Programació multimèdia i dispositius mòbils

CFGS.DAM.M08/0.15

Desenvolupament d'aplicacions multiplataforma

Generalitat de Catalunya
Departament d'Educació

ioc
institut obert
de catalunya

Aquesta col·lecció ha estat dissenyada i coordinada des de l'Institut Obert de Catalunya.

Coordinació de continguts
Eduard García Sacristán
Miguel Ángel Lozano Márquez

Redacció de continguts
Joan Climent Balaguer
Eduard García Sacristán
Samir Kanaan

Primera edició: setembre 2015

Dipòsit legal: DL B 12719-2016

Introducció

Si hi ha un mercat que ha revolucionat el món de la informàtica en els darrers cinc anys, aquest ha estat el de la telefonia mòbil i els dispositius portables. Si bé abans els telèfons ja eren programables, aquests eren considerats com unes plataformes minoritàries. Des de l'aparició de l'iPhone (únicament el 2007) tot ha sigut un canvi vertiginós on totes les companyies de tecnologia del món (de maquinari i programari) han volgut prendre posició. Tant és així que en aquests anys, a allò que semblava impossible se li ha donat un nom: l'era post-PC.

Actualment el mercat de dispositius portables ven més unitats que els ordinadors personals (en part pel període de renovació més curt) i és una molt bona oportunitat per als desenvolupadors que poden crear programes i distribuir-los de forma senzilla a través de les botigues d'aplicacions que estan integrades en tots aquests sistemes.

Android és el sistema operatiu més popular i amb una participació més àmplia en el mercat de dispositius mòbils i multimèdia. La programació per a sistemes Android es realitza en el llenguatge Java amb una sèrie de biblioteques i API proporcionats per la pròpia Google (desenvolupadora d'Android). La programació d'aplicacions per a Android, d'aplicacions que facin servir continguts multimèdia i videojocs, són tres àrees diferents que fan servir diferents classes del sistema.

En la unitat "Introducció a la programació de dispositius mòbils" veureu una introducció al sistema operatiu Android i les característiques fonamentals que té la programació per a aquest sistema. Veureu també l'entorn de desenvolupament en el qual programareu per a Android i desenvolupareu el vostre primer programa.

En la unitat "Programació avançada i comunicacions" veureu aspectes més complexos de la programació en Android, com ara la persistència de dades amb l'ús de bases de dades. Com que els dispositius mòbils solen tenir una connexió permanent a les xarxes sense fils, la programació de comunicació en xarxa serà un tema que cal veure amb detall.

En la unitat "Programació multimèdia" veureu els diferents aspectes de la programació del dispositiu que faci servir les seves capacitats multimèdia tant per reproduir (imatges, àudio i vídeo) com per capturar amb els objectes del dispositiu (càmera, micròfon...).

En la unitat "Desenvolupament de jocs per a dispositius mòbils" veureu una introducció a la programació de videojocs per a dispositius mòbils, dissenyant les diferents seccions de l'arquitectura d'un joc i implementant-les.

Per estudiar 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, en les activitats que es proposen sempre s'ha procurat donar un enfocament pràctic.

Resultats d'aprenentatge

En finalitzar aquest mòdul l'alumne/a:

Desenvolupament d'aplicacions per dispositius mòbils

1. Aplica tecnologies de desenvolupament per a dispositius mòbils avaluant les seves característiques i capacitats.
2. Desenvolupa aplicacions per a dispositius mòbils analitzant i fent servir les tecnologies i llibreries específiques.

Programació multimèdia

1. Desenvolupa programes que integren continguts multimèdia analitzant i fent servir tecnologies i llibreries específiques.

Desenvolupament de jocs per dispositius mòbils

1. Selecciona i prova motors de jocs analitzant l'arquitectura de jocs 2D i 3D.
2. Desenvolupa jocs 2D i 3D senzills fent servir motors de jocs.

Continguts

Desenvolupament d'aplicacions per dispositius mòbils

Unitat 1

Introducció a la programació de dispositius mòbils

1. Entorns de desenvolupament d'aplicacions mòbils
2. Programació de dispositius mòbils

Unitat 2

Programació avançada i de comunicacions

1. Programació avançada
2. Programació de comunicacions

Programació multimèdia

Unitat 3

Programació multimèdia

1. Programació multimèdia
2. Objectes multimèdia

Desenvolupament de jocs per dispositius mòbils

Unitat 4

Desenvolupament de jocs per dispositius mòbils

1. Programació de jocs en Android
2. Desenvolupament d'un joc

Introducció a la programació de dispositius mòbils

Eduard García Sacristán, Joan Climent Balaguer

Programació multimèdia i dispositius mòbils

Índex

Introducció

Un dels mercats amb major creixement en els darrers anys ha estat el de l'anomenada telefonia intel·ligent. Un dels aspectes fonamentals d'aquesta ha estat l'ampli ventall d'aplicacions disponibles per aquests dispositius, que tenen unes característiques de maquinari avançades. Android és un dels sistemes operatius punters en aquest àmbit i amb el qual és possible crear aplicacions de manera lliure i gratuïta. En aquesta unitat us endinsareu en la programació de dispositius mòbils i multimèdia en Android.

En l'apartat "Entorns de desenvolupament d'aplicacions mòbils" veureu una introducció al món de la programació de dispositius mòbils a l'actualitat, les característiques d'aquest tipus de programació i el sistema Android, i com instal·lar i configurar l'entorn de desenvolupament amb el que treballareu la resta del mòdul. També fareu el vostre primer programa.

En l'apartat "Programació de dispositius mòbils" veureu els conceptes bàsics que s'han de dominar per poder realitzar aplicacions en dispositius mòbils. Estudiareu el cicle de vida de l'aplicació i els components que es fan servir més habitualment en les aplicacions d'Android.

Per seguir els continguts d'aquest mòdul, és convenient anar fent les activitats i els exercicis d'autoavaluació i llegir els annexos (si n'hi ha). 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

En finalitzar aquesta unitat l'alumne/a:

1. Aplica tecnologies de desenvolupament per dispositius mòbils avaluant les seves característiques i capacitats.
    - Analitza les limitacions que planteja l'execució d'aplicacions als dispositius mòbils.
    - Descriu diferents tecnologies de desenvolupament d'aplicacions per dispositius mòbils.
    - Instal·la, configura i utilitza entorns de treball pel desenvolupament d'aplicacions per dispositius mòbils.
    - Descriu configuracions que classifiquen els dispositius mòbils segons les seves característiques.
    - Descriu perfils que estableixen la relació entre el dispositiu i l'aplicació.
    - Analitza l'estructura d'aplicacions existents per dispositius mòbils identificant les claus utilitzades.
    - Realitza modificacions sobre aplicacions existents.
    - Utilitza emuladors per comprovar el funcionament de les aplicacions.
2. Desenvolupa aplicacions per dispositius mòbils analitzant i fent servir les tecnologies i llibreries específiques.
    - Genera l'estructura de classes necessària per l'aplicació.
    - Analitza i utilitza les classes que modelen finestres, menús, esdeveniments i controls per al desenvolupament d'aplicacions gràfiques senzilles.
    - Realitza proves d'interacció usuari-aplicació per optimitzar les aplicacions desenvolupades a partir d'emuladors.
    - Documenta els processos necessaris pel desenvolupament de les aplicacions.

1. Entorns de desenvolupament d'aplicacions mòbils

Fa uns anys, un directiu d'una gran companyia d'informàtica va profetitzar el començament de l'*època post-PC*, un moment en què els usuaris deixaran de fer servir els ordinadors tal com els coneixem ara (bàsicament ordinadors de taula i ordinadors portàtils) per fer les tasques més habituals del dia a dia: comunicar-se (per correu electrònic, missatgeria o videoconferència), navegar per Internet o consumir continguts.

Aquest moment ja ha arribat, i la varietat de dispositius que tenim actualment ens permet realitzar fàcilment totes les nostres tasques diàries. De fet, la línia que separa uns dispositius d'uns altres cada vegada es fa més difícil de definir: els telèfons tenen cada vegada pantalles més grans, al mateix temps que les tauletes tàctils disposen de diferents mides, de forma que la diferència entre un telèfon gran i una tauleta petita és merament il·lusòria (com la diferència entre portàtils i *netbooks*). Per això, els sistemes operatius i la manera de programar-hi s'ha integrat, de forma que tots aquests dispositius mòbils multimèdia comparteixen les mateixes característiques i han integrat els mateixos sistemes i programari.

1.1 El mercat actual d'aplicacions mòbils

Avui en dia la *batalla dels dispositius mòbils* està més d'actualitat que mai. Apple va revolucionar el mercat de la telefonia mòbil amb el seu iPhone, deixant una sèrie de *víctimes* pel camí com van ser Palm, Windows CE i els telèfons de Nokia amb Symbian. Tota la resta del mercat ha estat cercant durant anys l'anomenat *iPhone Killer* (assassí de l'iPhone) o el telèfon que desbancaria el famós telèfon intel·ligent de la companyia de la poma com a telèfon amb més èxit. Finalment, ens hem trobat que no ha existit aquest telèfon, sinó que l'èxit d'altres plataformes (com ara Android, adquirit per Google l'any 2005) ha consistit a establir tot un ecosistema de dispositius compatibles amb què combatre la quota de mercat dels dispositius Apple. Els altres actors d'aquesta competició de mercat (Blackberry de RIM, Firefox OS de Mozilla, Windows Phone de la mà de Microsoft i Ubuntu Touch OS de Canonical, entre d'altres) tenen la difícil tasca de trobar el seu espai en un mercat on cada vegada és més difícil trobar visibilitat.

Tot i no vendre tants telèfons com altres companyies, els beneficis d'Apple superen amb escreix els de les altres companyies.

Actualment, el mercat de fabricants de mòbils està copat amb companyies que obtenen gran part dels seus beneficis de dispositius amb Android. Entre aquestes trobem les companyies que ja desenvolupaven telèfons abans de l'era iPhone com ara Samsung, LG o Sony i companyies que s'han incorporat posteriorment com ara Xiaomi (Mi), Oneplus o Lenovo per exemple.

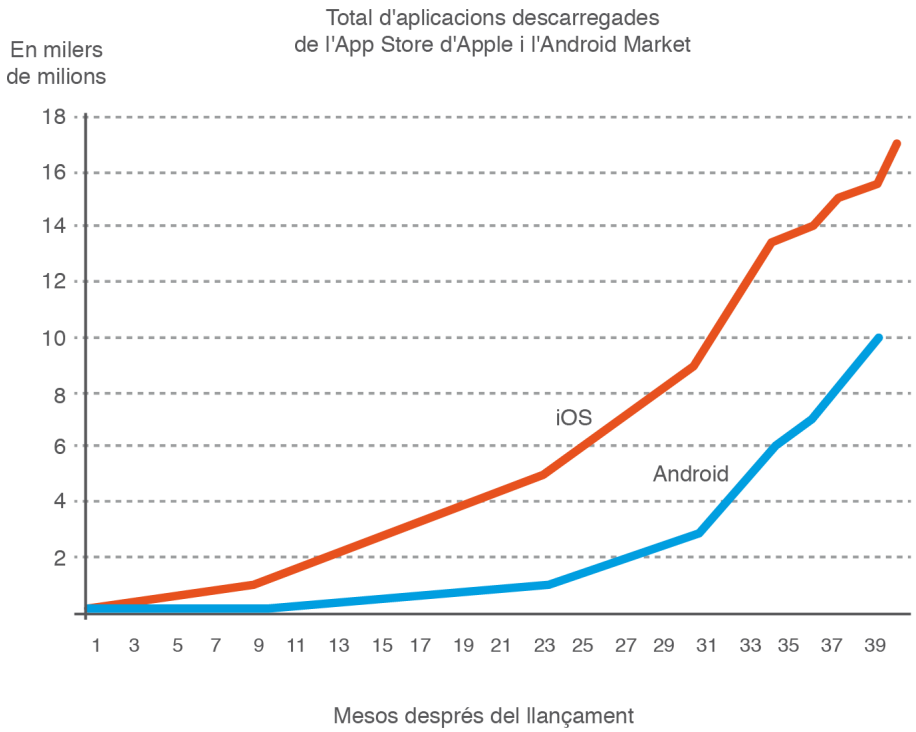
Quant als sistemes operatius dels anomenats telèfons intel·ligents, es pot veure la quota de mercat el desembre de 2014 a la taula 1.1.

Taula 1.1. Quota de mercat dels diferents sistemes operatius en 'smartphones' (desembre de 2014)

Sistema operatiu	Quota de mercat
Android	84,4%
iOS	11,7%
Windows Phone	2,9%
Blackberry	0,5%
Altres	0,6%

Una de les característiques que més va afiançar Apple amb el seu iPhone va ser la inclusió d'un *mercat d'aplicacions* integrat al propi dispositiu, on els usuaris poden cercar, analitzar i comprar aplicacions d'una manera molt senzilla. Aquest ecosistema d'aplicacions s'ha demostrat com un dels valors afegits més importants en els actuals dispositius mòbils, atès que un aparell amb un gran maquinari té una utilitat molt limitada si no ve acompanyat d'un programari que en faci un servei adequat. En aquest sentit, Google va entendre la importància de les aplicacions i ha facilitat la creació d'aplicacions i la seva publicació en llur mercat d'aplicacions. Així, a pesar d'haver llançat la seva plataforma d'aplicacions més tard, la quantitat d'aplicacions descarregades creix a un ritme molt alt, tal com es pot veure a la figura 1.1.

Figura 1.1. Aplicacions descarregades des del llançament de la plataforma



1.1.1 Història dels telèfons intel·ligents

El mercat de la telefonia mòbil va deixar de ser fa temps una promesa per establir-se com una de les principals formes de comunicació al mercat de consum actual. Així, si en un principi els telèfons ens permetien realitzar trucades telefòniques i enviar missatges, a poc a poc s'han anat introduint noves característiques i funcionalitats als dispositius mòbils fins arribar al punt actual, on és possible portar a la butxaca un autèntic ordinador personal.

Les tecnologies de transmissió sense fils **GPRS** (*General packet radio service*, servei de ràdio de paquets general) i **WAP** (*Wireless Application Protocol*, protocol d'aplicacions sense fil), un protocol estàndard per accedir a informació a través de xarxes sense fil, van proveïr unes primeres funcionalitats de navegació web als primers telèfons mòbils. La poca velocitat d'aquelles xarxes i, sobretot, el fet que l'experiència d'usuari no fos prou bona degut a la falta de pantalles de qualitat als telèfons, van fer que aquestes tecnologies (i els serveis creats per a elles) no es popularitzessin en excés.

Al final dels anys noranta, la majoria dels telèfons encara tenien unes funcionalitats molt limitades, i aquells usuaris que tenien necessitats més enllà del que podia oferir el seu telèfon també carregaven amb un dispositiu tipus **PDA** (*Personal Digital Assistant*, assistent digital personal), que oferia funcionalitats avançades, com ara: agenda, pantalles tàctils, expansió amb targetes de memòria, connexió sense fils i una sèrie d'aplicacions pròpies de la plataforma. La primera PDA de la història va ser la Psion Organizer desenvolupada l'any 1984 (figura 1.2). Aquestes PDA portaven un sistema operatiu propi, com per exemple Palm OS, BlackBerry OS o Pocket PC. Les darreres versions d'aquests sistemes integraven capacitats de missatgeria i telefonia. Aquests dispositius es podrien anomenar ja *smartphones* (telèfons intel·ligents).

La primera PDA

La primera PDA de la història va ser la Psion Organizer desenvolupada l'any 1984. (Psion és una companyia que va desenvolupar programes per al *mític* ordinador ZX Spectrum pels volts dels anys vuitanta). Tot i això, la primera vegada que es va fer servir l'expressió "PDA" va ser l'any 1992 amb la presentació del'Apple Newton.

FIGURA 1.2. Psion Organizer, la primera PDA de la història

Un ***smartphone*** (telèfon intel·ligent) és un telèfon mòbil de gamma alta construït sobre una plataforma mòbil amb capacitats i connectivitat superiors a les que té un telèfon convencional.

Encara que el mercat de les PDA estava dominat per l'empresa Palm (fabricant tant del maquinari com del sistema operatiu mòbil), Microsoft va treure al mercat Windows Mobile, basat en la plataforma Pocket PC 2000 que, gràcies a la possibilitat de llicenciar-lo per part d'altres empreses, va ser molt popular i l'any 2007 va obtenir un 42% del mercat.

Justament aquest any 2007, Apple va introduir l'iPhone (després de mesos de rumors i especulacions), que revolucionaria el mercat de telefonia mòbil. Moltes de les seves seves característiques s'han convertit en estàndards de facto a la indústria de telefonia mòbil, com ara:

- Pantalla capacitiva i multitàctil
- GPS
- Teclat en pantalla
- Integració amb alguns serveis de Google, especialment la cerca i Google Maps
- Connexió de dades permanent mitjançant 3G

Posicionament global

GPS (Global Positioning System, sistema de posicionament global) és un sistema de navegació basat en satèl·lits, que proporciona localització i hora en qualsevol tipus d'horatge, en qualsevol lloc de la terra, si es té visió directa de quatre o més satèl·lits GPS.

I un parell de característiques que no estaven a l'iPhone original però que es van afegir més endavant, com ara:

- Aplicacions de tercers
- Una botiga d'aplicacions on trobar totes les aplicacions de forma centralitzada (AppStore)

La resta de la indústria va trigar anys a oferir un producte que pogués competir amb l'iPhone (figura 1.3) tant a nivell de *software* com de *hardware*. Es va produir durant molt de temps una lluita per aconseguir restar quota de mercat a Apple en el sector dels *smartphones*.

Figura 1.3. L'iPhone original

Però el desenvolupament d'Android va començar, de fet, abans que l'iPhone veiés la llum. Google va comprar la companyia Android Inc. el 2005 i la primera distribució d'Android (el sistema operatiu mòbil) es va anunciar el 2007 per l'**Open Handset Alliance**. Des d'aleshores, la quantitat de fabricants de telèfons que fan servir Android com a sistema operatiu del seu maquinari (telèfons i tauletes tàctils, principalment, encara que existeixen multitud de dispositius que fan servir Android) ha crescut de forma exponencial. El novembre de 2011 existien més de 200 milions de dispositius Android, i Google va anunciar que el desembre de 2011 s'activaven més de 700.000 dispositius Android diaris; aquest índex ja arribava als 2 milions anuals el 2013 (font: www.talkandroid.com).

Una altra fita important en l'evolució dels dispositius mòbils va ser la irrupció de l'iPad, un *tablet* (tauleta tàctil) d'Apple que va sortir al mercat el 2010 i en molt poc temps va popularitzar l'ús d'aquest tipus de dispositius.

Qui s'encarrega de crear els estàndards?

L'Open Handset Alliance (que es podria traduir per Aliança d'auriculars oberts) és un consorci de 86 empreses de maquinari, programari i telecomunicacions dedicades a avançar estàndards oberts per a dispositius mòbils.

Un ***tablet*** és un ordinador portàtil, més gran que un telèfon mòbil o una PDA, integrat dins d'una pantalla tàctil plana i principalment operat mitjançant la seva pantalla tàctil (en lloc d'un teclat físic). Generalment, fa servir un teclat virtual en pantalla o un *stylus* per introduir text. El primer *tablet* que va gaudir d'un gran èxit comercial va ser l'iPad d'Apple.

Actualment existeixen diverses solucions de sistemes operatius per a dispositius portàtils. En aquesta llista anomenem les més importants, encara que el mercat està dominat per les dues primeres:

- Android: desenvolupat per Google i l'Open Handset Alliance. La majoria dels fabricants més importants (Samsung, HTC, Motorola, Sony…) tenen telèfons i tauletes que s'executen sobre Android. Té un 45,86% del mercat (dades de desembre de 2014).

- iOS: desenvolupat per Apple, està present únicament als productes de la marca Apple (iPhone, iPad i iPod); així i tot, té una presència del 43,15% al mercat, gràcies a la bona salut de vendes del sector de tauletes.

- BlackBerry OS: desenvolupat per RIM (*Research In Motion*), i present únicament als seus productes; ha baixat molt la seva quota de mercat, fins a l'1% actual.

- Windows Phone: desenvolupat per Microsoft i present fonamentalment als nous telèfons que Nokia va presentar a finals de 2011; la seva quota de mercat és molt petita actualment tot i que ha anat creixent gràcies a la seva major presència al món de les tauletes.

- Firefox OS: desenvolupat per Mozilla, és un sistema operatiu de codi obert que es basa en HTML5. L'impacte d'aquest sistema operatiu ha estat mínim a causa de la poca diversitat d'aplicacions disponibles.

En aquest materials hem decidit fer servir Android (figura 1.4) com a eina de desenvolupament per les següents característiques:

- És un projecte de codi lliure.

- És gratuït, i les seves eines de desenvolupament també ho són.

- Actualment és (juntament amb Apple iOS) la plataforma de dispositius mòbils més popular del mercat, amb una previsió de creixement encara més gran (és probable que en el futur la quota de mercat d'Android sigui encara superior a l'actual).

- El seu desenvolupament es realitza en Java, llenguatge molt popular i de fàcil aprenentatge.

Figura 1.4. Logo d'Android

Per tot això, pensem que Android és la plataforma ideal per introduir-se en el desenvolupament de la programació en dispositius mòbils i multimèdia.

1.1.2 Característiques d'Android

Android té una llicència de codi lliure i està disponible de forma gratuïta per als fabricants per a la seva adaptació. No existeixen configuracions fixes de maquinari i programari, encara que Android suporta les següents característiques:

- Missatgeria: suporta SMS (*Short Message Service*, servei de missatges curts) i MMS (*Multimedia Messaging Service*, servei de missatgeria multimèdia).
- Suport de maquinari: acceleròmetres, càmera, brúixola, sensor de proximitat i GPS.
- Emmagatzematge: fa servir SQLite, una base de dades relacional ultralleugera, per a l'emmagatzematge de dades.
- Formats: suporta multitud de formats multimèdia (imatges, so, vídeo...).
- Connectivitat: suporta xarxes GSM/EDGE, HSDPA, LTE, CDMA, UMTS, Bluetooh, Wi-fi i WiMAX.

En aquests moments, Android es fa servir en multitud de dispositius, com ara telèfons intel·ligents, *tablets*, lectors de llibres electrònics, *netbooks*, reproductors d'MP3 i MP4, televisors, rellotges...

L'Android OS (*sistema operatiu d'Android*) està dividit en cinc seccions. De la capa més baixa a la més alta, són les següents:

- **El kernel de Linux**: sobre el qual està basat Android. Aquesta capa conté tots els *drivers* de dispositius (a baix nivell) per als components de maquinari del dispositiu Android.
- **Llibreries (biblioteques de funcions**): conté el codi que proporciona les característiques principals de l'Android OS. Per exemple, les biblioteques de funcions de Webkit proporcionen funcionalitats de navegació web, i les d'SQLite d'accés a bases de dades.
- **Android Runtime:** està al mateix nivell que les llibreries i proporciona una sèrie de llibreries fonamentals que permeten als desenvolupadors escriure aplicacions d'Android fent servir Java. El *runtime* d'Android també inclou la màquina virtual Dalvik, que permet a cada aplicació Android executar-se en el seu propi procés (que és una instància d'aquesta màquina virtual). Les aplicacions Android estan compilades en executables Dalvik. A partir de la versió 5.0 (Lollipop) s'ha canviat la màquina virtual de Dalvik a ART, incrementant el rendiment general de les aplicacions.
- **Framework d'aplicacions:**proporciona als desenvolupadors les diferents característiques de l'Android OS per tal que les puguin fer servir en les seves aplicacions.

ART vs Dalvik

Mentre que Dalvik compila *just-in-time (JIT)* ART ho fa *ahead-of-time (AOT)*, és a dir, Dalvik compila les aplicacions al moment d'executar-les i ART ho fa en el moment de la instal·lació.

- **Aplicacions:** a la capa més alta, hi trobem les aplicacions que es distribueixen amb el dispositiu Android (com telèfon, contactes, navegador, etc.), així com altres aplicacions que es descarreguen i s'instal·len des de la botiga de Google, *Google Play Store*. Les aplicacions que escriurem estaran situades en aquesta capa.

1.1.3 Google Play

Un dels factors fonamentals que determina l'èxit d'una plataforma mòbil és la quantitat d'aplicacions que ofereix. Això va quedar demostrat amb l'iPhone d'Apple i el seu ecosistema d'aplicacions. De la mateixa manera, el fet que sigui fàcil per als desenvolupadors publicar les seves aplicacions i fàcil per als usuaris comprar-les determinarà la facilitat amb què creixerà el mercat d'aplicacions.

L'agost del 2008, Google va anunciar l'Android Market, el seu propi mercat d'aplicacions, que va ser rebatejat com a Google Play el març de 2012 (https://play.google.com/store). Els usuaris poden descarregar aplicacions de tercers directament des del seu dispositiu (a través de l'aplicació Play Store, preinstal·lada a la majoria de dispositius Android). Google Play disposa d'aplicacions de pagament, així com d'altres completament gratuïtes o gratuïtes amb micropagaments integrats.

El cicle de vida d'una aplicació d'Android es podria definir de la següent manera:

1. Descobriment de l'aplicació, tant des de Google Play (el més habitual) com d'una altra forma.
2. Descàrrega i instal·lació de l'aplicació.
3. Execució de l'aplicació.
4. Actualització de l'aplicació si existeixen versions més modernes (opcional).
5. Desinstal·lació de l'aplicació.

1.2 Dispositius mòbils

En primer lloc, hem de determinar què és programació per a dispositius mòbils i, per tant, què considerem com a dispositiu mòbil. En general, considerem que un dispositiu és *mòbil* si té les següents característiques:

- Té una mida reduïda i un pes lleuger. Per tant, la mobilitat és una de les seves principals característiques. Idealment, és un dispositiu que es pot portar a la butxaca o a una bossa petita i al que es pot accedir ràpidament.

- Té una connexió sense fils contínua en forma de connexió WiFi o connexió a Internet permanent fent ús de les xarxes de telefonia.
- Proveeix una alta capacitat d'interacció a l'usuari mitjançant una pantalla tàctil o una pantalla i un teclat.
- Té capacitat de processament i memòria interna.

Així, considerem dispositius mòbils els telèfons mòbils, els telèfons intel·ligents (*smartphones*), les PDA i els *tablets*. Descartem els ordinadors portàtils com a dispositius mòbils, atès que donats el seu pes i la seva mida són dispositius que no es poden portar a la butxaca.

Actualment existeixen també els dispositius *wearables*, que tenen menors dimensions que els dispositius mòbils i es sincronitzen amb els *smartphones* mitjançant la tecnologia *bluetooth* per afegir funcionalitats al sistema. En aquest sector trobem dispositius com els *smartwatch* o les polseres intel·ligents.

Google ha creat un sistema operatiu basat en Android per a aquest tipus de dispositius, *Android Wear*, que va ser presentat el 2014.

1.2.1 Limitacions en la programació de dispositius mòbils

Els dispositius mòbils tenen unes característiques particulars quant a les seves capacitats de processament i memòria. El fet que aquests dispositius siguin principalment mòbils i, per tant, d'unes mides i pes reduïts, determina les seves característiques de maquinari. Principalment, les seves limitacions, comparant-los amb els dispositius de taula, són les següents:

- La seva capacitat de processament és reduïda.
- La quantitat de memòria és reduïda (RAM i d'emmagatzematge).
- Les dimensions del dispositiu són reduïdes, i per tant el maquinari d'interacció amb l'usuari (pantalla i teclat) també ho és.
- Solen tenir una connexió permanent a Internet, i per tant s'ha de donar especial importància a la seguretat.
- La connexió a Internet, si és mòbil, té una velocitat limitada i es perd amb freqüència.

Totes aquestes característiques es van millorant a poc a poc, i els nous dispositius cada vegada tenen més capacitat de processament, més memòria, pantalles més grans i millors connexions a Internet. Així i tot, sempre aniran per darrere dels ordinadors de taula, i per tant s'han de tenir en compte les seves característiques quan programem sobre aquests dispositius.

1.3 Entorns de desenvolupament

El primer pas per començar a desenvolupar programes per a dispositius mòbils és instal·lar l'**Android SDK** (*Software Development Kit*, equip de desenvolupament de programari) i preparar l'entorn de desenvolupament per primera vegada.

Aquests passos poden canviar amb el temps, i estan especificats a la web d'Android (developer.android.com/sdk/installing.html); per tant, es recomana revisar que el procediment segueix sent vàlid.

Es poden desenvolupar aplicacions per a dispositius mòbils amb Android des de qualsevol sistema operatiu (GNU/Linux, MacOS, MS Windows) i amb diversos entorns de desenvolupament; l'oficial i més popular actualment és l'**Android Studio** tot i que també es poden desenvolupar aplicacions en Eclipse i en Netbeans. Escollirem per als materials desenvolupar amb Ubuntu 14.04 LTS 64 bits (*Long Term Support*, les versions LTS són suportades durant quatre anys per Ubuntu) i Android Studio.

Es recomana la utilització d'un sistema operatiu de 64 bits, Google ens informa que els arxius binaris de 32 bits desapareixeran en un futur. Si us és impossible fer servir una màquina de 64 bits podeu afegir l'opció `-force-32-bit` a *Run/Edit Configurations* a la pestanya de l'emulador o bé fer un `export ANDROID_EMULATOR_FORCE_32BIT=true` des del terminal; si voleu fer persistent aquest darrer pas, podeu crear un fitxer `.profile` al vostre directori *home* i afegir aquí l'`export`.

PhoneGap és un *framework open source* per desenvolupar aplicacions multiplataforma amb *HTML5*, *CSS* i *JavaScript*.

Cal fer menció a més, de l'existència de *frameworks* (entorns de treball) que ens permetran crear aplicacions per Android fent servir altres llenguatges de programació com ara: *C#*, *python* i fins i tot amb una combinació d'*HTML5* + *CSS* + *JavaScript*. Alguns exemples podrien ser: *Xamarin*, *Phonegap*, *dot42*, *Kivy*, etc.

Android Studio

Android Studio és un IDE (*Integrated Development Environment*, entorn de desenvolupament integrat) basat en l'IntelliJ IDEA que ha substituït oficialment a l'Eclipse com a eina per desenvolupar aplicacions.

1.3.1 Preparant l'ordinador

En primer lloc, heu de confirmar que el vostre ordinador compleix els requeriments del sistema per desenvolupar en Android. Hem de dir que les exigències sobre les característiques de la màquina són elevades (especialment per a l'execució del simulador). Els requeriments del sistema poden canviar al llarg del temps. Podeu trobar els requeriments actualitzats al web oficial d'Android: developer.android.com/sdk/requirements.html.

Els requeriments mínims han de ser:

Comuns:

- 2 GB de RAM com a mínim, 4 recomanats
- 2 GB lliures al disc dur
- 1280x800 de resolució de pantalla
- Oracle Java Development Kit (JDK) 7

Linux:

- Escriptori GNOME o KDE
- Llibreria CNU C (glibc) 2.11 o posterior

Windows:

- Microsoft Windows 8/7/Vista/2003 (32 o 64 bits)
- Opcional per a tenir acceleració per *hardware*: processador Intel amb Intel VT-x, Intel EM64T i Execute Disable (XD) bit

Mac OS X:

- Mac OS X 10.8.5 o superior fins a 10.9
- Java Runtime Environment (JRE) 6
- Opcional per a tenir acceleració per *hardware*: processador Intel amb Intel VT-x, Intel EM64T i Execute Disable (XD) bit

Entorn de desenvolupament suportat, Android Studio:

- Android Studio 1.0 o superior
- JDK 7 (Java Development Kit, equip de desenvolupament de Java) com a mínim. Amb el JRE (Java Runtime Environment, entorn d'execució de Java) no és suficient

La forma d'instal·lar l'entorn de desenvolupament també pot canviar amb les noves versions, per tant es recomana fer una ullada a la pàgina oficial d'Android per veure les instruccions d'instal·lació: developer.android.com/sdk/installing.html.

Per instal·lar el JDK a Ubuntu, de nou podeu revisar la darrera informació continguda a la documentació oficial: https://help.ubuntu.com/community/Java

Per defecte, la màquina de Java d'Oracle ja no està distribuïda amb Ubuntu per problemes de llicències. Les versions actuals d'Ubuntu porten instal·lada la implementació OpenJDK.

L'objectiu principal del projecte OpenJDK és produir una implementació de codi obert de la plataforma Java Standard Edition.

Podeu descarregar el JDK des de la pàgina oficial d'Oracle en format .tar.gz o .rpm però per facilitar la instal·lació ho farem des del repositori de webupd8 tal com ens recomanen en la documentació oficial d'Ubuntu.

Així, afegim el repositori ppa amb la següent comanda:

En cas de no disposar de la comanda **add-apt-repository** instal·leu python-software-properties: `sudo apt-get install python-software-properties`

```
sudo add-apt-repository ppa:webupd8team/java
```

Després actualitzarem la llista de repositoris i farem la instal·lació d'*oracle-java8-installer*:

```
sudo apt-get update
  sudo apt-get install oracle-java8-installer
```

Una vegada instal·lat, podrem alternar entre les versions de Java amb la comanda `sudo update-alternatives -config java` on podrem escollir entre l'OpenJDK i el JDK d'Oracle. Podeu comprovar quina versió de Java teniu instal·lada amb la comanda `java -version`.

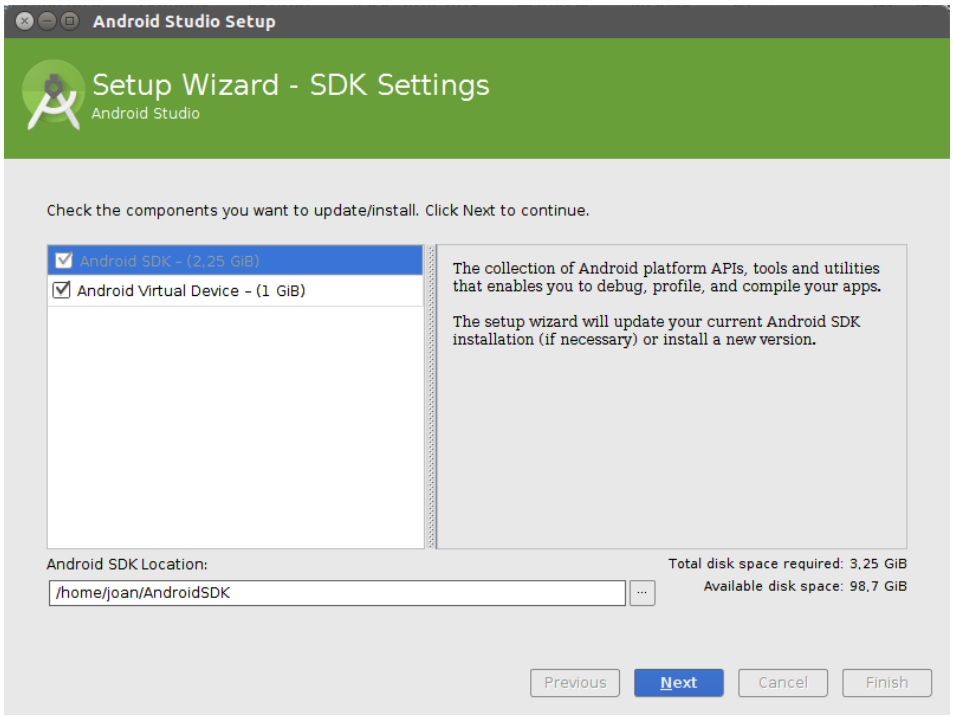
1.3.2 Instal·lant l'Android Studio i l'SDK

Podeu descarregar l'Android Studio des de la següent adreça: http://developer.android.com/sdk/index.html

Obrim un terminal, anem fins al directori de descàrrega i descomprimim el fitxer. Per tal d'executar l'Android Studio hem de canviar el directori a `android-studio/bin` i allà executar `./studio.sh`.

```
usuari@ubuntu:~/$ cd Descargas
  usuari@ubuntu:~/Descargas$ unzip android-studio-ide-*.zip
  usuari@ubuntu:~/Descargas$ cd android-studio/bin
  usuari@ubuntu:~/Descargas/android-studio/bin$ ./studio.sh
```

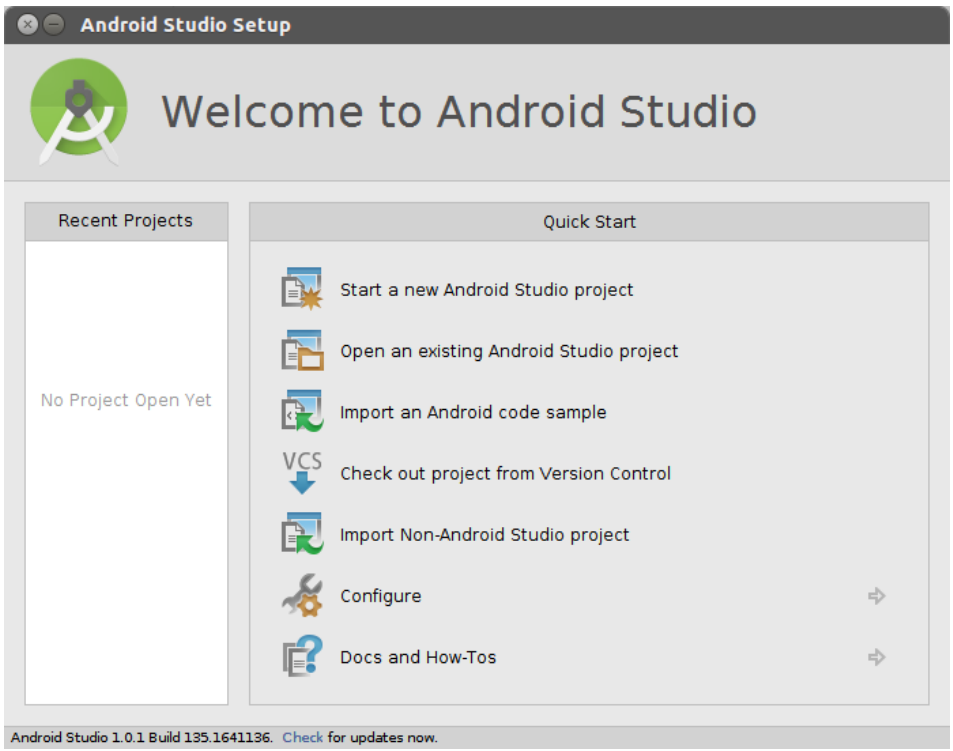
Figura 1.5. Android Studio la primera vegada que s'executa. Instal·lació 'Custom'



Seguint la configuració inicial instal·larem l'Android SDK, la darrera API d'Android i un emulador (figura 1.5), és a dir, tot allò que necessitem per desenvolupar aplicacions. Si ho volem, també disposem d'una versió *stand-alone* de l'SDK per descarregar des de http://developer.android.com/sdk/index.html#Other

Una vegada finalitzat l'assistent hauríem de veure la pantalla de la figura 1.6, a partir d'aquest moment ja podrem crear el nostre primer projecte.

Figura 1.6. Pantalla principal de l'Android Studio



Per tal de millorar el rendiment de l'emulador, si el nostre microprocessador és compatible, seria interessant instal·lar l'*Intel® Hardware Accelerated Execution Manager* seguint les instruccions del següent enllaç: Intel HAXM.

1.3.3 Gestió de l'SDK

L'últim pas en la configuració de l'SDK d'Android és la descàrrega dels components essencials per al nostre entorn de desenvolupament, que farem mitjançant una eina inclosa en el paquet, l'AVD Manager.

L'SDK fa servir una estructura modular que en separa les principals parts, els afegits, les eines, els exemples i la documentació en components instal·lables de forma separada. L'SDK Starter Package (paquet de començament d'SDK) que heu instal·lat inclou únicament la darrera versió de les eines de l'SDK. Podeu descarregar altres components, com les darreres versions de la plataforma d'Android quan es publiquen.

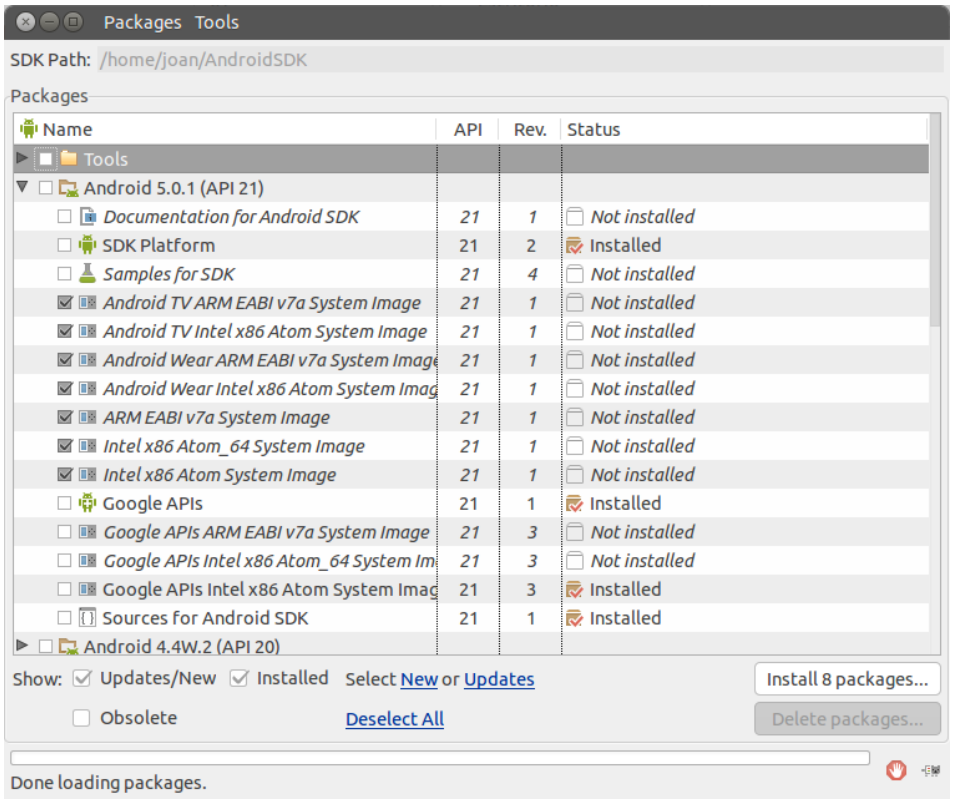
Podeu executar l'Android SDK de diverses maneres:

Figura 1.7. SDK Manager

- Des de la pantalla principal de l'Android Studio, seleccioneu *Configure/Android SDK Manager*.
- Des de qualsevol projecte, seleccioneu *Tools/Android/SDK Manager* o feu clic a la icona de la figura 1.7.
- Des de la consola, entreu dins del directori */tools* del directori de l'SDK d'Android i executeu `./android`.

Veureu un diàleg com el de la figura 1.8.

FIGURA 1.8. Paquets a instal·lar



Des d'aquí podeu seleccionar els components que voleu instal·lar o actualitzar.

Aquesta és una petita descripció dels components que hi podeu trobar:

- **SDK Tools**: conté les eines per comprovar i depurar les aplicacions Android. Aquestes eines estan instal·lades juntament a l'SDK d'Android i reben actualitzacions periòdiques. Es troben a la carpeta */tools* dins de la carpeta d'instal·lació de l'SDK.

- **Android SDK Platform-tools** (*eines de plataforma de l'SDK d'Android*): conté eines per desenvolupar i depurar l'aplicació dependents de la plataforma. Suporten les últimes característiques de la plataforma Android i, en general, s'actualitzen únicament quan hi ha una nova plataforma disponible. Es troben a la carpeta */platform-tools* dins de la carpeta d'instal·lació de l'SDK.

- **Plataformes Android:** hi ha plataformes de l'SDK disponibles per cada versió disponible d'Android. Cada plataforma Android conté una llibreria completa d'Android, imatges del sistema, codi d'exemple i *skins* de l'emulador (les diferents *pells* que simulen l'aparença externa dels dispositius, quan se simulen).

- **Samples:** conté el codi d'exemple i les aplicacions disponibles per a cada plataforma de desenvolupament d'Android. Si esteu començant a desenvolupar amb Android, assegureu-vos de descarregar els *samples* (exemples).

- **Documentació:** conté una còpia local de la darrera documentació de l'API

(*Applicacion Programming Interface*, interfície de programació d'aplicacions) d'Android.

