i qualitat (qualitat = 100), o qualsevol valor intermedi.

Amb aquestes instruccions, el que esteu fent en primer lloc és obtenir un nom de fitxer únic (gràcies al mètode `randomUUID()`) que estigui ubicat al directori "fotos" que hem seleccionat al codi anterior. Tot seguit, es crea un fitxer de sortida (`FileOutputStream fos`) per poder desar la imatge. Amb `Bitmap.compress()` es comprimeix i es desa la imatge al fitxer, i finalment es mostra un *toast* per veure el nom del fitxer.

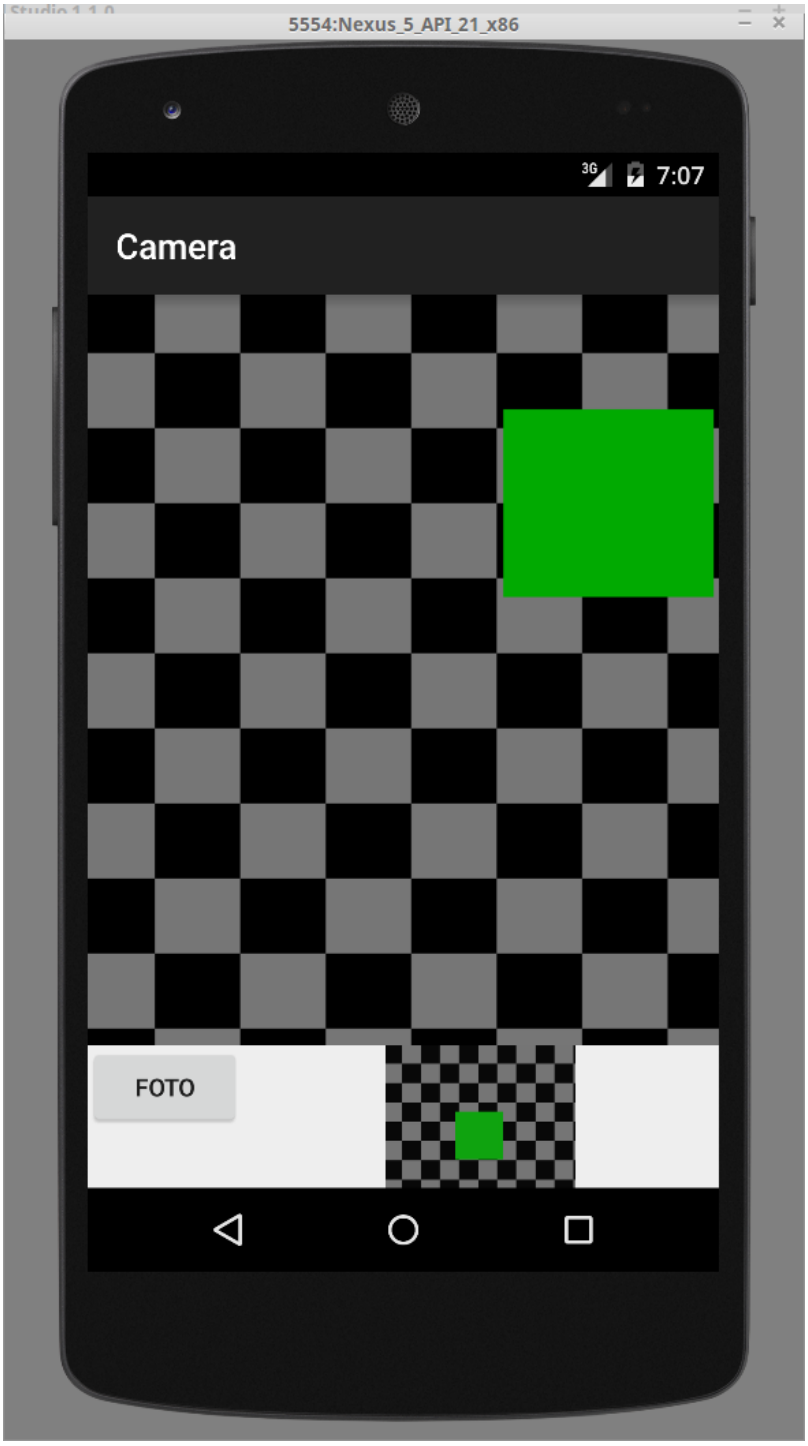
Finalment, fem un `flush()` i un `close()` de l'`OutputStream`, és a dir, forcem l'escriptura de dades que puguin quedar al *buffer*, i tanquem i alliberem el recurs.

Ara que ja ho teniu tot preparat, heu de canviar la crida a `camera.takePicture()` perquè faci servir aquest nou objecte:

```
camera.takePicture(shutterCB, null, jpegCB);
```

Si proveu d'executar l'aplicació i fer una foto, sentireu el clic i veureu els dos *toast*, amb la qual cosa s'haurà desat la vostra imatge. Podeu veure una captura de la pantalla a la figura 2.3.