Els afegits de tercers (*Third party Add-ons*) proporcionen components que permeten la creació d'un entorn de desenvolupament fent servir una llibreria externa d'Android específica (com, per exemple, la llibreria de Google Maps) o una imatge de sistema d'Android feta a mida (en anglès, *customized*).

1.3.4 Components recomanats

Google recomana la instal·lació de certs components per començar amb el desenvolupament d'Android.

En un **entorn bàsic** de desenvolupament hem de tenir, com a mínim:

- **SDK Tools**: amb la instal·lació realitzada des de l'assistent ja disposem d'aquest component.
- **SDK Platform-tools**: amb la instal·lació realitzada des de l'assistent ja disposem d'aquest component.
- **SDK Platform**: la instal·lació bàsica ens haurà descarregat la darrera versió de l'SDK Platform i una imatge del sistema d'un dispositiu. Amb això ja podreu compilar l'aplicació i executar un *Android Virtual Device* (*AVD*, dispositiu virtual Android). Per començar descarregueu, a banda de la darrera versió de la plataforma, la versió 2.3.3 (la versió 2.3.3 és compatible amb el 99,3% dels dispositius Android). Podeu descarregar més endavant altres versions per comprovar que el vostre programa és compatible amb aquestes.

Quina versió d'Android cal fer servir?

Quan desenvolupeu per a Android us pot sorgir el dubte de quina versió de la plataforma feu servir. Si feu servir la darrera versió disponible, és possible que el vostre programa estigui disponible únicament per a un percentatge reduït dels dispositius que hi ha al mercat. Per aquest motiu, és interessant en alguns casos fer servir versions anteriors perquè la vostra aplicació sigui compatible amb la major quantitat de dispositius possibles.

A banda d'aquests tres components que acabem d'especificar, la **instal·lació recomanada** és la següent:

- **Documentation**: la documentació dels components és útil perquè us permet treballar *offline* (sense connexió) i mirar la informació de referència de l'API des del mateix IDE.
- **Samples**: els exemples de components proporcionen codi que podeu fer servir per aprendre Android, carregar-lo com a projecte i executar-lo, o, fins i tot, reaprofitar-lo en els nostres projectes. Existeixen *samples* (exemples) per a cada versió de la plataforma, així doncs caldrà que escolliu els que corresponguin.

- **USB Driver**: aquest component únicament és necessari si es desenvolupa des de Windows i tenim un dispositiu Android sobre el qual volem instal·lar l'aplicació per la seva depuració i pel testeig. Per a plataformes GNU/Linux i Mac OSX no cal cap *driver* específic.

Per a una **instal·lació completa** es pot instal·lar també:

- **Google API**: dóna a la vostra aplicació accés a les llibreries externes de mapes, cosa que farà més senzill mostrar i manipular mapes a la vostra aplicació.
- **Plataformes addicionals de l'SDK**: si voleu publicar la vostra aplicació, haureu de descarregar les versions de la plataforma Android on voleu que la vostra aplicació s'executi. La recomanació és compilar l'aplicació sobre la versió més baixa d'Android que voleu que estigui suportada (per assegurar la compatibilitat), però comprovar-la (fer els tests) sobre la versió més alta on voleu que s'executi. Podeu comprovar les aplicacions en diferents plataformes executant-les en un AVD a l'emulador d'Android.

Una vegada instal·lada com a mínim la configuració bàsica dels components de l'SDK, podeu començar a desenvolupar aplicacions per a Android.

PATH del sistema

Podeu afegir opcionalment l'adreça de les carpetes /tools i /platform-tools dins de la carpeta de l'SDK al PATH del sistema. D'aquesta manera, tindreu un accés més fàcil a les eines. Per afegir les carpetes al PATH heu de consultar la documentació del vostre sistema operatiu. En sistemes basats en UNIX (com GNU/Linux i Mac OSX) heu de modificar el contingut del fitxer .bashrc que es troba a la vostra carpeta d'usuari i afegir les carpetes a la variable PATH; per exemple:

```
1 PATH=${PATH}:/home/usuari/android-sdk/tools:/home/usuari/android-
      sdk/platform-tools
```

2. Programació de dispositius mòbils

La programació de dispositius mòbils en Android té unes característiques particulars que la diferencien de la programació tradicional en equips de taula. La majoria d'aquestes característiques vénen donades pel fet que els dispositius mòbils tenen unes prestacions inferiors als equips de taula, i per tant el sistema operatiu ha hagut d'implementar unes solucions enginyoses per resoldre-les. D'altra banda, el sistema operatiu i l'API que es fan servir per a les aplicacions del sistema són únics. Aquestes aplicacions (a més a més) tendeixen a estar molt relacionades entre si. Per aquest motiu, és important veure en detall la forma de realitzar una aplicació i entendre els conceptes bàsics de la programació de dispositius mòbils.

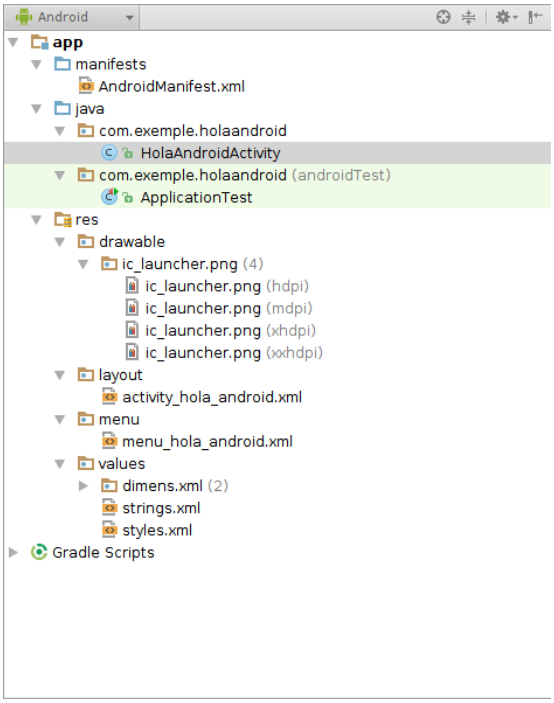
2.1 Conceptes previs

Existeixen tota una sèrie de conceptes que és interessant veure abans de començar directament amb la programació del dispositiu i l'explicació de les diferents classes que constitueixen l'API d'Android.

2.1.1 Parts d'una aplicació Android

Abans de res, si ja heu fet una aplicació bàsica amb Android ("Hola món!"), podeu mirar en el seu interior i examinar-ne cadascuna de les parts. En cas contrari, obriu el projecte "01 HolaAndroid" que trobareu als *Annexos* de la unitat. Quan obriu el projecte amb l'Android Studio veureu a la pestanya *Project* una sèrie de fitxers i carpetes com els que es mostren a la figura 2.1.

FIGURA 2.1. Vista Android del projecte. Podem canviar-la a Project i a Packages



La carpeta *Java* conté els fitxers Java amb el codi del projecte, en aquest cas "HolaAndroidActivity". Aquest fitxer conté el codi de l'activitat, i és aquí on escriureu el codi de la vostra aplicació.

El directori */res* conté tots els recursos que es fan servir a l'aplicació. Dins hi ha altres subdirectoris (diferents carpetes *drawable* per a diferents resolucions, *layout*, *values*). Les carpetes ***drawable***, en aquest moment només contenen les icones de l'aplicació en diferents resolucions. La carpeta *layout* conté el fitxer **activity_hola_android.xml**. Si l'obriu amb l'editor de text pla obtindreu un codi com 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=".
           HolaAndroidActivity">
7
8      <TextView android:text="@string/hello_world" android:layout_width="
           wrap_content"
9          android:layout_height="wrap_content" />
10 </RelativeLayout>
```

Fixeu-vos en el camp "**android:text="@string/hello_world"**". La variable "**@string/hello_world**" fa referència al camp "hello_world" que es troba al fitxer **strings.xml** dins de la carpeta *res/values*. El fitxer **strings.xml** conté les diferents cadenes de text que es fan servir al vostre projecte. Si l'obriu amb l'editor de text trobareu un codi semblant a aquest:

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

    <string name="app_name">Hola Android</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>

</resources>
```

Tot i que podeu ficar els camps de text com cadenes de caràcters constants, és recomanable que deseu totes les constants de text (els textos que apareixeran al vostre programa) dins del fitxer string.xml i que feu referència a aquests textos a partir de l'identificador "@string". D'aquesta manera, si heu de traduir la vostra aplicació a un altre idioma, tot el que necessitareu és traduir els textos del fitxer strings.xml amb l'idioma desitjat i tornar a compilar el projecte.

El directori */assets* conté altres recursos que es fan servir en l'aplicació, com ara fitxers de text, bases de dades... En aquest cas no el tenim disponible i l'haurem de crear manualment a la carpeta */res* en cas de voler fer-lo servir.

El fitxer **AndroidManifest.xml** del directori *manifests* és el fitxer de manifest de l'aplicació Android. Aquí hi ha especificats els permisos i altres característiques de l'aplicació.

2.2 Components bàsics d'una aplicació Android

Els blocs principals per construir les vostres aplicacions són els components que fareu servir per programar les aplicacions Android. Són elements conceptuals que ficareu junts per construir una aplicació en conjunt:

- Activitats
- *Fragments*
- Serveis
- Proveïdors de continguts
- Receptors de *broadcast*

2.2.1 Activitats

Podríem entendre l'***activity*** (activitat) de manera aproximada com cada una de les "pantalles" d'una aplicació. Una definició més precisa establiria que les activitats són cada un dels components de l'aplicació que proveeixen a l'usuari una pantalla amb la que aquest pugui interactuar.

Per exemple, una aplicació de missatgeria pot tenir una activitat que mostri la llista de contactes, una altra per enviar missatges i una altra per llegir missatges rebuts. Tot i que aquestes activitats treballen juntes dins de l'aplicació de missatgeria, cadascuna és independent de les altres.

A més a més, en Android, les aplicacions poden activar activitats (pantalles) d'altres aplicacions. Per exemple, l'aplicació de missatgeria a la que ens referíem podria obrir una activitat de l'aplicació de fotografia per enviar una foto juntament amb un missatge. De la mateixa manera, l'aplicació de fotografia podria obrir l'activitat d'una aplicació de correu electrònic per enviar la foto com a adjunt d'un correu.

Una activitat està implementada com una subclasse de la classe "Activity".

Podeu trobar més informació sobre la classe `Activity` a la *Guia del desenvolupador* d'Android:

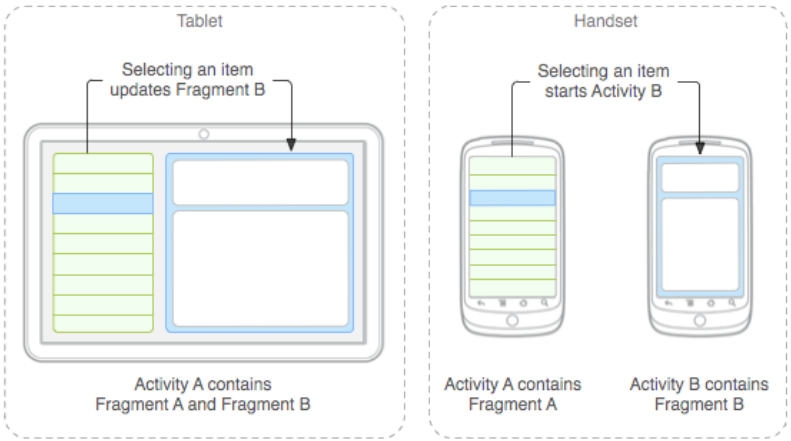
developer.android.com/guide/topics/fundamentals/activities.html.

2.2.2 'Fragments'

A la versió d'Android 3.0 (*Honeycomb*, una versió d'Android exclusiva per a tauletes) Google va incorporar el concepte de ***fragments***. Les majors dimensions de les pantalles en les tauletes permeten mostrar més informació que en un telèfon mòbil i per tant s'havia de trobar la manera de poder combinar diverses pantalles d'informació en una sola.

Un ***fragment*** és una porció d'una activitat. Podrem tenir diversos *fragments* i modificar-los durant l'execució de l'aplicació per tal de combinar-los segons les necessitats.

Figura 2.2. Exemple de "fragments" a Android



A la figura 2.2 podem veure com a la vista del telèfon mòbil una activitat disposa d'un sol *fragment* mentre que a la tauleta podem afegir dos *fragments* en cada activitat. Aquesta manera de treballar ens facilita la feina als desenvolupadors ja que, al tractar els *fragments* de manera modular, podem afegir i treure elements de l'activitat segons la quantitat d'espai lliure de què disposem i simplificar la programació per a dispositius amb diferents mides de pantalla.

2.2.3 Serveis

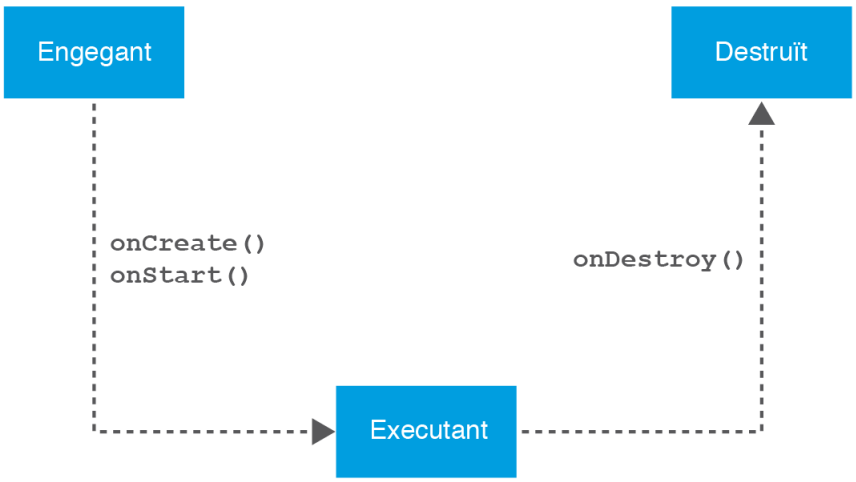
Un "***Service***" (servei) és un component que s'executa en el *background* (en el fons, que no es veu) per realitzar tasques per a treballs remots.

Els serveis s'executen, com dèiem, en el *background* i no tenen components d'interfície d'usuari. Poden fer les mateixes coses que una activitat, però no tenen interfície. Són útils per realitzar tasques durant un temps determinat, independentment de la pantalla. Per exemple, podeu voler agafar dades del receptor GPS del telèfon o reproduir música mentre esteu canviant a altres aplicacions. Aquests serveis poden haver estat iniciats per un altre component, com per exemple una activitat, i podran continuar en execució independentment del component que hagi fet la crida.

Els serveis d'Android no són el mateix que els serveis de Linux (també anomenats *daemons*), que representen un component del sistema operatiu de més baix nivell.

Els serveis tenen un cicle de vida molt més senzill que les activitats, com podreu veure si compareu la figura 2.3 amb la figura 2.5 que es mostra més endavant.

Figura 2.3. Cicle de vida d'un servei



Els serveis, o s'engeguen o s'aturen. A més a més, en general, el seu cicle de vida està més controlat pel programador, i no tant pel sistema.

Els serveis s'implementen com a classes derivades de la classe "Service". Podeu trobar més informació a la *Guia del desenvolupador* d'Android sobre aquesta classe, a developer.android.com/guide/topics/fundamentals/services.html.

2.2.4 Proveïdors de continguts

Els ***content providers*** (proveïdors de continguts) són interfícies destinades a compartir dades entre les aplicacions. Per defecte, Android executa les aplicacions de forma independent, de manera que les dades de l'aplicació estan aïllades de la resta d'aplicacions del sistema. Encara que es poden traspassar petites quantitats d'informació entre les aplicacions a través dels *intents*, els proveïdors de continguts són més adequats per compartir conjunts de dades entre aplicacions.

'Intent'
Un *intent* (en el sistema Android) és un mecanisme que permet als usuaris coordinar funcions de diferents aplicacions per tal de realitzar una tasca. Podeu trobar-ne més informació en l'enllaç http://developer.android.com/reference/android/content/Intent.html.

La informació es pot inserir, actualitzar, esborrar i consultar a un proveïdor de continguts. Android fa servir aquest mecanisme tota l'estona. Per exemple, els proveïdors de continguts de contactes (*contacts provider*) proporcionen la informació dels contactes del telèfon a diferents aplicacions. Així, una aplicació amb els permisos adequats pot consultar el *content provider* (`ContactsContract.Data`) per llegir i escriure informació sobre una persona en concret. Un altre exemple: el *Media Store* (Magatzem de mitjans), emmagatzema i serveix diferents tipus de dades per compartir, com fotos i música, entre les diferents aplicacions.

Els *content providers* també són útils per llegir i escriure dades que són privades a la vostra aplicació i no estan compartides. Per exemple, una aplicació de bloc de notes pot fer servir un *content provider* per desar notes.

La separació entre el magatzem de dades i la interfície d'usuari proporciona molta flexibilitat a Android. Per exemple, es podria instal·lar una aplicació alternativa per visualitzar els contactes, o una aplicació podria accedir des de l'escriptori a algunes de les dades de configuració del sistema (que també estan emmagatzemades en un proveïdor de continguts) per modificar-ne algunes en concret, com la d'activar i desactivar el Wi-Fi.

Els proveïdors de continguts tenen interfícies relativament senzilles amb mètodes estàndard per inserir, actualitzar, esborrar i consultar la informació, de manera molt semblant als mètodes d'accés a bases de dades.

Un *content provider* està implementat com una subclasse de "ContentProvider" i ha d'implementar una sèrie d'APIs que permeten a altres aplicacions fer transaccions. Per a més informació podeu consultar la secció de *content providers* de la *Guia del desenvolupador* d'Android: developer.android.com/guide/topics/providers/content-providers.html.

2.2.5 Receptors de 'broadcast'

'Broadcasting'

El terme *broadcasting*, popularitzat en els dècades de 1920 i 1930 per la implantació de la ràdio i la televisió, es fa servir en l'actualitat per referir-se a qualsevol emissió massiva de continguts d'un únic emissor a múltiples receptors.

Els ***broadcast Rreceivers*** (receptors de *broadcast*) són un tipus de components que responen a anuncis de sistema dirigits a totes les aplicacions del dispositiu (*broadcast*). En certa manera, és una implementació d'Android d'un sistema de subscripció d'esdeveniments. El receptor simplement es queda en estat latent esperant a activar-se quan s'esdevingui un esdeveniment al qual està subscrit.

El sistema està enviant missatges de *broadcast* tota l'estona. Per exemple, quan es rep una trucada, s'apaga la pantalla o la bateria té un nivell de càrrega baix, aquests esdeveniments són anunciats pel sistema. Un altre exemple: si voleu que un servei quedi engegat una vegada el sistema ha arrancat, podeu subscriure'l al *broadcast* que diu que el sistema ha acabat d'engegar-se.

Les aplicacions també poden enviar missatges de *broadcast*. Per exemple, una aplicació pot anunciar que s'han acabat de descarregar algunes dades i que aquestes ja estan disponibles per usar-se.

Els receptors de *broadcast* no tenen una representació visual, però quan es llancen executen una part de codi (com engegar una activitat o servei) i poden crear una notificació a la barra d'estat per alertar l'usuari quan ha ocorregut un esdeveniment de *broadcast*.

Un *broadcast receiver* està implementat com una subclasse de "BroadcastReceiver" i cada missatge de *broadcast* s'envia com un objecte `Intent`. Podeu trobar més informació a la *Guia del desenvolupador* d'Android a l'apartat de BroadcastReceiver, a developer.android.com/reference/android/content/BroadcastReceiver.html.

2.2.6 Activant els components

Tres dels cinc tipus de components (activitats, serveis i receptors de *broadcast*) s'activen per missatges asíncrons anomenats ***intents*** (intencions). Els *intents* lliguen els components individuals entre si durant l'execució, tant si els components pertanyen a la mateixa aplicació com si no. Podeu pensar en els *intents* com missatges que s'envien entre si els diferents components de les aplicacions (i que, de fet, poden servir per activar-ne alguns d'ells).

URI

Un URI (*Universal Resource Identifier*, identificador de recursos universal) és una cadena de caràcters que serveix per referenciar recursos en el sistema. Aquesta identificació permet la interacció amb representacions del recurs en el sistema. El mètode `Uri.parse(String)` serveix per analitzar i codificar un String en forma d'URI.

Un *intent* es crea amb un objecte de la classe `Intent`, que defineix un missatge per activar un component específic (*intent* explícit) o un tipus de component (*intent* implícit).

Per activitats i serveis, un *intent* defineix una acció que s'ha de realitzar (per exemple, "veure" o "enviar" alguna cosa) i pot especificar l'URI de les dades sobre les quals s'ha d'actuar. Per exemple, un *intent* pot enviar una petició d'obrir una pàgina web i especificar l'adreça de la web, per tal que l'activitat corresponent pugui obrir la pàgina correctament.

En altres casos es pot començar una activitat i esperar rebre un resultat. Per exemple, es podria iniciar una activitat per seleccionar una persona de l'agenda de contactes, i l'activitat retornaria el resultat en un *intent* (aquest inclouria l'URI apuntant al contacte seleccionat).

Per als Receptors de *broadcast*, un *intent* simplement defineix l'anunci que s'està transmetent; per exemple, un missatge a tot el dispositiu per indicar que el nivell bateria del dispositiu és baix inclouria únicament un String indicant aquest anunci.

Els proveïdors de continguts no s'activen per *intents*, sinó amb peticions d'un ***ContentResolver*** (Resolutor de continguts). El Resolutor de continguts tracta totes les transaccions directes amb el proveïdor de continguts, per tal que no ho hagi de fer directament el component que el fa servir. Això proporciona una capa d'abstracció entre el proveïdor de continguts i el component que sol·licita la informació (per seguretat).

Existeixen mètodes diferents per activar cada tipus de component:

- Les activitats es poden invocar creant un objecte de la classe `Intent` i passant-lo als mètodes `startActivity(Intent i)` o a `startActivityForResult (Intent i, int requestCode)` si voleu que l'activitat us retorni un resultat.
- Es pot iniciar un servei passant un *intent* a `startService(Intent service)`. Podeu enllaçar a un servei passant un *intent* a `bindService (Intent service, ServiceConnection conn, int flags)`.
- És possible iniciar un missatge de *broadcast* passant un *intent* als mètodes `sendBroadcast (Intent i)`, `sendOrderedBroadcast (Intent intent, String receiverPermission)` o `sendStickyBroadcast (Intent i)`.
- Es pot realitzar una consulta a un proveïdor de continguts amb una crida a `query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)` d'un ContentResolver.

Podeu consultar informació sobre tots aquests mètodes a la *Guia del desenvolupador* d'Android: developer.android.com/reference/packages.html.

2.2.7 El fitxer AndroidManifest.xml

Abans que el sistema Android iniciï un component de l'aplicació, el sistema ha de saber que aquest component existeix. Per anunciar al sistema els components que

farà servir l'aplicació existeix el fitxer **AndroidManifest.xml** (el fitxer de *manifest* de l'aplicació). L'aplicació ha de declarar tots els components en aquest fitxer, que ha d'estar a l'arrel del directori del projecte.

A més de declarar els components de l'aplicació, el fitxer de *manifest* fa les següents funcions, entre d'altres:

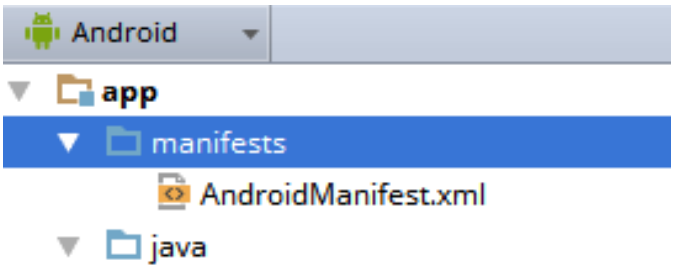
- Identifica els permisos que necessita l'aplicació (com ara permisos per accedir a Internet).
- Declara les característiques de maquinari i programari necessàries per executar correctament l'aplicació, com poden ser, entre d'altres:
  - La càmera.
  - La resolució mínima de la pantalla per poder funcionar.
  - Els dispositius d'entrada necessaris per executar l'aplicació (com el *pad* de quatre direccions).
  - Bluetooth.
  - Pantalla multitàctil.
- Declara les llibreries d'aplicació, a banda de les API d'Android, que s'han d'enllaçar (*link*) a l'aplicació, per exemple com la llibreria de Google Maps.

El fitxer XML del projecte d'*Hola Món* és el següent:

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

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".HolaAndroidActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

El fitxer *manifest* el podeu trobar a la carpeta *manifests* de qualsevol projecte, tal i com podeu veure a la figura 2.4.

Figura 2.4. Carpeta manifests d'un projecte



En l'element `<application>` hi podeu trobar altres atributs específics de l'aplicació, per exemple `android:icon` apunta al recurs de la icona que identifica l'aplicació i `android:label` apunta al recurs (dins del fitxer **string.xml**) amb el nom de l'aplicació.

L'element `<activity>` serveix per declarar una activitat. Així, l'atribut `android:name` especifica el nom de la subclasse (que hereta de la classe `Activity`) que tindrà el codi de l'activitat. Aquesta classe ha d'estar implementada i aparèixer al projecte. Els components de l'aplicació han d'estar declarats fent servir les etiquetes: `<activity>`, `<service>`, `<receiver>` i `<provider>`.

Aquells components que programeu i incloeu al projecte però no estiguin declarats al fitxer AndroidManifest.xml no es podran executar, atès que no són visibles pel sistema. Els receptors de *broadcast*, però, es poden declarar al *manifest* o bé crear-se dinàmicament amb codi i registrar-se al sistema tot cridant a `registerReceiver()`.

Gradle

Gradle és una eina per automatitzar la construcció de projectes. Podrem configurar fàcilment distintes versions de la nostra aplicació, per exemple: generar una versió *demo* o *lite* de la nostra aplicació, la versió "completa", per fer "debug", etc. A més, disposem d'una gran quantitat de *plugins* i la gestió de dependències millora respecte *ant*, l'antic sistema de construcció utilitzat a Android. Per més informació podeu consultar el següent enllaç: https://developer.android.com/tools/building/configuring-gradle.html.

La tasca principal del *manifest* és informar el sistema sobre els components de l'aplicació. Actualment, algunes de les característiques que abans trobàvem al fitxer de *manifest* han passat a formar part del fitxer `build.gradle (Module: app)`, que trobem dins de `Gradle Scripts`, els quals pengen a la vegada de l'arrel del projecte; és el cas de la declaració de la versió de la plataforma, el nivell mínim d'API o la versió de la nostra aplicació.

Aquest és el fitxer *build.gradle* d'*HolaAndroid*:

```
1  apply plugin: 'com.android.application'
2
3  android {
4      compileSdkVersion 21
5      buildToolsVersion "21.1.2"
6
7      defaultConfig {
8          applicationId "com.exemple.holaandroid"
9          minSdkVersion 10
10         targetSdkVersion 21
11         versionCode 1
12         versionName "1.0"
13     }
14     buildTypes {
15         release {
16             minifyEnabled false
17             proguardFiles getDefaultProguardFile('proguard-android.txt'), '
                   proguard-rules.pro'
18         }
19     }
20 }
```

```
21
22 dependencies {
23     compile fileTree(dir: 'libs', include: ['*.jar'])
24     compile 'com.android.support:appcompat-v7:21.0.3'
25 }
```

Per a més informació sobre el contingut i l'estructura del fitxer AndroidManifest.xml, podeu consultar la documentació de la *Guia del desenvolupador* d'Android: developer.android.com/guide/topics/manifest/manifest-intro.html.

Declarant les capacitats del component

Es pot fer servir un *intent* per activar activitats, serveis i receptors de *broadcast*. Per fer-ho es pot especificar a l'*intent* el nom del component que es vol activar (fent servir el nom de la classe de l'activitat que voleu activar, per exemple). Però existeix una altra forma d'activar activitats, a través del que es coneix com a ***intent actions* (*intents* d'acció)**.

Un *intent* d'acció descriu el tipus d'acció que es vol realitzar (i, opcionalment, les dades sobre les quals es vol realitzar) i delega al sistema la tasca de trobar un component en el dispositiu que pugui realitzar aquesta acció, i activar-lo. Per exemple, es pot llançar un *intent* on s'especifica que es vol visitar una pàgina web o fer una foto, i deixar que el sistema trobi l'activitat (de l'aplicació que sigui) que pugui realitzar aquesta acció. En el cas que hi hagi múltiples components que puguin realitzar l'acció, es demana a l'usuari quina d'elles vol fer servir.

Com pot saber el sistema quins components són apropiats per fer la tasca que es demana des de l'*intent*? Els components poden definir uns **filtres d'*intents* (*intent filters***) al fitxer de *manifest* que seran comparats pel sistema amb l'acció que s'ha demanat fer. Podeu, opcionalment, definir aquests filtres d'*intents* al *manifest* de la vostra aplicació per tal que el vostre component pugui respondre a *intents* d'altres aplicacions. Per fer-ho, heu d'afegir un element `<intent-filter>` com un fill de l'element de la declaració de component al fitxer XML.

Podeu trobar més informació sobre els filtres d'*intents* a la *Guia del desenvolupador* d'Android: developer.android.com/guide/topics/intents/intents-filters.html.

2.3 Activitats

A Android, una ***Activity*** (activitat) és un component d'aplicació que proporciona una finestra amb una interfície d'usuari de la vostra aplicació. Una aplicació també pot no tenir cap activitat.

Generalment, les aplicacions tenen més d'una activitat i el seu principal propòsit és el d'interactuar amb l'usuari. Des del moment que una activitat es mostra en pantalla fins al moment que es tanca, l'activitat passa per una sèrie de fases, conegudes com el ***cicle de vida*** de l'activitat. És molt important que entengueu el cicle de vida de l'activitat per tal que la vostra aplicació funcioni correctament.

Una ***Activity*** (activitat) és un component que proporciona una interfície gràfica amb la qual interactua l'usuari.

Podeu trobar la darrera informació oficial de Google sobre activitats a developer.android.com/guide/topics/fundamentals/activities.html.

Una aplicació consisteix en múltiples activitats que estan relacionades entre si. En general, existeix una activitat considerada l'**activitat principal** (o *main*) de l'aplicació, que és la que es mostra quan es llança l'aplicació per primera vegada. Aquesta activitat principal pot iniciar altres activitats per realitzar diferents accions. Cada vegada que es llança una activitat, l'activitat prèvia s'atura, però el sistema guarda l'activitat a la pila (anomenada *back stack*). Quan s'inicia una activitat, aquesta concentra el focus de l'aplicació.

Quan una activitat s'atura perquè una altra comença, la primera queda notificada a través dels **mètodes de *callback***, que són mètodes de l'activitat que són cridats quan aquesta canvia d'estat (si es crea, s'atura, es destrueix...) i que li permeten realitzar tasques en relació a aquest canvi d'estat. Per exemple, quan l'activitat s'atura aquesta hauria d'alliberar els recursos reservats i tancar les comunicacions de xarxa. Quan una activitat es llança, podria activar aquestes mateixes connexions de xarxa.

Callback

Una funció de *callback* (retrotrucada o devolució de trucada) és una referència a una porció de codi executable ("funció A") que es passa com a argument en la crida a un altre codi ("funció B"). D'aquesta manera, el codi (la "funció B") pot cridar la funció que se li passa com a argument ("funció A") quan ho necessiti. Per exemple, imagineu que voleu cridar una funció que realitza una cerca a una llista ("B"), i que quan trobi un element en concret voleu que cridi una funció vostra per mostrar-ho per pantalla ("A"). En aquest cas, es passaria a la crida de la funció B una referència a A. Quan B trobi l'element, farà una crida a A.

2.3.1 Treballant amb activitats

Per crear una activitat heu de crear una classe derivada de la classe `Activity`(o una subclasse d'aquesta). En la classe que heu creat heu d'implementar els mètodes de *callback* que el sistema cridarà en les transicions dels diferents estats del cicle de vida de l'activitat (quan es crea, es destrueix, etc.). Els dos mètodes més importants són:

- `onCreate()`: el sistema crida aquest mètode quan es crea l'activitat. Dins, heu d'inicialitzar els components inicials de la vostra aplicació. Aquí s'ha de cridar `setContentView()` per definir la distribució de la interfície d'usuari de l'activitat (coneguda com a *layout*).
- `onPause()`: el sistema crida aquest mètode, en primer lloc, per indicar que l'usuari està sortint de la vostra activitat. Encara que això no signifiqui necessàriament que l'activitat es destrueixi, cal que deseu tots els canvis

que voleu conservar de l'Aplicació, ja que és possible que l'usuari no torni a l'activitat.

La interfície d'usuari d'una activitat està formada per una jerarquia de **vistes** (objectes derivats de la classe `View`). Cada vista (*view*) controla un espai rectangular dins de l'espai de l'activitat i pot respondre a la interacció de l'usuari. Per exemple, una vista podria ser un botó que inicia una acció quan és premut per l'usuari.

A Android disposeu d'una sèrie de vistes que ja estan fetes que podeu fer servir per dissenyar la vostra activitat. Els ***widgets*** són vistes que proporcionen elements visuals i interactius en pantalla com un botó, un camp de text, un *checkbox* o una imatge. Els ***layouts*** (distribucions) també són vistes que deriven de la classe `ViewGroup`. Contenen altres vistes i proporcionen un tipus de distribució específica per a elles. Per exemple, un *layout linear* pot contenir altres botons (vistes) que es mostraran en forma de vista en la pantalla. Podeu crear subclasses derivades de `View` i `ViewGroup` (o les seves subclasses) per crear els vostres *widgets* i *layouts* i aplicar-los a les vostres activitats.

Els *layouts* de les diferents activitats de l'aplicació estan definits a una sèrie de fitxers XML que es troben a la carpeta */res/layout* del projecte. Existeix un fitxer XML per cada *layout*, sent el fitxer **main.xml** el que es crea per defecte per a la primera activitat de l'aplicació. Podeu carregar els *layouts* com la interfície gràfica de la vostra activitat amb una crida a la funció `setContentView (int layoutResID)` que té un únic argument, l'identificador del *layout* que es vol carregar.

A més de les activitats, els ***intents*** són un altre concepte únic d'Android. Aquests, bàsicament, permeten a diferents activitats de diferents aplicacions treballar conjuntament com si totes pertanyessin a la mateixa aplicació.

2.3.2 Cicle de vida d'una activitat

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

Per crear una activitat primer heu de crear una classe de Java que sigui descendent de la classe base `Activity`. Per seguir l'explicació, creeu un nou projecte.

Fixeu-vos en la definició de la classe que trobeu quan creeu un nou projecte:

```
1 public class ActivitatPrincipal extends ActionBarActivity {
2
3     @Override
4     protected void onCreate(Bundle savedInstanceState) {
5         super.onCreate(savedInstanceState);
6         setContentView(R.layout.activity_main);
7     }
8 }
```

La classe `ActivitatPrincipal` hereta d'`ActionBarActivity` que a la vegada és un descendent d'`Activity`. La classe `Activity` defineix una sèrie d'*events* que defineixen el cicle de vida de l'aplicació:

- `onCreate()`: es crida quan l'activitat s'ha creat per primera vegada.
- `onStart()`: es crida quan l'activitat és visible per a l'usuari.
- `onResume()`: es crida quan l'activitat comença a interaccionar amb l'usuari.
- `onPause()`: es crida quan l'activitat actual es pausa i l'activitat anterior és represa.
- `onStop()`: es crida quan l'activitat ja no és visible per a l'usuari.
- `onDestroy()`: es crida abans que l'activitat sigui destruïda pel sistema (manualment o pel sistema per estalviar memòria).
- `onRestart()`: es crida quan l'activitat s'ha aturat i s'ha tornat a iniciar.

Un ***handler*** (controlador) és un mètode o funció que s'executa per tractar un esdeveniment ocorregut al sistema.

Per defecte, l'activitat que es crea al nou projecte conté l'*event* `onCreate()`. Dins d'aquest mètode (és un *handler* de l'*event*) hi ha el codi que serveix per mostrar els elements de la **UI** (*user interface*, interfície d'usuari) en pantalla. El cicle de vida d'una activitat i les diferents fases per les quals passa són les següents:

- Estat inicial
- Estat en execució
- Estat pausat
- Estat aturat (*stop*)
- Estat destruïda

Estat inicial

Quan una activitat s'està engegant passa a través de tota una sèrie de crides a mètodes de *callback* on podeu introduïr codi. Tot seguit, passarà a l'estat d'execució. Aquest procés de creació de l'activitat (la transició entre l'estat inicial i l'estat en execució) és una de les operacions més costoses en temps de computació i, per tant, afecta la vida de la bateria del dispositiu. Aquest és el motiu pel qual les activitats no es destrueixen automàticament quan es deixen de mostrar, ja que l'usuari pot voler tornar-hi aviat.

Estat en execució

Una activitat en execució és la que està actualment en pantalla i oferint interacció a l'usuari. També diem que l'activitat està *en focus*, és a dir, que totes les interaccions de l'usuari (com tocar la pantalla, escriure al teclat, pressionar els diferents botons del telèfon) són tractades per aquesta activitat. Per aquest motiu, hi ha una única activitat en estat d'execució en cada moment. L'activitat en execució és la que té prioritat en termes de memòria i recursos, per tal d'oferir a l'usuari un temps de resposta reduït.

Estat pausat

Quan una activitat no té el *focus* (l'usuari no hi està interactuant) però encara està visible en pantalla, diem que està en estat de pausa. No és una situació freqüent, ja que la pantalla del dispositiu sol ser petita i l'activitat, generalment, o bé la fa servir tota o bé no la fa servir en absolut. Però de vegades es mostra un diàleg en pantalla,fent que l'activitat que estava en execució quedi en pausa. A més, totes les activitats que es tancaran (estat *stop*) passen abans per l'estat de pausa. Les activitats en estat de pausa encara tenen una prioritat alta en termes de memòria i altres recursos, perquè romanen visibles i no es poden amagar.

Estat aturat ('stop')

Una activitat està en estat aturat quan no és visible però encara està en memòria. Una activitat aturada es pot tornar a mostrar (i, per tant, tornaria a ser una activitat en execució) o es pot destruir definitivament alliberant l'espai que ocupava a la memòria. El sistema desa les activitats en estat aturat, ja que és probable que l'usuari les vulgui tornar a executar aviat, i reiniciar una activitat aturada és més ràpid que engegar-la des de zero. Això és així perquè els objectes ja estan carregats en memòria i és més fàcil portar-los al capdavant de la pantalla. Les activitats aturades es poden treure de memòria en qualsevol moment (ja que el sistema pot necessitar la memòria que estan ocupant).

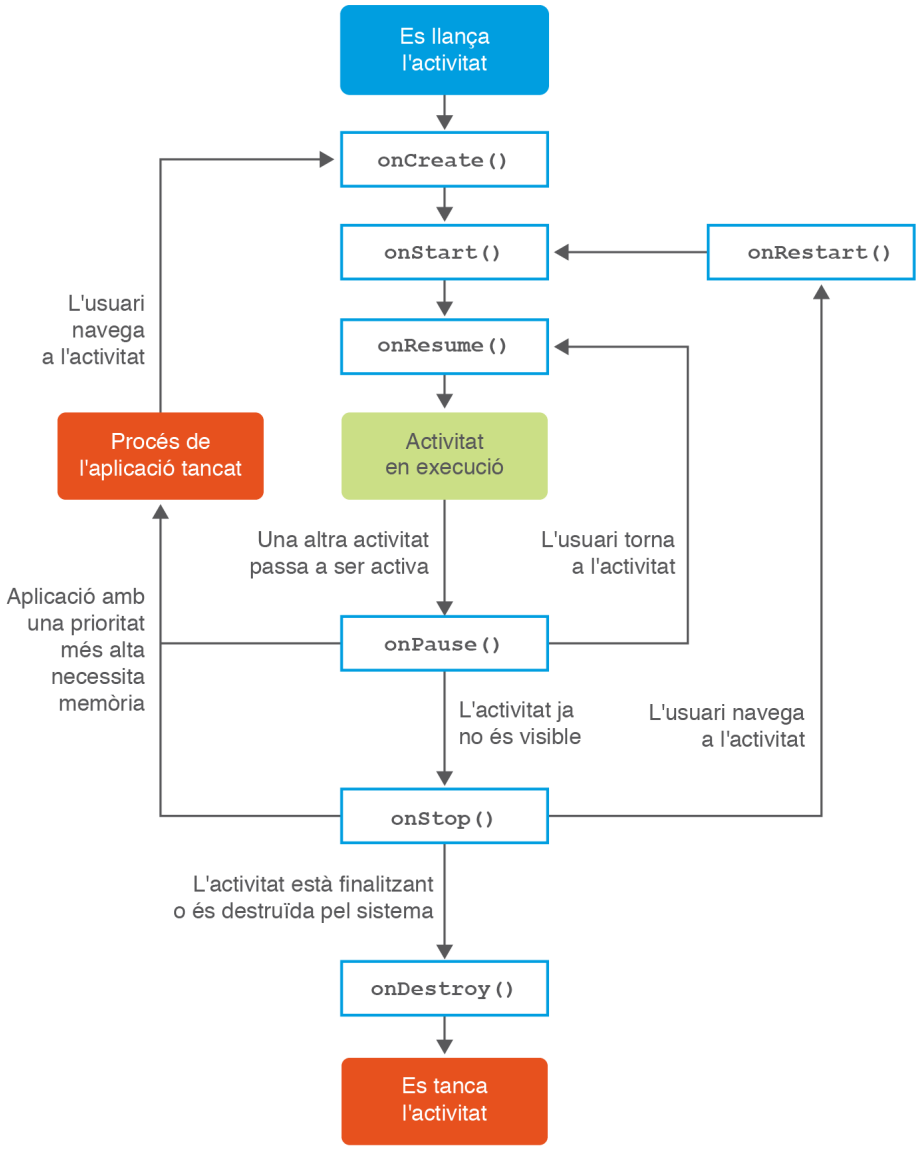
Estat destruïda

Una activitat destruïda és aquella que ja no està en memòria. L'*activity manager* ha decidit que aquesta activitat ja no és necessària i ha alliberat l'espai de memòria que estava ocupant. Abans que es destrueixi es poden fer certes accions, com salvar informació important de l'activitat.

Cicle de vida de l'activitat

La figura 2.5 mostra les etapes del cicle de vida de l'activitat.

Figura 2.5. Cicle de vida d'una activitat



En execució i aturada

El fet que una activitat estigui en execució no vol dir necessàriament que estigui fent moltes coses. Podria estar senzillament esperant interacció per part de l'usuari. Igualment, una activitat en estat aturat no vol dir que necessàriament no estigui fent res. Els noms dels estats fan referència a com d'activa està l'activitat respecte a l'entrada de l'usuari. En altres paraules, si l'activitat està visible, si té el focus o si no és visible en absolut.

Què fa l'activitat en cada estat? Per veure i entendre les diferents fases del cicle de vida d'una activitat, el millor és crear un projecte i interactuar-hi, i veure quan ocorren els diferents esdeveniments.

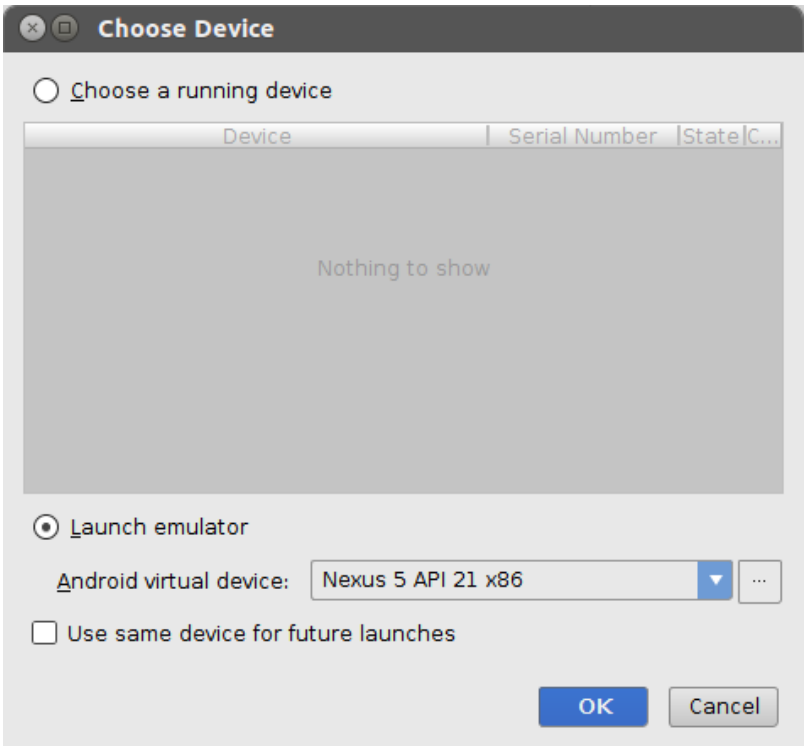
Una vegada creat el nou projecte, el fitxer **ActivitatPrincipal** quedarà:

```
1  package cat.xtec.ioc.cicledevida;
2
3  import android.support.v7.app.ActionBarActivity;
4  import android.os.Bundle;
5  import android.util.Log;
6
7
8  public class ActivitatPrincipal extends ActionBarActivity {
9
10     String tag = "Events";
11
12     @Override
13     protected void onCreate(Bundle savedInstanceState) {
```

```
14          super.onCreate(savedInstanceState);
15          setContentView(R.layout.activity_main);
16
17          Log.d(tag, "A l'event onCreate()");
18      }
19
20      @Override
21      public void onStart() {
22          super.onStart();
23          Log.d(tag, "A l'event onStart");
24      }
25
26      public void onRestart() {
27          super.onRestart();
28          Log.d(tag, "A l'event onRestart()");
29      }
30
31      public void onResume() {
32          super.onResume();
33          Log.d(tag, "A l'event onResume()");
34      }
35
36      public void onPause() {
37          super.onPause();
38          Log.d(tag, "A l'event onPause()");
39      }
40
41      public void onStop() {
42          super.onStop();
43          Log.d(tag, "A l'event onStop()");
44      }
45
46      public void onDestroy() {
47          super.onDestroy();
48          Log.d(tag, "A l'event onDestroy()");
49      }
50
51  }
```

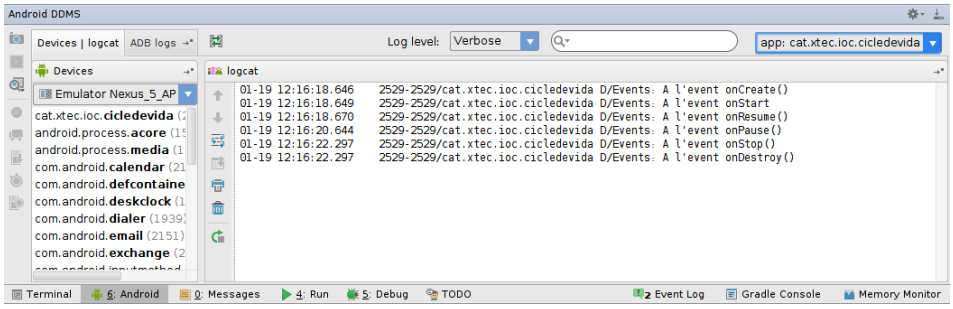
Abans d'executar el projecte feu clic al menú *Run/Edit Configurations* i a l'opció *Target device* assegureu-vos que està marcada l'opció *Show chooser dialog*. Això farà que cada vegada que executeu el programa se us demani on s'executarà d'entre les diferents opcions possibles del sistema (màquines virtuals o dispositius físics connectats a l'ordinador). Feu clic a *Run* per executar el projecte. Veureu una pantalla com la de la figura 2.6. En aquest cas en concret, no disposem de cap dispositiu ni emulador en funcionament, i se'ns proposa executar una màquina virtual d'un Nexus 5.

FIGURA 2.6. Selecció de dispositiu on s'executarà l'aplicació



Una vegada fet això, podeu executar el projecte per depurar amb el botó dret el projecte al *Package Explorer* i l'opció *Debug As/Android Application*. Seguiu aquests *events* amb el gràfic de la figura 2.5 ("Cicle de vida d'una activitat"). Podeu veure també els esdeveniments de *log* a la finestra *LogCat* de l'Android Studio, tal com podeu veure a la figura 2.7.

FIGURA 2.7. Log d"events'



Podeu provar d'afegir filtres fent clic al desplegable de la part superior dreta i escollint *Edit filter configuration*.

Així, quan es carrega l'aplicació per primera vegada es mostrarà al *logcat* el següent:

```
1 D/Events: A l'event onCreate()
2 D/Events: A l'event onStart()
3 D/Events: A l'event onResume()
```

Les inicialitzacions de les variables de la vostra aplicació les podeu fer a l'*event* `onCreate`, que com vèiem al codi anterior és el primer en executar-se. Quan

premeu el botó de tornar enrere al simulador, els *events* que es mostren al *log* seràn aquests:

```
D/Events: A l'event onPause()
D/Events: A l'event onStop()
D/Events: A l'event onDestroy()
```

Podeu observar com quan premem la tecla *enrere* l'aplicació es destrueix (crida el mètode `onDestoy()`) de manera que si tornem a executar-la, ja sigui des del menú o des de la tecla *recents* (prement durant una estona el botó *home* en aquells telèfons on no existeixi aquesta tecla), es tornaran a produir els següents esdeveniments:

```
D/Events: A l'event onCreate()
D/Events: A l'event onStart()
D/Events: A l'event onResume()
```

Si mentre estem a l'aplicació premem el botó *home* i tornem a executar-la, la seqüència d'esdeveniments serà la següent:

```
D/Events: A l'event onPause()
D/Events: A l'event onStop()
D/Events: a l'event onRestart()
D/Events: A l'event onStart()
D/Events: A l'event onResume()
```

En aquest cas no es fa cap crida a `onDestroy()` i per tant, al tornar a tenir l'aplicació en primer pla no s'executa el mètode `onCreate()`. Quan una aplicació està pausada o aturada i tornem a ella, executarà el mètode `onRestart()`, seguits d'`onStart()` i `onResume()`.

Hem vist com l'activitat es destrueix quan es prem el botó de tornar enrere (*Back button*). És molt important aquest punt, ja que l'estat de l'activitat es perd. Per tant, heu d'afegir codi addicional per preservar l'estat de l'activitat abans no es destrueixi (en qualsevol dels *events* que s'executen abans de destruir l'aplicació) i tornar a restituir l'estat quan es torni a obrir. Per exemple, les dades que voleu desar quan l'aplicació deixa d'estar en execució les podríeu desar a `onPause()` i tornar a carregar-les des d'`onResume()`. </newcontent>

L'*event* `onPause()` és cridat als dos escenaris, quan l'activitat s'envia al fons (*background*) i quan és eliminada amb el botó *Back*. Quan l'activitat es torna a engegar sempre es crida `onStart()` i `onResume()`, independentment de si ha tornat del fons (*background*) o si s'ha creat de nou.

2.4 'Intents'

Els ***intents*** són missatges enviats entre els diferents elements que conformen les aplicacions d'Android. Les activitats, serveis i *broadcast receivers* (receptors de *broadcast*) són activats a través d'aquests missatges. Els *intents* li indiquen a una activitat que s'iniciï, o a un servei que comenci o s'aturi, o són simplement

missatges de *broadcast* (dirigits a totes les aplicacions del dispositiu). En definitiva, serveixen per comunicar components de la mateixa aplicació o d'altres.

> Els ***intents*** serveixen per comunicar diferents components de la mateixa aplicació o d'aplicacions diferents. Generalment fan referència a una acció que s'ha de realitzar i van acompanyats de dos elements: l'acció que s'executarà i la informació que aquesta necessita.

L'*intent* és un objecte de la classe `Intent`, i conté la informació de l'operació que es vol realitzar o, en el cas dels *broadcasts*, la descripció d'algun esdeveniment que ha tingut lloc en el sistema. Existeixen diferents maneres d'enviar els *intents* en funció del tipus de component al qual van dirigits.

Per adreçar un *intent* a una activitat, podeu fer servir els mètodes `Context.startActivity (Intent intent)` o `Activity.startActivityForResult (Intent intent, int requestCode)`, depenent de si espereu que l'activitat us retorni alguna informació o no.

Per iniciar un servei o enviar informació a un servei podeu fer servir el mètode `Context.startService (Intent service)`. Es pot fer servir `Context.bindService (...)` per establir una connexió entre el servei i el component que l'està cridant.

Per adreçar un *intent* a tots els receptors de *broadcast* podeu fer servir `Context.sendBroadcast (Intent intent)`, `Context.sendOrderedBroadcast (...)` o `Context.sendStickBroadcast (Intent intent)`.

En cada cas, el sistema Android troba l'activitat, servei o receptor de *broadcast* que respondrà a l'*intent*, instanciant-lo si cal.

2.4.1 Objecte "Intent"

Un `Intent` és un objecte que conté informació d'interès per al component que el rep (com l'acció que ha de realitzar i les dades sobre les quals l'ha de realitzar). També conté informació per al sistema Android (com la categoria del component que ha de tractar l'*intent* i les instruccions sobre com llançar el component). Conté la següent informació:

- Nom del component
- Acció
- Dades
- Categoria

- Extres
- *Flags*

Nom del component

El nom del component que ha de tractar l'*intent* és un objecte `ComponentName`. El nom del component és opcional. Si s'especifica, l'*intent* és enviat a la instància de la classe especificada. Si no s'especifica, el sistema trobarà un component per tractar l'*intent* (en base a una altra informació especificada a l'objecte `Intent`).

El component s'especifica amb `setComponent()`, `setClass()` o `setClassName()`.

Acció

L'acció és un *string* que especifica l'acció a realitzar o, en el cas dels *intents broadcast*, una acció que ha tingut lloc i que s'està anunciant. La classe `Intent` defineix alguns tipus d'accions estàndards, com les que es poden veure a la taula 2.1. Podeu veure tots els tipus de constants d'acció a la *Guia del desenvolupador* d'Android, a la pàgina de la classe `Intent`.

Taula 2.1. Diferents tipus d'accions

Constant	Component al qual va dirigit l'*intent*	Acció
`ACTION_CALL`	Activitat	Realitza una trucada.
`ACTION_EDIT`	Activitat	Mostra dades a l'usuari per a edició.
`ACTION_MAIN`	Activitat	Inicia com a activitat principal d'una aplicació. No s'espera valor de retorn.
`ACTION_SYNC`	Activitat	Sincronitza dades des del dispositiu al servidor.
`ACTION_VIEW`	Activitat	Mostra dades a l'usuari.
`ACTION_BATTERY_LOW`	Receptor de *broadcast*	Advertència de que la bateria del dispositiu està baixa.
`ACTION_HEADSET_PLUG`	Receptor de *broadcast*	S'han connectat o desconnectat els auriculars del dispositiu.
`ACTION_SCREEN_ON`	Receptor de *broadcast*	S'ha encès la pantalla.
`ACTION_AIRPLANE_MODE_CHANGED`	Receptor de *broadcast*	S'ha modificat el mode avió del dispositiu.

Podeu definir i llegir l'acció d'un objecte `Intent` amb els mètodes `setAction()` i `getAction()`.

Dades

Es tracta de la informació relativa a les dades sobre les quals s'ha de realitzar l'acció (especificant l'URI) i el tipus MIME d'aquestes dades.

MIME

MIME (*Multipurpose Internet Mail Extensions*, extensions de correu d'Internet multipropòsit) és un estàndard d'Internet que es va dissenyar originalment per especificar el tipus de les dades adjuntes a un correu electrònic, però que actualment serveix per especificar el tipus de dades en general.

Depenent del tipus d'acció, l'URI especificarà unes dades o altres. Per exemple, quan l'acció és `ACTION_CALL` el camp de dades serà de tipus "tel:" i anirà acompanyat d'un URI amb el número de telèfon al que s'ha de trucar. O si l'acció és `ACTION_VIEW`, el camp de dades pot ser del tipus "http:" i s'aportarà l'adreça del lloc web que es vol visitar.

Moltes vegades, el tipus de dades es pot deduir de l'URI, per exemple mirant si és la URL d'un web o una imatge del dispositiu. Però es pot definir específicament el tipus de les dades a l'objecte *intent*.

El mètode `setData (Uri data)` serveix per especificar l'URI de les dades, i el mètode `setType (String type)` serveix per especificar el tipus MIME de les dades. També podeu fer servir el mètode `setDataAndType (Uri data, String type)` per especificar ambdós. Per llegir l'URI i el tipus MIME, podeu fer servir `getData()` i `getType()`, respectivament.

Categoria

És una cadena que conté informació addicional sobre el tipus de component que ha de tractar l'*intent*. La classe `Intent` defineix algunes categories, entre les quals es troben les de la taula 2.2.

TAULA 2.2. Alguns dels tipus de categories definides a l"intent'

Constant	Significat
`CATEGORY_PREFERENCE`	L'activitat és un panell de preferències.
`CATEGORY_APP_MUSIC`	L'activitat ha de ser capaç de reproduir o manipular música.
`CATEGORY_APP_MESSAGING`	L'aplicació ha de ser capaç d'enviar i rebre missatges.
`CATEGORY_APP_BROWSER`	L'activitat ha de ser capaç de navegar per Internet.

Podeu comprovar la llista completa de categories de la definició de la classe `Intent` a la *Guia del desenvolupador* d'Android.

Podeu fer servir els mètodes `addCategory(String category)`, `removeCategory(String category)` i `getCategories()` per afegir, esborrar i obtenir una llista de les categories de l'objecte *intent*, respectivament.

Extres

Els extres són parells clau-valor que aporten informació addicional que s'ha d'enviar al component perquè tracti l'*intent*. Així, diferents URI tenen associats alguns extres en particular. Per exemple, l'acció `ACTION_HEADSET_PLUG`, que correspon a una canvi en l'estat de la connexió dels auriculars del dispositiu, té un camp extra indicant si els auriculars s'han endollat o no. Un altre exemple, l'acció `ACTION_TIMEZONE_CHANGED`, que correspon al fet que s'ha canviat l'ús horari, té un extra *time-zone* per especificar l'ús horari que s'ha definit.

La classe `Intent` té una sèrie de mètodes que es fan servir per afegir extres, els podeu consultar a la *Guia del desenvolupador* d'Android. També es pot fer servir un objecte del tipus `Bundle` per afegir extres. Un objecte de la classe `Bundle` (literalment de l'anglès, 'paquet' o 'feix') permet emmagatzemar una sèrie de dades (que poden ser de diferent tipus). Les variables es poden afegir al `Bundle` simplement donant un nom i la variable que es vol guardar. En certa manera, recorda els *arrays* amb què es passen variables entre els formularis en PHP: un *array* de *tuples* [nom, valor]. Podeu fer servir els mètodes `putExtras(Bundle extras)` i `getExtras()` per afegir i obtenir extres de l'*intent*.

'Flags' (banderes)

Es poden especificar *flags* de diferents tipus que indiquin al sistema com ha de llançar l'activitat i com tractar-la després que s'hagi engegat. Tots aquests *flags* estan definits a la classe `Intent`.

Enviament de 'broadcasts'

Les aplicacions fan servir els objectes `Intent` per enviar *broadcasts* i activar components del sistema. Per exemple, a la taula 2.3 teniu alguns exemples d'*intents* que criden a serveis estàndard del sistema, amb el seu significat.

Taula 2.3. Diferents 'intents'

Aplicació objectiu	URI de l'*intent*	Acció de l'*intent*	Resultat
Navegador	http://ioc.xtec.cat	`ACTION_VIEW`	Obre en un navegador l'adreça especificada.
Navegador	"text"	`ACTION_WEB_SEARCH`	Es realitza una cerca a Google de la cadena especificada.
Marcador	tel:*número de telèfon*	`ACTION_CALL`	Truca al número de telèfon especificat.
Marcador	tel:*número de telèfon*	`ACTION_DIAL`	Marca el número de telèfon especificat (però no truca).
GoogleMaps	geo:*latitud*,*longitud* geo:*latitud*,*longitud*?z=*zoom* geo:0,0?q=*adreça* geo:0,0?q=*pizzeria*	`ACTION_VIEW`	Obre GoogleMaps a la localització especificada o fent la cerca demanada.

2.4.2 Filtres d"intents'

Hi ha dos tipus d'*intents* diferents:

- Explícits: on s'especifica el component explícitament pel seu nom (el camp *nom del component* de l'`Intent` té un valor introduït). Generalment, no podem saber el nom específic dels components d'altres aplicacions, per això

els *intents* explícits generalment es fan servir per cridar components de la nostra pròpia aplicació.

- Implícits: aquests no tenen definit el nom del component al qual van dirigits. Generalment, es fan servir per activar components d'altres aplicacions.

Els *intents* explícits s'envien a través del sistema a una instància de la classe definida, únicament es fa servir el nom del component (definit al `ComponentName` de l'objecte `Intent`) per determinar quin és el component al qual va dirigit l'*intent*.

Com els *intents* implícits no especifiquen quin és el component al qual va dirigit l'*intent*, el sistema ha de trobar el millor component per tractar l'*intent*. Per fer-ho, compara els continguts definits a l'objecte `Intent` amb unes estructures definides als components que serveixen per *anunciar* al sistema les capacitats del component. Aquestes estructures serveixen per dir al sistema quin tipus d'*intents* és capaç de tractar el component i s'anomenen ***intent filters*** (filtres d'*intents*).

> Els ***intent filters*** serveixen per dir al sistema les capacitats del component. El sistema sabrà a quin tipus d'*intent* pot respondre el component comparant les característiques de l'*intent* amb el filtre d'*intent*.

Per exemple, podeu crear una activitat que serveixi per veure pàgines web. Podeu *anunciar* al sistema que la vostra activitat està oberta a rebre *intents* per visitar llocs web. La propera vegada que es faci un *intent* per visitar un web des del vostre dispositiu, la vostra activitat serà una de les candidates per obrir el web (en el cas que hi hagi diversos components per atendre l'`Intent`, el sistema demanarà a l'usuari quin vol fer servir).

Els filtres permeten que els components puguin rebre *intents* implícits. Si un component no té *intent filters*, únicament pot rebre *intents* explícits (amb el seu nom). Un component que defineix filtres d'*intents* pot rebre *intents* implícits i explícits.

Un component té un filtre diferent per cada treball que pot fer. Per exemple, una aplicació multimèdia pot tenir diferents tipus de filtres per anunciar que pot obrir diferents tipus de fitxers (fotos, àudio, vídeo, etc.).

Un filtre és una instància de la classe `IntentFilter`. El sistema comprova els components per veure quin pot rebre l'*intent* implícit, per aquest motiu ha de conèixer les capacitats del component abans d'engegar-lo. Per això els filtres s'han de definir al fitxer **AndroidManifest.xml** com a elements `<intent-filter>`. L'única excepció són els filtres per als receptors de *broadcast* que es registren dinàmicament cridant el mètode `Context.registerReceiver()`.

Un filtre té camps similars als de l'objecte `Intent`: acció, dades i categoria. Els *intents* implícits es comparen amb els camps del filtre. Perquè l'*intent* sigui enviat al component, la comparació amb els tres camps del filtre ha de ser vàlida. Es realitza un test amb cada camp per separat:

- Acció

- Categoria
- Dades (l'URI i el tipus de dades)

Els extres i els *flags* no es fan servir per determinar quin component rep l'`Intent`.

Acció

L'element `<intent-filter>` en el fitxer de *manifest* mostra les accions del filtre com a subelements `<action>`. Per exemple:

```
<intent_filter>
  <action android:name="android.settings.SOUND_SETTINGS"/>
  <action android:name="android.settings.WIFI_SETTINGS"/>
  <action android:name="android.intent.action.CALL"/>
</intent_filter>
```

Encara que els objectes `Intent` únicament defineixen una acció, els filtres poden definir-ne més d'una. La llista no pot estar buida, ha de tenir almenys un element `<action>` o bloquejarà tots els *intents*. Per tal que l'objecte `Intent` passi el test del filtre d'acció ha de coincidir una de les accions definides al filtre. Si l'objecte `Intent` no especifica una acció, passarà el test d'acció automàticament i es comprovaran la resta de camps.

Categoria

L'element `<intent-filter>` també defineix les categories com a subelements, per exemple:

```
<intent_filter>
  <categoryandroid:name="android.intent.category.DEFAULT"/>
  <categoryandroid:name="android.intent.category.BROWSABLE"/>
</intent_filter>
```

Fixeu-vos que es fan servir les cadenes completes per definir les accions i les categories (a diferència del que hem vist per a la creació de l'objecte `Intent`). Així, quan a l'objecte `Intent` definiu la categoria `CATEGORY_BROWSABLE`, al filtre d'*intent* s'ha de definir `android.intent.category.BROWSABLE`.

Perquè l'*intent* passi la comprovació de categoria, totes les categories definides a l'objecte `Intent` han d'estar presents al filtre d'*intents*.

Existeix una petita característica que cal aclarir. Tots els *intents* implícits que s'envien a `startActivity()` es tracten pel sistema com si tinguessin la categoria `CATEGORY_DEFAULT` definida. Per tant, les activitats que volen rebre *intents* implícits han d'incloure `android.intent.category.DEFAULT` en el seu filtre d'*intents*.

Els filtres `android.intent.action.MAIN` i `android.intent.category.LAUNCHER` són excepcions. Defineixen activitats que comencen noves tasques i que estan representades a la pantalla llançadora

d'aplicacions (la pantalla on l'usuari pot escollir entre totes les aplicacions que vol llançar). Poden incloure `android.intent.category.DEFAULT` en la llista de categories, encara que no ho necessiten.

Dades

La definició de dades està especificada com un subelement del filtre d'*intents*.

```
<intent-filter>
  <data android:mimeType="image/*"/>
  <data android:scheme="http" android:type="video/*"/>
</intent-filter>
```

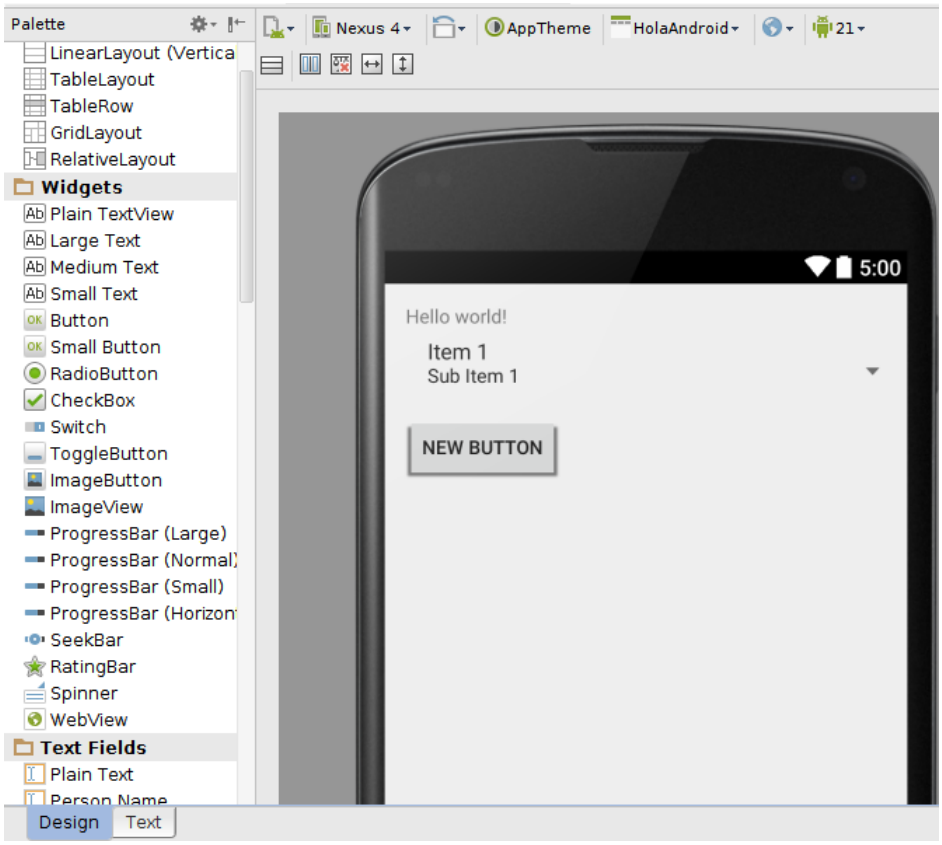
Cada element `<data>` pot especificar un URI i un tipus de dades (MIME). L'atribut `type` de l'element `<data>` especifica un tipus MIME de les dades. En l'exemple, el primer element `<data>` diu al sistema que el component pot obtenir una imatge i mostrar-la. El segon element és un filtre que defineix un esquema i un tipus de dades, que en aquest cas li diu al sistema Android que el component pot obtenir dades de vídeo de la xarxa i mostrar-les.

Com que la majoria de les dades es proporcionen amb proveïdors de continguts, els filtres que especifiquen tipus de dades no especifiquen URI. Tant l'objecte `Intent` com el filtre poden usar un comodí "*" per especificar un subtipus. Per exemple, ''`text/*`'' o ''`audio/*`'' indica que se seleccionarà qualsevol subtipus de `text` o d'`audio` respectivament.

2.5 Interfície d'usuari d'Android

Les activitats serveixen per interactuar amb l'usuari i mostren la interfície d'usuari (o **UI**, *User Interface*) de la vostra aplicació, que pot tenir elements gràfics (*widgets*, en anglès) com botons, caixes de text, etiquetes... La UI està definida al fitxer **activity_*.xml**, a la carpeta *res/layout* del vostre projecte. Si l'obriu a l'editor de l'Android Studio, veureu dues pestanyes: *Design* i *Text*, que mostren el contingut del fitxer XML i la interpretació gràfica del mateix, com es pot veure a la figura 2.8.

FIGURA 2.8. Edició de la UI des d'Android Studio



Quan s'executa l'aplicació es carrega el fitxer XML amb la UI al controlador d'*event* `onCreate()` de la vostra classe d'activitat, fent servir el mètode `setContentView()`.

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_hola_android);
    }
```

En la fase de compilació, cada element del fitxer XML és compilat en una classe d'UI d'Android equivalent amb els seus mètodes i atributs.

Crear la UI

En general, el més senzill és crear la UI mitjançant un fitxer XML, encara que és possible crear la UI mitjançant programació. En alguns casos, com els videojocs, aquesta darrera opció és preferible.

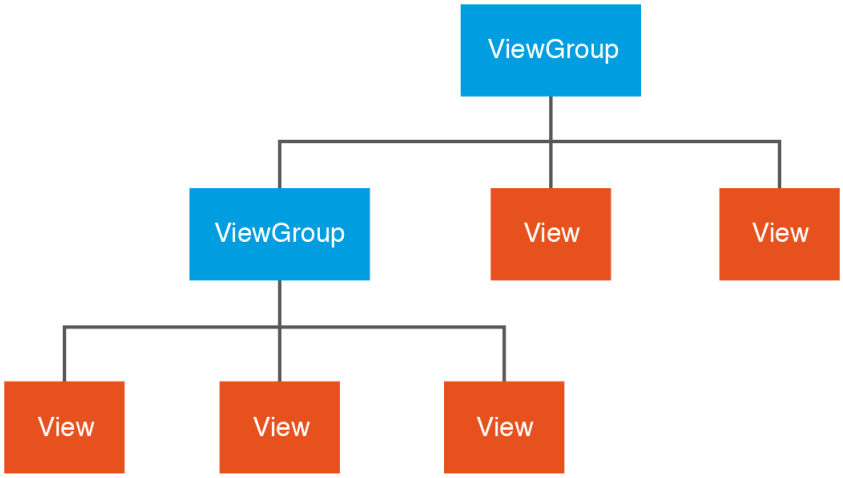
La UI a Android es construeix fent servir objectes `View` (vistes) i `ViewGroup` (grups de vistes). Existeixen moltes *Views* i *ViewGroups*, tots derivats de la classe `View`.

Els objectes `View` són la unitat bàsica de la UI de la plataforma Android. La classe `View`(android.view.View) serveix de classe base per a totes les subclasses anomenades ***widgets***, que ofereixen objectes UI totalment implementats com a camps de text i botons.

La classe `ViewGroup`(android.view.ViewGroup) serveix de classe base per a les subclasses anomenades ***layouts***, que proporcionen diferents tipus de disposicions dels elements. Una o més *Views* es poden agrupar juntes dins d'un *ViewGroup*. El *ViewGroup* (que en si mateix és un tipus de *View*, ja que hereta d'aquesta classe) proporciona una disposició en la qual es mostrarà la seqüència de *Views*.

A Android, es defineix la UI d'una activitat fent servir una jerarquia de *Views* i *ViewGroups*, tal com es pot veure a la figura 2.9.

FIGURA 2.9. Jerarquia que defineix una UI d'una activitat



Per enllaçar l'arbre jeràrquic de vista a la pantalla, l'activitat ha de cridar `setContentView()` i passar-li com a referència l'objecte al node arrel de la jerarquia. El node arrel de la jerarquia va cridant els seus nodes fills (els altres *Views* i *ViewGroups*) per tal que, finalment, tots es dibuixin en pantalla.

Android té els diferents tipus de *ViewGroups*:

- LinearLayout
- GridLayout
- TableLayout
- RelativeLayout
- FrameLayout
- ScrollView

El més habitual és combinar els diferents tipus de *layout* per crear la UI que volem (tal com es mostra a la figura 2.9, inserint-ne uns dins dels altres).

2.5.1 'Layout'

La forma més comuna de definir el *layout* de l'activitat i expressar la jerarquia de *Views* és a través d'uns fitxers XML de *layout*, que proporcionen una estructura del *layout* que resulta fàcil de seguir pel programador. Cada element XML és un objecte `View` o `ViewGroup`. Els objectes `View` serien les fulles de l'arbre i

els objectes `ViewGroup` les branques de l'arbre (fixeu-vos en la figura 2.9). Per exemple, el fitxer XML d'un *layout* lineal vertical amb quatre botons seria el següent:

```
<?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">

  <Button
    android:id="@+id/btnMapa"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Mapa"/>

  <Button
    android:id="@+id/btnWeb"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Web"/>

  <Button
    android:id="@+id/btnContactes"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Contactes"/>

  <Button
    android:id="@+id/btnTrucar"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Trucar"/>

</LinearLayout>
```

Fixeu-vos com el `LinearLayout` conté els elements Button. Es podria niuar un altre `LinearLayout` (o un altre tipus de *ViewGroup*) dins seu per afegir complexitat al *layout*. Si voleu saber més sobre els fitxers de *layout* XML, podeu consultar la *Guia del desenvolupador* d'Android a developer.android.com/guide/topics/ui/declaring-layout.html.

Si voleu veure les diferents característiques dels diferents *layouts*, les podeu trobar a developer.android.com/guide/topics/ui/layout-objects.html.

2.5.2 Creació del 'layout'

Per mostrar els diferents tipus de *layout* que existeixen es poden crear UI mitjançant l'edició d'un fitxer XML o mitjançant l'editor gràfic que proporciona Android Studio. Per obrir-lo, obriu el fitxer **.xml** que es troba a la carpeta *res/layout* del projecte.

Disposem de dues maneres fer de servir els *layout* existents:

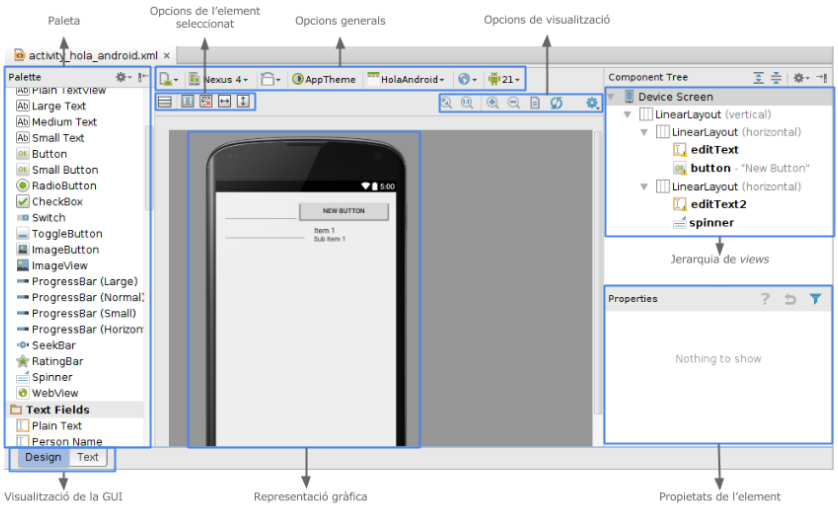
- Mitjançant l'edició d'un fitxer XML

- Mitjançant l'editor gràfic que proporciona Android Studio

Per fer servir l'editor gràfic d'Android Studio cal que obriu el fitxer **XML** que es troba a la carpeta *res/layout* del projecte.

A la figura 2.10 podeu veure les diferents parts de l'editor.

FIGURA 2.10. Editor de GUI a Android Studio



Les diferents parts són les següents:

- **Visualització de GUI** (*graphic user interface*, interfície gràfica d'usuari): amb aquestes pestanyes podeu escollir si voleu veure la GUI com un fitxer de text de tipus XML (pestanya "Text") o bé la seva representació gràfica (pestanya "Design").
- **Opcions generals**: inclou algunes opcions per visualitzar el *layout*, segons:
  - El tipus de dispositiu: mida de pantalla i resolució.
  - El seu format (emmarcat o apaïsat).
  - El seu idioma.
  - La versió de l'SDK.
  - El tema disponible.
- **Jerarquia de *Views***: a la dreta podeu veure la jerarquia de *Views* i *ViewGroups* que formen part de la GUI. Podeu fer clic a cadascuna d'elles per seleccionar-les i així modificar les seves propietats al requadre corresponent. Feu servir aquesta representació per tenir clar com està estructurada la vostra GUI.
- **Representació gràfica**: aquí podeu veure com es mostrarà la vostra interfície gràfica. Podeu seleccionar els *Views* i *ViewGroups* i desplaçar-los per les diferents parts de la GUI. Igualment, fent clic sobre cada element podreu modificar les seves propietats.

- **Paleta**: aquí teniu els diferents *Views* i *ViewGroups* que podeu fer servir a la vostra GUI. Per afegir qualsevol d'aquests *Views*, els podeu arrossegar amb el ratolí a la representació gràfica del GUI o dins de la jerarquia de *Views*. De vegades és més senzill introduir els *Views* a la jerarquia a través de la paleta. Feu clic a les diferents carpetes (*Widgets*, *Text Fields*, *Layouts*...) per fer una ullada als diferents elements gràfics amb els quals podeu treballar.
- **Opcions de visualització**: us permeten modificar la forma en què es visualitza la representació gràfica de la GUI.
- **Opcions de l'element seleccionat**: ofereix diferents opcions que es poden aplicar a l'element seleccionat. Les opcions varien entre els diferents *Views*; per tant, quan seleccioneu un *View* cal que us fixeu en quines són les seves opcions.

Android Studio dóna ajuda contextual sobre el cursor del ratolí. Si deixeu el cursor sobre els diferents botons de l'editor, us apareixerà una petita etiqueta explicant què fa aquest botó.

Els *Views* i *ViewGroups* tenen una sèrie d'atributs comuns que podeu veure a la taula 2.4.

Taula 2.4. Atributs comuns de View i ViewGroup

Atribut	Descripció.
`layout_width`	Amplada del View.
`layout_height`	Alçada del View.
`layout_marginTop`	Especifica l'espai extra a dalt del View.
`layout_marginBottom`	Especifica l'espai extra a baix del View.
`layout_marginLeft`	Especifica l'espai extra a l'esquerra del View.
`layout_marginRight`	Especifica l'espai extra a la dreta del View.
`layout_gravity`	Especifica com els Views fills es posicionen.
`layout_weight`	Especifica quant de l'espai extra del *layout* s'hauria de reservar per al View.
`layout_x`	Especifica la coordenada *x* del View.
`layout_y`	Especifica la coordenada *y* del View.

Atributs del View

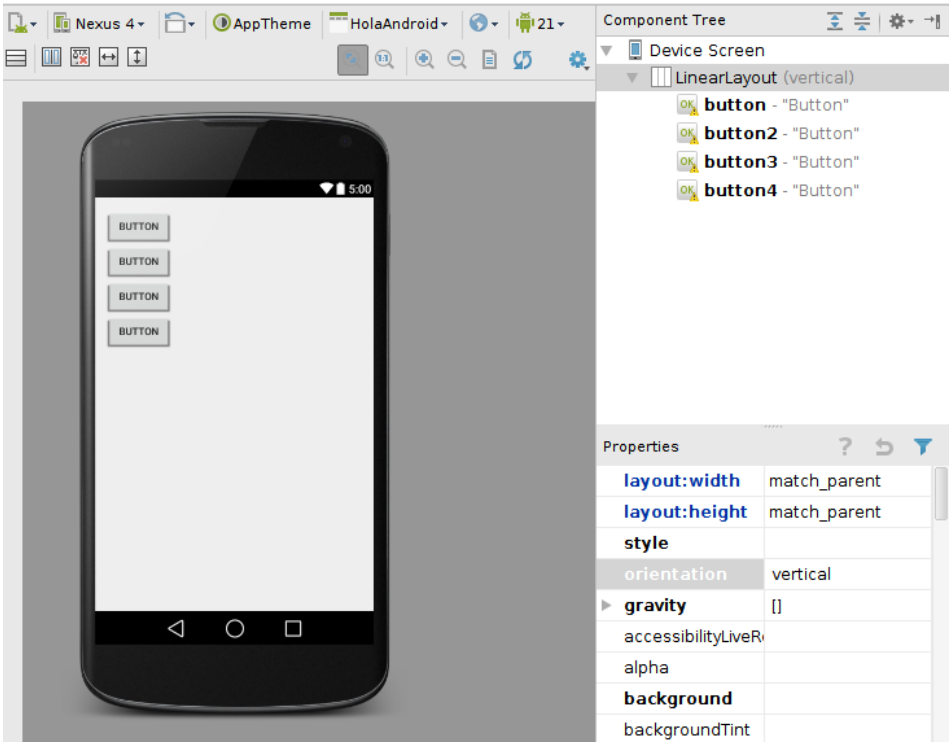
Alguns dels atributs del View únicament són aplicables quan el View està dins d'un ViewGroup. Per exemple, els atributs `layout_weight` i `layout_gravity`únicament s'apliquen quan el View està dins d'un `LinearLayout` o `TableLayout`.

Podeu veure els diferents tipus de *layouts* disponibles a la *Guia del desenvolupador* d'Android: developer.android.com/resources/tutorials/views/index.html.

Un `LinearLayout` mostra els *Views* en una única fila o columna, vertical o horitzontalment. Per exemple, a la figura 2.11 podeu veure un `LinearLayout` on hem introduït quatre botons.

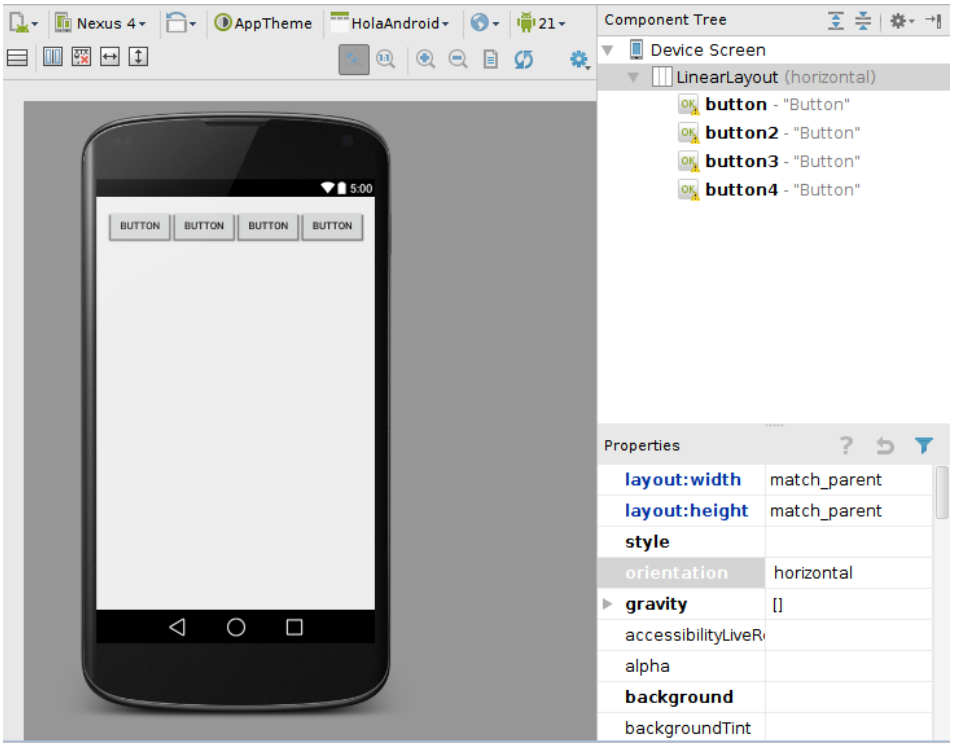
Tot seguit veurem el funcionament de `LinearLayout`; hem triat aquest *layout* perquè és el més senzill de fer servir. Si feu una ullada a la part de la documentació oficial que parla dels diferents *layouts* podreu veure altres estils gràfics disponibles.

FIGURA 2.11. Amb un 'Linear Layout' els components es distribueixen linealment



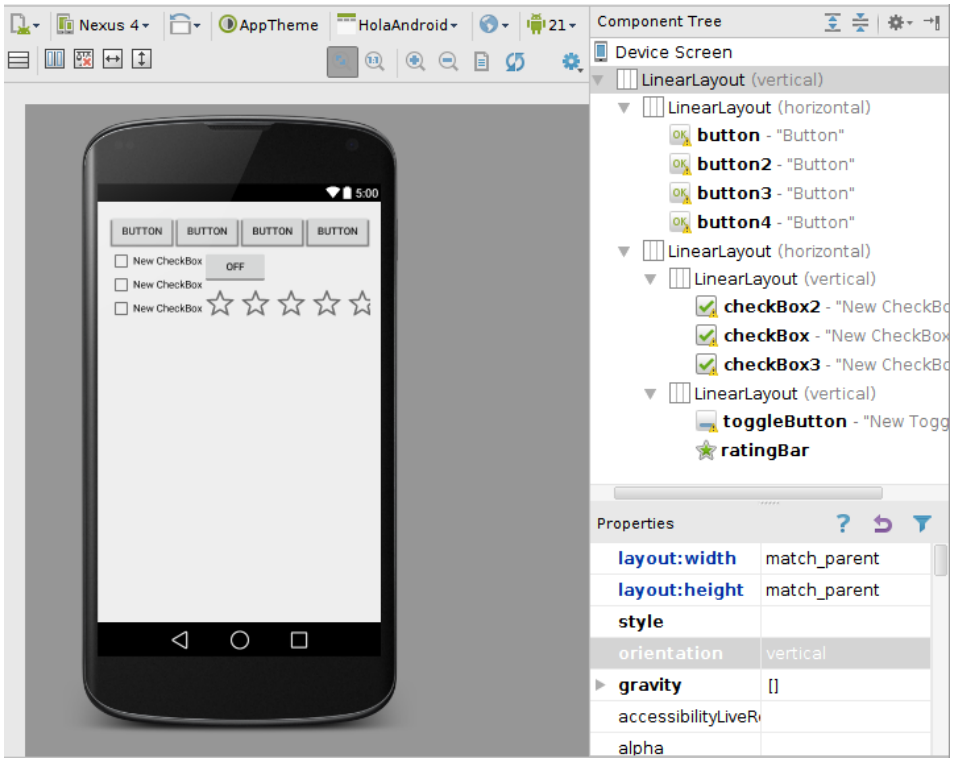
Clicant el botó dret del ratolí sobre `LinearLayout` podeu veure i modificar a la jerarquia de *Views* algunes de les seves propietats. Per exemple, si canvieu l'orientació a *Orientation/Horizontal* veureu com els botons es disposen de manera horitzontal, tal com es pot veure a la figura 2.12.

FIGURA 2.12. 'Linear Layout' horitzontal



Podeu niuar els *layouts* els uns dins dels altres per crear disposicions més complexes, tal com es pot veure a la figura 2.14.

Figura 2.13. 'Linear Layout' niuats



Podeu modificar les propietats de cada `View` seleccionant-les en el requadre *Properties*. Per exemple, podeu canviar el valor de l'ID que identifica el `View`, o les seves dimensions d'amplada i alçada. En la figura 2.14 podeu veure tres botons amb amplades diferents. El primer té l'amplada que ve per defecte, al segon li hem assignat el valor `Fill Parent` (omplir pare) a la propietat `Layout Width`(amplada del *layout*), i al tercer li hem assignat per aquesta mateixa propietat el valor de 100 dp (100 punts de pantalla).

Figura 2.14. Diferents valors de 'Layout Width'

Finalment, a partir del *layout* que creeu mitjançant les eines gràfiques es generarà un fitxer XML amb el qual el sistema Android sabrà com crear la interfície gràfica en temps d'execució. Per exemple, el *layout* de la figura 2.14 correspon al següent fitxer XML:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/LinearLayout1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">

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

    <Button
        android:id="@+id/button2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Button"/>

    <Button
        android:id="@+id/button3"
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:text="Button"/>

</LinearLayout>
```

Al començament és normal que feu servir l'editor gràfic per crear els vostres *layouts*, però quan aneu agafant experiència veureu que editar directament el fitxer XML fa molt més ràpida l'edició del *layout*.