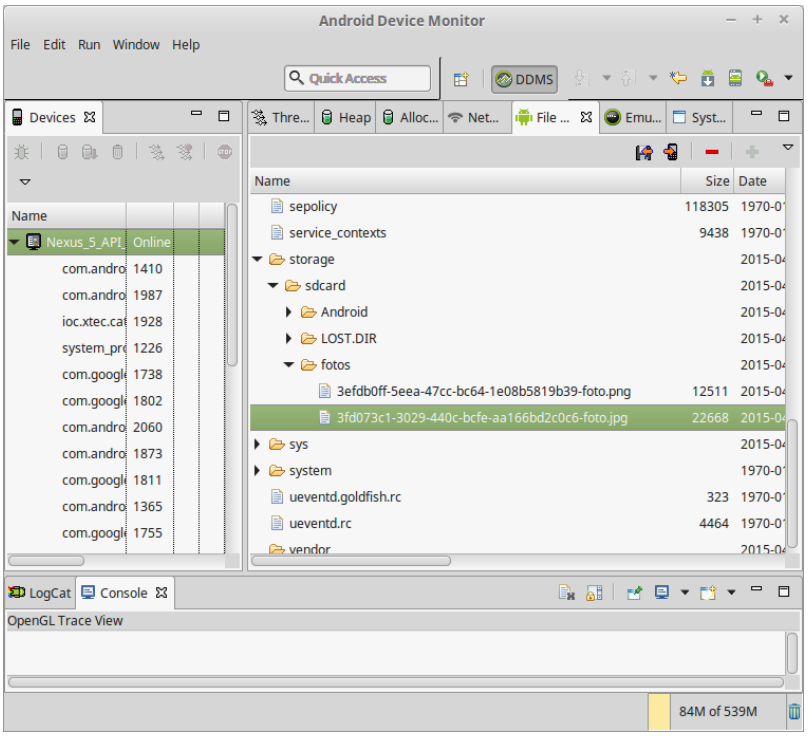
Figura 2.3. Aplicació de fotos



Per tal d'accedir a la imatge podeu obrir el monitor de l'emulador Android anant a *Tools/Android/Android Device Monitor*, on heu d'accedir a la pestanya *File Explorer* per tal de veure els fitxers del dispositiu virtual Android.

Aneu a la carpeta */storage/sdcard/fotos*, tal com es veu a la figura 2.4. En aquesta carpeta hi trobareu les imatges que ha capturat la càmera. A la part superior de la finestra hi ha un botó amb un disquet i una fletxa al damunt que serveix per transferir un fitxer de l'emulador al vostre ordinador. Proveu de copiar alguna de les imatges per comprovar que la foto correspon a la que heu pres amb la càmera.

FIGURA 2.4. Imatges capturades al sistema d'arxius de l'emulador



Per guardar la foto en format PNG només haurem de canviar el nom del fitxer i el `Bitmap.CompressFormat`; així, el codi del `PictureCallback` quedaria de la següent manera:

JPEG i PNG

Alguns formats d'imatge, com ara el JPEG, permeten desar amb més o menys qualitat les imatges. PNG sempre desa amb màxima qualitat i, per tant, el paràmetre *qualitat* s'ignora.

```
1  Camera.PictureCallback jpegCB = new Camera.PictureCallback() {
2              // Aquest mètode és cridat quan la foto està feta
3              @Override
4              public void onPictureTaken(byte[] data, Camera cam) {
5                  // A "data" arriben les dades de la foto
6                  if (data != null) {
7                      // Converteix els píxels en bitmap
8                      Bitmap bm = BitmapFactory.decodeByteArray(data, 0, data.
                           length);
9                      // Mostra la imatge a l'ImageView de l'aplicació
10                     ImageView miniatura = (ImageView) findViewById(R.id.
                           miniatura);
11                     miniatura.setImageBitmap(bm);
12
13                     //Crea el directori fotos en cas de no existir
14                     File directori = new File(Environment.
                           getExternalStorageDirectory().toString() + "/fotos/");
15                     if (!directori.exists()) {
16                         directori.mkdir();
17                     }
18
19                     /* Desa la imatge en format PNG */
20                     try {
21                         // Genera un nom únic per la imatge
22                         String nomFitx = directori.getAbsolutePath() + "/" +
                               UUID.randomUUID().toString() + "_foto.png";
23
24                         // Generem la sortida a partir del nom del fitxer
25                         FileOutputStream sort = new FileOutputStream(nomFitx);
26                         // Comprimeix la imatge com a JPG
27                         bm.compress(Bitmap.CompressFormat.PNG, 100, sort);
28                         // Mostra el nom del fitxer on s'ha desat
29                         Toast.makeText(getApplicationContext(), nomFitx, Toast.
                               LENGTH_SHORT).show();
30                         // Allibera l'OutputStream
```

```
                        sort.flush();
                        sort.close();

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

                // Torna a activar la vista prèvia de la càmera
                camera.startPreview();
            }
        };
```

Amb unes senzilles operacions heu vist com convertir dades multimèdia d'un format a un altre.

2.2 Mostrar imatges de la targeta de memòria

A banda de mostrar imatges que s'hagin afegit com a recursos de l'aplicació, Android permet visualitzar les imatges que hi ha emmagatzemades a la memòria del dispositiu, com per exemple a la targeta de memòria, que és on s'acostumen a emmagatzemar les imatges capturades amb la càmera de fotos o descarregades d'Internet.

Podeu descarregar el codi corresponent a aquesta activitat en l'annex anomenat "Imatges de la targeta de memòria" de la secció "Annexos".

Així doncs, en aquest exemple veureu com es pot triar una imatge emmagatzemada a la targeta SD del dispositiu, que és on es guarden les fotografies fetes amb la càmera, per exemple. Comenceu un nou projecte Android. En primer lloc afegiu el permís `<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>` abans de l'etiqueta `application` del fitxer AndroidManifest.xml i prepareu la interfície de l'aplicació perquè inclogui un ***ImageView*** que us mostrarà la imatge.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@mipmap/ic_launcher" />

</LinearLayout>
```

Al mètode `onCreate()` de l'activitat s'hi han d'afegir les següents instruccions, que el que fan és engegar una nova activitat que permet triar una imatge. A més, es pot recollir el resultat d'aquesta activitat:

```
Intent i = new Intent(Intent.ACTION_PICK, android.provider.MediaStore.Images.
    Media.EXTERNAL_CONTENT_URI);
startActivityForResult(i, ACTIVITAT_SELECCIONAR_IMATGE);
```

Com veieu, es fa servir una constant per identificar l'activitat; cal declarar-la com a constant de la classe (és a dir, abans de l'`onCreate()`):

```
1 private static final int ACTIVITAT_SELECCIONAR_IMATGE = 1;
```

El valor concret que li doneu a la constant no és important, mentre no assigneu el mateix valor a més d'una constant que es faci servir per engegar activitats noves.

Amb tot això, quan s'obre la vostra aplicació immediatament s'inicia una activitat que us deixa triar alguna foto de la targeta SD, tal com es veu a la figura 2.5.

Figura 2.5. Activitat de selecció d'imatges

Un cop trieu una de les imatges, aquesta activitat es tancarà i retornarà la imatge seleccionada a la vostra activitat, però si ho proveu ara mateix veureu que en comptes de mostrar la imatge seleccionada segurament apareixerà la icona d'Android (imatge per defecte de l'**ImageView**). Per què? Doncs perquè no hem recollit el resultat de l'activitat. Per fer-ho, cal afegir-hi el següent mètode:

```
protected void onActivityResult(int requestCode, int resultCode, Intent intent) {
    super.onActivityResult(requestCode, resultCode, intent);

    switch (requestCode) {
        case ACTIVITAT_SELECCIONAR_IMATGE:
            if (resultCode == RESULT_OK) {
                Uri seleccio = intent.getData();
                String[] columna = {MediaStore.Images.Media.DATA};

                Cursor cursor = getContentResolver().query(seleccio,
                    columna, null, null, null);
                cursor.moveToFirst();

                int indexColumna = cursor.getColumnIndex(columna[0]);
                String rutaFitxer = cursor.getString(indexColumna);
                cursor.close();

                Bitmap imatge = BitmapFactory.decodeFile(rutaFitxer);
                /* Reduïm la imatge per no tenir problemes de visualització
                    .
                   Calculem l'alçada per mantenir la proporció amb una
                       amplada de 1080 píxels
                */
                int alt = (int) (imatge.getHeight() * 1080 / imatge.
                    getWidth());
                Bitmap reduit = Bitmap.createScaledBitmap(imatge, 1080, alt
                    , true);