Unitats de mesura

Per especificar la mida d'un element a la interfície gràfica d'Android es poden fer servir les següents unitats de mesura:

- **dp** (*density-independent pixel*, píxel independent de la densitat). És la recomanada per especificar la mida dels *Views* en el vostre *layout*.
- **sp** (*scale-independent pixel*, píxel independent de l'escala). Similar a la dp i recomanada per especificar mides en general.
- **pt** (*point*, punt). Un punt està especificat com 1/72 d'una polsada; està basat en la mida física de la pantalla.
- **px** (pixel). Correspon als píxels reals en pantalla. Aquesta mida no es recomana, atès que la vostra interfície gràfica es pot mostrar de formes diferents en els diferents dispositius.

2.5.3 'Layouts' XML

El *layout* defineix la forma en què els elements gràfics de l'activitat es mostren a l'usuari. Es pot declarar el *layout* de dues maneres diferents:

- **Declarant els elements gràfics a un fitxer XML**. Aquest fitxer contindrà les *Views* i les seves subclasses que definiran l'UI. L'avantatge d'aquest sistema és que permet separar la presentació de l'aplicació del codi que la controla. Així, podeu modificar la UI (o adaptar-la a un altre dispositiu) sense haver de recompilar el projecte. Per exemple, podeu crear diferents *layouts* XML per a diferents mides de pantalla, diferents orientacions de la pantalla o diferents llenguatges.
- **Instanciant els elements del *layout* en temps d'execució**. Es poden crear els elements gràfics des del mateix programa. Això és útil quan voleu crear una UI en funció d'alguna cosa que no sabeu en el moment de dissenyar l'aplicació (per exemple, en funció d'unes dades que rebeu per Internet).

Podeu fer servir qualsevol dels dos mètodes (tot i que generalment fareu servir el primer), i fins i tot ambdós a la vegada. Podeu dissenyar la UI de l'aplicació i després, en temps d'execució, modificar l'estat dels elements que estan en pantalla.

En general, el vocabulari XML per declarar interfícies d'usuari té una estructura similar als noms corresponents a les classes i mètodes que representen. Així, és fàcil saber a quines classes i atributs corresponen els elements de l'XML (encara que no sempre són idèntiques).

Es poden crear *layouts* d'UI i els elements que contenen ràpidament, fent servir un fitxer XML de la mateixa manera que es creen les pàgines web amb HTML, amb una sèrie d'element niuats. Cada fitxer *layout* ha de tenir un element arrel, que ha de ser un objecte `View` o `ViewGroup`. Una vegada definit l'element arrel, podeu afegir *widgets* o objectes de *layout* com a elements fills de l'arrel per crear a poc a poc la jerarquia de *Views* que definirà el vostre *layout*. Per exemple, podeu veure tot seguit el fitxer XML d'un *layout* que usa un `LinearLayout` i que inclou a dins un `TextView` i un `Button`:

```
<?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">

<TextView
  android:id="@+id/text"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Hola, sóc un TextView"/>

<Button
  android:id="@+id/button"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Hola, sóc un Button"/>
</LinearLayout>
```

Podeu trobar els fitxers d'android.R a dins del directori *app/build/generated/source/r/debug*, accessibles des de la vista *Project* de l'explorador del projecte.

Carregar el recurs XML

Els fitxers XML de *layout* són compilats a la vostra aplicació dins d'un recurs *View*. Això es produeix quan, en iniciar-se l'aplicació, és cridat el mètode de *callback* `onCreate()` de la vostra activitat, mètode a través del qual es carrega el recurs del *layout* des del codi. Per dur a terme aquest procés, crideu el mètode `setContentView (int layoutResID)`, que té com a argument un identificador de recurs de *layout*. Aquest ID, el podeu trobar a la classe que conté els recursos `android.R`. Concretament, els identificadors de *layouts* es troben dins de `R.layout.`*`nom_del_fitxer_layout`*`.` Així, si el nostre *layout* està contingut al fitxer **principal.xml**, trobareu l'identificador a `R.layout.principal`. La forma de carregar aquest recurs seria la següent:

```
1 public void onCreate(Bundle savedInstanceState)
2 {
3   super.onCreate(savedInstanceState);
4   setContentView(R.layout.principal);
5 }
```

Atributs

Els objectes de les classes `View` i `ViewGroup` tenen els seus propis atributs. Alguns d'ells són comuns a tots els *Views*, com l'ID, mentre que d'altres són específics de cada tipus de *View*; per exemple els `TextView` tenen l'atribut `textSize`, que no està present en tots els *Views*.

Tots els objectes `View` tenen un número d'ID al qual estan associats, i que identifiquen el *View* de forma unívoca. Aquest ID està present en forma de *string* al fitxer XML, encara que en compilar l'aplicació és referenciat mitjançant un *Integer*. Aquest atribut és present a tots els objectes `View` i derivats d'ell. La sintaxi per definir l'ID al fitxer XML és la següent:

```
1 android:id="@+id/el_meu_boto"
```

El símbol + diu al compilador que es tracta d'un nou recurs i que s'ha de crear i afegir als recursos del projecte (que es troben al fitxer "R.java"). Quan es fa referència a un ID de recurs d'Android, s'ha de fer així:

```
1 android:id="@android:id/empty"
```

Introduint el *namespace* del paquet, esteu referenciant l'ID de la classe de recursos `android.R`.

La forma habitual de crear Views i referenciar-los des de l'aplicació és la següent:

En primer lloc, cal definir el `View` en el fitxer XML de *layout* i assignar-li un ID únic:

```
1 <Button android:id="@+id/botoAcceptar"
2   android:layout_width="wrap_content"
3   android:layout_height="wrap_content"
4   android:text="@string/acceptar"/>
```

Tot seguit creeu la instància de l'objecte `View` (o classe derivada d'ell) que correspongui i referencieu-lo al del *layout* (generalment es fa servir la funció de *callback* `onCreate()`):

```
1 Button Boto =(Button) findViewById(R.id.botoAcceptar);
```

A partir d'aquest moment ja podeu començar a treballar amb el botó des del codi.

2.5.4 'Widget' i events de teclat

Un ***widget*** és un objecte `View` que serveix d'interfície d'interacció amb l'usuari. Existeixen multitud de *widgets* ja implementats a Android, com botons, *checkboxes*, caixes de text, etc. El programador pot crear els seus propis *widgets* definint una classe que sigui descendent de `View` o d'un *widget* ja existent (per exemple, pot heretar les propietats d'una caixa de text).

Una vegada afegits els *widgets*, caldrà programar la interacció entre el programa i l'usuari. Cal que captureu els *events* de l'objecte `View` amb què l'usuari està interactuant i fer que succeeixi quelcom (executar una part de codi).

La classe `View` (de la qual els *widgets* hereten les propietats) té una sèrie de **mètodes de*callback*** que serveixen per tractar els *events* de la interfície gràfica. Aquests mètodes són cridats pel sistema quan l'acció corresponent succeeix en aquest objecte. Per exemple, quan es toca un botó, el sistema crida el mètode de *callback* `onTouchEvent()`.

La classe `View` té una sèrie d'**interfícies** anomenades `On<quelcom>Listener`, cadascuna amb un mètode de *callback* anomenat `on<quelcom>()`. Per exemple, `OnClickListener` (per tractar els clics a un *View*) defineix el mètode `onClick()`.

Aquestes interfícies, anomenades ***event listeners*** (*escoltadors* d'esdeveniments), us permeten capturar la interacció de l'usuari amb la UI. Si voleu que el vostre `View` sigui notificat en ser clicat, heu d'implementar l'interfície `OnClickListener`, definir el mètode `onClick()`, que establirà l'acció que s'ha de realitzar quan es faci clic, i registrar-lo al View amb `setOnClickListener()`.

> Un ***event listener*** és una interfície de la classe `View` que conté un únic mètode de *callback*. Aquests mètodes seran cridats pel sistema Android quan el `View` (al qual està registrat l'*event listener*) s'activa per la interacció de l'usuari.

Per exemple, podeu definir un *event listener* amb un mètode de *callback* i registrar-lo a un `View` de tipus botó. Quan l'usuari premi el botó, el sistema cridarà el mètode de *callback* definit i podreu executar el codi que vulgueu.

Per això heu de definir un *event listener* i registrar-lo al View. Un *event listener* és un mètode que es cridarà pel sistema quan es produeixi un esdeveniment en concret, per exemple quan es premi un botó.

Per registrar un event listener, en primer lloc heu d'obtenir una referència al View sobre el que voleu definir la funció de callback. Per obtenir la referència podeu fer servir el mètode `findViewById(int id)` de l'objecte de l'activitat. L'argument del mètode (l'identificador) és el que està definit al fitxer AndroidManifest.xml i el podeu obtenir a partir de l'objecte `R.id`.

Per exemple, per obtenir una referència a un botó anomenat "buttonSi" faríeu el següent:

```
Button btnSi;

//Listener delbotoSi
btnSi = (Button) findViewById(R.id.buttonSi);
```

Podeu trobar tots els identificadors de *widgets* de la vostra activitat a la variable `R.id`. Una vegada tingueu la referència al botó, fareu una crida al mètode `setOnClickListener (View.OnClickListener l)`. Aquest mètode registra el *callback* que serà invocat quan aquest View sigui clicat. Com a únic argument, se li passa la funció de *callback* que s'executarà.

Ho podeu fer de dues maneres diferents. A la primera, senzillament, definiu la funció de *callback* a la mateixa crida a `setOnClickListener()`. Per exemple:

```
btnSi.setOnClickListener(

  new OnClickListener()
  {
    @Override
    publicvoid onClick(View v)
    {
      //Codi que s'executarà quan es faci clic
      Toast.makeText(c, "Boton si", Toast.LENGTH_LONG).show();
    }
  }
);
```

En aquest codi es cridasetOnClickListener() i com a argument es fa una implementació de la interfície `OnClickListener()` que té un únic mètode per definir, onClick(). Dins d'aquest mètode estarà el codi que s'executarà quan es premi aquest botó. En aquest cas, el codi correspon a mostrar un missatge per pantalla amb `Toast`.

Això s'hauria de fer per cada View que existeixi al *layout* i amb què es vulgui interactuar.

La segona forma consisteix a implementar `OnClickListener` com a part de l'activitat. Això evitarà haver de crear totes les implementacions de `OnClickListener`, simplement diem que la nostra activitat implementa la interfície `OnClickListener` afegint-ho a la seva definició amb:

```
public class ProvabotonsActivity extends Activity implements OnClickListener {
```

Ara, l'activitat ja implementa la *interface*, i únicament s'ha de sobreescriure el mètode `OnClick(View v)`. En aquest cas es cridarà aquest mètode independentment de quin View hagi estat clicat, per tant el primer que s'haurà de fer és comprovar quin ha sigut l'objecte clicat, com es pot veure al següent codi:

```
public class ProvabotonsActivity extends Activity implements OnClickListener {

Button btnSi, btnNo;

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

  //Definim els listeners
  btnSi = (Button)findViewById(R.id.buttonSi);
  btnSi.setOnClickListener(this);
  btnNo = (Button)findViewById(R.id.buttonNo);
  btnNo.setOnClickListener(this);
}

public void onClick(View v)
{
  // Fer alguna cosa quan s'ha polsat un View.

    // Comprovem quin ha estat el View clicat.
    if(v== btnSi)
    {
      //Executem el codi
     Toast.makeText(this, "Botó si", Toast.LENGTH_LONG).show();
    }
    else if(v == btnNo)
    {
      Toast.makeText(this, "Botó no", Toast.LENGTH_LONG).show();
    }
  }
}
```

A les *interfaces* d'*event listeners* existeixen el següents mètodes de *callback*:

- `onClick()`: de `OnClickListener`. Es crida quan l'usuari fa clic en l'ítem.
- `onLongClick()`: de `OnLongClickListener`. Es crida quan l'usuari fa una pulsació llarga damunt de l'ítem.
- `onFocusChange()`: de `OnFocusChangeListener`. Es crida quan l'usuari navega en o fora de l'ítem fent servir els cursors o el *trackball*.
- `onKey()`: de `OnKeyListener`. Es crida quan l'usuari té el focus a l'ítem i prem o solta una tecla del dispositiu.
- `onTouch()`: de `OnTouchListener`. Es crida quan l'usuari fa una acció de *tocar* l'ítem (inclou prémer, soltar o un gest sobre l'ítem).
- `onCreateContextMenu()`: de `OnCreateContextMenuListener`. Es crida quan un menú contextual es crea sobre l'ítem (resultat d'una pulsació llarga o de la pulsació de la tecla de menú).

Programació avançada i de comunicacions

Eduard García Sacristán, Joan Climent Balaguer

Programació multimèdia i dispositius mòbils

Índex

Introducció

La programació d'aplicacions per dispositius mòbils, en general, consta de més funcionalitats que unes simples pantalles que s'envien entre elles informació introduïda per l'usuari. Una vegada tenim clar l'esquema de l'aplicació i la transició entre pantalles, és el moment d'anar més enllà i atribuir-li un seguit de particularitats específiques.

Android ens proporciona una API i unes llibreries de classes que donen al programador l'oportunitat de crear aplicacions completes. A banda de la programació de la interfície gràfica de l'aplicació (amb totes les seves complexitats), existeixen altres elements que poden enriquir la vostra aplicació, com ara la persistència de dades i la comunicació de la vostra aplicació amb Internet.

En l'apartat "Programació avançada" veureu tècniques avançades de programació en Android. S'hi tractaran diferents formes d'obtenir persistència de dades, mitjançant bases de dades i proveïdors de continguts. També veureu la manera de preparar les aplicacions per a la seva publicació i distribució.

En l'apartat "Programació de comunicacions" veureu com treballar amb les diferents llibreries de comunicacions que ofereix Android per a la programació de les seves aplicacions. Atès que Android és un sistema dissenyat per a dispositius mòbils, la connexió contínua és un dels seus aspectes fonamentals. Vegeu la comunicació web i els serveis de missatgeria instantània i multimèdia.

Per seguir els continguts d'aquest mòdul, és convenient anar fent les activitats i els exercicis d'autoavaluació i llegir els annexos (si n'hi ha). 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

En finalitzar aquesta unitat, l'alumne/a:

1. Desenvolupa aplicacions per a dispositius mòbils analitzant i fent servir les tecnologies i llibreries específiques.
    - Utilitza les classes necessàries per a la connexió i comunicació amb dispositius sense fils.
    - Utilitza les classes necessàries per a l'intercanvi de missatges text i multimèdia.
    - Utilitza les classes necessàries per establir connexions i comunicacions HTTP i HTTPS.
    - Utilitza les classes necessàries per establir connexions amb magatzems de dades garantint la persistència.
    - Realitza proves d'interacció usuari-aplicació per optimitzar les aplicacions desenvolupades a partir d'emuladors.
    - Empaqueta i desplega les aplicacions desenvolupades en dispositius mòbils reals.
    - Documenta els processos necessaris per al desenvolupament de les aplicacions.

1. Programació avançada

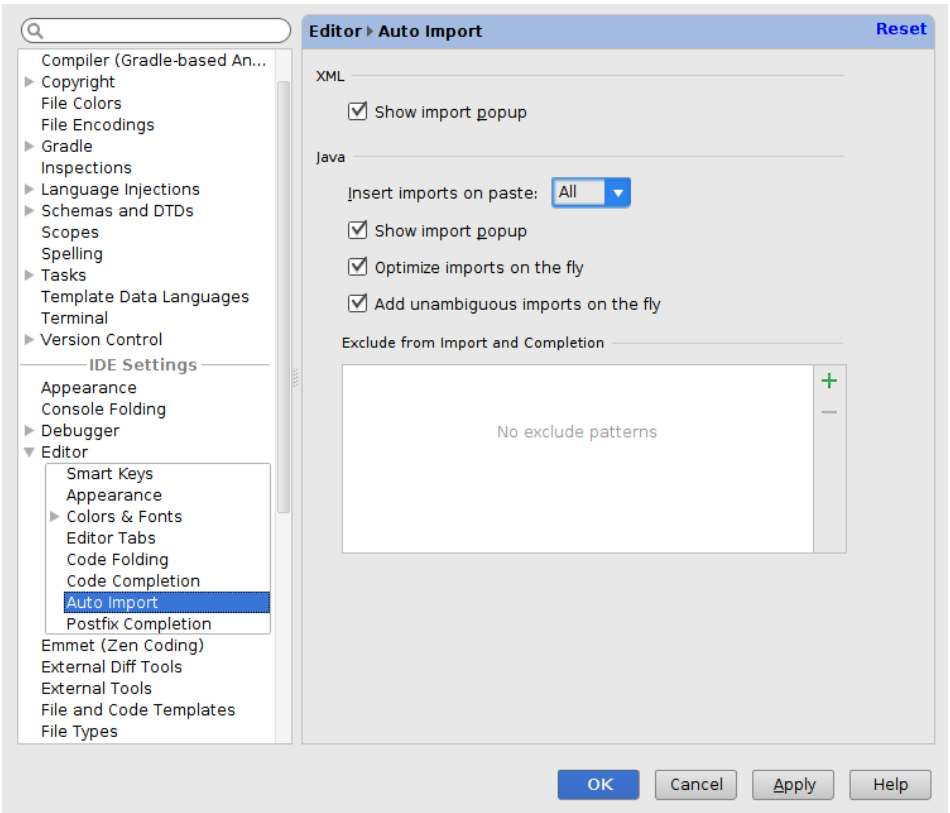
Quan hàgiu creat l'esquema d'una senzilla aplicació, és el moment d'afegir funcionalitats avançades, com una interfície d'usuari avançada, l'accés a les dades de l'aplicació (mitjançant bases de dades o proveïdors de continguts) o la persistència de les dades de la mateixa. Així mateix, quan penseu que teniu l'aplicació suficientment desenvolupada, és el moment de publicar l'aplicació en qualsevol de les diferents formes que teniu per fer-ho.

1.1 Imports a l'Android Studio

Per facilitar la programació amb l'Android Studio modificarem les preferències per automatitzar els *imports* a mesura que anem escrivint o que enganxem codi, així obtindrem una major agilitat a l'hora d'escriure codi.

Accedirem a les preferències des de *File/Settings* i al submenú *Editor/Auto Import* haurem de marcar els *checkboxes Optimize imports on the fly* i *Add unambiguous imports on the fly*. A més, podem modificar el valor d'*Insert imports on paste* d'*Ask* a *All* (vegeu la figura 1.1).

FIGURA 1.1



Amb aquestes modificacions l'Android Studio farà tots els *imports* automàticament, és per això que sempre haurem de comprovar que l'import sigui el correcte per evitar comportaments no desitjats.

1.2 Persistència

És probable que vulgueu que la vostra aplicació pugui desar algunes dades entre les diferents execucions de l'aplicació. Per exemple, podeu voler desar les preferències de l'aplicació per tal que la propera vegada que l'executeu tingui la mateixa aparença que la darrera vegada que es va executar. Existeixen diferents formes d'obtenir persistència en les dades de l'aplicació en diferents execucions:

- Un mecanisme lleuger anomenat **preferències compartides** (*shared preferences*) per desar petites quantitats de dades.
- El sistema de fitxers tradicional.
- Una base de dades relacional SQLite.

Per desar poca informació, la millor forma de fer-ho és amb les preferències compartides. Android incorpora l'objecte `SharedPreferences`, que serveix per desar i llegir dades persistents en la forma clau-valor de dades primitives. Podeu fer servir `SharedPreferences` per desar qualsevol tipus de dades primitives: *boolean*, *float*, *int*, *long* i *string*. Aquestes dades es mantindran entre sessions, encara que la vostra aplicació s'hagi tancat. A més, seran desades automàticament en un fitxer XML.

Per obtenir un objecte`SharedPreferences` a la vostra aplicació podeu fer servir el mètode `getSharedPreferences()` amb dos arguments: el nom del fitxer de preferències i el mode d'operació. Per exemple:

```
1 private SharedPreferences prefs;
2 //Obtenir l'objecte SharedPreferences
3 prefs = getSharedPreferences("FitxerPreferences", MODE_PRIVATE);
```

En el cas que vulgueu fer servir un únic fitxer de preferències, podeu cridar el mètode `getPreferences()`, on no cal especificar el nom del fitxer.

Per escriure valors al fitxer de preferències:

1. Crideu el mètode `edit()` per obtenir un objecte `SharedPreferences.Editor`.
2. Afegiu valors amb els mètodes que us permeten escriure valors primitius, com `putBoolean()` o `putString()`. Aquests mètodes tenen dos arguments. El primer és un *string* que defineix la clau, i el segon és el valor que es vol desar.

3. Confirmeu els valors amb `commit()`.

Per exemple:

```
SharedPreferences.Editor editor = prefs.edit();

editor.putInt("Edat", 23);
editor.putString("NomUsuari", "Fidel");
```

Per llegir els valors de preferències podeu fer servir els mètodes anàlegs `getBoolean()`, `getString()` o `getInt()` de la classe `SharedPreferences`. Aquests mètodes tenen dos arguments: el primer és el nom de la clau que esteu cercant al fitxer de preferències. El segon és el valor per defecte que es farà servir en cas que no es trobi la clau en el fitxer de preferències. Per exemple:

```
//Carregarlespreferències
SharedPreferences prefs = getSharedPreferences("FitxerConfiguracio",
    MODE_PRIVATE);

int Edat = prefs.getInt("Edat", 18);
String nom = prefs.getString("NomUsari", "Usuari");
```

El lloc més adequat per carregar i desar les preferències de l'aplicació serien els mètodes `onCreate()` i `onStop()` (el moment en què l'aplicació s'engega i el moment en què es tanca).

1.3 Treballant amb bases de dades

Android proporciona un sistema de bases de dades relacionals basat en SQLite que podeu fer servir a les vostres aplicacions. Quan la quantitat de dades a desar és important, o bé es volen fer cerques sobre les dades, o la informació està relacionada entre si, és més adequat tenir-la estructurada en forma d'una base de dades. Per exemple, podríeu tenir una base de dades amb informació de diferents fabricants relacionada amb els diferents productes que aquests produeixen. Fent servir bases de dades, podeu assegurar la integritat de les dades especificant relacions entre els diferents conjunts de dades.

Android fa servir el sistema de bases de dades SQLite. La base de dades que podeu crear per a la vostra aplicació únicament estarà disponible per a la pròpia aplicació. La resta d'aplicacions del dispositiu no hi podran accedir, per tant no podeu fer servir la base de dades per compartir dades entre aplicacions.

Diem que una aplicació té un accés transparent a les dades quan podem accedir a la informació a través de mètodes sense haver de conèixer la seva implementació. Aquesta manera de treballar ens permet modificar l'estructura interna de la informació sense que les aplicacions que la fan servir hagin de modificar el seu codi per funcionar correctament.

Treballar amb bases de dades a Android pot ser complicat. Per aquest motiu, heu de crear una classe que servirà per encapsular l'accés a les bases de dades i, així, simplificar el codi de la vostra aplicació (que tindrà un accés a les dades transparent a la seva implementació, a través d'aquesta classe).

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

1.3.1 Classe DBInterface

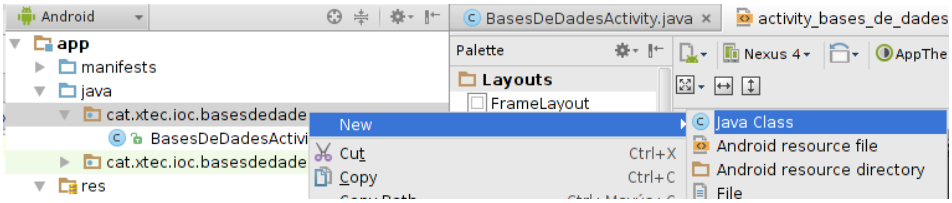
Creeu un nou projecte amb les següents dades:

- **Application name**: BasesDeDades
- **Company domain**: cat.xtec.ioc
- **Blank Activity**: BasesDeDadesActivity

Deixeu la resta d'opcions amb els valors per defecte.

Ara creareu una classe que us servirà d'interfície amb l'accés a les bases de dades de l'aplicació. Aquesta classe tindrà el nom DBInterface. Creeu una aplicació, i dins d'aquesta, creeu una nova classe fent clic amb el botó dret dins del paquet del vostre projecte i seleccionant *New/Class*, com es pot veure a la figura 1.2.

FIGURA 1.2. Creant una nova classe



Aquesta classe crearà, obrirà, farà servir i tancarà una base de dades SQLite. Creareu una base de dades anomenada BDClients que contindrà una única taula, *contactes*. Aquesta taula tindrà únicament tres camps: _id, nom i email, tal com es pot veure a la taula 1.1.

TAULA 1.1. Taula de contactes

_id	nom	email
1	John	jcoltrane@atlantic.com
2	Miles	mdavis@bluenote.com

En primer lloc, cal que definiu una sèrie de constants de text que serviran per establir alguns identificadors i camps (i així no caldrà haver-los de repetir per tot el codi, amb el perill d'equivocar-se en algun moment).

```
1  package ioc.xtec.cat.basesdedades;
2
3  import android.content.ContentValues;
4  import android.content.Context;
5  import android.database.Cursor;
6  import android.database.SQLException;
7  import android.database.sqlite.SQLiteDatabase;
8  import android.database.sqlite.SQLiteOpenHelper;
9  import android.util.Log;
10
11
12 public class DBInterface {
13     //Constants
```

```
14      public static final String CLAU_ID = "_id";
15      public static final String CLAU_NOM = "nom";
16      public static final String CLAU_EMAIL = "email";
17
18      public static final String TAG = "DBInterface";
19
20      public static final String BD_NOM = "BDClients";
21      public static final String BD_TAULA = "contactes";
22      public static final int VERSIO = 1;
23
24      public static final String BD_CREATE =
25              "create table " + BD_TAULA + "( " + CLAU_ID + " integer primary key
                    autoincrement, " +
26                    CLAU_NOM + " text not null, " + CLAU_EMAIL + " text not
                          null);";
27
28      private final Context context;
29      private AjudaBD ajuda;
30      private SQLiteDatabase bd;
31  }
```

</newcontent>

Concretament, la constant `BD_CREATE` conté la cadena que es farà servir per crear la taula *contactes* dins de la base de dades.

La classe `AjudaBD`, la creareu més tard. El constructor de la nostra classe, l'únic que fa és crear un objecte `AjudaBD` i guardar en una variable el context en què s'està executant la classe:

```
1  public DBInterface(Context con)
2    {
3      this.context = con;
4      ajuda = new AjudaBD(context);
5    }
```

`Context` és una classe implementada pel sistema Android que dóna accés a recursos i classes específics de l'aplicació.

A Android existeix la classe `SQLiteOpenHelper`, que és una classe que serveix d'ajuda per gestionar la creació de bases de dades i gestió de versions. Creareu la classe `AjudaBD` que hereta d'aquesta:

```
1  private static class AjudaBD extends SQLiteOpenHelper {
2          AjudaBD(Context con) {
3              super(con, BD_NOM, null, VERSIO);
4          }
5
6          @Override
7          public void onCreate(SQLiteDatabase db) {
8              try {
9                  db.execSQL(BD_CREATE);
10             } catch (SQLException e) {
11                 e.printStackTrace();
12             }
13         }
14
15         @Override
16         public void onUpgrade(SQLiteDatabase db, int VersioAntiga, int
                VersioNova) {
17             Log.w(TAG, "Actualitzant Base de dades de la versió" + VersioAntiga
                    + " a " + VersioNova + ". Destruirà totes les dades");
18             db.execSQL("DROP TABLE IF EXISTS " + BD_TAULA);
19
```

```
            onCreate(db);
        }
    }
```

El constructor d'`AjudaDB` crida el constructor d'`SQLiteOpenHelper`, el qual crea un objecte d'ajuda per crear, obrir i gestionar la base de dades. Mireu la documentació d'`SQLiteOpenHelper` per obtenir ajuda sobre els seus mètodes. Aquesta classe s'ocupa d'obrir la base de dades si aquesta existeix o crear-la en cas contrari, i actualitzar-la si és necessari.

El mètode `onCreate()` crea una nova base de dades. El mètode `onUpgrade()` és cridat quan s'ha d'actualitzar la base de dades. El que fa és eliminar-la (fer un *drop* de la taula) i tornar-la a crear.

De tornada a la classe `DBInterface`, el següent pas és definir els diferents mètodes per obrir i tancar la base de dades.

```
//Obre la BD

public DBInterface obre() throws SQLException {
    bd = ajuda.getWritableDatabase();
    return this;
}

//Tanca la BD

public void tanca() {
    ajuda.close();
}
```

El mètode `getWritableDatabase()` crea i/o obre una base de dades. La primera vegada que es crida s'obre la base de dades i es crida `onCreate()`. Aquí és on es crea la base de dades, cridant `execSQL()` amb la cadena de creació de la base de dades. Una vegada creada, la base de dades queda en *caché*, i per tant es pot cridar aquest mètode per obrir-la.

A continuació definireu els mètodes per modificar la base de dades. Per inserir un contacte fareu servir el mètode `insert(String table, String nullColumnHack, ContentValues values)`. Els arguments són aquests:

- `String table`: taula on es vol inserir un element.
- `String nullColumnHack`: un argument opcional que deixarem a *null*. Podeu consultar la *Guia del desenvolupador* d'Android per consultar el seu ús.
- `ContentValues values`: un objecte de la classe `ContentValues` que serveix per emmagatzemar valors que poden ser processats per un `ContentResolver`.

Aquest mètode retorna l'ID de la fila que s'ha inserit o un -1 si hi ha hagut una errada. El nostre mètode per inserir un contacte crearà un `ContentValues` amb els valors de la fila a inserir i farà la crida a `insert()`.

```
//Insereix un contacte

public long insereixContacte(String nom, String email
{
ContentValues initialValues = new ContentValues();
initialValues.put(CLAU_NOM, nom);
initialValues.put(CLAU_EMAIL, email);
return bd.insert(BD_TAULA ,null, initialValues);

}
```

Per esborrar un element de la taula fareu servir el mètode `delete (String table, String whereClause, String[] whereArgs)`. El significat dels arguments és el següent:

- `String table`: taula de la qual s'esborrarà el registre.
- `String whereClause`: la clàusula `WHERE` que s'aplicarà per esborrar de la base de dades. Si se li passa *null*, esborrarà totes les files de la taula.
- `String[] whereArgs`: arguments de la clàusula `WHERE`. Aquest mètode retorna la quantitat de files afectades per la clàusula `WHERE`.

```
//Esborra un contacte

public boolean esborraContacte(long IDFila)
{
return bd.delete(BD_TAULA, CLAU_ID + " = " + IDFila, null) > 0;
}
```

El vostre mètode retornarà un valor booleà que indica si s'ha esborrat algun element o no.

Per retornar un contacte fareu servir el mètode `query()`, que té els següents arguments:

- `boolean distinct`: serà *true* si voleu que cada fila sigui única, o *false* en cas contrari.
- `String table`: defineix la taula respecte a la qual voleu executar la sentència de *query*.
- `String[] columns`: admet una llista de les columnes de la taula que retornarà el mètode.
- `String selection`: estableix un filtre que defineix quines files retornar, amb el format de clàusula d'SQL, WHERE (sense incloure la paraula `WHERE` a l'*string*).
- `String[] selectionArgs`: permet afegir els arguments de la selecció, si no els heu introduït directament a la cadena.
- `String groupBy`: estableix un filtre que defineix com s'agrupen les files. Té el mateix format que la clàusula d'SQL: GROUP BY (sense incloure les paraules GROUP BY). Si se li passa un *null*, les files que es retornin no estaran agrupades.

- `String having`: estableix un filtre que declara quins grups de files incloure al cursor. Té el mateix format que la clàusula d'SQL: HAVING (sense incloure la paraula HAVING). Si se li passa un *null*, s'inclouran tots els grups.
- `String orderBy`: indica com ordenar les files. Té el mateix format que la clàusula d'SQL: ORDER BY (sense incloure les paraules ORDER BY). Si se li passa un *null* retorna les files amb l'ordre per defecte.
- `String limit`: especifica el límit de files retornades pel *query*, amb el format de la clàusula d'SQL: LIMIT. Si se li passa un *null*, no existeix límit.

Aquest mètode retorna un objecte de la classe `Cursor`, que proporciona accés de lectura i escriptura al resultat retornat per una consulta (un *query*) a la base de dades.

```
1  //Retorna un contacte
2
3  public Cursor obtenirContacte(long IDFila) throws SQLException {
4    Cursor mCursor = bd.query(true, BD_TAULA, new String[] {CLAU_ID, CLAU_NOM,
         CLAU_EMAIL}, CLAU_ID + " = " + IDFila, null, null, null, null, null);
5
6    if(mCursor != null) {
7      mCursor.moveToFirst();
8    }
9
10   return mCursor;
11
12 }
```

Per obtenir tots els contactes, feu servir una altra versió de *query* que no inclogui el primer booleà.

```
1  //Retorna tots els contactes
2
3  public Cursor obtenirTotsElsContactes()
4  {
5    return bd.query(BD_TAULA, new String[] {CLAU_ID, CLAU_NOM, CLAU_EMAIL}, null,
         null, null, null, null);
6  }
```

Fixeu-vos que Android fa servir un objecte de la classe `Cursor` per valor de retorn de les consultes a la base de dades (els *queries*). Penseu en el `Cursor` com un apuntador al conjunt de resultats obtinguts de la consulta a la base de dades. L'ús d'un `Cursor` permet a Android gestionar d'una forma més eficient les files i les columnes.

Finalment, per actualitzar un registre de la taula fareu servir el mètode`update()` amb els següents arguments:

- `String table`: estableix la taula a actualitzar.
- `ContentValues values`: introdueix un objecte de la classe `ContentValues` amb el valor de les columnes a actualitzar.
- `String whereClause`: inclou la clàusula `WHERE` opcional que s'ha d'aplicar quan es fa l'actualització. Si se li passa *null* actualitzarà totes les files.

- `String[] whereArgs`: estableix els arguments del `WHERE`.

El codi resultant és el següent:

```
//Modifica un contacte

public boolean actualitzarContacte(long IDFila, String nom, String email) {
  ContentValues args = new ContentValues();
  args.put(CLAU_NOM, nom);
  args.put(CLAU_EMAIL, email);
  returnbd.update(BD_TAULA, args, CLAU_ID + " = " + IDFila, null) > 0;
}
```

1.3.2 Fent servir la classe "DBInterface"

Alguns dels *layouts* s'han omès, els podreu trobar al codi font del projecte.

Quan hàgiu creat la classe que us servirà d'ajuda per treballar amb les bases de dades, és el moment de fer-la servir. El següent exemple mostra una aplicació que fa servir la classe d'interfície de base de dades. És molt senzilla, però serveix per il·lustrar la forma d'ús d'aquesta classe.

L'activitat principal té un *layout* amb cinc botons, un per a cada opció de l'aplicació, tal com es pot veure a la figura 1.3.

Figura 1.3. Layout de l'aplicació

El codi que correspon a aquest *layout* és el següent:

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

```
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/btnAfegir"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Afegir" />

    <Button
        android:id="@+id/btnObtenir"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Obtenir" />

    <Button
        android:id="@+id/btnObtenirTots"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Obtenir Tots" />

    <Button
        android:id="@+id/btnActualitzar"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Actualitzar" />

    <Button
        android:id="@+id/btnEsborrar"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Esborrar" />

</LinearLayout>
```

L'aplicació activarà una activitat per a cadascun dels botons, equivalent a les diferents "pantalles" de l'aplicació. Aquestes diferents activitats han d'estar definides a l'AndroidManifest.xml. Recordeu que heu de definir al fitxer de *manifest* els filtres d'*intents* per a cada activitat, per tal que les diferents activitats s'activin quan es llancin els *intents* adequats per a cada una d'elles. Aquest és el document de *manifest* de l'aplicació:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ioc.xtec.cat.basesdedades" >

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".BaseDeDadesActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".Afegir"
```

```
            android:label="@string/title_activity_afegir" >
            <intent-filter>
                <action android:name="cat.xtec.ioc.AFEGIR" />

                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
        <activity
            android:name=".Obtenir"
            android:label="@string/title_activity_obtenir" >
            <intent-filter>
                <action android:name="cat.xtec.ioc.OBTENIR" />

                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
        <activity
            android:name=".Esborrar"
            android:label="@string/title_activity_esborrar" >
            <intent-filter>
                <action android:name="cat.xtec.ioc.ESBORRAR" />

                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
        <activity
            android:name=".Actualitzar"
            android:label="@string/title_activity_actualitzar" >
            <intent-filter>
                <action android:name="cat.xtec.ioc.ACTUALITZAR" />

                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

L'activitat principal implementa la interfície `OnClickListener` per gestionar les accions que es deriven quan l'usuari prem els botons. Tot seguit teniu el codi parcial que correspon a la definició de la classe i de les variables que farem servir a l'activitat:

```
public class BasesDeDadesActivity extends Activity implements OnClickListener {

  Button btnAfegir, btnObtenir, btnObtenirTots, btnActualitzar, btnEsborrar;
  DBInterface bd;
```

El mètode `onCreate()` senzillament carrega el *layout*, crea l'objecte d'interfície de la base de dades i crea els *listeners* dels botons.

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

  bd = new DBInterface(this);

  //Listeners dels botons
  btnAfegir = (Button) findViewById(R.id.btnAfegir);
  btnAfegir.setOnClickListener(this);

  btnObtenirTots = (Button) findViewById(R.id.btnObtenirTots);
  btnObtenirTots.setOnClickListener(this);

  btnObtenir = (Button) findViewById(R.id.btnObtenir);
  btnObtenir.setOnClickListener(this);
```

```
16
17    btnEsborrar = (Button) findViewById(R.id.btnEsborrar);
18    btnEsborrar.setOnClickListener(this);
19
20    btnActualitzar = (Button) findViewById(R.id.btnActualitzar);
21    btnActualitzar.setOnClickListener(this);
22
23  }
```

A continuació implementarem cadascuna de les accions que corresponen als botons:

- Afegir
- Obtenir
- Obtenir tots
- Actualitzar
- Esborrar

Afegir

En primer lloc, cal definir el *listener* del botó:

```
1  //Afegir
2  if (v == btnAfegir) {
3    startActivity(new Intent("cat.xtec.ioc.AFEGIR"));
4  }
```

Això llançarà un *intent* per a l'activitat que permetrà afegir elements a la base de dades. Aquesta activitat té un disseny molt senzill, incorpora tan sòls els *widgets* necessaris per afegir un nou element, com es pot veure a la figura 1.4.

Figura 1.4. Activitat d'afegir contactes

Aquest és el *listener* del botó *Afegir*:

```
1  @Override
2      public void onClick(View v) {
3          if(v == btnAfegir){
```

```
            //Obrim la base de dades
            bd = new DBInterface(this);
            bd.obre();

            //Inserim el contacte
            if(bd.insereixContacte(editNom.getText().toString(), editEmail.
                getText().toString()) != -1) {
                Toast.makeText(this, "Afegit correctament", Toast.LENGTH_SHORT)
                    .show();
            } else {
                Toast.makeText(this, "Error a l'afegir", Toast.LENGTH_SHORT).
                    show();
            }
            bd.tanca();
            finish();
        }
    }
```

Bàsicament, el que es fa és:

1. Obrir la base de dades.
2. Inserir un element amb les dades que obteniu de les caixes de text. Fixeu-vos com comprova el valor de retorn d'`insereixContacte()` per saber si s'ha pogut inserir el contacte nou correctament o no.
3. Tancar la base de dades.
4. Tancar l'activitat.

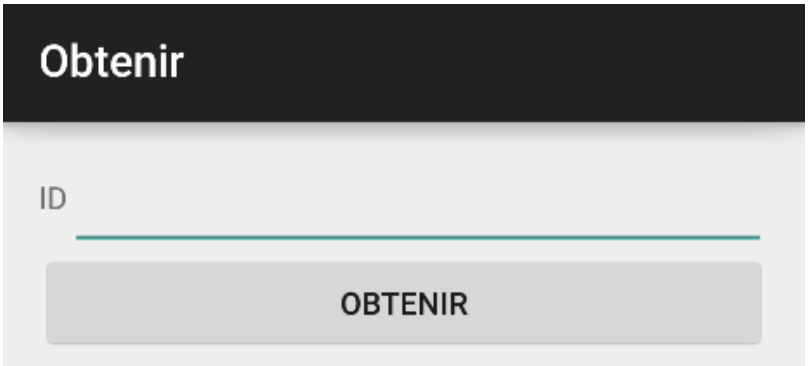
Obtenir

Tot seguim definirem un nou *listener*, aquest cop pel botó d'*Obtenir*.

```
btnObtenir = (Button) findViewById(R.id.btnObtenir);
btnObtenir.setOnClickListener(this);

...

else if (v == btnObtenir) {
            startActivity(new Intent("cat.xtec.ioc.OBTENIR"));
        }
```

Això us permetrà llançar una activitat en la que els únics *widgets* seran una caixa de text i el botó per obtenir el contacte, com es pot veure a la figura 1.5.

Figura 1.5. Activitat per obtenir un contacte



Al codi de l'activitat l'única cosa que es programa és el *listener* del botó *Obtenir*. El codi és el següent:

```
@Override
    public void onClick(View v) {
        //Botó obtenir
        if (v == btnObtenir) {
            Cursor c;

            //Obrim la base de dades
            bd = new DBInterface(this);
            bd.obre();

            //Aquest és l'identificador que està escrit a la caixa de text
            long id = Long.parseLong(editID.getText().toString());

            //Crida a la BD
            c = bd.obtenirContacte(id);

            // Comprovem si hi ha hagut algun resultat
            if (c.getCount() != 0) {
                //Mostrem el contacte
                Toast.makeText(this, "id: " + c.getString(0) + "\n" + "Nom: " +
                     c.getString(1) + "\n Email: " + c.getString(2), Toast.
                    LENGTH_SHORT).show();
            } else {
                Toast.makeText(this, "id inexistent!", Toast.LENGTH_SHORT).show
                    ();

            }
            //Tanquem la BD
            bd.tanca();

            //Tanca l'activitat
            finish();
        }
    }
```

El que hem fet és:

1. Obrir la base de dades
2. Obtenir l'identificador que està escrit a la caixa de text
3. Cridar la base de dades
4. Comprovar si hi ha hagut algun resultat
5. Mostrar el contacte
6. Tancar la base de dades
7. Tancar l'activitat

El més normal seria mostrar el contacte a través d'una altra activitat dissenyada per mostrar contactes, però per fer el programa més senzill hem optat per mostrar el contacte amb un missatge per pantalla.

Obtenir tots

L'opció *Obtenir tots* mostrarà tots els contactes per pantalla, un a un. En una aplicació de veritat el més habitual seria introduir els contactes en un `ListView`

o un `ContentProvider`, però en aquest exemple volem mostrar únicament el funcionament de les bases de dades, per tant qualsevol forma de mostrar els contactes ens serà suficient. Atès que el programa no necessita cap altra informació extra per mostrar tots els contactes (i que, per tant, no es demanarà cap informació a l'usuari), aquesta opció no obrirà una altra activitat, sinó que mostrarà els contactes un a un per pantalla. Dins del *listener* del botó obtenim tots els contactes de la base de dades i fem un recorregut amb el cursor que se'ns retorna per anar mostrant tots els contactes.