                // Afegim la imatge reduïda a l'Imageview
                ImageView imageView = (ImageView) findViewById(R.id.
                    imageView1);
                imageView.setImageBitmap(reduit);
            }
    }
}
```

Vegem pas a pas què fa aquest mètode per mostrar la imatge seleccionada. En primer lloc, és important el nom del mètode, `onActivityResult()`, perquè serà cridat quan alguna activitat engegada amb `startActivityForResult()` acabi. El mètode rep tres paràmetres: `requestCode`, que és el codi amb què heu iniciat l'activitat (la constant definida prèviament); `resultCode`, que és el codi d'èxit o error de l'activitat; i finalment, `intent`, que dóna accés a les dades de l'activitat.

En primer lloc es crida el mètode corresponent de la superclasse perquè faci les operacions bàsiques. Tot seguit, s'ha de mirar quin és el codi de l'activitat (perquè poden haver-se'n llançat vàries) amb el `switch`, i comprovar que l'activitat ha tornat amb èxit (que `resultCode` sigui `RESULT_OK`). Si no, us podria aparèixer algun missatge d'error.

Un cop comprovat això ja es pot començar a extreure la imatge. Penseu que l'activitat pren una estructura de dades complexa, una taula (`seleccio`) amb moltes columnes d'informació; n'extraiem la columna `DATA`, que és la que ens interessa. Tot seguit fem servir un **cursor** (una mena d'índex a la taula) per

accedir a la posició que ens interessa (el primer element), i finalment llegim el seu contingut per obtenir el nom del fitxer d'imatge (`rutaFitxer`).

Només queda llegir i carregar la imatge a l'aplicació (variable `imatge`) i accedir a l'***ImageView*** per tal que la mostri; abans, però, la reduirem per no tenir problemes de memòria que impedeixin la seva representació. Ara ja podreu veure la imatge quan la seleccioneu.

2.3 Enregistrar àudio

Els dispositius mòbils poden ser eines útils com a enregistradores de so, perquè estan equipats amb un micròfon i disposen d'espai per emmagatzemar so. Per tal de fer servir aquesta prestació, crearem una aplicació que, quan es premi un botó, començarà a enregistrar so pel micròfon del dispositiu; un cop fet això, permetrà reproduir el so per tal de comprovar que s'ha enregistrat el que es desitjava.

L'**emulador d'Android** no accedeix al micròfon, per la qual cosa si executeu aquesta aplicació a l'emulador es crearà un fitxer d'àudio vàlid però en silenci; cal executar-la, doncs, en un dispositiu per provar-la.

L'enregistrament de so a Android s'aconsegueix mitjançant un objecte ***MediaRecorder***, que serveix per enregistrar àudio i vídeo d'una manera relativament senzilla. Un cop s'hagi desat el fitxer d'àudio, es reproduirà fent servir un objecte de tipus ***MediaPlayer***, que és el complementari del *MediaRecorder*: permet reproduir àudio i vídeo a les aplicacions Android.

Podeu descarregar el codi corresponent a aquesta activitat en l'annex anomenat "Enregistrar àudio" de la secció "Annexos".

El fitxer d'àudio es desarà a la carpeta compartida per música i so, de manera que qualsevol altra aplicació hi tingui accés. Així doncs, els permisos que requereix l'aplicació seran dos: accedir al micròfon i escriure fitxers a l'emmagatzemament extern. Creeu un nou projecte d'Android i afegiu els permisos següents al fitxer **AndroidManifest.xml**:

```
1 <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
2 <uses-permission android:name="android.permission.RECORD_AUDIO"/>
```

D'altra banda, la interfície d'aquesta aplicació és molt senzilla: només cal crear dos botons, un per iniciar o aturar la gravació i un altre per reproduir l'arxiu de so. El **layout.xml** ha de ser el següent:

```
1  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools" android:layout_width="
           match_parent"
3      android:layout_height="match_parent" android:paddingLeft="@dimen/
           activity_horizontal_margin"
4      android:paddingRight="@dimen/activity_horizontal_margin"
5      android:paddingTop="@dimen/activity_vertical_margin"
6      android:paddingBottom="@dimen/activity_vertical_margin" tools:context=".
           MainActivity">
7
8      <ToggleButton
9          android:id="@+id/botoGravar"
10         android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:textOff="Gravar"
        android:textOn="Aturar"
        android:text="ToggleButton"
        android:onClick="onClickBotoGravar"/>

    <Button
        android:id="@+id/botoReproduir"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="15dp"
        android:layout_toRightOf="@+id/botoGravar"
        android:text="Reproduir"
        android:onClick="onClickBotoReproduir"/>

</RelativeLayout>
```

Per gravar fareu servir un **ToggleButton**, que té una llumeneta que s'encén quan el botó està "activat". En aquest cas el fareu servir per indicar que s'està gravant. Com podeu veure, cada botó té associat un mètode diferent que s'executarà quan el cliqueu (propietat `onClick`).

El funcionament general de l'aplicació serà el següent: amb dues variables booleanes es controlarà si l'aplicació està gravant o no i si està reproduint o no l'arxiu d'àudio. També es necessiten atributs de l'activitat per recordar el nom del fitxer d'àudio i els objectes **MediaRecorder** i **MediaPlayer** que es faran servir. Al mètode `onCreate()` de l'activitat es definirà el nom del fitxer d'àudio. Per tant, el començament del codi de l'aplicació serà:

```
private static String mNomFitxer = null;

    private MediaRecorder mRecorder = null;
    private MediaPlayer mPlayer = null;
    private boolean mGravant = false;
    private boolean mReproduint = false;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // Generar el nom del fitxer
        mNomFitxer = Environment.getExternalStorageDirectory() + "/gravacio.3gp
            ";

    }
```

A la secció "Gravació en altres formats i codificacions" veurem com emmagatzemar el so en altres formats populars, com ara l'**MP4**.

Com podeu veure, l'extensió de l'arxiu d'àudio és **3gp**, que és un format d'àudio i vídeo molt habitual per a dispositius mòbils. Quan es creï el ***MediaRecorder*** s'haurà d'especificar que el format sigui aquest.

La primera funcionalitat que afegirem és la gravació amb el micròfon. Quan l'usuari de l'aplicació clica el botó *Gravar*, es crida el mètode `onClickBotoGravar()` tal com heu vist al *layout* de l'aplicació. D'altra banda, aquest mètode comprova l'estat actual (gravant o no, amb l'atribut `mGravant`) i segons el seu valor atura la gravació (cridant el mètode `aturaGravacio()`) o l'engega amb el mètode `comencaGravacio()`. Finalment, canvia a l'altre estat (si estava gravant deixa de gravar i viceversa).

```
public void onClickBotoGravar(View view) {
        if (mGravant) {
            aturaGravacio();
        } else {
            comencaGravacio();
        }
        // Canvia a l'altre estat
        mGravant = !mGravant;
    }
```

El mètode `comencaGravacio()` té com a tasques principals crear el **MediaRecorder**, configurar-lo perquè gravi el so del micròfon al fitxer i format desitjat, i finalment posar en marxa la gravació. Els paràmetres més importants que cal indicar al MediaRecorder són:

- Font d'àudio amb el mètode `setAudioSource()`, en aquest cas el micròfon. Hi ha altres valors possibles, els més interessants dels quals són `MediaRecorder.AudioSource.VOICE_UPLINK` (gravació del so que envia l'usuari en una conversa telefònica) i `MediaRecorder.AudioSource.VOICE_DOWNLINK` (gravació del so que rep l'usuari en una conversa telefònica, és a dir, el que diu l'altre interlocutor).
- Format del fitxer d'àudio que es gravarà amb el mètode `setOutputFormat()`, en aquest cas 3GP (`MediaRecorder.OutputFormat.THREE_GPP`). Altres valors interessants són `MediaRecorder.OutputFormat.AAC_ADTS`, un format d'alta qualitat sovint utilitzat per a l'àudio de pel·lícules, o `MediaRecorder.OutputFormat.MPEG_4`, per guardar l'àudio en format MP4, que és un format suportat per molts dispositius i programes.
- Nom del fitxer d'àudio que es gravarà, mitjançant `setOutputFile()`.
- Codificació d'àudio, mitjançant `setAudioEncoder()`. En aquest cas es fa servir `MediaRecorder.AudioEncoder.AMR_NB`, però hi ha d'altres codificacions amb més qualitat, com ara `MediaRecorder.AudioEncoder.AMR_WB`, o bé `MediaRecorder.AudioEncoder.AAC`.

Codificació i compressió

No heu de confondre el format de fitxer multimèdia (que només indica de quina manera s'emmagatzema al disc), amb la codificació o compressió, que indica de quina manera es comprimeixen l'àudio i el vídeo per tal que no ocupi molta memòria. Molts formats (també anomenats contenidors) suporten diferents codificacions.

```
private void comencaGravacio() {
        // Crea el MediaRecorder i especifica la font d'audio, el format
        // de sortida i el fitxer, i el codificador d'audio
        mRecorder = new MediaRecorder();
        mRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
        mRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
        mRecorder.setOutputFile(mNomFitxer);
        mRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);

        // En enllestir la gravació poden sorgir problemes, per tant cal
        // preveure excepcions
        try {
            mRecorder.prepare();
        } catch (IOException e) {
            e.printStackTrace();
        }

```

```
        // Si s'ha pogut disposar tot correctament, es comença a gravar
        mRecorder.start();
    }
```

D'altra banda, per aturar la gravació s'ha d'aturar el MediaRecorder i alliberar-lo perquè alliberi el micròfon i el deixi a l'abast d'altres aplicacions:

```
private void aturaGravacio() {
        mRecorder.stop();
        mRecorder.release();
        mRecorder = null;
    }
```

Pel que fa a la reproducció del fitxer d'àudio que s'ha gravat, el mètode `onClickBotoReproduir()` és molt semblant a `onClickBotoGravar()`, ja que mira si ja s'està reproduint el fitxer o no, i segons sigui aquest valor atura o engega la reproducció.

```
public void onClickBotoReproduir(View v) {
        if (mReproduint) {
            aturaReproduccio();
        } else {
            comencaReproduccio();
        }
        // Canvia a l'altre estat
        mReproduint = !mReproduint;
    }
```

Els següents dos mètodes fan la feina d'iniciar i aturar la reproducció del so. El primer, `comencaReproduccio()`, crea un *MediaPlayer*, especifica el fitxer multimèdia que farà servir amb `setDataSource()`, prepara el reproductor i el posa en marxa. Com que aquest procés pot fallar (per exemple, si el fitxer no està disponible), s'han d'encerclar les operacions amb un `try... catch`.

D'altra banda, per aturar la reproducció només cal alliberar el *MediaPlayer*, que és el que fa el mètode `aturaReproduccio()`.