```
else if (v == btnObtenirTots) {
            //Obrir base de dades
            bd.obre();

            //Crida la BD per obtenir tots els contactes
            Cursor c = bd.obtenirTotsElsContactes();

            //Movem el cursor a la primera posició
            if (c.moveToFirst()) {
                do {
                    //Mostrem contactes...
                    MostraContacte(c);
                    //... mentre puguem passar al següent contacte
                } while (c.moveToNext());
            }
            //Tanquem la BD
            bd.tanca();

            Toast.makeText(this, "Tots els contactes mostrats", Toast.
                LENGTH_SHORT).show();
        }
```

El que hem fet és:

1. Obrir la base de dades
2. Cridar la BD per obtenir tots els contactes
3. Moure el cursor a la primera posició
4. Mostrar els contactes mentre es pugui
5. Tancar la base de dades
6. Mostrar un missatge per pantalla quan s'hagi acabat de mostrar els contactes.

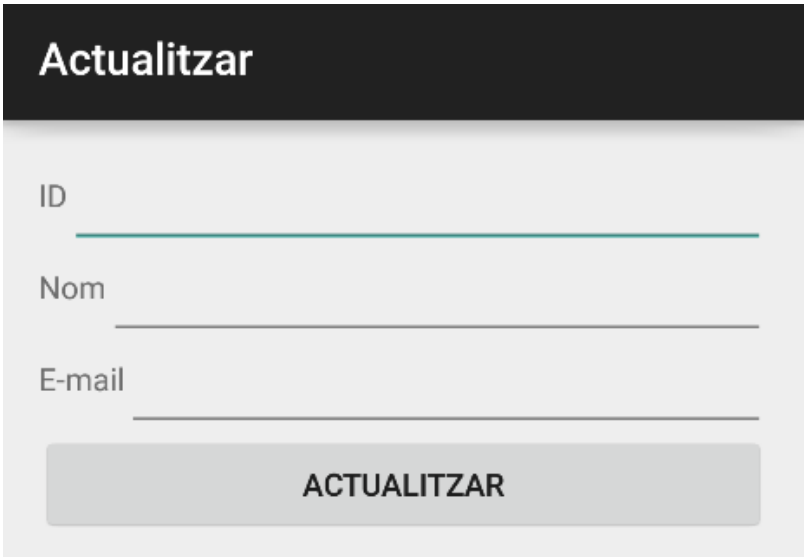
El mètode `MostraContacte()` simplement fa una crida a Toast per mostrar el contacte per pantalla.

```
public void MostraContacte(Cursor c)
{
  Toast.makeText(this, "id: "+ c.getString(0) + "\n" + "Nom: "+ c.getString(1)
      + "\n Email: " + c.getString(2), Toast.LENGTH_SHORT).show();
}
```

Actualitzar

L'opció d'actualitzar permet modificar les dades d'un contacte a partir del seu identificador. Quan hàgiu creat el *listener*, es mostrarà una activitat com la que es pot veure a la figura 1.6.

FIGURA 1.6. Activitat actualitzar




```
@Override
    public void onClick(View v) {
        if (v == btnActualitzar) {
            long id;

            //Obtenim la BD
            bd = new DBInterface(this);
            bd.obre();

            //Identificador de la caixa de text
            id = Long.parseLong(editID.getText().toString());

            //Crida a la BD
            boolean result = bd.actualitzarContacte(id, editNom.getText().
                toString(), editEmail.getText().toString());

            //Comprovem el resultat, si s'ha pogut actualitzar la BD o no
            if (result)
                Toast.makeText(this, "Element modificat", Toast.LENGTH_SHORT).
                    show();
            else
                Toast.makeText(this, "No s'ha pogut modificar l'element", Toast
                    .LENGTH_SHORT).show();

            //Tanquem la BD
            bd.tanca();

            //Tanca l'activitat.
            finish();

        }

    }
```

El que hem fet és:

1. Obrir la base de dades
2. Obtenir l'identificador de la caixa de text
3. Cridar la BD
4. Comprovar el resultat per determinar si s'ha pogut actualitzar la BD o no
5. Tancar la base de dades
6. Tancar l'activitat

El funcionament del codi és molt semblant. Obteniu les dades que necessiteu des dels *widgets* de l'activitat (les caixes de text) i feu la crida a la base de dades. En aquest cas, a més a més, es comprova el valor de retorn de la funció per saber si s'ha pogut actualitzar la base de dades correctament.

Esborrar

La funció d'esborrar té una estructura molt similar a les altres funcions. Obté d'una caixa de text l'ID del contacte que es vol esborrar i fa una crida a la base de dades per esborrar el contingut corresponent. Es comprova el valor de retorn per saber si el contacte s'ha eliminat correctament.

```
@Override
    public void onClick(View v) {
        if (v == btnEsborrar) {
            //Obrim la BD
            bd = new DBInterface(this);
            bd.obre();

            //Obtenim l'ID de la caixa de text
            long id = Long.parseLong(editID.getText().toString());

            //Cridem la BD
            boolean result = bd.esborraContacte(id);

            //Comprovem el resultat de l'operació
            if (result)
                Toast.makeText(this, "Element esborrat", Toast.LENGTH_SHORT).
                    show();
            else
                Toast.makeText(this, "No s'ha pogut esborrar l'element", Toast.
                    LENGTH_SHORT).show();
            //Tanquem la BD
            bd.tanca();

            //Tanquem l'activitat
            finish();
        }
    }
```

El que hem fet és:

1. Obrir la base de dades

2. Obtenir l'identificador de la caixa de text
3. Cridar la BD
4. Comprovar el resultat de l'operació
5. Tancar la base de dades
6. Tancar l'activitat

1.4 Proveïdors de continguts

Els *content providers* (proveïdors de continguts) donen accés a una sèrie estructurada de dades i serveixen d'interfície de dades estàndard per connectar dades d'un procés amb el codi que s'està executant en un altre procés. Els proveïdors de continguts són la forma recomanada de compartir dades entre aplicacions. Són magatzems de dades que serveixen per compartir dades entre aplicacions. Es comporten de forma similar a una base de dades (podeu fer consultes, editar el contingut, afegir i esborrar), però a diferència d'aquestes fan servir diferents formes per emmagatzemar les seves dades. Aquestes poden estar en una base de dades, en fitxers o fins i tot a la xarxa, però per al procés que les fa servir això és transparent (i ens resulta indiferent).

Android fa servir una sèrie de proveïdors de continguts, estàndards del sistema, que poden fer servir la resta d'aplicacions, per exemple:

- *Browser*: emmagatzema dades com els marcadors del navegador, l'historial de navegació, etc.
- *CallLog*: emmagatzema dades com crides perdudes, detall de les trucades, etc.
- *ContactsContract*: emmagatzema informació dels contactes: nom, *email*, telèfon, foto, etc.
- *MediaStore*: emmagatzema dades multimèdia com imatges, àudio i vídeo.
- *Settings*: emmagatzema dades de configuració i preferències del dispositiu com el *bluetooth*, wifi, etc.

Podeu trobar una llista de proveïdors (del paquet `android.provider`) a la *Guia del desenvolupador* d'Android, que podeu trobar aquí:

http://developer.android.com/reference/android/provider/package-summary.html.

A banda d'aquests proveïdors de continguts, podeu crear els vostres propis.

Quan vulgueu accedir a les dades d'un proveïdor de continguts haureu de servir un objecte `ContentResolver` en el context de la vostra aplicació. Aquest objecte treballa com a client amb un objecte proveïdor, que treballa com a servidor:

un objecte `ContentProvider`. El proveïdor rep les dades dels clients, realitza la tasca i retorna els resultats. Heu de crear un `ContentProvider` si voleu compartir dades de la vostra aplicació amb altres, però per fer servir dades d'altres aplicacions simplement necessiteu un `ContentResolver`.

1.4.1 Accedint al proveïdor de continguts

El `ContentProvider` presenta dades a les aplicacions, com una o més taules similars a les que trobem a les bases de dades relacionals. Cada fila representa un element del proveïdor de continguts, i cada columna una dada concreta del mateix element de la fila.

L'aplicació accedeix a les dades del proveïdor a través de l'objecte client `ContentResolver`. Aquest objecte conté mètodes que criden altres mètodes (per cert, amb el mateix nom) a la classe `ContentProvider`. Per exemple, per obtenir una llista de contactes podeu cridar el mètode `ContentResolver.query()`, i aquest quest cridarà el mètode `query()` de `ContentProvider`. El mètode `query()` té una sèrie d'arguments que serveixen per definir la consulta que es vol fer al proveïdor. Aquests arguments tenen un paral·lelisme amb les opcions d'una consulta *query* a una base de dades:

- `Uri uri`: URI que representa la taula d'on s'obtindran les dades.
- `String[] projection`: llista de les columnes que es retornaran per cada fila de la taula.
- `String selection`: filtre que indica quines files retornar amb el mateix format que la clàusula `WHERE` d'SQL (sense la paraula "WHERE"). Si es passa un *null* retornarà totes les files de l'URI.
- `String[] selectionArgs`: en l'argument anterior (*selection*), en lloc d'incloure el valor de les columnes, directament podeu indicar "= ?" en la selecció. Els "?" seran substituïts pels valors de l'*array* selectionArgs en l'ordre en què apareixen a *selection*.
- `String sortOrder`: estableix com s'ordenen les files, amb el mateix format que la clàusula d'SQL `ORDER BY` (excloent les paraules "ORDER BY"). Si es passa *null* es farà servir l'ordre per defecte.

Es poden consultar els arguments i el seu equivalent en un `SELECT` d'SQL a la taula 1.2.

TAULA 1.2. Equivalència d'arguments entre query() i una consulta SELECT d'SQL

Argument de `query()`	**Equivalent en** `SELECT`**d'SQL**
`Uri uri`	`FROM` taula
`String[] projection`	columna, columna, columna
`String selection`	`WHERE` col = valor

TAULA 1.2 (continuació)

Argument de `query()`	**Equivalent en** `SELECT`**d'SQL**
`String[] selectionArgs`	No existeix un equivalent
`String sortOrder`	`ORDER BY` col, col,...

El mètode retorna un objecte de la classe `Cursor`, posicionat abans de la primera entrada o *null* en cas de trobar algun problema.

L'URI especifica les dades en el proveïdor. Els URI inclouen el nom del proveïdor (anomenat *authority*, 'autoritat') i un nom que apunta a una taula (*path*). El `ContentProvider` fa servir el *path* de l'URI per escollir la taula a la qual accedirà (té un *path* per a cada taula). L'estructura d'un URI és aquesta:

```
1 <prefix>://<authority>/<path>/<id>
```

on:

- El *prefix* per als proveïdors de continguts sempre és: `content://`.
- *authority* especifica el nom del proveïdor de continguts. Per als estàndards del sistema, per exemple: media, call_log, browser. Per a proveïdors de tercers, millor introduir un nom de domini complet com: com.android.ioc.
- El *path* especifica la taula dins del proveïdor.
- L'*id* és un identificador únic per a una fila concreta de la taula especificada al *path*.

A la taula 1.3 es poden veure alguns proveïdors de continguts del sistema.

TAULA 1.3. Alguns exemples de proveïdors de continguts del sistema.

String	Descripció
content://media/internal/images	Llista de les imatges en la memòria del dispositiu
content://media/external/images	Llista de les imatges en la memòria externa del dispositiu (per exemple, en la targeta SD)
content://call_log/calls	Llista de trucades fetes amb el dispositiu
content://browser/bookmarks	Llista dels marcadors del navegador web

D'altra banda, molts proveïdors de continguts tenen una constant amb l'URI d'accés a ells mateixos. Per exemple, en lloc de crear vosaltres l'URI per accedir al diccionari de paraules del dispositiu podeu fer servir el que ell mateix té definit: `UserDictionary.Words.CONTENT_URI`. O per accedir a la llista de contactes, `ContactsContract.Contacts.CONTENT_URI`.

Per obtenir dades d'un proveïdor, la vostra aplicació necessita d'un permís de lectura per part del proveïdor. No es pot demanar aquest permís durant l'execució, així que per obtenir-lo caldrà que feu servir l'element `<uses-permission>` en el vostre fitxer de *manifest* especificant el permís que voleu obtenir del proveïdor. En quedar definit al *manifest*, quan l'usuari instal·la aquesta aplicació li està

donant els permisos implícitament. Per saber exactament de quins permisos consta el proveïdor així com seu nom, consulteu la documentació del proveïdor. Per exemple, per demanar que la vostra aplicació tingui permisos de lectura sobre el proveïdor de contactes del dispositiu heu d'indicar-ho al fitxer de *manifest* de la següent manera, abans de l'etiqueta `<application>`:

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

El següent pas és realitzar una consulta al proveïdor per obtenir les seves dades. El següent codi mostra com fer una consulta per obtenir tots els contactes de la llista de contactes:

```
//Les columnes que volem obtenir per a cada element.
String[] projection = new String[] {
  ContactsContract.Contacts._ID,
  ContactsContract.Contacts.DISPLAY_NAME,
  ContactsContract.Contacts.HAS_PHONE_NUMBER
};

//Condició: volem obtenir totes les files (per això és null).
String where = null;
String[] whereArgs = null;

//Ordre: que estiguin ordenats de forma ascendent.
String sortOrder = ContactsContract.Contacts.DISPLAY_NAME + " COLLATE LOCALIZED
     ASC";

Cursor c = getContentResolver().query
(
  ContactsContract.Contacts.CONTENT_URI,
  projection, // Columnes per obtenir de cada fila
  where,      // Criteri de selecció
  whereArgs,  // Criteri de selecció
  sortOrder   // Ordre
);
```

Aquest codi obtindrà tota la llista de contactes i retornarà un cursor al resultat, la variable c. Quan hàgiu obtingut el cursor, heu de mirar el seu valor per saber si hi ha hagut un error, o si el cursor té dades o no. Ho podeu comprovar de la següent manera:

```
//Si hi ha hagut un error
  if (c == null){
  //Codi per tractar l'error, escriure logs, etc.

  }
  //Si el cursor està buit, el proveïdor no ha trobat resultats.
  elseif (c.getCount() < 1) {
    //El cursor està buit, el content provider no té elements.
    Toast.makeText(this, "No hi ha dades", Toast.LENGTH_SHORT).show();
  } else {
    //Dades obtingudes
    Toast.makeText(this, "OK", Toast.LENGTH_SHORT).show();
  }
```

En aquests moments teniu un cursor a les dades obtingudes del proveïdor, així que podríeu recórrer les dades amb el cursor i treballar-hi. Per exemple, el següent codi mostra com obtenir l'identificador i el nom, i com saber si el contacte té número de telèfon:

```
//Mentre tenim un nou element al cursor
while(c.moveToNext()) {
  //Obtenir ID
  String contactId = c.getString(c.getColumnIndex(ContactsContract.Contacts._ID
      ));

  //Obtenir nom
  String nomContacte = c.getString(c.getColumnIndex(ContactsContract.Contacts.
      DISPLAY_NAME));

  //Saber si té telèfon
  String hasPhone = c.getString(c.getColumnIndex(ContactsContract.Contacts.
      HAS_PHONE_NUMBER));

  //Mostrar
  Toast.makeText(this, "id: "+ contactId + "\n" + "Nom: "+ nomContacte + "\n Té
       telèfon: " + hasPhone , Toast.LENGTH_SHORT).show();
}

c.close();
```

Per obtenir una columna a partir del cursor heu de fer servir el mètode `Cursor.getString(int ColumnIndex)`, al qual se li passa l'índex de la columna que voleu obtenir. Per obtenir el número de columna a partir de l'identificador del camp que voleu, feu servir `Cursor.getColumnIndex(String columnName)`. Tot això ho podeu fer a la vegada:

```
String nomContacte = c.getString(c.getColumnIndex(ContactsContract.Contacts.//
    DISPLAY_NAME//));
```

El codi anterior us mostra per pantalla la informació de tots els contactes del proveïdor tal com estan al cursor. Però, què passa si voleu filtrar aquesta informació? Treballant amb SQL ho podeu fer amb una sentència *WHERE columna = valor*. Això ho podeu definir al mètode `query()` amb els arguments `where` i `whereArgs`. En realitat, podríeu obtenir fàcilment la mateixa sentència `WHERE` modificant el valor de l'argument `where`. Si voleu obtenir al cursor únicament els contactes que tinguin telèfon, per exemple, podeu definir l'argument com:

```
String where = ContactsContract.Contacts.HAS_PHONE_NUMBER + "= 1";
```

O, per obtenir únicament els contactes amb nom IOC:

```
String where = ContactsContract.Contacts.DISPLAY_NAME + "= 'IOC'";
```

Per qüestions de seguretat i per evitar que l'usuari pugui introduir codi SQL maliciós, es pot fer servir el caràcter "?" a la variable `where` dins de la sentència. Els caràcters "?" seran substituïts amb els valors de l'*array* d'*strings* `whereArgs`. Per exemple, la sentència anterior seria equivalent a:

```
String where = ContactsContract.Contacts.DISPLAY_NAME + "= ?";
String[] whereArgs = {"IOC"};
```

SQL Injection

L'objectiu de `whereArgs` i de la substitució dels "?" és impedir la inclusió de codi SQL maliciós dins de la sentència (vegeu aquest enllaç a la Wikiquipedia). Imagineu que creeu un codi per accedir a tots els contactes que tinguin un nom igual a una variable que

introdueix l'usuari ("WHERE name = valor_variable"). I aquest usuari, en lloc d'introduir el nom d'una persona, introdueix: "nothing; DROP TABLE *". La sentència SQL resultant, en sers executada: "WHERE name = nothing. DROP TABLE *" aconseguiria esborrar totes les taules de la base de dades (si tingués permís per fer-ho).

Fins ara heu mostrat el nom, l'identificador i una variable que diu si el contacte té o no telèfon. Però obtenir el telèfon i el correu electrònic és una mica més complicat. Atès que a la llista un contacte pot tenir diversos números de telèfon i diverses adreces de correu electrònic, aquestes no estan emmagatzemades com variables estàtiques sinó com un proveïdor de continguts dins del proveïdor de continguts dels contactes. Per tant, per obtenir-les s'ha de fer una altra consulta i obtenir un cursor a la llista de telèfons i correus electrònics, respectivament. El següent codi recorre la llista de telèfons i correus i els va desant a una variable de tipus `String` (es podrien mostrar o fer qualsevol altra cosa amb ells). Al final, la variable es queda amb l'últim telèfon i correu que serà el que es mostri del contacte (encara que es podria mostrar el primer, o tots).

```
String telefon = null;
String email = null;

if (hasPhone.compareTo("1") == 0) {
  // Obtenim els telèfons
  Cursor telefons = getContentResolver().query(
  ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
  null,
  ContactsContract.CommonDataKinds.Phone.CONTACT_ID +" = "+ contactId,
  null,
  null);

  //Recorrem els telèfons
  while (telefons.moveToNext()) {
    telefon = phones.getString(phones.getColumnIndex( ContactsContract.
        CommonDataKinds.Phone.NUMBER));
  }

//Tanquem el cursor
  telefons.close();
}

//Obtenir cursor correus
Cursor emails = getContentResolver().query(ContactsContract.CommonDataKinds.
    Email.CONTENT_URI, null, ContactsContract.CommonDataKinds.Email.CONTACT_ID
     + " = " + contactId, null, null);

//Recorrem els correus
 while (emails.moveToNext()) {
  email = emails.getString(emails.getColumnIndex(ContactsContract.
      CommonDataKinds.Email.DATA));
}
//Tanquem el cursor
emails.close();

//Mostrar
Toast.makeText(this, "id: "+ contactId + "\n" + "Nom: "+ nomContacte + "\n
    Telefon: " + telefon + "\n email: " + email, Toast.LENGTH_SHORT).show();
```

1.4.2 Inserint dades

La forma d'inserir dades a un proveïdor de continguts és similar a la consulta. Per fer-ho, existeix el mètode `ContentResolver.insert()`. Aquest mètode insereix una nova fila al proveïdor de continguts i retorna l'URI de la fila creada. Els valors de la fila queden definits amb un objecte de la classe `ContentValues`. Per exemple, el següent codi serveix per inserir una nova paraula al diccionari personal de l'usuari del dispositiu. Perquè funcioni, s'han de donar permisos d'escriptura al diccionari de dades del dispositiu inserint la següent sentència al document de *manifest*:

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

El següent fragment de codi mostra com s'insereix un nou valor al proveïdor de continguts:

```
Uri UriNou;

ContentValues Valors = new ContentValues();

//Creem el valor de la nova entrada
Valors.put(UserDictionary.Words.APP_ID, "com.android.ioc");
Valors.put(UserDictionary.Words.LOCALE, "es_ES");
Valors.put(UserDictionary.Words.WORD, "Hospitalet");
Valors.put(UserDictionary.Words.FREQUENCY, "100");

//Inserim
UriNou = getContentResolver().insert(
  UserDictionary.Words.CONTENT_URI,
  Valors
);
```

1.5 Publicant aplicacions

Publicar és el procés de fer les vostres aplicacions d'Android disponibles per als usuaris. La publicació consta principalment de dues tasques:

- Preparar l'aplicació per a la distribució, compilant una versió de l'aplicació per a distribució.
- Distribuir l'aplicació als usuaris, on es publicita, ven i distribueix la versió del programa als usuaris.

El procés de publicació es realitza després d'haver comprovat el funcionament de l'aplicació en un entorn de depuració.

Preparar l'aplicació consta d'una sèrie de passos:

- Compileu i signeu la versió de distribució de l'aplicació. L'Android Studio proporciona tot el necessari per fer-ho.
- Traieu les crides a `Log`. També, quan feu la generació del fitxer .apk assegureu-vos de seleccionar a *build type* l'opció *release* i no *debug*. A més, afegiu o modifiqueu els valors dels atributs `versionCode` i `versionName` que estan a l'element `defaultConfig` del `build.gradle`
- Abans de distribuir l'aplicació, heu de comprovar la versió de distribució d'aquesta. Idealment, l'hauríeu de comprovar almenys en un telèfon i una tauleta.
- Assegureu-vos que tots els recursos (imatges, vídeos...) de l'aplicació estan actualitzats.
- Prepareu els servidors remots i els serveis si és que la vostra aplicació depèn d'aquests.
- Creeu una icona per a l'aplicació.
- Si voleu, podeu preparar un EULA (*End User License Agreement*, acord de llicència d'usuari final) per protegir la vostra propietat intel·lectual i la vostra persona.

Quan acabeu de preparar l'aplicació, creareu un fitxer signat .apk que podreu distribuir als usuaris.

1.5.1 Preparar l'aplicació

Per publicar una aplicació heu de crear un paquet que els usuaris puguin instal·lar i executar en els seus dispositius Android (versió *release*). El paquet de publicació de la versió conté els mateixos components que el fitxer .apk de depuració (codi font compilat, recursos, fitxer de *manifest*...) i es construeix amb les mateixes eines. Però el fitxer .apk que es publica està signat amb el vostre certificat i està optimitzat.

Per preparar l'aplicació per ser publicada, generalment es realitzen les següents tasques:

1. Preparació de materials i recursos.
2. Configuració de l'aplicació per a la publicació.
3. Compilació de l'aplicació per a la publicació.
4. Preparació dels servidors externs i recursos
5. Comprovació de l'aplicació per a la publicació.

Claus criptogràfiques

El sistema Android requereix que cada aplicació instal·lada estigui signada digitalment amb un certificat propietat del seu desenvolupador (un certificat del qual el desenvolupador en tingui la clau privada). Android fa servir aquest certificat per identificar l'autor i establir relacions de confiança entre aplicacions.

Un requeriment per publicar a Google Play és que l'aplicació estigui signada amb una clau criptogràfica amb un període de validesa que acabi després del 22 d'octubre de 2033.

Icona de l'aplicació

La vostra aplicació ha de tenir una icona i ha de seguir les recomanacions sobre com han de ser aquestes, tal com podeu trobar a la *Guia del Desenvolupador* d'Android:

http://developer.android.com/guide/practices/ui_guidelines/icon_design_launcher.html

La icona de l'aplicació serveix per identificar-la i pot aparèixer a la pantalla principal, al *launcher* del dispositiu, a *les meves descàrregues*, quan es gestionen les aplicacions instal·lades... A més a més, si publiqueu l'aplicació a Google Play es mostrarà la icona als usuaris (en aquest cas també heu de publicar una versió d'alta resolució de la icona).

Altres continguts

Potser voleu preparar un EULA (*End User License Agreement*, acord de llicència d'usuari final). També, si voleu publicar la vostra aplicació a Google Play, heu de generar un text explicatiu del que fa i algunes captures de pantalla.

Configurar l'aplicació per a alliberament

Aquestes són recomanacions de configuracions de l'aplicació:

- Escolliu un bon nom per al *package* de l'aplicació: no es podrà modificar després d'haver-la desplegat. Es pot configurar des del fitxer de *manifest*.
- Netegeu els directoris del projecte de fitxers innecessaris.
- Actualitzeu les configuracions del fitxer de *manifest*. Doneu valor als atributs `versionCode` i `versionName` del build.gradle tal com està explicat a la *Guia del desenvolupador* d'Android (http://developer.android.com/tools/building/configuring-gradle.html).
- Treballeu en la compatibilitat de l'aplicació. Afegiu suport per a diferents pantalles i versions d'Android.
- Actualitzeu les URL dels servidors i els serveis.

Construir l'aplicació signada

Si esteu fent servir Android Studio, podeu fer servir l'assistent d'exportació per crear una clau i exportar un fitxer .apk signat. Una clau privada adequada per signar l'aplicació ha de complir les següents característiques:

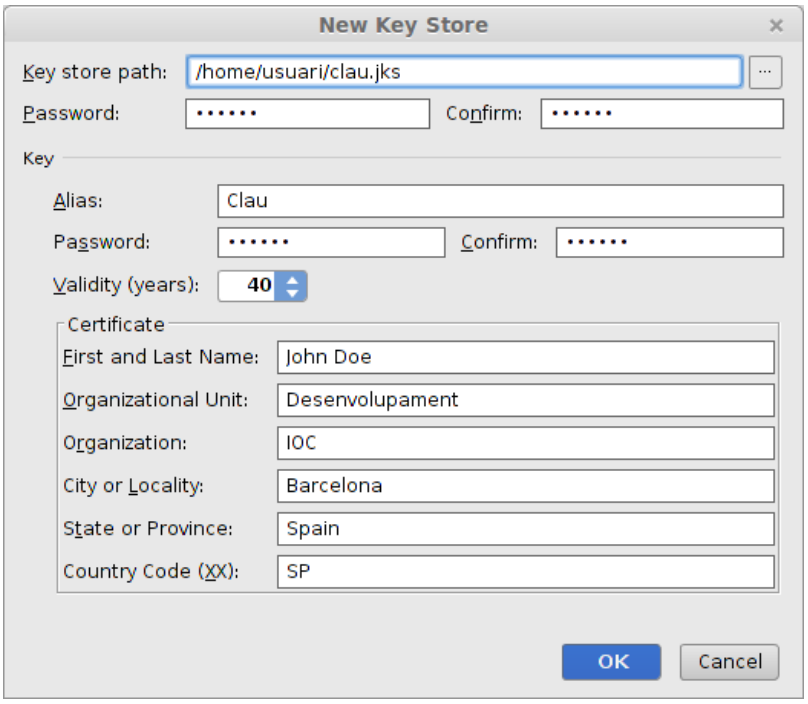
- És de la vostra possessió
- Representa la persona, corporació o organització que s'ha d'identificar amb l'aplicació
- Té un període de validesa que supera el temps de vida esperat de l'aplicació. Es recomana un període de validesa d'almenys 25 anys
- La clau no és la clau per defecte creada per les eines de l'SDK

Si no disposeu d'una clau, en podeu generar una amb l'ordre *keytool*. Tant si ho feu amb *keytool* o amb l'AndroidStudio heu de proporcionar alguns arguments:

- *Alias*: un nom o àlies per a la clau.
- *Password*: contrasenya de la clau.
- *Validity*: temps de validesa. En Android Studio està expressat en anys, i en *keytool* en dies.
- *First and Last Name*: nom i cognoms de la persona que signa.
- *Organization Unit*: organització.
- *City*: ciutat.
- *State or Province*: estat o província.
- *Country Code*: codi de país.

Per crear una clau i signar-la amb Android Studio seleccioneu *Build/Generate Signed APK...*, i ompliu les diferents opcions que us demana, tal com es pot veure a la figura 1.7. Podeu escollir una clau que ja tingueu desada al disc o crear-ne una de nova.

FIGURA 1.7. Signant una aplicació



En el cas que feu servir llibreries de tercers (per exemple, la llibreria externa de Google Maps), és possible que necessiteu altres claus. En el cas de Google Maps, heu de registrar la vostra aplicació al servei de Google Maps i obtindreu una clau de l'API de Maps.

Comprovació de funcionament

Feu una comprovació del funcionament de l'aplicació, idealment en un dispositiu real (preferiblement un telèfon i una tauleta). A la *Guia del desenvolupador* d'Android hi trobareu consells de les coses que es poden comprovar de les aplicacions abans de distribuir-les: http://developer.android.com/guide/topics/testing/what_to_test.html.

1.5.2 Distribuir aplicacions

Podeu distribuir la vostra aplicació de diferents maneres. La forma habitual de distribuir una aplicació és a través d'un mercat d'aplicacions (abans anomenat Android Market, ara Google Play). Però també podeu distribuir les aplicacions directament.

Distribuint l'aplicació mitjançant Google Play

Google Play ha substituït el Google Market com la forma més habitual de distribuir aplicacions d'Android. És una plataforma que us permet publicitar, vendre i

distribuir la vostra aplicació d'Android a usuaris de tot el món. Quan publiqueu a través de Google Play teniu accés a una sèrie d'eines de desenvolupador que us permeten analitzar les vendes i les tendències del mercat, així com controlar a qui es distribuirà la vostra aplicació. Per distribuir una aplicació a Google Play cal que seguiu aquests tres passos:

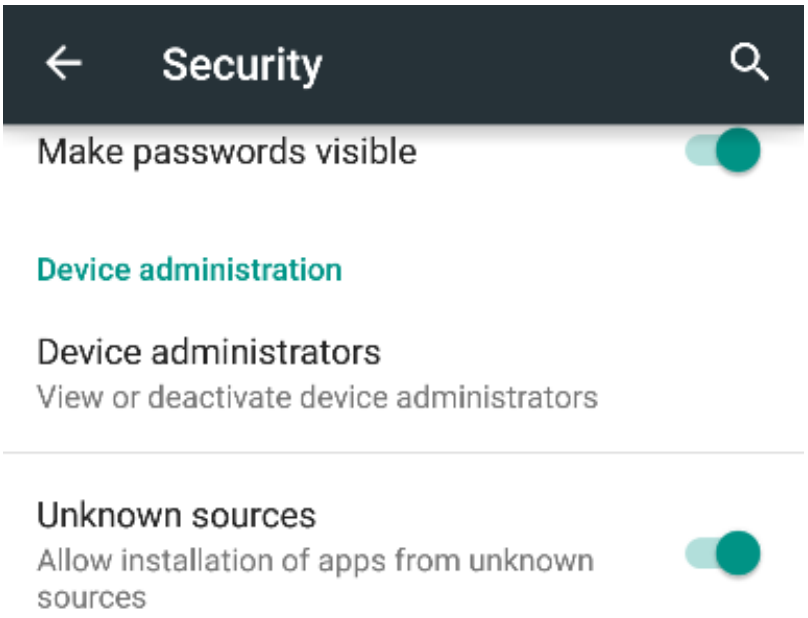
Per a una informació més detallada de com publicar a Google Play, podeu consultar la darrera informació publicada per Google a http://developer.android.com/distribute/tools/lau checklist.html.

1. Prepareu materials promocionals per vendre l'aplicació com captures de pantalla, vídeos, gràfics i un text explicatiu.
2. Configureu les opcions i pugeu els continguts. Configurant alguns ajustaments de Google Play podeu escollir en quins països voleu que estigui disponible la vostra aplicació, el llistat de llenguatges que voleu fer servir i el preu que voleu cobrar en cada país. També podeu configurar detalls de l'aplicació com el tipus d'aplicació, la categoria i el tipus de contingut. Quan l'hàgiu acabat, podeu pujar els continguts promocionals de la vostra aplicació com a esborrany.
3. Publiqueu la versió definitiva de l'aplicació. Si l'esborrany pujat és correcte, podeu fer clic a *Publicar* i en uns minuts la vostra aplicació estarà disponible per descarregar arreu del món.

Publicar l'aplicació des de la vostra pròpia web

Podeu distribuir la vostra aplicació fent-la descarregable des d'una pàgina web emmagatzemada en un servidor de la vostra elecció. Tot el que heu de fer és crear el fitxer .apk definitiu signat i crear un enllaç directe al fitxer des de la vostra pàgina web. Quan els usuaris visitin la pàgina des d'un dispositiu Android i descarreguin l'aplicació, el sistema Android detectarà el tipus de fitxer i instal·larà l'aplicació al dispositiu. Això únicament es podrà fer si l'usuari ha configurat el seu dispositiu per permetre la instal·lació d'aplicacions desconegudes, com es pot veure a la figura 1.8.

FIGURA 1.8. Opció d'habilitar aplicacions d'origen desconegut a Android Lollipop



Publicar a través de la vostra pròpia web és una forma senzilla de publicar l'aplicació, però aquesta no tindrà la visibilitat que tenen les aplicacions a Google Play, i si voleu cobrar per ella n'haureu de gestionar vosaltres mateixos el pagament.

Publicant per correu electrònic

La forma més fàcil i ràpida de publicar una aplicació en un dispositiu és enviar el fitxer .apk signat per correu electrònic i obrir-lo des d'un dispositiu Android. El dispositiu reconeixerà el tipus de fitxer i instal·larà l'aplicació. Aquesta és la forma més còmoda de provar les vostres aplicacions en un telèfon o d'enviar la vostra aplicació a un grup reduït de destinataris (per exemple, la resta de membres del vostre grup de treball).

2. Programació de comunicacions

Cada vegada és més habitual que les aplicacions facin ús de la comunicació per Internet, integrant-se en les xarxes socials o utilitzant serveis web. En aquest apartat veureu com fer ús de les comunicacions del dispositiu per tal de comunicar-vos amb l'exterior.

2.1 Comunicacions en Android

El primer pas per treballar amb l'API de comunicacions en Android és demanar els permisos per a la vostra aplicació al document de *manifest*. Això ho podeu fer afegint les següents línies en l'esmentat document:

```
<uses–permission android:name="android.permission.INTERNET"/>
<uses–permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

Això permet a la vostra aplicació tenir accés a Internet (obrir *sockets* de xarxa) i a informació de l'estat de la xarxa (per exemple, per saber si el Wi-Fi està activat o desactivat).

Abans de fer un intent de connexió a la xarxa, hauríeu de comprovar si el dispositiu té una connexió de xarxa disponible i funcionant correctament. Per això heu de fer servir objectes de la classe `ConnectivityManager` i `NetworkingInfo` de la següent manera:

```
//Obtenim un gestor de les connexions de xarxa
  ConnectivityManager connMgr = (ConnectivityManager) getSystemService(Context.
      CONNECTIVITY_SERVICE);

  //Obtenim l'estat de la xarxa
  NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();

  //Si està connectat
  if (networkInfo != null && networkInfo.isConnected()) {
    //Xarxa OK
    Toast.makeText(this,"Xarxa ok", Toast.LENGTH_LONG).show();
  } else {
    //Xarxa no disponible
    Toast.makeText(this,"Xarxa no disponible", Toast.LENGTH_LONG).show();
  }
```

Amb `getSystemService()` obteniu un objecte `ConnectivityManager` que serveix per gestionar la connexió de xarxa. Amb aquest objecte podeu obtenir un objecte `NetworkInfo`, amb `getActiveNetworkInfo()` que retorna una instància de `NetworkInfo`, que representa la primera interfície de xarxa que pot trobar o un *null* si cap connexió està connectada.

A `NetworkInfo` teniu el mètode `isConnected()`, que indica si existeix connectivitat a la xarxa i si és possible establir connexions.

En el cas que vulgueu obtenir informació individual dels diferents tipus de xarxes, ho podeu fer de forma individual amb `getNetworkingInfo()`, amb un argument que és una constant que indica el tipus de xarxa que voleu comprovar, com es pot veure en el següent codi:

```
1 //Obtenim l'estat de la xarxa mòbil
2 NetworkInfo networkInfo = connMgr.getNetworkInfo(ConnectivityManager.
      TYPE_MOBILE);
3 boolean connectat3G = networkInfo.isConnected();
4
5 //Obtenim l'estat delaxarxaWi-Fi
6 networkInfo = connMgr.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
7 boolean connectatWifi = networkInfo.isConnected();
```

És molt important que sempre comproveu la connexió a la xarxa i no doneu per fet que la xarxa està disponible. Penseu que les operacions de xarxa a un dispositiu mòbil solen ser mòbils, 3G o Wi-Fi, i existeixen possibilitats reals de caiguda de la connexió. Per tant, heu de fer aquestes comprovacions quan engegueu l'aplicació o quan hi torneu després d'haver-ne sortit. El lloc ideal per fer-ho seria al mètode `onStart()` de la vostra aplicació.

Imagineu que engegueu l'aplicació i es fa la comprovació de connexió al començament, i trobeu que existeix connexió. Si l'usuari executa l'aplicació de configuració del dispositiu i activa el mode d'avió, la vostra aplicació seguirà funcionant correctament, ja que quan l'usuari torni a la vostra aplicació `onStart()` tornarà a ser cridat i detectareu que la xarxa està desconnectada.

Però què passa quan la connexió a Internet canvia d'estat mentre esteu dins de la vostra aplicació? Això pot passar perquè el dispositiu perdi la cobertura o la connexió a la xarxa. Com que no es torna a cridar `onStart()`, no podríeu comprovar la connectivitat. Per això, el que heu de fer és registrar un *broadcastReceiver* (receptor *broadcast* o multidifusió) que escoltarà els canvis en l'estat de la connexió i comprovarà quin és l'estat de la xarxa.

En primer lloc, creeu una classe que hereti de `BroadcastReceiver` i implementeu el mètode abstracte `onReceive()`.

```
1 public class ReceptorXarxa extends BroadcastReceiver {
2
3   @Override
4   public void onReceive(Context arg0, Intent arg1) {
5     //Actualitzar l'estat de la xarxa
6     ActualitzaEstatXarxa();
7   }
8 }
```

Si aquesta classe es farà servir dins de la vostra activitat no cal que creeu la classe com a un fitxer .java independent al vostre projecte, pot estar definida dins de la classe de la vostra aplicació. Dins del mètode `onReceive()` actualitzareu l'estat de la xarxa a la vostra aplicació.

El següent pas és registrar el receptor de *broadcast* perquè escolti els missatges relacionats amb els canvis de la connectivitat de la xarxa. Per fer-ho, creeu una instància de la classe receptora i registreu-la amb un filtre d'*intent* per escoltar únicament els missatges de *broadcast* sobre la connectivitat del dispositiu.

```
1  private ReceptorXarxa receptor;
2
3  @Override
4  publicvoid onCreate(Bundle savedInstanceState) {
5    (...) //Resta de codi onCreate()
6
7  IntentFilter filter = new IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION)
       ;
8
9  receptor = new ReceptorXarxa();
10 this.registerReceiver(receptor, filter);
```

El mètode `registerReceiver()` registra un receptor de *broadcast* per executar-se al fil principal de l'aplicació. El receptor de *broadcast* escoltarà qualsevol *intent* de *broadcast* que coincideixi amb el filtre definit en el fil principal de l'aplicació.

En aquest moment, els missatges de *broadcast* que anunciïn els canvis en la connectivitat de la xarxa seran rebuts pel vostre receptor, el qual actualitzarà l'estat de la xarxa de l'aplicació.

Tenir un receptor de *broadcast* que és cridat constantment o innecessàriament pot derivar en un consum excessiu dels recursos del sistema (entre d'altres, de la bateria). Si declareu el receptor de *broadcast* en el document de *manifest*, aquest pot engegar l'aplicació automàticament, encara que aquesta no estigui en execució.

Per aquest motiu, el més adequat seria registrar el receptor de *broadcast* quan es crea l'aplicació `onCreate()` i donar-lo de baixa al mètode `onDestroy()`.

```
1  public void onDestroy() {
2    super.onDestroy();
3
4    //Donem de baixa el receptor de broadcast quan es destrueix l'aplicació
5    if (receptor != null) {
6      this.unregisterReceiver(receptor);
7    }
8  }
```

2.2 Mostrar pàgines web amb un 'widget'

Possiblement, el servei de comunicació més utilitzat avui en dia són les pàgines web, i per aquest motiu començarem mostrant com podem visualitzar-les dins les nostres aplicacions.

El procediment és molt senzill perquè hi ha un component anomenat **WebView** que ja fa gairebé tota la feina, només cal dir-li quina adreça volem visitar.

Creeu un nou projecte Android. Com que la vostra aplicació accedirà, lògicament, a Internet, heu d'afegir-hi el permís corresponent al fitxer **AndroidManifest.xml** (tot seguit del `uses-sdk`):

```
<uses–permissionandroid:name="android.permission.INTERNET"/>
```

Guardeu i aneu al **layout.xml** per afegir el component `WebView`, a més d'un `EditText` per introduir l'adreça web i un botó per anar-hi:

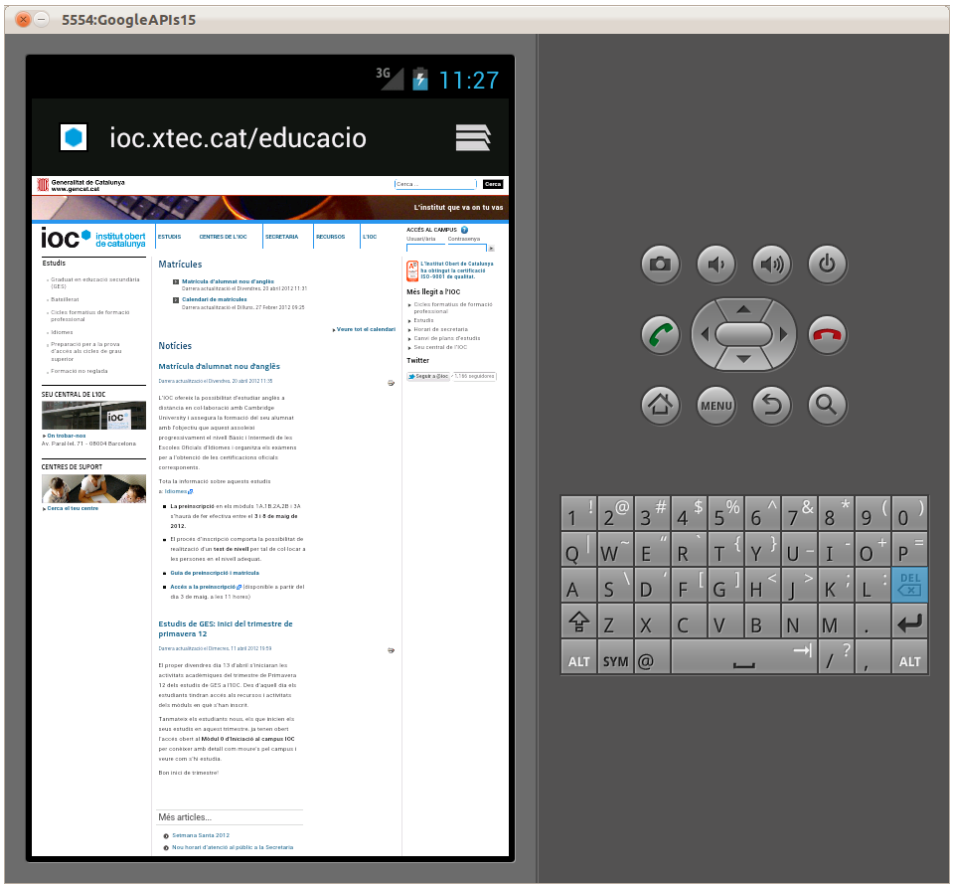
```
<?xml version="1.0" encoding="utf–8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical">

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

    <EditText
      android:id="@+id/editText1"
      android:layout_width="264dp"
      android:layout_height="wrap_content"
      android:hint="Introdueix l&apos;adreça web">

      <requestFocus/>
    </EditText>

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

  </LinearLayout>

  <WebView
    android:id="@+id/webView1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>

</LinearLayout>
```

Només cal anar al codi de l'activitat per afegir la funcionalitat necessària. Bàsicament, es tracta d'escriure el mètode `onClickAnar()` que s'activarà quan cliquem el botó, i que ha de dir al `WebView` que obri una pàgina:

```
public void onClickAnar(View v) {
  WebView webView = (WebView) findViewById(R.id.webView1);
  EditText editText = (EditText)findViewById(R.id.editText1);

  String adreca = editText.getText().toString();
  webView.loadUrl(adreca);
}
```

Amb això podem executar l'aplicació, indicar un web i anar-hi, tal com es pot veure a la figura 2.1.

Figura 2.1. Obrint la pàgina web de l'IOC



Observareu que, si escriviu una adreça sense el prefix "http://", l'aplicació dóna un error. Això és perquè realment l'adreça completa ha d'incloure aquest prefix (que és el protocol d'aplicació). Per tant, podeu afegir el següent codi perquè ho faci, editant el mètode `onClickAnar()`:

```
String adreca = editText.getText().toString();

if(!adreca.startsWith("http://") && !adreca.startsWith("https://")) {
  adreca = "http://" + adreca;
  editText.setText(adreca);
}

webView.loadUrl(adreca);
```

Podeu comprovar com fent doble clic a la pàgina la vista s'apropa (fa zoom) i us permet arrossegar-la per tal d'anar-vos desplaçant per tots els seus continguts segons us calgui, així que podeu clicar enllaços i fer qualsevol altra activitat amb les pàgines que aneu obrint.

2.3 Model dels fils

Quan s'engega una aplicació, el sistema Android posa en marxa un nou procés per a l'aplicació amb un únic fil d'execució.

Processos del sistema

Un procés és una instància d'un programa que s'està executant. Conté el codi i els recursos que està fent servir en aquests moments (per exemple, la memòria reservada). Depenent del sistema operatiu, pot contenir diversos fils d'execució que s'executen concurrentment.

Per defecte, tots els components de l'aplicació s'executen en aquest fil d'execució. Però en alguns casos pot ser convenient executar diferents tasques de l'aplicació en fils d'execució independents.

Fils d'execució

Un fil d'execució és la unitat de processament més petita que es pot programar en un sistema operatiu. Poden existir múltiples fils dins d'un procés i compartir recursos (com memòria, codi i context), cosa que no succeeix entre diferents processos.

Quan una aplicació accedeix a Internet o a altres recursos externs, es produeix un retard de durada variable però que pot arribar a ser de molts segons, o fins i tot d'alguns minuts, des que l'aplicació demana les dades fins que aquestes hi arriben.

Si l'aplicació funciona amb un únic fil d'execució, aquest es pot quedar aturat en la crida a un mètode (per exemple, alguna que descarregui dades d'un servidor extern, el qual pot tardar a enviar la informació). En aquest cas, l'execució de l'aplicació es quedarà bloquejada fins que aquest mètode retorni les dades i així es continuï executant la resta del codi.

Si això ocorregués, els usuaris tindrien la impressió que l'aplicació s'ha quedat penjada i segurament la tancarien i no tornarien a obrir-la. És necessari que l'aplicació continuï responent a les accions de l'usuari i, si el temps d'espera és molt llarg, convé que mostri algun tipus d'animació o barra de progrés per tal que l'usuari vegi que l'aplicació està funcionant.

El mecanisme que permet que les aplicacions puguin fer dues o més operacions alhora són els fils (*threads*, en anglès).

Crear i sincronitzar dos fils és una tasca relativament complexa, i per aquest motiu la llibreria d'Android ens facilita la classe `AsyncTask`, que es fa càrrec de la major part de la feina per nosaltres.

Quan es fa servir un *widget* com `WebView` no cal un `AsyncTask`, perquè el `WebView` ja genera un fil propi internament.

El que fa l'`AsyncTask` és crear un altre fil que podeu fer servir per a qualsevol operació que preveieu que pugui tenir una durada notable i que, per tant, pugui bloquejar la interfície d'usuari. Concretament, qualsevol operació d'accés a Internet hauria de fer-se en un fil separat obligatòriament, i la manera més senzilla és fer servir una `AsyncTask`.

`AsyncTask` s'hauria de fer servir, idealment, per a operacions curtes (d'alguns segons, com a molt). Si necessiteu que els fils estiguin executant-se durant períodes de temps llargs, es recomana que feu servir l'API del paquet `java.util.concurrent`.

Una tasca asíncrona es defineix com una tasca que s'executa en el fil de *background* (fil BG, per *background*, és a dir, que s'executa pel darrere) i els resultats de la qual es mostren en el fil de la interfície d'usuari (l'anomenarem fil UI, de *User Interface*).

Per crear una `AsyncTask` heu de crear una nova classe que derivi d'`AsyncTask`, i implementar els mètodes descrits a continuació. El més important és saber que alguns mètodes s'executen al fil UI i d'altres al fil que fa l'operació en el *background*. Això és fonamental perquè només es poden modificar les vistes de l'aplicació (escriure valors, mostrar missatges, fer avançar barres de progrés o mostrar altre tipus de continguts) des del fil UI.

Un `AsyncTask` es defineix fent servir tres tipus genèrics:

- Params: el tipus dels paràmetres enviats a la tasca durant l'execució.
- Progress: el tipus d'unitats de progrés publicades durant la computació en el *background*. Això serveix per si voleu mostrar un progrés de la tasca feta mentre aquesta s'executa en el *background*. Per exemple, podríeu animar una barra per mostrar el progrés de la càrrega d'un fitxer.
- Result: el tipus del resultat de la computació en el *background*.

No sempre es fan servir tots els tipus en una tasca asíncrona. Per marcar un tipus que no es farà servir, simplement utilitzeu el tipus *void*. Per exemple, la següent classe asíncrona defineix que tindrà un argument de tipus `String` i retornarà un `Bitmap`. No farà servir cap tipus de progrés de la tasca:

```
1 private class CarregaImatge extends AsyncTask<String, Void, Bitmap> {
2   ...
3 }
```

Quan s'executa una tasca de forma asíncrona, aquesta realitza quatre passos:

- `onPreExecute()` (UI): executat a l'UI immediatament després que la tasca és executada. Permet inicialitzar els elements de la interfície que siguin necessaris per a la tasca, com ara crear una barra de progrés.
- `doInBackground(Params...)` (BG): és cridada després que `onPreExecute()` acaba d'executar-se. Posa en marxa l'operació que voleu executar en un altre fil en el *background*. Aquest mètode rep els paràmetres necessaris per fer l'operació asíncrona. El resultat de l'execució s'ha de retornar i es passarà de tornada en l'últim pas. Dins de `doInBackground()` es pot cridar `publishProgress(Progress...)` per publicar una o més unitats de resultat. Aquests valors es publiquen en el fil UI, a `onProgressUpdate(Progres...)`.
- `onProgressUpdate(Progress...)` (UI): és cridat en el fil UI després d'haver cridat `publishProgress(Progress...)`. Us permet actualitzar algun element de la interfície per mostrar el progrés de l'operació mentre l'operació en el *background* encara està en execució, com ara actualitzar una animació o fer avançar una barra de progrés.
- `onPostExecute(Result)` (UI): aquest mètode és cridat quan l'execució en el *background* finalitza. Rep el resultat de l'operació que ha executat l'`AsyncTask` i com que s'executa al fil UI permet visualitzar els resultats obtinguts (com ara una imatge, una pàgina web o una llista de *tweets*).

Per fer servir `AsyncTask` us heu de crear la vostra classe, que hereti d'`AsyncTask` i implementi el mètode de *callback* `doInBackground()`, que s'executa en un fil en el *background*. Per actualitzar la vostra interfície d'usuari (o UI, *user interface*) heu d'implementar el mètode `onPostExecute()` que incorpora els resultats d'haver executat `doInBackground()` i s'executa en el fil UI, i per tant pot actualitzar l'estat de la vostra interfície sense problemes. Quan tingueu la classe creada, per llançar el fil de l'`AsyncTask` heu d'elaborar un nou objecte de

la subclasse que heu creat i cridar el mètode `execute()`, passant-li la informació necessària perquè pugui fer l'operació.

Per exemple, la següent classe descarrega una imatge de la xarxa, mitjançant el mètode propi `Bitmap DescarregaImatgeDeLaXarxa(String url)`, i fa servir el `Bitmap` per mostrar-lo a un *widget*:

```
1  private class CarregaImatge extends AsyncTask<String, Void, Bitmap> {
2
3    @Override
4    protected Bitmap doInBackground(String... url) {
5      //Descarreguem la imatge d'Internet
6      returnDescarregaImatgeDeLaXarxa(url);
7    }
8
9    protected void onPostExecute(Bitmap result) {
10   //Rebem el resultat de doInBackground (el Bitmap)
11   //i el fem servir per carregar la imatge a l'UI
12     mImageView.setImageBitmap(result);
13   }
14 }
```

Existeixen algunes regles per tal que `AsyncTask` funcioni correctament:

- La classe `AsyncTask` s'ha de crear al fil UI.
- La instància de la tasca s'ha de crear al fil UI.
- `execute(Params...)` s'ha d'executar des del fil UI.
- Mai heu de cridar `onPreExecute()`, `onPostExecute(Result)`, `doInBackground(Params...)` o `onProgressUpdate(Progress...)` directament.
- La tasca es pot executar únicament una vegada al mateix temps (si n'intenteu executar una altra, en llançarà una excepció).

2.4 Connexions HTTP

El protocol HTTP (*HyperText Transfer Protocol*, protocol de transferència d'hipertext) és un protocol estàndard d'Internet que serveix per a la transferència d'hipertext. És el protocol base sobre el qual està fonamentada la WWW (*World Wide Web*, 'xarxa d'extensió mundial'). Fent servir HTTP a la vostra aplicació podreu realitzar diverses tasques, com descarregar-vos informació de text i binària (com ara pàgines web o imatges, respectivament).

Un *hipertext* és un text mostrat a un ordinador o un altre dispositiu electrònic amb enllaços (*hyperlinks*) a altres textos, que són d'accés immediat pel lector a través d'un clic del ratolí o de la pantalla tàctil. A banda de text, pot contenir taules, imatges i altres formats de representació de continguts multimèdia.

També es pot fer servir el protocol HTTPS (*HTTP Secure*, HTTP segur) per a comunicacions segures a través d'una xarxa d'ordinadors. HTTPS proporciona

autentificació del lloc web i encriptació bidireccional en les comunicacions entre el client i el servidor.

Android inclou dos clients HTTP que podeu fer servir a la vostra aplicació: `HttpURLConnection` i el client d'Apache `HttpClient`. Ambdós suporten HTTPS, fluxos de pujada i descàrrega i IPv6. Segons articles de la pròpia Google, el client HTTP d'Apache és més estable a les versions d'Android Eclair (versió 2.1) i Froyo (versió 2.2). Però per a les versions Gingerbread (versió 2.3) i posteriors recomana fer servir `HttpURLConnection`, en el que enfocarà els seus esforços futurs.

Una connexió `HttpURLConnection` permet enviar i rebre dades per la web. Les dades poden ser de qualsevol tipus i mida, fins i tot podeu enviar i rebre dades de les quals no coneixeu la mida prèviament. Per usar aquesta classe heu de seguir els següents passos:

- Obtingueu un objecte nou `HttpURLConnection` cridant el mètode `URL.openConnection()` i forçant el seu resultat a `HttpURLConnection`.
- Prepareu la petició. La principal propietat de la petició és la seva URI. Les capçaleres de la petició poden tenir altres metadades com credencials, tipus de contingut, galetes de sessió, etc.
- Si aneu a pujar dades a un servidor, la instància ha d'estar configurada amb `setDoOutput(true)`. Heu de transmetre dades escrivint al flux de dades retornat per `getOutputStream()`.
- Llegiu la resposta. Les capçaleres de la resposta generalment inclouen metadades com el tipus de dades de la resposta i la seva mida, la data de modificació o galetes de sessió, entre d'altres. El cos de la resposta es pot llegir del flux retornat per `getInputStream()`. Si la resposta no té cos (*body* en anglès), el mètode retorna un flux buit.
- Desconnecteu. Quan hàgiu llegit la resposta, heu de tancar l'`HttpURLConnection` amb `disconnect()`.

Per exemple, amb el següent codi estaríeu llegint el contingut del web de l'IOC:

```
URL url =new URL("http://ioc.xtec.cat/");
  HttpURLConnection urlConnection =(HttpURLConnection) url.openConnection();
  try {
    InputStream in = new Buffered InputStream(urlConnection.getInputStream());
    treballarAmbElFluxDeDades(in);
  finally {
    urlConnection.disconnect();
  }
}
```

2.4.1 Comunicacions segures amb HTTPS

Si crideu `openConnection()` amb una URL que comenci amb la cadena "https", se us retornarà un objecte `HttpsURLConnection` (fixeu-vos que no és el mateix que l'anterior, té una "s" al final del nom). Aquesta classe us permet especificar uns `HostnameVerifier` i `SSLSocketFactory` necessaris per al protocol HTTPS. Podeu consultar el funcionament d'aquesta classe a la *Guia del desenvolupador* d'Android http://developer.android.com/reference/javax/net/ssl/HttpsURLConnection.html.

2.4.2 Pujant dades a un servidor

Per pujar dades a un servidor web heu de configurar la connexió per sortida amb `setDoOutput(true)`. Per millorar el rendiment de la vostra aplicació cal que crideu `setFixedLengthStreamingMode(contentLength)` quan conegueu per avançat la mida del cos de les dades que voleu enviar, o `setChunkedStreamingMode(chunkLength)` quan no la sabeu (té com a argument la mida de *chunk*, porció de dades, o 0 per fer servir la mida estàndard). Si no ho feu, `HttpURLConnection` copiarà les dades completes a enviar en un *buffer* (memòria cau) abans d'enviar-les, desaprofitant així memòria i fent que l'aplicació vagi més lenta. Per exemple, el codi següent estableix una connexió a un servidor i realitza una descàrrega i una pujada de dades. Les funcions `EscriureFluxDeDades()` i `LlegirFluxDeDades()` seran funcions pròpies que treballaran amb l'`OutputStream` i l'`InputStream`.

```
1  HttpURLConnection urlConnection =(HttpURLConnection) url.openConnection();
2    try{
3      urlConnection.setDoOutput(true);
4      urlConnection.setChunkedStreamingMode(0);
5
6      OutputStream out=new BufferedOutputStream(urlConnection.getOutputStream());
7      EscriureFluxDeDades(out);
8
9      InputStream in=new BufferedInputStream(urlConnection.getInputStream());
10     LlegirFluxDeDades(in);
11   finally {
12     urlConnection.disconnect();
13   }
14 }
```

2.5 Missatgeria

Els serveis de missatgeria SMS (*Short Message Service*, servei de missatges curts) es troben entre els serveis de telefonia més populars. El seu cost reduït (en comparació amb les trucades) ha fet que hagi esdevingut la forma més comuna

per realitzar comunicacions senzilles durant anys. Darrerament, amb l'aparició dels *smartphones* i la seva connexió contínua a Internet, han aparegut serveis de missatgeria gratuïts a través de la xarxa de dades 3G (com Whatsapp), que han esdevingut una alternativa per a la comunicació de missatges entre dispositius que tenen connexió a Internet. En qualsevol cas, tots els telèfons permeten l'ús d'SMS, també els *smartphones*, tot i que aquest servei ha evolucionat cap a altres funcionalitats, com ara la validació del *login* en dues passes, la confirmació d'operacions bancàries, informacions del govern o d'empreses de subministraments...

Android ja porta inclosa per defecte una aplicació per enviar i rebre missatges SMS. Però potser voleu integrar les capacitats SMS dins de la vostra aplicació. Tot seguit teniu alguns exemples d'aplicacions Android que podrien fer servir el servei SMS:

- Una aplicació que enviï un missatge de felicitació automàticament als nostres contactes el dia del seu aniversari.
- Una aplicació que enviï un missatge quan algun valor de la borsa pugi o baixi d'un cert límit.
- Una aplicació que enviï regularment la posició en la qual es troba el telèfon (per a persones amb alzheimer o nens petits).

Com veieu, les possibilitats són infinites i depèn únicament de la imaginació del desenvolupador.

2.5.1 Enviament d'SMS

Enviar missatges SMS des del vostre programa és molt senzill. En primer lloc, heu de declarar que la vostra aplicació tindrà permisos per enviar SMS. Per fer-ho, afegiu la següent declaració al fitxer de *manifest* del projecte:

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

El *layout* de l'activitat principal és molt senzill, amb unes caixes de text per introduir el número de telèfon i el text a enviar i un botó per enviar el missatge:

```
<?xml version="1.0"encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/RelativeLayout1"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical">

  <TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/editTelèfon"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
```

```
    android:text="Telefon"
    android:textAppearance="?android:attr/textAppearanceLarge"/>

  <EditText
    android:id="@+id/editTelèfon"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentRight="true"
    android:layout_alignParentTop="true"
    android:layout_toRightOf="@+id/textView1"
    android:layout_weight="1"
    android:ems="10"
    android:inputType="phone">

    <requestFocus/>
  </EditText>

  <TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_below="@+id/editTelèfon"
    android:text="Missatge"
    android:textAppearance="?android:attr/textAppearanceLarge"/>

  <EditText
    android:id="@+id/editMissatge"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_below="@+id/textView2"/>

  <Button
    android:id="@+id/btnEnviarSMS"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_below="@+id/editMissatge"
    android:text="Enviar SMS"/>

</RelativeLayout>
```

Fixeu-vos que aquest *layout* està definit com un`RelativeLayout`. Podríeu obtenir un resultat similar amb un `LinearLayout`.

Les operacions amb el servei SMS estan gestionades mitjançant la classe `SMSManager`, que us permet enviar missatges de dades, text i missatges per parts. Per obtenir un objecte de la classe `SMSManager` heu de cridar el mètode `SMSManager.getDefault()`, per exemple:

```
SmsManager sms = SmsManager.getDefault();
```

Quan hàgiu aconseguit un objecte `SMSManager`, podeu fer servir el mètode `sendTextMessage`, que té els següents arguments:

- `String destinationAddress`: el número de telèfon on voleu enviar el missatge.
- `String scAddress`: el número del centre de servei d'SMS. Si introduïu *null*, es farà servir el que té definit el telèfon a la seva configuració.
- `String text`: text a enviar.

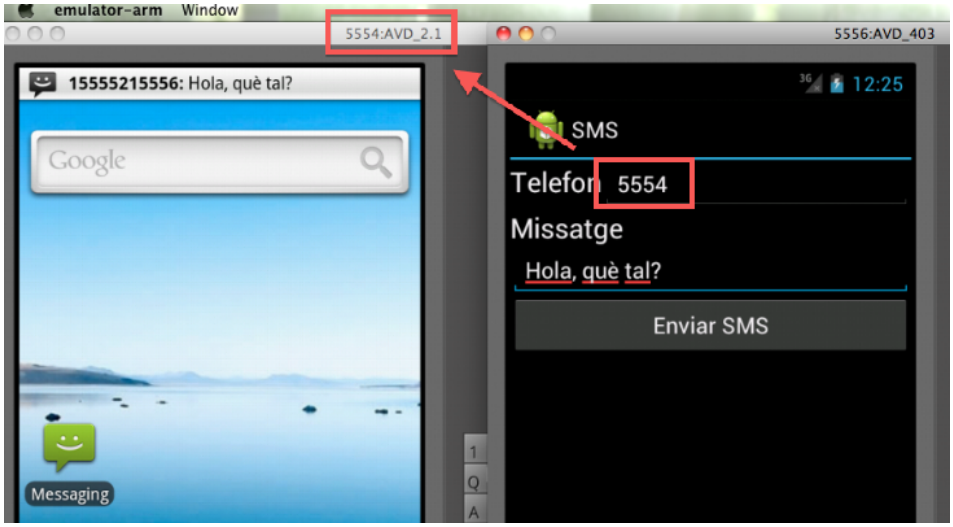
- `PendingIntent sentIntent`: si està a *null*, no fa res. Si li passeu un `PendingIntent`, aquest serà enviat com a missatge de *broadcast* al dispositiu quan el missatge hagi estat enviat, o bé quan falli l'enviament. En el segon cas, aquest Intent us permetrà saber el tipus d'error que s'ha produït.
- `PendingIntent deliveryIntent`: si és *null* no fa res. Si passeu un `Intent`, aquest serà enviat com a missatge de *broadcast* al dispositiu quan el missatge ha estat lliurat al destinatari.

La forma més fàcil d'enviar un missatge és simplement passar el número de telèfon i el missatge, com es veu a continuació:

```
1 sms.sendTextMessage(numeroTelefon, null, missatge, null, null);
```

Podeu fer una prova de funcionament des del simulador. Executeu dos simuladors des de l'AVD Manager i envieu un missatge SMS de l'un a l'altre. Per fer-ho, heu de fer servir el número de port de la instància de simulador Android (està a la capçalera de la finestra) com a número de telèfon, com es pot veure a la figura 2.2.

Figura 2.2. Enviant un missatge entre dues instàncies de l'emulador



2.5.2 Rebre SMS

Per rebre missatges dins de la vostra aplicació heu de fer servir un objecte `BroadcastReceiver`. Quan es rep un SMS al dispositiu, s'envia un missatge de *broadcast* anunciant la recepció. És aquest missatge el que podeu obtenir per fer alguna acció quan es rep un SMS.

Per rebre missatges SMS, en primer lloc heu de modificar el fitxer de *manifest* per indicar que l'aplicació té permisos per rebre SMS amb el codi. També heu de registrar al fitxer de *manifest* el receptor de *broadcast* que fareu servir:

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

```
3   package="com.android.ioc"
4   android:versionCode="1"
5   android:versionName="1.0">
6
7   <uses-sdk android:minSdkVersion="15"/>
8
9   <uses-permission android:name="android.permission.RECEIVE_SMS"/>
10
11  <application
12    android:icon="@drawable/ic_launcher"
13    android:label="@string/app_name">
14    <activity
15      android:name=".RebreSMSActivity"
16      android:label="@string/app_name">
17      <intent-filter>
18        <actionandroid:name="android.intent.action.MAIN"/>
19        <categoryandroid:name="android.intent.category.LAUNCHER"/>
20      </intent-filter>
21    </activity>
22
23    <receiver android:name=".SMSReceiver">
24      <intent-filter>
25        <action android:name="android.provider.Telephony.SMS_RECEIVED"></action
              >
26      </intent-filter>
27    </receiver>
28  </application>
29
30  </manifest>
```

Cal que indiqueu el nom del receptor de *broadcast* i, dins del filtre d'*intents*, quin tipus de missatge de *broadcast* estareu escoltant (en aquest cas el missatge de rebuda d'SMS, `SMS_RECEIVED`).

Dins de la carpeta */src* del projecte, afegiu una nova classe dins del paquet de la vostra aplicació i anomeneu-la `SMSReceiver`. Aquesta classe heretarà de `BroadcastReceiver`. La classe `BroadcastReceiver` és la classe base que rebrà els *Intents* enviats mitjançant el mètode `sendBroadcast()`. La implementació del receptor ja ha estat anunciada a través del *manifest* de l'aplicació amb l'etiqueta `<receiver>`. Una altra forma de registrar el receptor de *broadcast* és mitjançant `Context.registerReceiver()`.

La classe que creeu, que hereta de `BroadcastReceiver`, ha d'implementar el mètode `onReceive(Context c, Intent i)`. Aquest és el mètode que és cridat quan el `BroadcastReceiver` rep un *Intent* de *broadcast*. Aquest mètode rep dos arguments: el context on el receptor s'està executant (on s'ha cridat) i l'*Intent* que s'està rebent.

L'objecte `BroadcastReceiver` únicament és vàlid durant la crida al mètode `onReceive()`. Una vegada que el codi retorna d'aquesta funció, el sistema considera l'objecte terminat i deixa d'estar actiu. Això té una repercussió important en el que es pot fer dins de la implementació del mètode `onReceive()`. Per exemple, no es poden fer coses que requereixen una operació asíncrona, ja que per fer-ho heu de sortir de la funció per manegar l'operació asíncrona, però en el moment en què se surt d'`onReceive()`, el `BroadcastReceiver` ja no està actiu i, per tant, el sistema és lliure de finalitzar el procés abans que l'operació asíncrona s'hagi completat.

Aquesta és la definició de la classe que fareu servir per escoltar els *Intents* de *broadcast*:

```
package com.android.ioc;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

public class SMSReceiver extends BroadcastReceiver {
  public void onReceive(Context context, Intent intent) {

  }
}
```

Quan es rep un missatge SMS, el mètode `onReceive()` és cridat. El missatge SMS està contingut dins de l'objecte `Intent` (el segon argument que rep onReceive()) a través dels extres de l'`Intent`, que són un objecte `Bundle`. Reviseu la definició de la classe `Intent` a la *Guia del desenvolupador* d'Android per veure la seva definició i els extres, a http://developer.android.com/reference/android/content/Intent.html.

Aquest és el codi que obté el `Bundle` de l'`Intent`.

```
//Obtenim els extres de l'Intent
Bundle bundle = intent.getExtras();

//Si hi ha extres
if(bundle !=null) {

}
```

Els missatges estan emmagatzemats en un *array* d'`Objects` en el format PDU.

Unitat de Descripció de Protocol

PDU (*Protocol Description Unit*, unitat de descripció de protocol) és un dels mètodes definits per la indústria per enviar i rebre missatges SMS. No cal que feu una dissecció de la PDU, ja que la classe `SMSMessage` hi pot treballar directament. Per a més informació sobre l'estructura d'una PDU podeu consultar: www.dreamfabric.com/sms.

Per obtenir l'*array* d'objectes a partir del `Bundle` cal que feu servir:

```
//obtenir llistat d'SMS
Object[] pdus = (Object[]) bundle.get("pdus");
```

Per extraure cada missatge, heu de fer servir el mètode estàtic `SmsMessage.createFromPdu()`, que us retorna un `SmsMessage` a partir d'un PDU (dels que trobeu a l'array d'`Objects`). El següent codi mostra com obtenir el missatge 0 (en forma d'`SmsMessage`) a partir de l'*array* d'objectes obtingut:

```
SmsMessage missatge = null;

missatge = SmsMessage.createFromPdu((byte[])pdus[0]);
```

Quan tingueu l'objecte `SmsMessage` podeu obtenir accés al cos de missatge o el número de telèfon que l'ha enviat amb `getMessageBody()` i `getOriginatingAddress()`, respectivament.

Finalment, el codi següent mostra tot el codi del mètode `onReceive()`. Penseu que, com que es pot haver rebut més d'un missatge, la lectura dels missatges de l'*array* d'objectes es fa dins d'un bucle *for*, per llegir tots els missatges si és el cas.

```
public void onReceive(Context context, Intent intent) {

  //Rebre l'SMS

  //Obtenim els extres de l'Intent
  Bundle bundle = intent.getExtras();

  SmsMessage[] missatges = null;
  String s = "";

  //Si hi ha extres
  if(bundle !=null) {
    //obtenir llistat d'SMS
    Object[] pdus = (Object[]) bundle.get("pdus");

    SmsMessage missatge = null;

    for(int i=0; i<pdus.length; i++) {
      missatge = SmsMessage.createFromPdu((byte[])pdus[i]);
      s += "SMS de " + missatge.getOriginatingAddress();
      s += " :";
      s += missatge.getMessageBody().toString();
      s += "\n";
    }

  Toast.makeText(context, s, Toast.LENGTH_SHORT).show();
  }

}
```

2.5.3 Enviant un SMS mitjançant un 'Intent'

Podeu fer servir la classe `SmsManager` per enviar missatges SMS des de la vostra aplicació. Però si no ho voleu programar, podeu invocar l'aplicació de missatgeria inclosa en el mateix sistema Android perquè enviï el missatge per vosaltres.

Per activar l'aplicació de missatgeria des de la vostra aplicació podeu llançar un objecte *Intent* amb el tipus `MIME ''vnd.android-dir/mms-sms''`. En aquest cas, l'únic que heu de fer és crear un objecte `Intent`, afegir les dades del missatge que voleu enviar com a extres: `''address''` en el cas del número de telèfon (se'n pot introduir més d'un, separats per punts i coma) i `''sms_body''` en el cas del cos del missatge. També heu d'introduir el tipus de l'`Intent` com `''vnd.android-dir/mms-sms''`. Una vegada fet això, simplement invoqueu`startActivity()` amb l'`Intent` creat, com es pot veure al següent codi:

```
Intent i = new Intent(android.content.Intent.ACTION_VIEW);
i.putExtra("address", editTelefon.getText().toString());
i.putExtra("sms_body",editMissatge.getText().toString());
i.setType("vnd.android-dir/mms-sms");
startActivity(i);
```

2.5.4 Seguretat

Quan feu servir els receptors de *broadcast* heu de pensar que aquests són una porta d'entrada a la vostra aplicació per a la resta d'aplicacions del dispositiu. Heu de considerar que altres aplicacions poden abusar de l'ús de missatges i, per tant, heu de tenir en compte alguns conceptes de seguretat per a la vostra aplicació:

- L'espai de noms d'`Intent` és global, per tant assegureu-vos que els noms d'acció dels *intents* estiguin escrits en un espai de noms vostre, o sense adonar-vos-en podríeu entrar en conflicte amb altres aplicacions.

- Quan registreu un receptor de *broadcast* amb `registerReceiver()`, *qualsevol* aplicació pot enviar *broadcasts* a la vostra. Podeu controlar qui pot enviar *broadcast* a través de permisos.

- Quan publiqueu un receptor al *manifest* de l'aplicació i especifiqueu filtres d'`Intent` per aquest, qualsevol altra aplicació pot enviar *broadcast*. Per evitar que altres aplicacions enviïn missatges de *broadcast* a la vostra, podeu fer-la "no disponible" per la resta amb `android:exported=''false''` al fitxer de *manifest*.

- Quan feu servir `sendBroadcast(Intent)` o algun altre mètode similar, normalment qualsevol altra aplicació pot rebre aquests *broadcasts*. Podeu controlar qui rep els *broadcasts* mitjançant els permisos. A partir de la versió *Ice Cream Sandwich* d'Android podeu restringir el *broadcast* a una única aplicació.

Cap d'aquests problemes existeix quan es fa servir `LocalBroadcastManager()`, ja que els *intents broadcast* en aquest cas mai surten del procés de l'aplicació.

Es poden forçar els permisos en l'enviament i recepció *broadcast*. Per forçar els permisos quan s'envien missatges de *broadcast* compteu amb l'argument `String receiverPermission` dels mètodes `sendBroadcast()` i `sendOrderedBroadcast()`. Si no definiu aquest argument com a *null*, únicament els receptors als quals s'ha permès aquest permís (amb l'etiqueta de petició `<uses-permission>` al fitxer de *manifest*) podran rebre el *broadcast*.

Per establir els permisos sobre allò que pot rebre l'aplicació, heu d'introduir a l'argument `String broadcastPermission` un valor no *null* quan registreu el receptor amb el mètode `registerReceiver()`. També ho podeu fer estàticament dins de l'etiqueta `<receiver>` del fitxer de *manifest* de l'aplicació. Únicament els emissors de *broadcast* que tinguin aquest permís concedit (amb la petició amb l'etiqueta `<uses-permission>` al fitxer de *manifest*) podran enviar missatges a l'aplicació.

Podeu trobar més informació sobre seguretat a la *Guia del desenvolupador* d'Android, a http://developer.android.com/guide/topics/security/permissions.html.

2.5.5 Enviant un MMS mitjançant 'Intent'

Els MMS (*Multimedia Messaging System*, sistema de missatgeria multimèdia) són missatges de text semblant als SMS però amb un contingut extra de tipus multimèdia (foto, àudio, vídeo...). El procés d'enviament d'un missatge MMS és semblant al de l'SMS mitjançant un `Intent`, però en aquest cas s'han d'afegir una sèrie d'extres per indicar les dades multimèdia que es volen afegir al missatge. Per començar, l'`Intent` es crea amb el constructor d'`Intent` que té com a arguments una acció i un URI de les dades que es volen enviar:

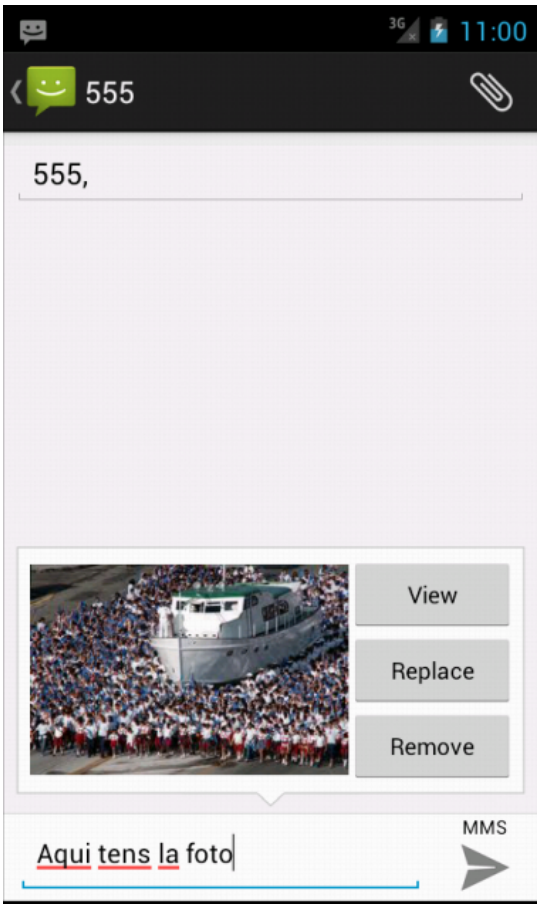
```
1 Intent intentMMS = new Intent(Intent.ACTION_SEND, Uri.parse("mms://"));
```

Després s'afegeixen els extres. A més del número de telèfon i el cos del missatge, afegiu el contingut multimèdia que es vol afegir al missatge. Per fer-ho agregueu un contingut extra del tipus que aneu a afegir. En aquest exemple de codi, s'inclou una imatge que es troba en la targeta SD del dispositiu.

```
1  intentMMS.putExtra("address", editTelefon.getText().toString());
2
3  intentMMS.putExtra("sms_body",
4  editMissatge.getText().toString());
5  String url = "file://mnt//sdcard//foto.jpeg";
6
7
8  intentMMS.setType("image/jpeg");
9
10
11 intentMMS.putExtra(Intent.EXTRA_STREAM, Uri.parse(url));
12
13 startActivity(Intent.createChooser(intentMMS, "MMS:"));
```

Quan s'executa l'aplicació, es fa una crida que es tractada per l'aplicació de missatgeria del sistema. Es fa servir aquesta aplicació per enviar un missatge MMS amb les dades que s'han passat a través de l'`Intent`, com es pot veure a la figura 2.3.

FIGURA 2.3. Enviament de missatgeria MMS



2.6 Analitzant codi XML

Un format molt útil per intercanviar informació entre dos dispositius és mitjançant XML. XML és un llenguatge de marques que defineix una sèrie de regles per codificar documents en un format que és comprensible, tant per les màquines (ordinadors, telèfons...) com per les persones. És un format que es fa servir freqüentment per compartir dades a Internet. Per exemple, els webs que tenen contingut que s'actualitzi freqüentment, com els blogs o pàgines de notícies, solen proporcionar codi XML per tal que programes externs puguin mostrar les actualitzacions de la pàgina. Per poder fer servir aquestes dades, les aplicacions s'han de descarregar les dades XML (per exemple, amb una connexió HTTP) i, posteriorment, analitzar el codi XML per obtenir les dades i tractar-les dins del programa com vulgui.

Existeixen diferents analitzadors d'XML que podeu fer servir a Android. Google recomana fer servir `XmlPullParser`, la documentació del qual podeu trobar a la *Guia del desenvolupador* d'Android:

http://developer.android.com/reference/org/xmlpull/v1/XmlPullParser.html.

Tot seguit veurem un exemple d'utilització de l'analitzador a través d'un exemple. L'objectiu és descarregar les noticies del web StackOverflow.com. El *feed*

d'RSS d'aquest web sobre Android ordenat per les últimes entrades el podeu trobar a http://stackoverflow.com/feeds/tag?tagnames=android&sort=newest i té la següent forma:

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:creativeCommons="http://backend
    .userland.com/creativeCommonsRssModule" xmlns:re="http://purl.org/atompub/
    rank/1.0">
<title type="text">newest questions tagged android - Stack Overflow</title>
...
  <entry>
  ...
  </entry>
  <entry>
    <id>http://stackoverflow.com/q/11530594</id>
    <re:rank scheme="http://stackoverflow.com">0</re:rank>
    <title type="text">No puc compilar</title>
    <category scheme="http://stackoverflow.com/feeds/tag?tagnames=android&amp;
        sort=newest/tags" term="android"/><category scheme="http://
        stackoverflow.com/feeds/tag?tagnames=android&amp;sort=newest/tags"
        term="android-emulator"/>
    <author>
      <name>aturing</name>
      <uri>http://stackoverflow.com/users/803649</uri>
    </author>
    <link rel="alternate" href="http://stackoverflow.com/questions/11530594/
        launch-of-android-emulator-failing" />
    <published>2012-07-17T20:44:06Z</published>
    <updated>2012-07-17T20:47:59Z</updated>
    <summary type="html">
      No trobo el botó per compilar l'aplicació. Algú em pot ajudar?
    </summary>
  </entry>
  <entry>
  ...
  </entry>
...
</feed>
```

Si us hi fixeu, dins de les etiquetes `<entry>` i `</entry>` teniu tota la informació d'una entrada en el web.

Les entrades les emmagatzemem com una llista d'objectes de la classe `Entrada`, que conté tres *strings* per les diferents parts que us interessen de les notícies.

```
//Aquesta classe representa una entrada de notícia de l'RSS Feed
public static class Entrada {
  public final String titol; //Títol de la notícia
  public final String enllac; //Enllaç a la notícia completa
  public final String resum; //Resum de la notícia

  private Entrada(String title, String summary, String link) {
    this.titol = title;
    this.resum = summary;
    this.enllac = link;
  }
}
```

Tot seguit crearem una altra classe, `StackOverflowXmlParser`, per analitzar el contingut de l'XML. El mètode analitzat serveix per començar l'anàlisi de l'XML i rep un `InputStream` (que heu obtingut a través d'una connexió HTTP) com a entrada. El mètode retorna una llista d'objectes `Entrada`:

```
public List<Entrada> analitza(InputStream in) throws XmlPullParserException,
    IOException {
    try {
      //Obtenim analitzador
      XmlPullParser parser = Xml.newPullParser();

      //No fem servir namespaces
      parser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES, false);

      //Especifica l'entrada de l'analitzador
      parser.setInput(in, null);

      //Obtenim la primera etiqueta
      parser.nextTag();

      //Retornem la llista de notícies
      return llegirNoticies(parser);
    } finally {
      in.close();
    }
  }
```

L'objecte `XmlPullParser` l'obteniu amb una crida a `Xml.newPullParser()`. Una vegada especificades les seves característiques i el flux de dades d'entrada de l'analitzador crideu a `nextTag()` per obtenir el primer *tag* (etiqueta) del document XML. El mètode `next()` avança l'analitzador al següent *event* del document XML. Retorna un valor que indica el tipus d'*event* de l'estat de l'analitzador (el podeu obtenir també amb `getEventType()`). El mètode `nextTag()` ignora els espais en blanc i retorna la següent etiqueta de l'XML.

El mètode és l'encarregat de llegir totes les notícies i retornar el `List` d'entrades.

```
private List<Entrada> llegirNoticies(XmlPullParser parser) throws
    XmlPullParserException, IOException {

  List <Entrada>llistaEntrades = new ArrayList<Entrada>();

  //Comprova si l'event actual és del tipus esperat (START_TAG) i del nom "feed
      "
  parser.require(XmlPullParser.START_TAG, ns, "feed");

  //Mentre que no arribem al final d'etiqueta
    while (parser.next() != XmlPullParser.END_TAG) {
      //Ignorem tots els events que no siguin un començament d'etiqueta
      if (parser.getEventType() != XmlPullParser.START_TAG) {
        //Saltem al següent event
        continue;
      }

      //Obtenim el nom de l'etiqueta
      String name = parser.getName();

      // Si aquesta etiqueta és una entrada de notícia
      if (name.equals("entry")) {
        //Afegim l'entrada a la llista
        llistaEntrades.add(llegirEntrada(parser));
    } else {
      //Si és una altra cosa la saltem
      saltar(parser);
    }
  }
  return llistaEntrades;
}
```

El mètode `parser.require()` comprova si l'*event* actual és del tipus adequat i amb el nom que esperem, en aquest cas, "feed"; ho podeu comprovar en l'estructura del codi XML.

A continuació s'aniran llegint etiquetes dins del bucle *while* fins que es trobi el final d'etiqueta. S'ignoraran tots els *events* llegits que no siguin de començament d'etiqueta amb el codi:

```
//Ignora fins que no trobem un començament d'etiqueta
if (parser.getEventType() != XmlPullParser.START_TAG) {
  continue;
}
```

El mètode `parser.getEventType()` retorna el tipus d'esdeveniment que ha llegit l'analitzador. Existeixen els següents tipus d'*events* definits a `XmlPullParser`:

- `START_TAG`: es llegeix un començament d'etiqueta XML.
- `TEXT`: es llegeix text. Es pot obtenir el seu contingut amb el mètode `parser.getText()`.
- `END_TAG`: es llegeix el final d'una etiqueta XML.
- `END_DOCUMENT`: no hi ha més *events* disponibles.

A continuació es cercarà entre les etiquetes fins que es trobi l'etiqueta *entry*. El mètode `parser.getName()` retorna el nom de l'etiqueta, es compara aquest nom per veure si és *entry* (fixeu-vos dins l'estructura del fitxer XML com les entrades estan contingudes dins d'una parella d'etiquetes `<entry> </entry>`).

Tota la resta d'etiquetes seràn ignorades. Una vegada trobada una secció *entry* de l'XML, es cridarà el mètode encarregat de llegir una entrada. El mètode `llegirEntrada()` retornarà un objecte `Entrada` que serà afegit al `List` retornat per `llegirNoticies()`.

```
//Analitza el contingut d'una entrada. Si troba un títol, resum o enllaç, crida
     els mètodes de lectura
//propis per processar-los. Si no, ignora l'etiqueta.

private Entrada llegirEntrada(XmlPullParser parser) throws
    XmlPullParserException, IOException {
  String titol = null;
  String resum = null;
  String enllac = null;

  //L'etiqueta actual ha de ser "entry"
  parser.require(XmlPullParser.START_TAG, ns, "entry");

  //Mentre no acabi l'etiqueta d'"entry"
  while (parser.next() != XmlPullParser.END_TAG) {
    //Ignora fins que no trobem un començament d'etiqueta
    if (parser.getEventType() != XmlPullParser.START_TAG) {
      continue;
    }

    //Obtenim el nom de l'etiqueta
    String etiqueta = parser.getName();

    //Si és un títol de notícia
```

```
23      if (etiqueta.equals("title")) {
24        titol = llegirTitol(parser);
25
26      //Si l'etiqueta és un resum de notícia
27      } elseif (etiqueta.equals("summary")) {
28        resum = llegirResum(parser);
29
30      //Si és un enllaç
31      } elseif (etiqueta.equals("link")) {
32        enllac = llegirEnllac(parser);
33      } else {
34        //les altres etiquetes les saltem
35        saltar(parser);
36      }
37    }
38
39    //Creem una nova entrada amb aquestes dades i la retornem
40    return new Entrada(titol, resum, enllac);
41  }
```

El mètode `llegirEntrada()` comprova que està dins de l'etiqueta *entry* amb una crida a `parser.require()`. Posteriorment, comprova el nom de l'etiqueta que troba i es crida els mètodes `llegirTitol()`, `llegirResum()` i `llegirEnllac()` per llegir els diferents tipus de contingut de l'entrada. Tots ells retornen un `String` que fareu servir finalment en la crida al constructor d'`Entrada`.