```
private void comencaReproduccio() {
        mPlayer = new MediaPlayer();
        try {
            mPlayer.setDataSource(mNomFitxer);
            mPlayer.prepare();
            mPlayer.start();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private void aturaReproduccio() {
        mPlayer.release();
        mPlayer = null;
    }
```

Ja podeu executar l'aplicació i veure com grava el fitxer **gravacio.3gp** a la carpeta */storage/sdcard/*. A la figura 2.6 podeu veure una captura de pantalla de l'aplicació.

FIGURA 2.6. Aplicació de gravació d'àudio en marxa

2.3.1 Gravació en altres formats i codificacions

Un cop feta l'aplicació per gravar àudio, és molt senzill modificar-la perquè els fitxers de sortida estiguin en altres formats i amb altres codificacions. Per exemple, imagineu que necessiteu fer una gravació amb més qualitat i format MP4. Només heu de fer tres canvis a l'aplicació de gravació d'àudio.

En primer lloc, el fitxer on es gravarà hauria de tenir l'extensió **mp4**. Editeu les línies corresponents a l'`onCreate()` i feu el següent:

```
mNomFitxer= Environment.getExternalStorageDirectory() + "/gravacio.mp4";
```

A banda d'això, s'ha de canviar el format del fitxer i la codificació que s'utilitza (per fer-ne servir una amb més qualitat de so). Aneu al mètode `comencaGravacio()` i editeu-hi les línies següents per especificar el format **MP4** i la codificació **AAC**:

```
mRecorder.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
....
mRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AAC);
```

Fet! Amb només tres línies podeu especificar fàcilment el format del fitxer d'àudio que es gravarà.

2.4 Gravació de vídeo

Com que els dispositius mòbils disposen de càmera i micròfon, també ofereixen la possibilitat de gravar vídeos. Com en el cas de la fotografia, és possible gravar vídeos fent servir l'aplicació de gravació que hi hagi instal·lada o bé programar directament tots els detalls per tal de previsualitzar el vídeo i indicar a la càmera que comenci a gravar. En aquest cas, farem servir l'aplicació de gravació de vídeo perquè acostuma a incloure moltes funcionalitats i els usuaris ja estan acostumats a fer-la servir.

El procediment per activar l'aplicació de gravació des de les vostres aplicacions és l'habitual: executar un ***intent*** amb el tipus de tasca que es necessita i recollir el seu resultat per veure si tot ha funcionat correctament; i, en aquest cas, s'ha gravat el vídeo o, altrament, si s'ha cancel·lat la gravació.

En el cas que s'hagi gravat el vídeo correctament, l'aplicació oferirà la possibilitat de reproduir-lo per tal que l'usuari pugui comprovar el seu contingut. Com és habitual, les vostres aplicacions podrien fer qualsevol altra operació amb el vídeo gravat.

Podeu descarregar el codi corresponent a aquesta activitat en l'annex anomenat "Gravació de vídeo" de la secció "Annexos".

Creeu un nou projecte Android anomenat GravaVideo amb les opcions per defecte i editeu el seu *layout* perquè quedi així:

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

    <Button
        android:id="@+id/botoGravar"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Gravar vídeo"
        android:onClick="onClickBotoGravar" />

    <Button
        android:id="@+id/botoReproduir"
```

```
16          android:layout_width="fill_parent"
17          android:layout_height="wrap_content"
18          android:text="Reproduir"
19          android:enabled="false"
20          android:onClick="onClickBotoReproduir" />
21
22      <VideoView
23          android:id="@+id/visorVideo"
24          android:layout_width="fill_parent"
25          android:layout_height="wrap_content" />
26  </LinearLayout>
```

Aquest projecte conté dos botons, un per indicar que s'inicïi l'aplicació de gravació de vídeo, i un altre per reproduir el vídeo quan es rebi. Aquest segon botó restarà desactivat (`enabled=''FALSE''`) fins que hi hagi un vídeo disponible, com es mostra a la figura 2.7 (és allò que veuríeu si provéssiu d'executar l'aplicació en aquest moment).

FIGURA 2.7. Aplicació de gravació i reproducció de vídeo

Haurem d'afegir el següent permís al fitxer AndroidManifest.xml per tal que l'aplicació tingui permís per llegir a l'emmagatzemament extern:

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

El que ha de fer el botó de gravar és crear i executar un **Intent** que demani l'aplicació de gravació de vídeo que l'usuari tingui instal·lada per defecte.

Aleshores, el començament de l'aplicació haurà de ser com mostra el codi a continuació: es defineix una constant (`INTENT_GRAVAR_VIDEO`) per identificar l'*intent* que engegarà el gravador de vídeo (li podeu assignar un número qualsevol), i un Uri per rebre l'adreça del vídeo.

Un Uri és un *Universal Resource Identifier*, un nom que identifica un recurs, en aquest cas el vídeo gravat.