```
1   //Llegeix el títol d'una notícia del feed i el retorna com String
2   private String llegirTitol(XmlPullParser parser) throws IOException,
        XmlPullParserException {
3     //L'etiqueta actual ha de ser "title"
4     parser.require(XmlPullParser.START_TAG, ns, "title");
5
6     //Llegeix
7     String titol = llegeixText(parser);
8
9     //Fi d'etiqueta
10    parser.require(XmlPullParser.END_TAG, ns, "title");
11    return titol;
12  }
13
14  //Llegeix l'enllaç d'una notícia del feed i el retorna com String
15  private String llegirEnllac(XmlPullParser parser) throws IOException,
        XmlPullParserException {
16    String enllac = "";
17
18    //L'etiqueta actual ha de ser "link"
19    parser.require(XmlPullParser.START_TAG, ns, "link");
20
21    //Obtenim l'etiqueta
22    String tag = parser.getName();
23
24    //Obtenim l'atribut rel (mirar l'XML d'Stackoverflow)
25    String relType = parser.getAttributeValue(null, "rel");
26
27    //Si l'enllaç és link
28
29    if (tag.equals("link")) {
30      //Obtenim l'enllaç del valor de l'atribut "href". Revisar format XML
            stackoverflow
31      if (relType.equals("alternate")) {
32        enllac = parser.getAttributeValue(null, "href");
33        parser.nextTag();
34      }
35    }
36
37    //Fi d'etiqueta
38    parser.require(XmlPullParser.END_TAG, ns, "link");
```

```
39
40    return enllac;
41  }
42
43  //Llegeix el resum d'una notícia del feed i el retorna com String
44  private String llegirResum(XmlPullParser parser) throws IOException,
        XmlPullParserException {
45
46    //L'etiqueta actual ha de ser "summary"
47    parser.require(XmlPullParser.START_TAG, ns, "summary");
48
49    String resum = llegeixText(parser);
50
51    parser.require(XmlPullParser.END_TAG, ns, "summary");
52    return resum;
53  }
54
55  //Extrau el valor de text per les etiquetes títol, resum
56  private String llegeixText(XmlPullParser parser) throws IOException,
        XmlPullParserException {
57
58    String resultat = "";
59
60    if (parser.next() == XmlPullParser.TEXT) {
61      resultat = parser.getText();
62      parser.nextTag();
63    }
64
65    return resultat;
66
67  }
```

En aquests mètodes, el que es fa és llegir el contingut de l'XML i retornar un `String`. El mètode per obtenir l'enllaç és una mica diferent, ja que els enllaços en aquest XML no vénen dins d'una etiqueta, sinó com un atribut de la mateixa, tal com es pot observar en aquesta secció d'XML:

```
1  <link rel="alternate" href="http://stackoverflow.com/questions/tagged/?tagnames
       =xml&amp;sort=newest" type="text/html" />
```

Per obtenir el valor de l'atribut heu de fer servir el mètode `parser.getAttributeValue()`.

Així doncs, l'analitzador anirà recorrent el fitxer XML tot analitzant les etiquetes i per cada *entry* crearà un objecte de la classe `Entrada` que quedarà emmagatzemat a un `List <Entrada>`, que serà retornat. Quan tinguem tota la informació de les entrades, aquestes s'hauran de mostrar en la nostra aplicació. Existeixen diferents formes de fer-ho: amb objectes creats dinàmicament, amb `ListView`... la que durem a terme nosaltres consisteix a crear un codi HTML a partir de la informació, que mostrarem en un `WebView`.

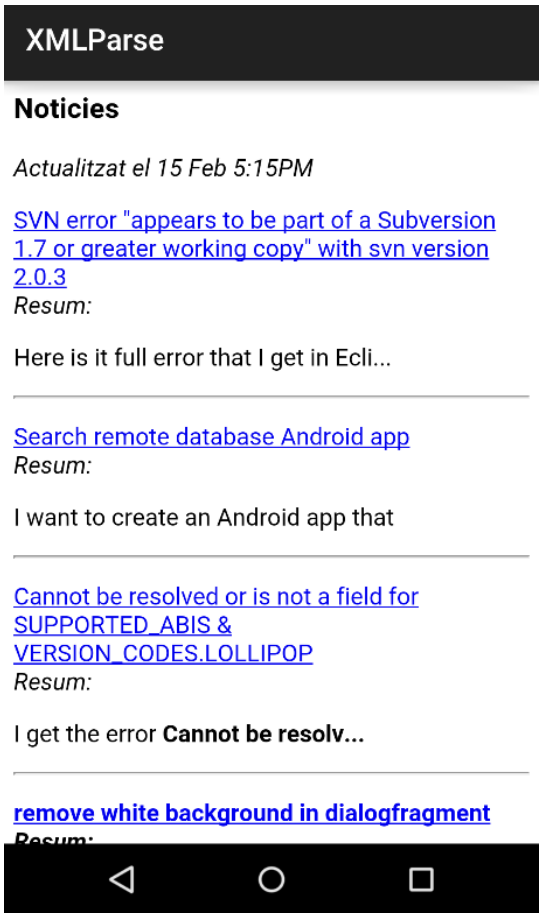
```
1  ...
2  //Llista de d'entrades de notícies
3    List<Entrada> entrades = null;
4
5  //Cadena on construirem el codi HTML que mostrarà el widget webView
6    StringBuilder htmlString = new StringBuilder();
7
8  ...
9
10   entrades = analitzador.analitza(stream);
11
12 ...
```

```
13
14 //analitzador.parse() retorna una llista (entrades) d'entrades de notícies (
       objectes
15 //de la classe Entrada). Cada objecte representa un post de l'XML Feed. Ara es
       processen
16 //les entrades de la llista per crear un codi HTML. Per cada entrada es crea un
        enllaç a la notícia completa
17
18 //Si tenim notícies
19 if(entrades != null) {
20
21
22 //Creem l'HTML a partir dels continguts del List<Entrada>
23
24 //Per indicar quan s'ha actualitzat l'RSS
25
26 Calendar ara = Calendar.getInstance();
27 DateFormat formatData = new SimpleDateFormat("dd MMM h:mmaa");
28
29 //Títol de la pàgina
30 htmlString.append("<h3> Noticies </h3>");
31
32 //Data d'actualització
33 htmlString.append("<em>Actualitzat el " + formatData.format(ara.getTime()) + "
       </em>");
34
35 //Per cada notícia de la llista
36 for (Entrada noticia : entrades) {
37   //Creem un títol de la notícia que serà un enllaç HTML a la notícia completa
38   htmlString.append("<p> <a href='");
39   htmlString.append(noticia.enllac);
40   htmlString.append("'>" + noticia.titol + "</a>");
41
42   //Si la notícia té un resum, l'afegim (els primers 70 chars)
43   String breu = noticia.resum.substring(0, 70);
44
45   htmlString.append("<br><i>Resum:</i>"+breu+"...");
46   htmlString.append("</p> <hr>");
47     }
48   }
49 ...
50   setContentView(R.layout.main);
51
52   //Mostra la cadena HTML a l'UI a través del WebView
53   WebView myWebView = (WebView) findViewById(R.id.webView1);
54
55   myWebView.loadData(htmlString.toString(), "text/html", null);
```

Es van llegint totes les entrades del `List` i generant el codi HTML per tal que es mostri a l'aplicació tal com es pot veure a la figura 2.4.

Podeu descarregar el codi de la solució en el document XMLParse.zip, que trobareu en la secció "Annexos" del web del mòdul.

Figura 2.4. Lector d'RSS en funcionament



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

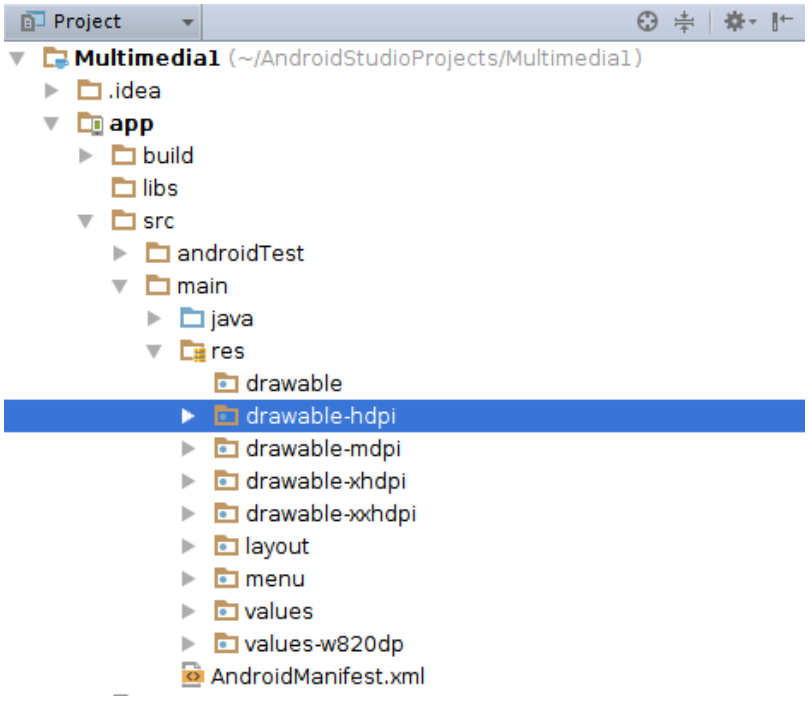
El format Portable Network Graphics (PNG) és un format d'imatge lliure (sense patents) i sense pèrdua de qualitat que, a més, permet transparència. Per aquestes raons és el format d'imatge recomanat per a les aplicacions Android, enfront d'altres formats populars com ara el JPEG.

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



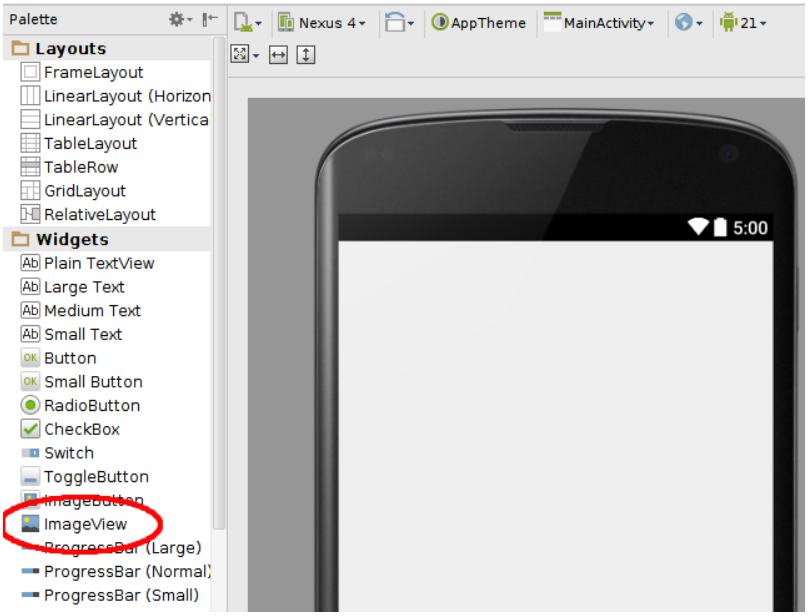
Quan hàgiu trobat aquesta subcarpeta, heu d'arrossegar-hi la imatge que voleu visualitzar per tal que passi a formar part de l'aplicació. Us apareixerà un diàleg per modificar el nom de la imatge i la seva ruta; cal que accepteu.

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.

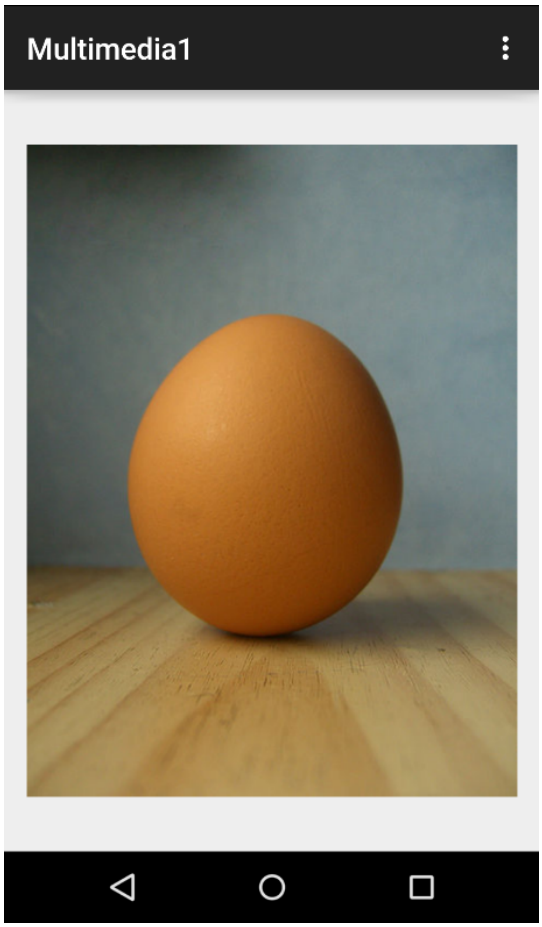
FIGURA 1.2. Localització del control "ImageView" a l'editor de layout



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:

```
<ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/imageView"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:src="@drawable/ou"
        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:

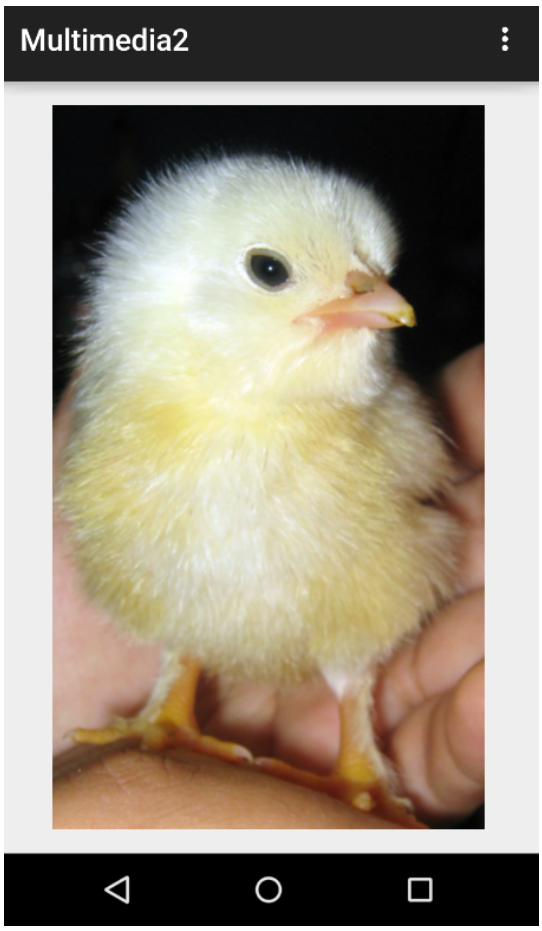
```
1 @Override
2 public void onClick(View v) {
3   // Canviem la imatge assignant-li l'altre recurs
4   ImageView visorImatge = (ImageView) findViewById(R.id.imageView);
5   visorImatge.setImageResource(R.drawable.pollastre);
6 }
```

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.

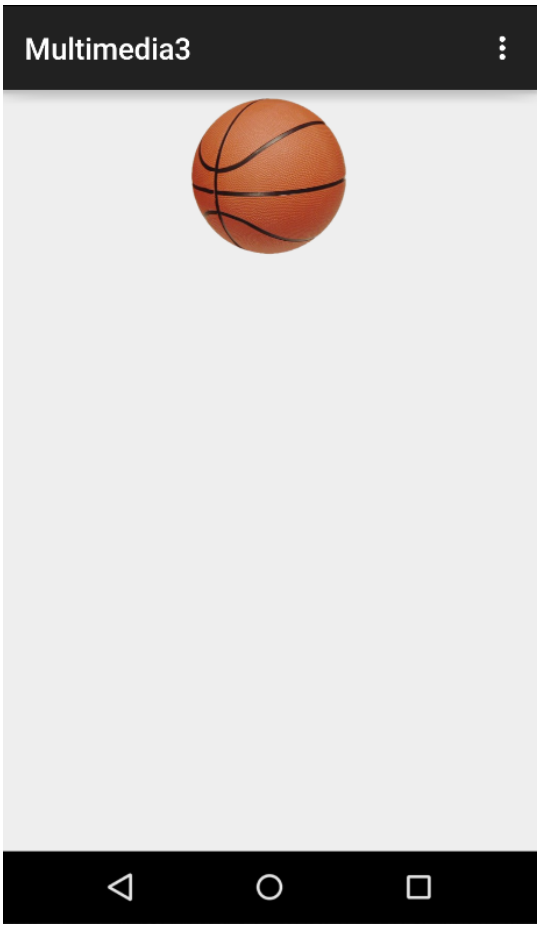
Per crear una animació per interpolació només cal que definiu l'estat inicial d'una imatge (de fet, es pot aplicar a qualsevol altra vista, com ara un text) i l'estat final desitjat, i quant ha de trigar la imatge en passar de l'estat inicial al final, i l'interpolador ja fa la resta. Els paràmetres 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:

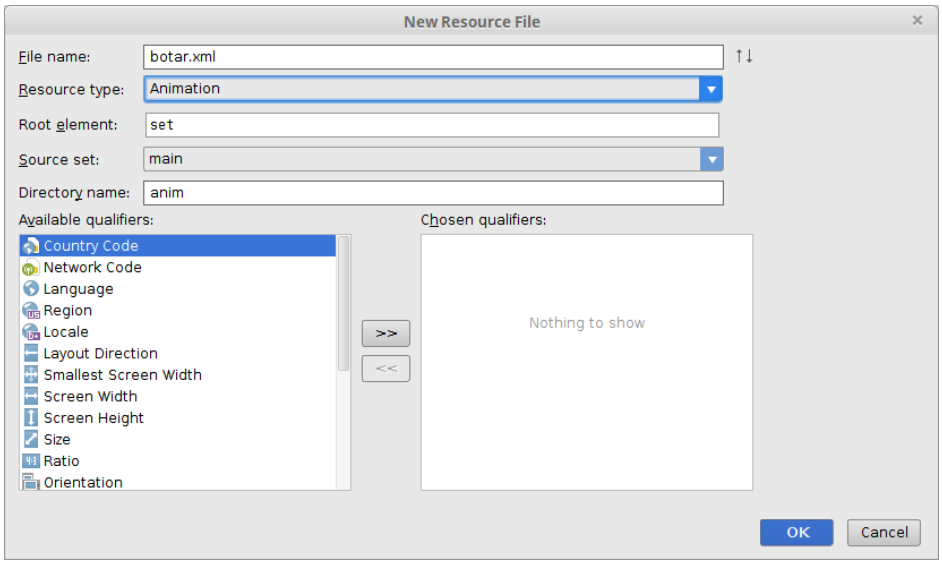
- Fer botar la pilota.
- Màgia: una pilota que apareix.
- Fer rodar la pilota.
- Fer venir de lluny la pilota.

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

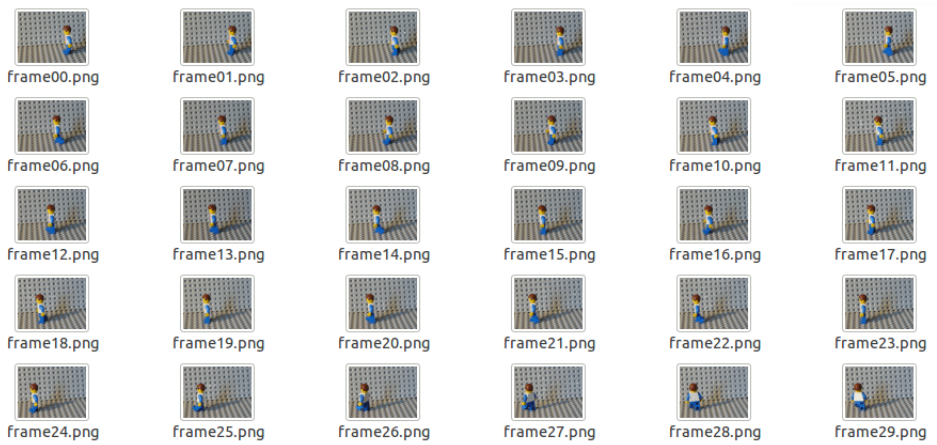
Cadascuna de les imatges de la seqüència rep el nom de fotograma, i al cap i a la fi el cine, la televisió i els videojocs, el que fan és mostrar fotogrames a alta velocitat (com a mínim, 25 per segon) per tal de donar-nos la impressió que estem veient moviment continu.

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

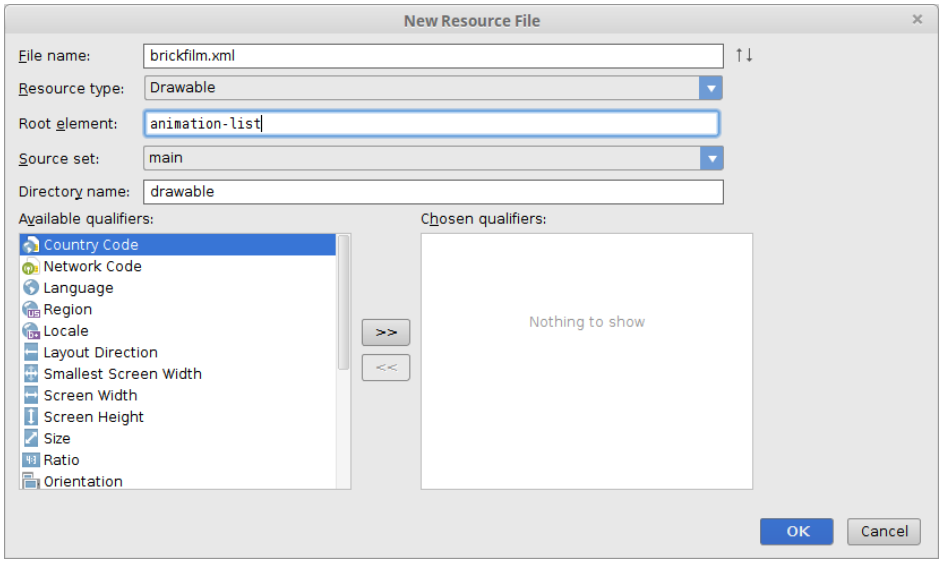


Amb vuit o deu imatges és suficient per apreciar l'efecte, no cal que feu servir moltes més imatges.

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):

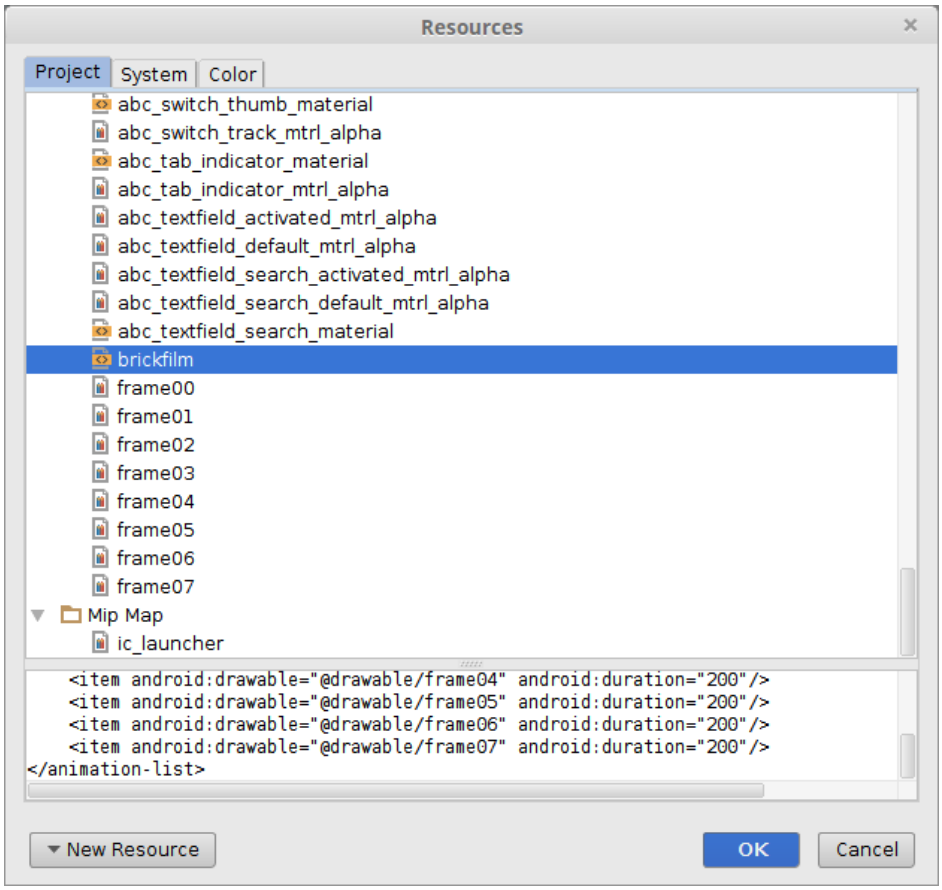
```
1 <?xml version="1.0"encoding="utf-8"?>
2 <animation-list xmlns:android="http://schemas.android.com/apk/res/android">
3   <item android:drawable="@drawable/frame00" android:duration="200"/>
4   <item android:drawable="@drawable/frame01" android:duration="200"/>
5   <item android:drawable="@drawable/frame02"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

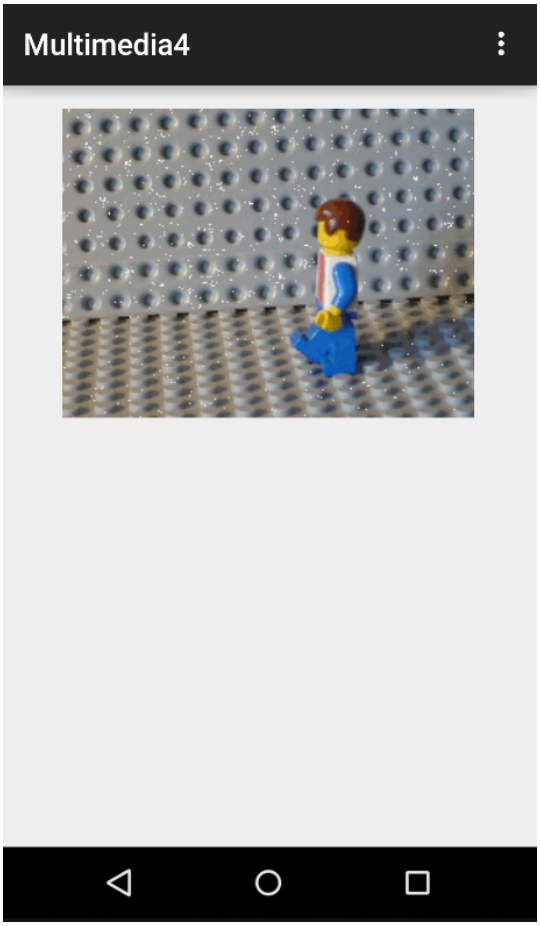


Si proveu d'executar l'aplicació només veureu el primer fotograma; cal dir-li a l'animació que s'executi. Per fer-ho, aneu al MainActivity.java i afegiu les següents instruccions al final del mètode `onCreate()`:

```
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));
```

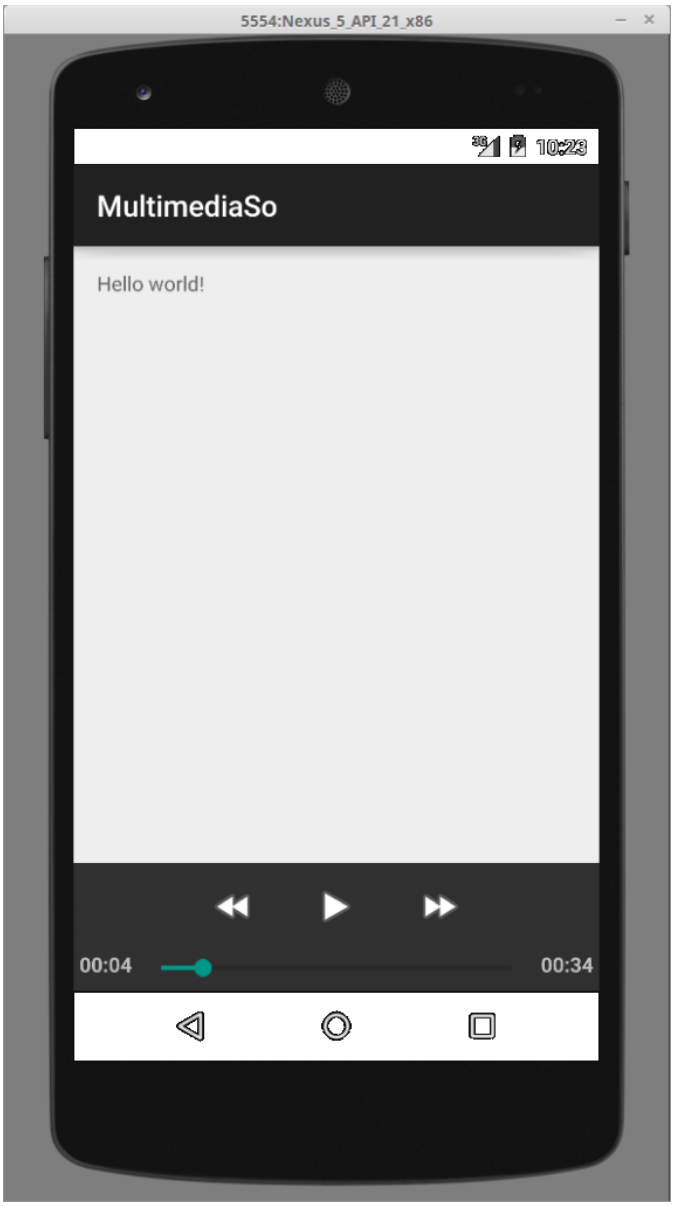
Amb aquestes instruccions, el que esteu fent és crear el *MediaPlayer* per reproduir la pista d'àudio, i després crear el *MediaController* per disposar d'uns controls per a l'àudio. A continuació associeu el codi del *MediaController* a l'activitat actual, i finalment definiu a quina vista de l'aplicació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

```
9                 Toast.LENGTH_LONG).show();
10
11    }
```

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:

```
1  @Override
2      public void onProviderDisabled(String provider) {
3          Toast.makeText(getApplicationContext(),
4                  "GPS desactivat per l'usuari",
5                  Toast.LENGTH_LONG ).show();
6      }
7
8      @Override
9      public void onProviderEnabled(String provider) {
10         Toast.makeText(getApplicationContext(),
11                 "GPS habilitat per l'usuari",
12                 Toast.LENGTH_LONG).show();
13     }
14
15     @Override
16     public void onStatusChanged(String provider, int status, Bundle extras) {
17
18         String missatge = "";
19         switch (status) {
20             case LocationProvider.OUT_OF_SERVICE:
21                 missatge = "GPS status: Out of service";
22                 break;
23             case LocationProvider.TEMPORARILY_UNAVAILABLE:
24                 missatge = "GPS status: Temporarily unavailable";
25                 break;
26             case LocationProvider.AVAILABLE:
27                 missatge = "GPS status: Available";
28                 break;
29         }
30
31         Toast.makeText(getApplicationContext(),
32                 missatge,
33                 Toast.LENGTH_LONG).show();
34     }
35
36 }
```

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:

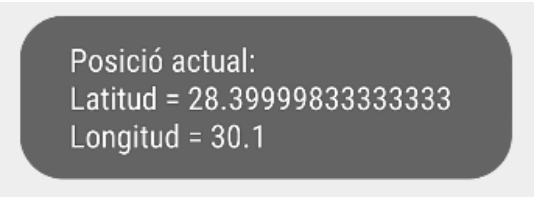
```
1  telnet localhost 5554
2  geo fix 30.1 28.4
```

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!

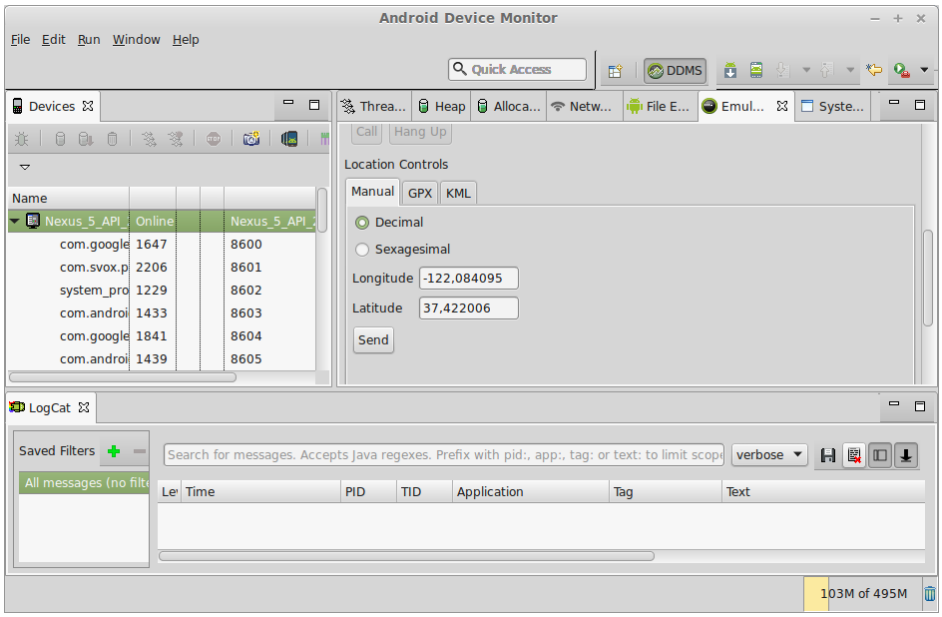
La **latitud** és la distància (angular) d'un punt a l'equador, és a dir, a l'eix sud-nord. La **longitud** és la distància (angular) d'un punt al meridià de Greenwich, és a dir, a l'eix est-oest.

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

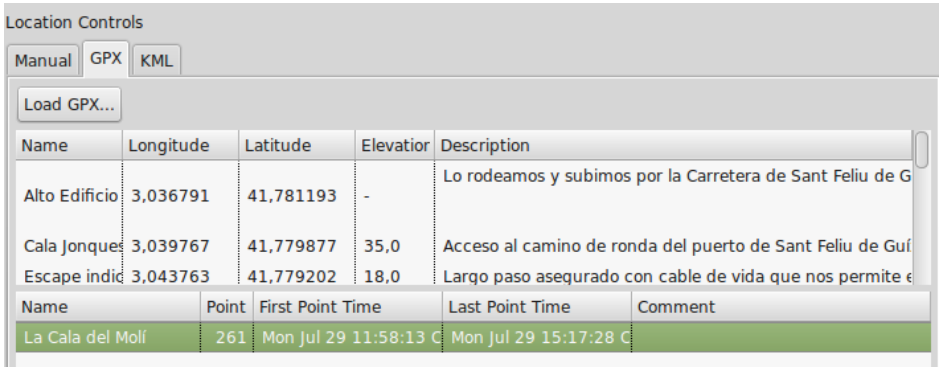


Si introduïu una coordenada i premeu el botó *Send* l'enviareu a l'emulador. Però aquesta eina us ofereix una manera molt millor d'introduir coordenades GPS: fer servir fitxers **GPX**.

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.

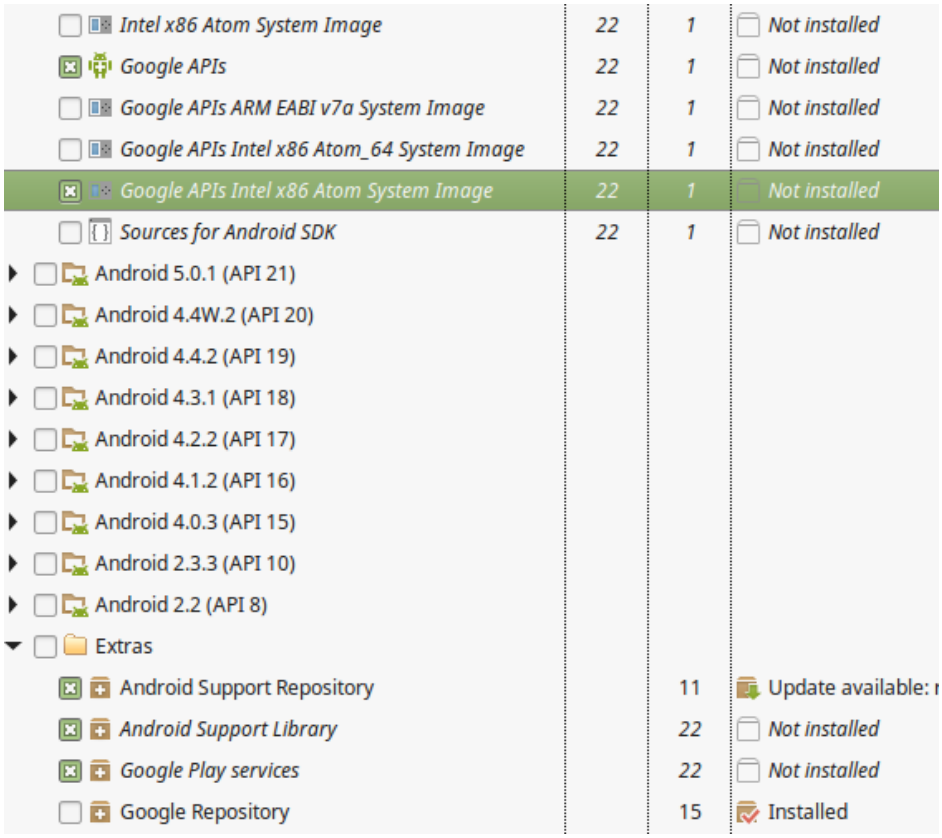
Com probablement ja sabeu, la font més important de mapes i imatges per satèl·lit és **Google Maps/Earth**, i atès que Google ha estat la creadora d'Android ho ha posat força fàcil perquè feu servir els seus mapes a les aplicacions Android.

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:

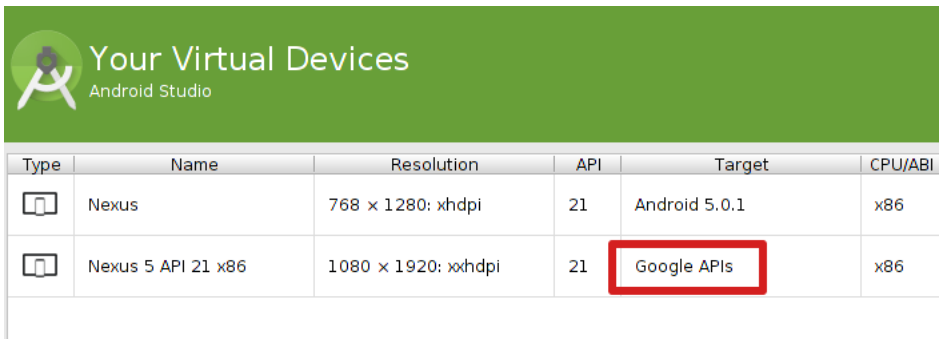
- **Google APIs Intel x86 Atom System Image** de la darrera versió de l'SDK.
- **Google APIs** de la darrera versió de l'SDK.
- **Google Play services**, que trobareu a la secció *extras*.

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

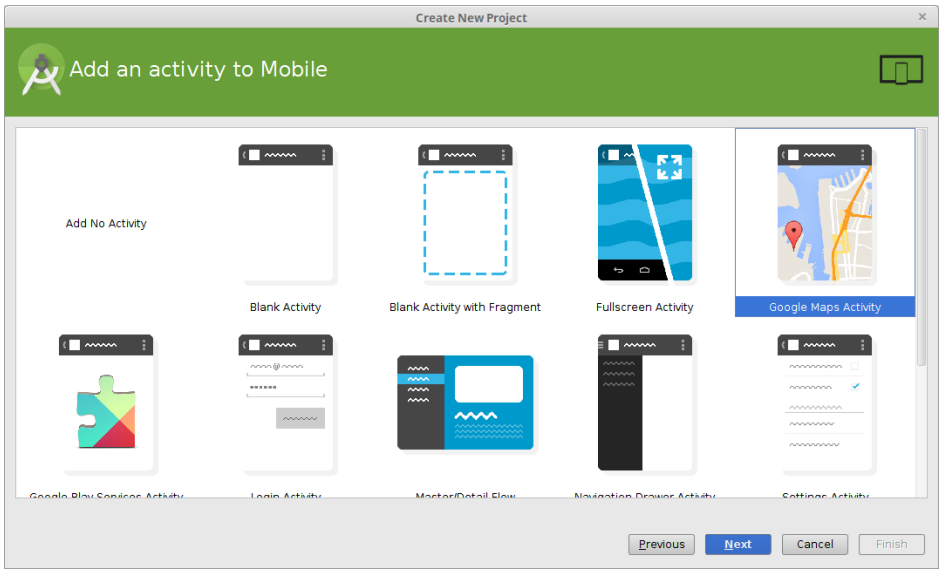


Ara que ja ho tenim tot llest per poder crear el projecte, introduïu la següent configuració:

- *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.

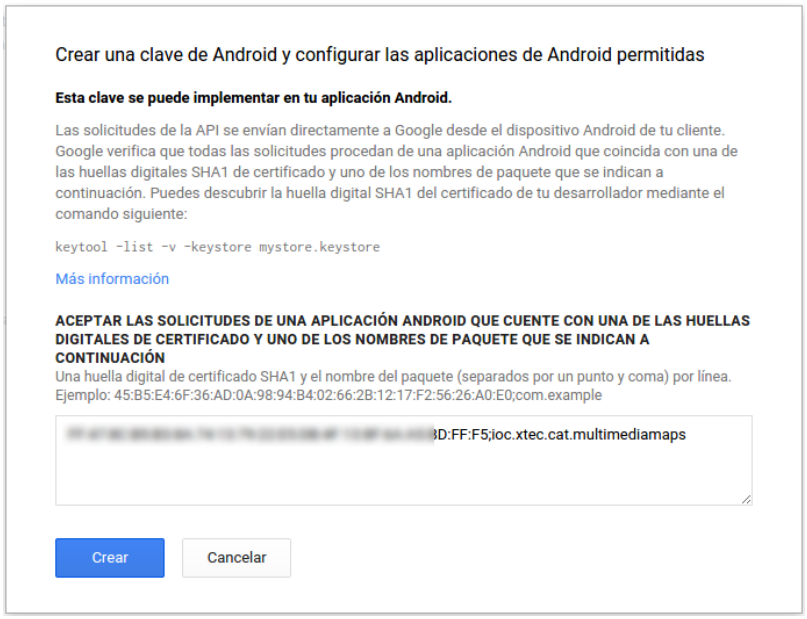
FIGURA 1.17. Selecció de l'activitat



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:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ioc.xtec.cat.multimediamaps" >

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
    <!--
 The ACCESS_COARSE/FINE_LOCATION permissions are not required to use
         Google Maps Android API v2, but are recommended.
    -->
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <meta-data
            android:name="com.google.android.gms.version"
            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:

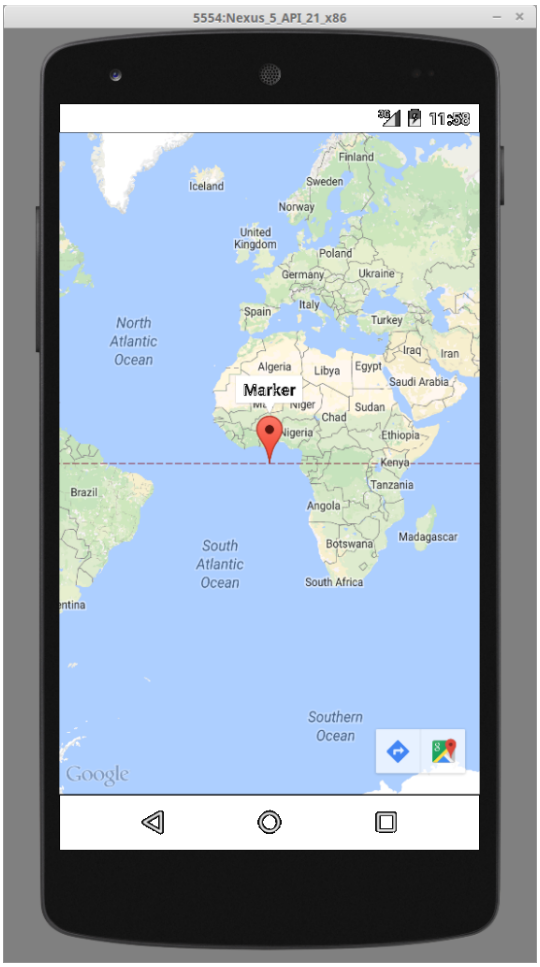
- `android.permission.INTERNET`: per accedir a Internet, necessari per descarregar els mapes.
- `android.permission.ACCESS_NETWORK_STATE`: per detectar canvis en la xarxa.
- `android.permission.WRITE_EXTERNAL_STORAGE`: permet escriure a l'emmagatzemament del telèfon, en aquest cas és necessari per descarregar 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.

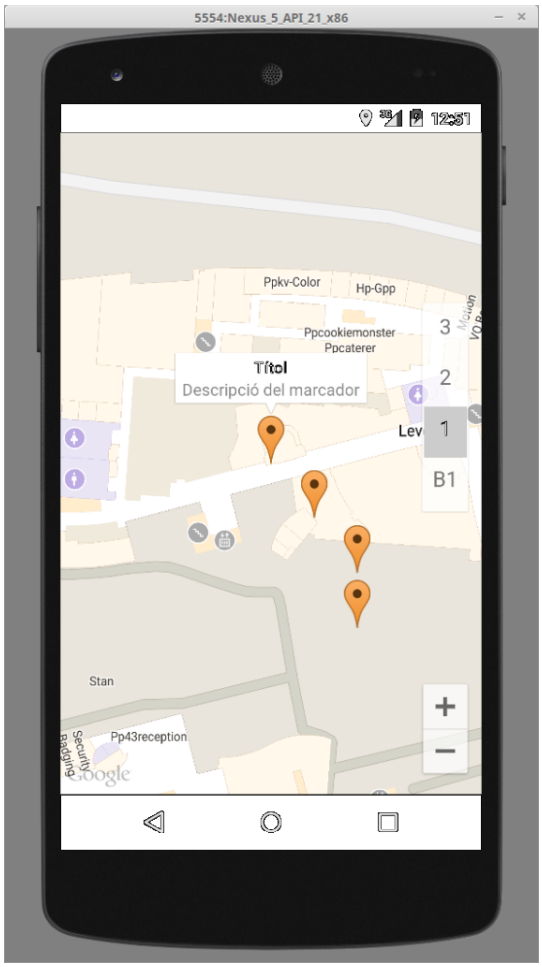
El codi del mètode `onLocationChanged` quedarà de la següent manera:

```
@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

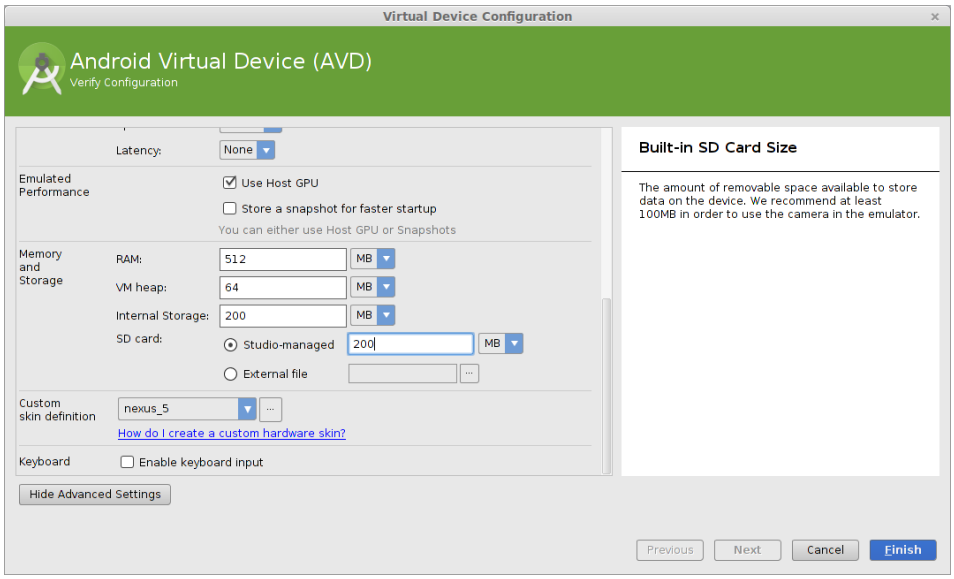
Podeu descarregar el codi corresponent a aquesta activitat en l'annex anomenat "Aplicació càmera de fotos" de la secció "Annexos".