El mètode `onCreate()` és força estàndard: simplement activa el *layout* que heu creat abans. D'altra banda, el mètode `onClickBotoGravar()` només ha de crear l'*intent* que demana una captura de vídeo i llançar una activitat amb aquest *intent*. Com que se li passa la constant `INTENT_GRAVAR_VIDEO`, quan torni l'aplicació sabrà a què correspon el resultat rebut.

```
public class MainActivity extends ActionBarActivity {

    // Una constant per identificar l'intent de gravar vídeo
    final static int INTENT_GRAVAR_VIDEO = 1;
    // Aquí tornarà l'adreça del vídeo gravat
    Uri uriVideo = null;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void onClickBotoGravar(View view) {
        // Es crea l'intent i es llança
        Intent intent = new Intent(android.provider.MediaStore.
            ACTION_VIDEO_CAPTURE);
        startActivityForResult(intent, INTENT_GRAVAR_VIDEO);
    }
  }
```

Quan l'*intent* que acabeu de llançar torna, es crida el mètode `onActivityResult()`, que rep tres paràmetres: `requestCode`, que és el codi amb el qual l'heu creat (`INTENT_GRAVAR_VIDEO`); `resultCode`, que indica si l'*intent* ha acabat bé (RESULT_OK) o ha estat cancel·lat per l'usuari (`RESULT_CANCELED`), i `data`, que torna les dades de l'*intent*, en aquest cas l'URI del vídeo gravat.

Amb aquesta informació, el mètode comprova que l'*intent* hagi acabat bé, que sigui l'*intent* de gravació de vídeo que s'ha engegat abans, i en cas afirmatiu mostra amb un *toast* l'URI del vídeo i activa el botó de reproducció (que estava desactivat per defecte); si no, mostra un missatge per indicar a l'usuari que la gravació s'ha cancel·lat.

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (resultCode == RESULT_OK) {
```

```
            if (requestCode == INTENT_GRAVAR_VIDEO) {

                uriVideo = data.getData();
                Toast.makeText(this,
                        uriVideo.getPath(),
                        Toast.LENGTH_LONG).show();
                // Activar el botó de reproduir vídeo perquè ja hi ha un vídeo
                Button botoReproduir = (Button) findViewById(R.id.botoReproduir
                    );
                botoReproduir.setEnabled(true);
            }
        } else if (resultCode == RESULT_CANCELED) {
            uriVideo = null;
            Toast.makeText(this,
                    "Gravació cancel·lada!",
                    Toast.LENGTH_LONG).show();
        }
    }
```

Al `resultCode` d'una activitat també s'hi poden fer servir valors propis per indicar diferents resultats.

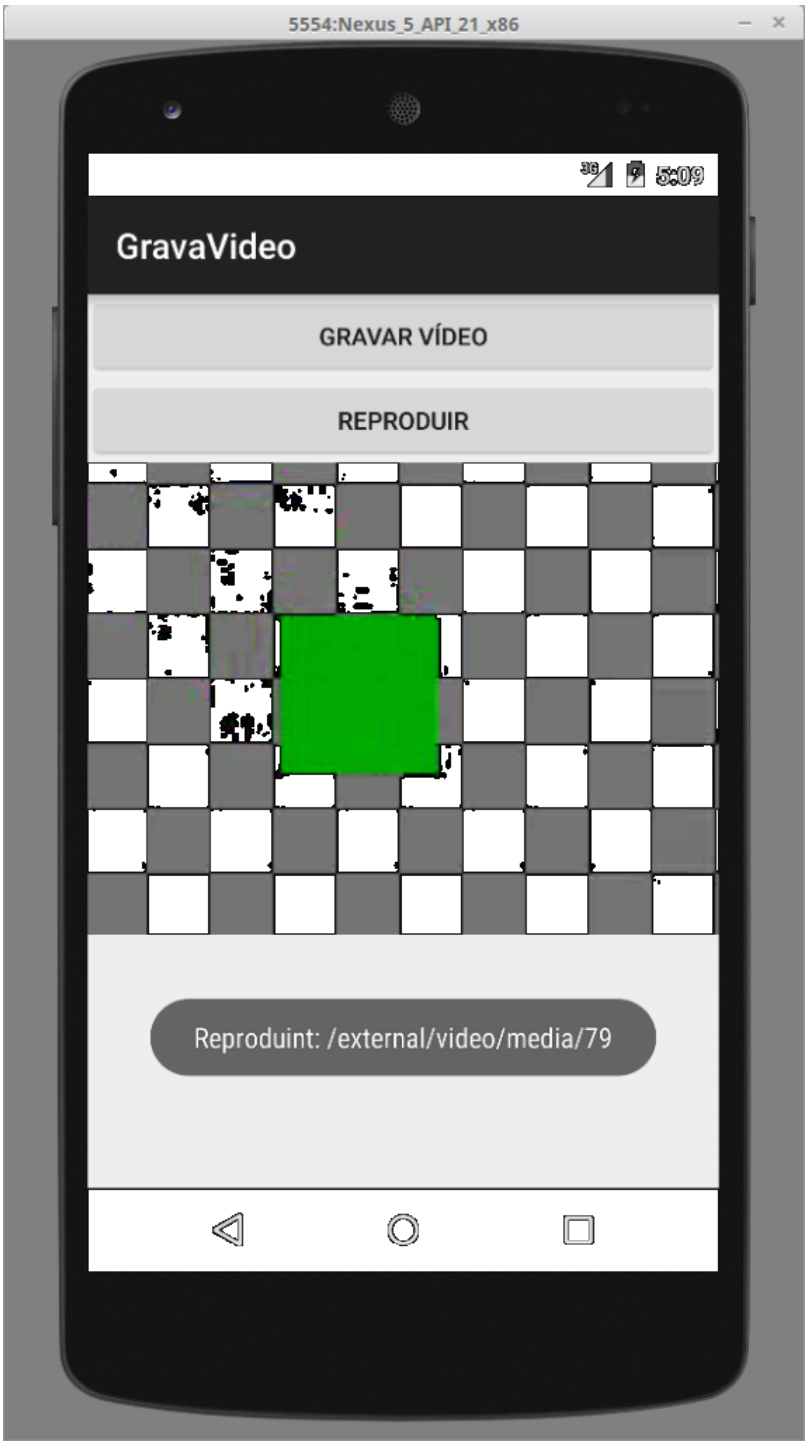
La segona funcionalitat de l'aplicació és mostrar el vídeo que s'ha gravat quan l'usuari premi el botó *Reproduir*, que s'ha activat quan l'aplicació ha gravat un vídeo amb èxit (perquè si aquest botó hagués estat disponible i l'usuari l'hagués clicat, s'hauria produït un error).

La funcionalitat del mètode que s'executa en clicar el botó de reproduir és molt senzilla: simplement mostra un *toast* amb l'URI del vídeo i tot seguit accedeix al **VideoView**, carrega el vídeo i comença la reproducció.

```
public void onClickBotoReproduir(View view) {
        // Primer mostra un toast amb l'URI del vídeo
        Toast.makeText(this,
                "Reproduint: " + uriVideo.getPath(),
                Toast.LENGTH_LONG).show();

        // Després accedeix al visor, carrega el vídeo i el reprodueix
        VideoView visorVideo = (VideoView) findViewById(R.id.visorVideo);
        visorVideo.setVideoURI(uriVideo);
        visorVideo.start();
    }
```

Un cop completada l'aplicació, podeu provar d'executar-la i comprovar el seu comportament. A la figura 2.8 es pot veure la gravació de vídeo engegada (amb l'aplicació estàndard instal·lada per l'usuari del dispositiu).

Figura 2.8. Aplicació de gravació de vídeo en el moment de reproduir la gravació



Un cop el vídeo s'ha gravat, la vostra aplicació us mostra l'URI del vídeo, que és un identificador simbòlic que permet accedir-hi, com es veu a la figura 2.8. No es tracta pas de l'adreça del fitxer de vídeo, que per defecte es gravarà a la carpeta */storage/sdcard/DCIM/Camera/* del dispositiu amb un nom com ara VID_20150101_121310.mp4, generat automàticament amb la data i l'hora.

Comproveu-ho fent servir el monitor de l'emulador: *Tools/Android/Android Device Monitor*.

Com podeu comprovar també, l'aplicació permet gravar més d'un vídeo; com que feu ús de l'aplicació estàndard de gravació, cada nou vídeo que graveu rep un nom diferent (amb la data i l'hora) i, com podeu veure a les *toast* que van sortint, una URI diferent per tal que l'aplicació els pugui distingir.

Tot i que feu servir l'aplicació estàndard amb la seva configuració per defecte, també és possible passar-li alguns paràmetres per controlar determinats aspectes del vídeo generat, com ara la qualitat, la seva durada màxima i la mida màxima del vídeo gravat. Aquestes opcions s'han d'especificar un cop creat **l'intent**, però abans d'executar-lo:

```
1  public void onClickBotoGravar(View view) {
2          // Es crea l'intent i es llança
3          Intent intent = new Intent(android.provider.MediaStore.
               ACTION_VIDEO_CAPTURE);
4          // Qualitat del vídeo: 1 és alta, 0 baixa
5          intent.putExtra(MediaStore.EXTRA_VIDEO_QUALITY, 1);
6          // Durada màxima del vídeo en segons
7          intent.putExtra(MediaStore.EXTRA_DURATION_LIMIT, 10);
8          // Tamany màxim del vídeo en bytes
9          intent.putExtra(MediaStore.EXTRA_SIZE_LIMIT, 100000);
10         // Començar la gravació
11         startActivityForResult(intent, INTENT_GRAVAR_VIDEO);
12     }
```