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.

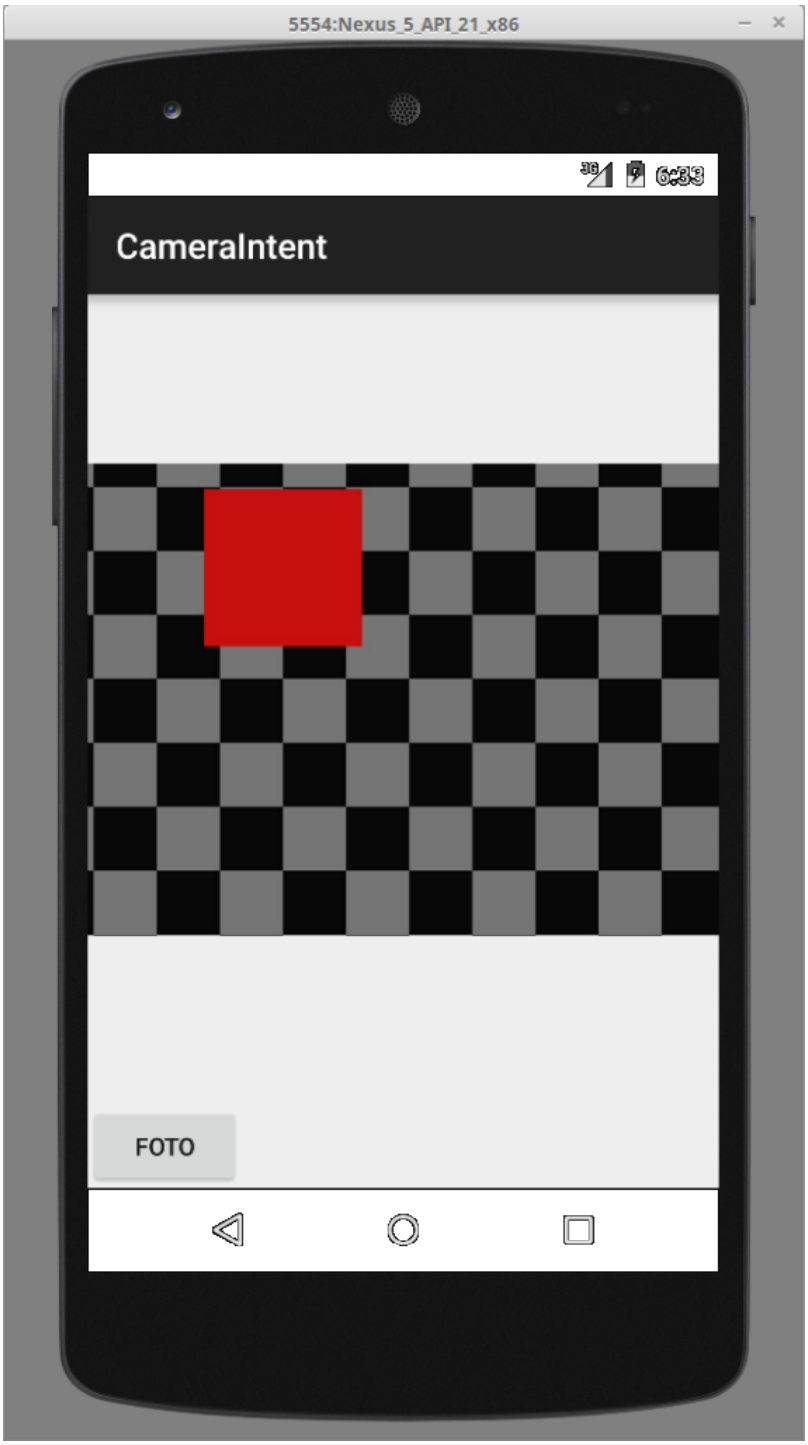
Un cop comprovat això, el codi accedeix a la imatge mitjançant el `ContentResolver`, que dóna accés als continguts del dispositiu, en aquest cas la foto que ha fet la càmera. Com que la càrrega de la imatge pot fallar, s'ha de fer dins d'un `try... catch` per tal de tractar les possibles excepcions. 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.

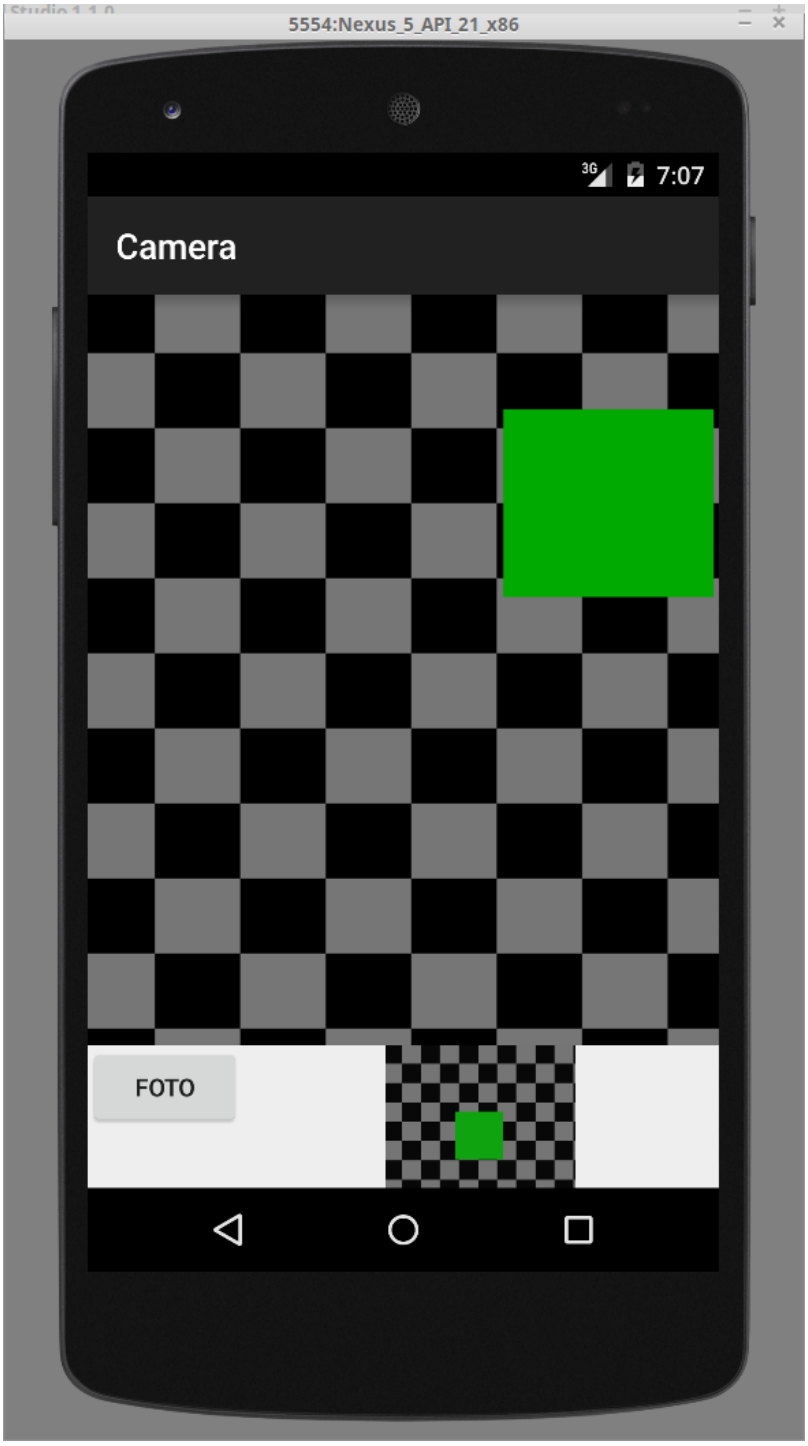
Finalment, fem un `flush()` i un `close()` de l'`OutputStream`, és a dir, forcem l'escriptura de dades que puguin quedar al *buffer*, i tanquem i alliberem el recurs.

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.

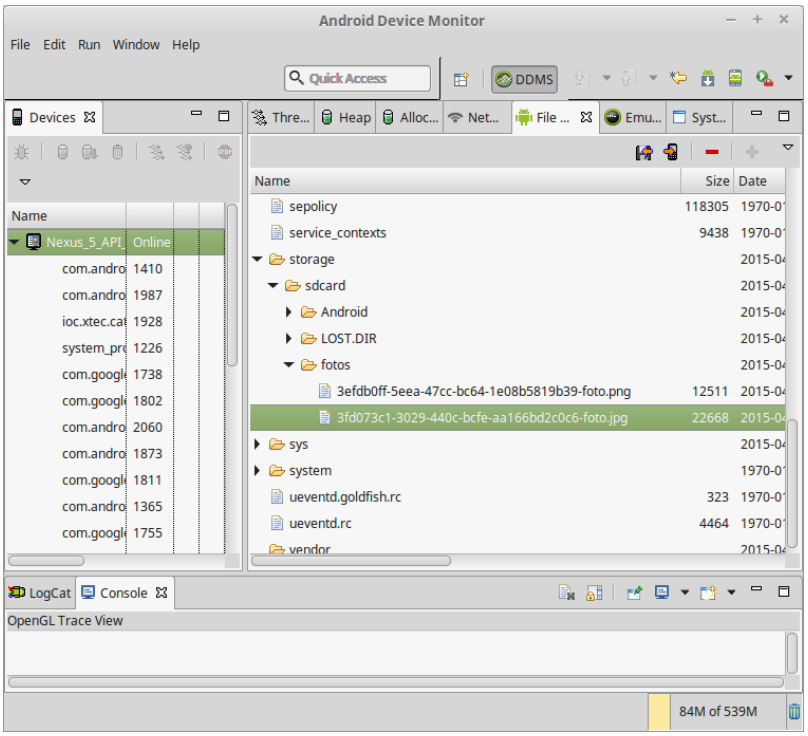
Figura 2.3. Aplicació de fotos



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**:

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

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="
        match_parent"
    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" tools:context=".
        MainActivity">

    <ToggleButton
        android:id="@+id/botoGravar"
        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'inicií 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.

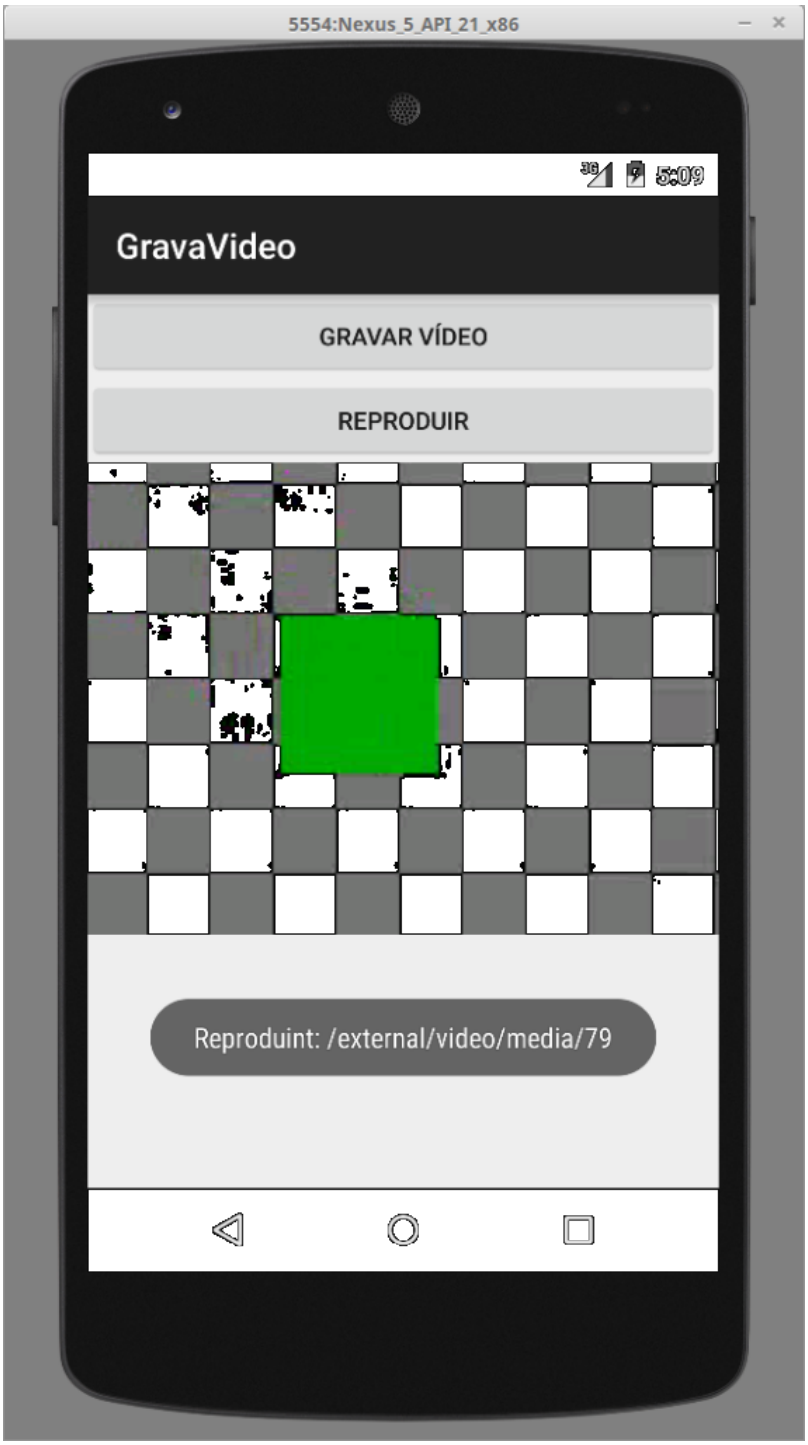
La segona funcionalitat de l'aplicació és mostrar el vídeo que s'ha gravat quan l'usuari premi el botó *Reproduir*, que s'ha activat quan l'aplicació ha gravat un vídeo amb èxit (perquè si aquest botó hagués estat disponible i l'usuari l'hagués clicat, s'hauria produït un error).

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     }
```

Desenvolupament de jocs per dispositius mòbils

Joan Climent Balaguer

Programació multimèdia i dispositius mòbils

Índex

Introducció

Avui en dia els videojocs són un dels elements més importants de totes les plataformes, tant al sector dels telèfons mòbils com dels ordinadors de sobretaula i portàtils. El fet de disposar d'una major varietat de jocs pot ser decisiva en l'elecció dels usuaris per una o altra plataforma, i això les empreses que hi ha al darrere dels sistemes operatius ho saben i dediquen gran part del seu esforç a cuidar aquest sector.

En la programació de dispositius mòbils ens trobem amb algunes diferències respecte la programació de jocs per a ordinador o per a consoles. Algunes de les més importants són:

- Les dimensions de la pantalla són reduïdes
- Tenim menys elements d'entrada d'informació i entre ells disposem dels sensors, que no són habituals en ordinadors
- Hem d'adaptar el joc tenint en compte els diferents tipus de dispositius disponibles
- S'han de prioritzar aspectes com el consum energètic o l'espai que ocupa el joc a causa de les característiques dels dispositius mòbils

En l'apartat "Programació de jocs en Android" farem un repàs als conceptes bàsics de la programació de videojocs i analitzarem els principals mètodes d'entrada de què disposem a la plataforma Android: sensors, pantalla tàctil i el teclat.

En l'apartat "Desenvolupament d'un joc" aprendrem a crear un joc des de zero fent servir un motor de jocs: LibGDX. Aprendrem a crear personatges, a assignar-los gràfics, a aplicar-los transformacions, etc.

Per seguir els continguts d'aquesta unitat és molt recomanable anar creant i provant l'aplicació pas a pas, provant les diferents configuracions i mètodes per assegurar-se que s'entén tot el que es va realitzant. Necessitareu també consultar l'ajuda contextual de l'Android Studio i la documentació de LibGDX de manera regular per descobrir noves funcions dels objectes utilitzats.

Resultats d'aprenentatge

En finalitzar aquesta unitat, l'alumne/a:

1. Selecciona i prova motors de jocs analitzant l'arquitectura de jocs 2D i 3D.

- Descriu els conceptes fonamentals de l'animació 2D i 3D.
- Identifica els elements que componen l'arquitectura d'un joc en 2D i 3D.
- Analitza els components d'un motor de jocs.
- Analitza entorns de desenvolupament de jocs.
- Analitza diferents motors de jocs, les seves característiques i funcionalitats.
- Identifica els blocs funcionals d'un joc existent.
- Defineix i executa els processos de *render*.
- Reconeix la representació lògica i espacial d'una escena gràfica sobre un joc existent.

2. Desenvolupa jocs 2D i 3D senzills fent servir motors de jocs.

- Estableix la lògica d'un nou joc.
- Crea objectes i defineix els fons.
- Instal·la i utilitza extensions per al tractament d'escenes.
- Utilitza instruccions gràfiques per determinar les propietats finals de la superfície d'un objecte o imatge.
- Incorpora so als diferents esdeveniments del joc.
- Desenvolupa i implanta jocs per a dispositius mòbils.
- Realitza proves de funcionament i optimització dels jocs desenvolupats.
- Documenta les fases de disseny i desenvolupament dels jocs creats.

1. Programació de jocs en Android

La programació de videojocs és una de les parts més complexes a dins del món de la informàtica ja que requereix de molts coneixements tècnics que a més comprenen molts sectors. Per tenir-ne una idea bàsica, per crear un joc es necessita:

- Definir el concepte del joc (equip creatiu)
- Escriure el guió del joc (guionistes)
- Dissenyar els personatges, escenaris i nivells (dissenyadors gràfics)
- Crear els efectes sonors, música i veus (equip de so)
- Programar la lògica del joc (programadors)

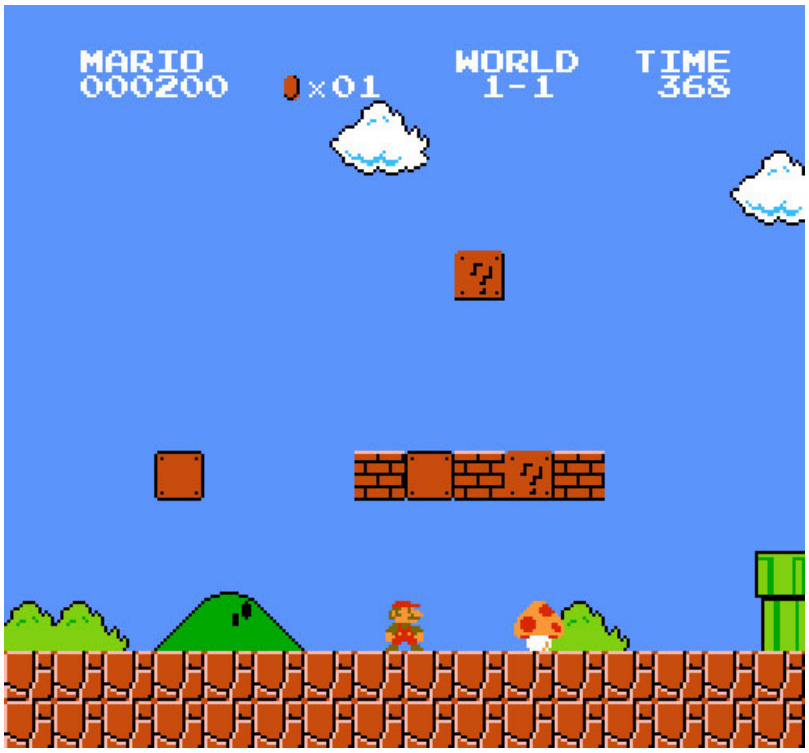
I a més, una vegada estigui tot llest, s'ha de comprovar que tot funcioni correctament per a finalment passar a la postproducció.

Com podem veure, la creació d'un videojoc no és trivial i implica el treball i la coordinació de moltes persones.

En aquesta unitat acabarem desenvolupant un joc, anirem fent poc a poc alguns dels passos descrits a la llista anterior, llavors, abans de posar-nos a programar, haurem de decidir la temàtica del nostre joc. Per fer això necessitarem conèixer alguns dels distints gèneres que existeixen:

- Plataformes: un dels gèneres per excel·lència als jocs en 2 dimensions (2D), molt popular gràcies a jocs com *Donkey Kong* o *Super Mario Bros* (vegeu figura 2.1). En aquest gènere l'objectiu és el d'anar esquivant obstacles i en la majoria dels casos eliminar els enemics que ens anem trobant pel camí. Amb la popularització dels jocs en 3 dimensions (3D), els jocs de plataformes han disminuït respecte altres gèneres però sense arribar a desaparèixer.

FIGURA 1.1. Joc de plataformes: Super Mario Bros



Un *mod* (videojocs) és una modificació d'un joc en el qual fent servir la mateixa base, es canvien els objectius i fins i tot la jugabilitat de l'original.

- Tirs: jocs en el quals l'objectiu principal és atacar els enemics fent servir armes de foc. Entre els seus subgèneres més coneguts trobem:

  – En primer persona: en anglès *First Person Shooter* (*FPS*), són jocs en els quals es visualitza allò que el protagonista veuria a la realitat (perspectiva subjectiva). Aquest tipus de perspectiva combinada amb l'ús d'armes donen nom al gènere. Avui en dia és molt popular als jocs amb opció multijugador, és a dir, en aquells que els usuaris juguen entre ells i no contra personatges controlats per una intel·ligència artificial. Alguns dels jocs més coneguts d'aquest subgènere serien: *Doom*, *Half Life* i el seu *mod Counter Strike* o la saga *Call of duty*.

  – En tercera persona: en anglès *Third Person Shooter* (*TPS*), són molt similars en concepte als *FPS* a excepció que la càmera se situa a la part posterior del personatge, és a dir, veiem totalment o parcialment el cos d'aquest. Alguns exemples serien *Tomb Raider*, les sagues *Resident Evil* o el *Gears of war*.

  – *Shoot 'em up*: jocs en els quals habitualment controlem un vehicle que s'enfronta amb una gran quantitat d'enemics a qui ha de destruir. En aquest subgènere és molt comú trobar perspectives zenitals o laterals i gràfics en 2D. Exemples de jocs d'aquest gènere serien: *Thunder Force*, *R-Type* o *1942*.

- Aventures gràfiques: molt populars durant els anys 90, tenen com a principal objectiu anar avançant en la història que ens proposen bé sigui mitjançant la resolució de trencaclosques, mitjançant diàlegs amb personatges o fent servir objectes. Entre les aventures gràfiques més populars trobem la saga

Monkey Island, *Broken Sword*, *The day of the tentacle* o *The longest journey* entre altres.

- Conducció/carreres: entraria a dins del gènere de simuladors. En aquest tipus de jocs, bé sigui amb una perspectiva en primera o tercera persona, el jugador condueix un vehicle per competir amb altres jugadors i arribar el primer a la meta. Alguns exemples d'aquest gènere serien la saga *Need for speed*, *Gran Turismo* o els distints *Mario Kart*.

Joc casual

Un joc casual és un joc en què la jugabilitat és molt senzilla, no necessita gaire temps de dedicació ni habilitats especials per jugar-lo. Ja existia abans de les plataformes mòbils però amb aquestes ha pres una gran importància.

Molts dels jocs presents avui en dia a les plataformes mòbils són considerats "jocs casuals", aquest tipus de jocs tenen també un gènere associat. Exemples més evidents d'aquest tipus de jocs són els coneguts *Angry Birds* i *Candy Crush Saga*, tots dos tenen una mecànica simple i nivells molt curts que ens permetran jugar tot i no tenir molt de temps disponible, per exemple entre una parada i una altra del metro.

1.1 Conceptes bàsics de la programació de jocs

Abans de començar amb la programació de videojocs hi ha una sèrie de conceptes que caldria conèixer. Fem un repàs d'alguns que ens poden ser útils per entendre el funcionament dels jocs o bé que ens ajudaran en el seu procés de creació.

1. Píxels i resolució de pantalla

Totes les pantalles existents al mercat, incloses les dels telèfons mòbils i tauletes, estan dividides en píxels.

> Un **píxel** és la menor unitat que podem representar en una pantalla. Una pantalla està formada per milions de píxels, dels quals necessitem conèixer-ne la posició respecte aquesta i el color que té.

La **resolució** d'una pantalla ens informarà de la quantitat de divisions horitzontals i verticals de què disposem, i això determinarà el número de píxels que tindrà. Per exemple, quan ens diuen que un telèfon mòbil té una resolució de 1080 x 1920 ens estan informant de la quantitat de divisions que té la pantalla en ample i alt respectivament (mirant el telèfon en *portrait* o les tauletes en *landscape*). Si multipliquem les dues divisions, ens donarà el total de píxels, amb la resolució de l'exemple anterior tindríem un total de 2073600 píxels.

Si a més tenim en compte les dimensions de pantalla, podem calcular la **densitat de píxels** (mesurada en *ppi*, *píxels per inch*) de la pantalla, és a dir, quants píxels hi ha per cada polzada de pantalla; a major densitat, més nítida serà la visualització de la pantalla. Per calcular la densitat de píxels hem d'aplicar la següent fórmula:

$$ppi = \frac{\sqrt{r_h^2 + r_v^2}}{d_i}$$

on r_h és la resolució horitzontal, r_v la vertical i d_i la diagonal de la pantalla, és a dir, les polzades que té. Si sabem que la resolució de l'exemple anterior era d'un dispositiu amb una pantalla de 4,95", la densitat de píxels seria:

$$ppi = \frac{\sqrt{1080^2 + 1920^2}}{4,95} \approx 445ppi$$

.

La profunditat de color fa referència al número de bits que dediquem a representar els colors, a major número de bits, més colors es podran fer servir. Per exemple, amb una profunditat de color de 16 bits es poden representar 2^{16} colors, és a dir, 65536 colors.

2. *Sprites* i animacions

Un ***sprite*** és una imatge en mapa de bits d'un determinat ample i alt. De cada píxel que forma aquesta imatge cal guardar el color que tindrà així que l'espai ocupat per un *sprite* serà el total de píxels de la imatge multiplicat per la seva profunditat de color. Una imatge de 200x200 píxels amb una profunditat de color de 32 bits ocuparia 1.280.000 bits, que passat a bytes i posteriorment a kibibytes farien un total de 156,25 KiB.

Zoòtrop

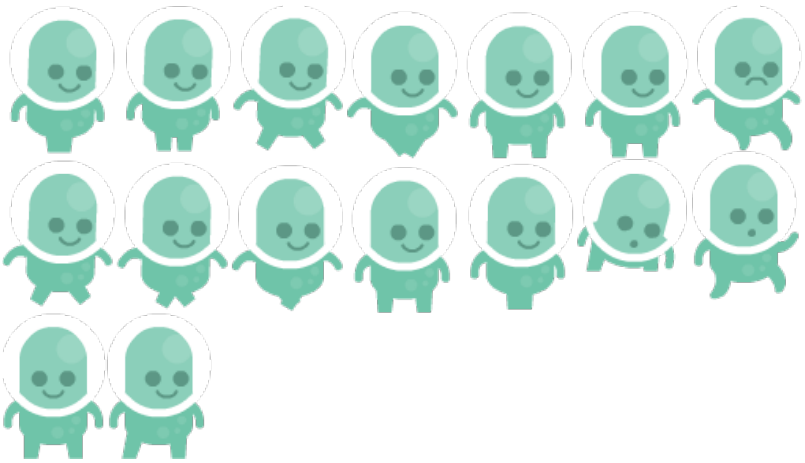
Per tal de reduir la memòria que poden ocupar els *sprites* en ser redimensionats, es fa servir el concepte d'*sprite sheet*: un mapa de bits que conté totes les imatges que es faran servir a la nostra aplicació, del qual traurem tota aquella informació innecessària (com per exemple píxels transparents o colors que són innecessaris). Al final haurem d'aconseguir fer més lleugers els recursos gràfics amb la qual cosa obtindrem diverses millores: reduirem l'espai utilitzat per la nostra aplicació i en millorarem el rendiment. Al següent vídeo podeu veure un exemple d'aquest concepte:

https://www.youtube.com/embed/crrFUYabm6E?controls=1

El zoòtrop va ser un dispositiu que produïa la il·lusió de moviment i que és considerat un dels precursors del cinema. Aquest dispositiu era circular i podíem observar una animació creada a partir de fotogrames a través d'un visor vertical.

Per tal de realitzar animacions, necessitarem crear els *sprites* dels objectes dels diferents estats de l'animació, tal com es feia a un zoòtrop. Afegirem tots els *sprites* a l'*sprite sheet* i després els anirem canviant a la nostra aplicació per donar la sensació de moviment. Un exemple el trobem a la figura 2.3.

FIGURA 1.2. Sprite sheet



3. *Frame rate*

El *frame rate* fa referència a la quantitat d'imatges (*frames*) que es mostren per segon. És una mesura de freqüència i la seva unitat són els *frames per second* (*FPS*).

4. *Polling / Event handling*

El terme *polling* fa referència a una consulta periòdica d'un valor, habitualment obtingut d'un dispositiu de *hardware* (un dispositiu d'entrada o un sensor, per exemple).

La gestió de l'entrada de l'usuari o l'estat dels sensors es pot fer mitjançant *polling* o mitjançant *event handling* (gestió d'esdeveniments).

- Amb *polling* el que fem és comprovar l'estat actual d'un dispositiu d'entrada, per exemple en un bucle del joc comprovem si s'ha pitjat la pantalla o una tecla.

- El sistema *event handling* ens permet gestionar la informació d'entrada a través d'*events* (esdeveniments, en català), és a dir, important, quan la necessitem conèixer, la seqüència dels *events*, per exemple, quan passem el dit per la pantalla ens interessa primer tractar el moment en què el dit toca la pantalla, després tractem el desplaçament del dit i finalment gestionem quan l'usuari aixeca el dit de la pantalla. Serien, per exemple, els *events*: `touchDown`, `touchDragged` i `touchUp`.

5. Gràfics 3D

Per tal de desenvolupar aplicacions que disposin de gràfics en tres dimensions haurem de recórrer a APIs gràfiques. Actualment les principals són:

- **OpenGL**: *Open Graphics Library* és una API multiplataforma i multillenguatge per a la creació de gràfics en 2D i 3D. La versió 4.5 va ser publicada l'11 d'agost de 2014.

- **Direct3D**: API gràfica de Microsoft a dins de les llibreries *DirectX*. Ens permet crear gràfics en 3D per a les plataformes i sistemes operatius de Microsoft: videoconsoles Xbox i sistemes operatius Windows (inclosos els sistemes operatius per a dispositius mòbils).

OpenGL ES (*OpenGL for Embedded Systems*) és una versió reduïda d'OpenGL dissenyada per a dispositius mòbils o consoles de videojocs. La numeració és distinta entre OpenGL i OpenGL ES.

Les versions d'Android Lollipop 5.X disposen de la versió d'OpenGL ES 3.1. Si observem el "*hello world*" d'OpenGL que podem trobar a la pàgina de desenvolupadors http://developer.android.com/training/graphics/opengl/index.html podrem veure que la programació en 3D no és trivial i que requereix de molts coneixements i temps. Al tutorial enllaçat se'ns expliquen els passos per dibuixar figures geomètriques i per afegir-los moviment.

1.2 Entrada de dades

Un dels aspectes més importants dels videojocs és la manera amb què recollim la informació de l'usuari. A banda de la pantalla tàctil i dels teclats i botons físics, els dispositius mòbils acostumen a incorporar molts sensors que ens faciliten aquesta interacció i que estan dividits en tres categories:

- Sensors de moviment: on trobem acceleròmetres, sensors de gravetat, giroscopis, etc.
- Sensors d'entorn: baròmetres, termòmetres i sensors de llum.
- Sensors de posició: sensors d'orientació i magnetòmetres.

Cadascun dels sensors ens podrà ser útil segons la finalitat de la nostra aplicació però sempre haurem de tenir en compte que no tots els dispositius incorporen tots els sensors. Android ens facilitarà l'accés a la informació sobre quins sensors tenim disponibles, les característiques de cadascun d'ells i l'obtenció de la informació que ens proporcionen a través d'un *framework* propi que forma part del *package* `android.hardware`.

1.2.1 Pantalla tàctil

La pantalla tàctil és el dispositiu d'entrada d'informació per excel·lència als dispositius mòbils. A través d'ella interactuem amb el dispositiu la major part del temps.

Des de la perspectiva de la programació d'aplicacions, per controlar què ha fet l'usuari, disposem del mètode `onTouchEvent(MotionEvent event)` de la classe `Activity` o bé del mètode `onTouch(View v, MotionEvent event)` de la interfície `OnTouchListener`. La diferència entre els dos és: mentre que al mètode `onTouchEvent` l'aplicació registrarà totes les pulsacions que es produeixin en la nostra activitat, independentment del *view* sobre el qual estem polsant, implementant la interfície `onTouchListerner` podrem especificar de quins *views* ens interessa conèixer les pulsacions, fent el corresponent `setOnTouchListener`.

Podeu descarregar el projecte de la següent aplicació a l'apartat "Touch Events" dels annexos.

Per entendre els *events* que es produeixen cada cop que l'usuari interactua amb la pantalla, crearem una aplicació, tot iniciant un nou projecte de nom **TouchEvents** i de domini **cat.xtec.ioc**.

El *layout* de la nostra aplicació tindrà una capsa de text que mostrarà els *events* que s'aniran generant (i que ens permetrà fer *scroll*), un *LinearLayout* que controlarà les accions de l'usuari amb la pantalla tàctil i una altra capsa de text que mostrarà la direcció en la qual l'usuari ha arrossegat el dit. Si posem aquests elements a dins d'un *LinearLayout* ens quedarà el següent codi:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <ScrollView
        android:id="@+id/scroll"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_weight="1"
        android:fillViewport="true"
        android:scrollbars="vertical">

        <TextView
            android:id="@+id/accions"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:scrollbars="vertical"
            android:text=""
            android:textAppearance="?android:attr/textAppearanceSmall" />
    </ScrollView>

    <LinearLayout
        android:id="@+id/linear"
        android:layout_width="match_parent"
        android:layout_height="fill_parent"
        android:layout_weight="1"
        android:background="#ff88eeff"
        android:orientation="vertical">


        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:text=""
            android:id="@+id/direccio" />
    </LinearLayout>

</LinearLayout>
```

Observeu que el *textView* d'identificador "accions" és a dins de l'element *ScrollView*. Això ens permetrà fer *scroll* en cas de produir-se *overflow*, és a dir, que el text no càpiga a l'espai assignat.

Cada cop que l'usuari faci una pulsació o arrossegui el dit pel *LinearLayout* d'identificador "linear", es generaran una sèrie d'*events* que quedaran recollits al *textView* "accions".

Abans de passar a la part del codi, modificarem l'entrada de menú *settings* per donar-li una utilitat. Aquesta netejarà el contingut del *textView* "accions". Editem el fitxer */res/menu/menu_main.xml* per deixar-lo de la següent manera:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools" tools:context=".MainActivity">
    <item android:id="@+id/action_clear" android:title="@string/action_clear"
  android:orderInCategory="100" app:showAsAction = "ifRoom"/>
</menu>
```

Hem fer tres modificacions de l'element *item*:

- `android:id=''@+id/action_settings''` per `android:id=''@+id/action_clear''`: canviem el nom de l'identificador.
- `android:title=''@string/action_settings''` per `android:title=''@string/action_clear''`: canviem el text que es mostrarà a la pantalla.
- i app:showAsAction = "never" per app:showAsAction = "ifRoom": fem que l'element de menú sigui visible a l'`ActionBar` sempre que hi hagi lloc disponible.

Com que hem canviat l'*string* que es mostrarà per pantalla, haurem d'incorporar-la al fitxer */res/values/strings.xml* i podrem eliminar els recursos que no es facin servir, quedant de la següent manera:

```
<resources>
    <string name="app_name">TouchEvents</string>

    <string name="action_clear">Neteja</string>
</resources>
```

Una vegada ja tenim les vistes i el menú preparats, donem funcionalitat a la nostra aplicació. Com que volem enregistrar les pulsacions a un *view* concret haurem d'implementar la interfície `OnTouchListener` i per això, haurem d'afegir el corresponent `implements` en la definició de la classe:

```
public class MainActivity extends ActionBarActivity implements View.
    OnTouchListener {
```

Definirem les variables corresponents als *views* que volem modificar i afegirem dos *float* que guardaran l'inici d'un *event* d'arrossegar (*drag*): "iniciX" i "iniciY". Per últim necessitarem un enter ("numAccio") que actuarà com a comptador dels *events* que s'han generat.

Codi de definició de les variables:

```
TextView accions, direccio;
    float iniciX, iniciY;
    int numAccio;
```

Al mètode `onCreate`, inicialitzarem les variables i afegirem l'`OnTouchListener` al *textView* d'accions, que quedarà de la següent manera:

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

        // Afegim el "Listener" dels events de "touch" sobre el layout inferior
            .
        LinearLayout linear = (LinearLayout) findViewById(R.id.linear);
        linear.setOnTouchListener(this);

        // Inicialitzem les variables
```

```
        accions = (TextView) findViewById(R.id.accions);
        direccio = (TextView) findViewById(R.id.direccio);
        iniciX = iniciY = 0;
        numAccio = 0;

    }
```

Pel que fa al menú, deixem la funció `onCreateOptionsMenu` sense modificar i controlem si l'usuari ha fet clic a l'opció "Neteja" des de la funció `onOptionsItemSelected`, si l'usuari vol esborrar: netejarem el *textView* i posarem el comptador a 0:

```
@Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //Netegem el TextView i posem el comptador a 0
        if (id == R.id.action_clear) {
            accions.setText("");
            numAccio = 0;
            return true;
        }

        return super.onOptionsItemSelected(item);
    }
```

Tota la gestió de les pulsacions de l'usuari la farem a la funció `onTouch(View v, MotionEvent event)` en la qual disposem de dos arguments: la vista sobre la que s'ha fet una pulsació i el `MotionEvent` generat, que entre d'altra informació, ens permetrà saber quin *event* s'ha realitzat, sent alguns dels més importants els següents:

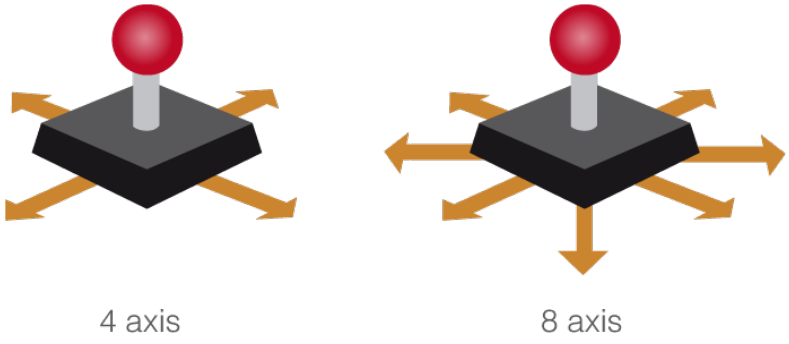
- **ACTION_DOWN**: es produeix quan l'usuari inicia una pulsació, és a dir, al moment exacte que el dit fa contacte en la pantalla.
- **ACTION_UP**: quan finalitzem la pulsació, quan el dit deixa de tocar la pantalla.
- **ACTION_MOVE**: entre els *events* ACTION_DOWN i ACTION_UP hi ha hagut algun canvi en la posició del dit, si hi ha moviment és perquè l'usuari ha desplaçat el dit per la pantalla durant la pulsació.
- **ACTION_POINTER_DOWN**: s'ha produït una pulsació d'un dit que no és el principal (passem a generar *events multi touch*). Quan aquest dit s'aixeca de la pantalla genera un **ACTION_POINTER_UP**.
- **ACTION_POINTER_INDEX_MASK**: conjuntament amb **ACTION_POINTER_INDEX_SHIFT** ens permetrà saber quin índex té la pulsació en un *event multi touch*.

Per controlar un *event* d'arrossegament de dit farem servir els *events* `ACTION_DOWN` (on guardarem la posició inicial del dit a la pantalla) i `ACTION_UP` (que guardarà la posició final i ens permetrà calcular la direcció). Si volem

saber quin desplaçament ha fet l'usuari, haurem de descartar tota la informació innecessària i calcular la trajectòria segons ens interessi; en *4-axis* o en *8-axis* (vegeu figura 2.4).

La coordenada (0,0) de la pantalla se situa a la part superior esquerra del dispositiu.

FIGURA 1.3. Moviments en 4 i 8 axis



- **4-axis**: els valors possibles són dalt, dreta, baix i esquerra. Per saber la direcció en què l'usuari ha arrossegat el dit haurem de veure en quin eix (horitzontal o vertical) i en quina direcció ho ha fet. Com que és molt complex realitzar un desplaçament perfecte en un únic eix, haurem de descartar la informació que ens arriba de l'altre; una manera de fer-ho és detectar a quin eix s'ha produït un major desplaçament fent la resta entre els valors absoluts dels punts inicials i els punts finals. Per realitzar aquesta implementació haurem de guardar els punts X i Y inicials quan es produeix l'*event* `ACTION_DOWN` i recollir les posicions finals a l'`ACTION_UP`. Una vegada sabem l'eix, podrem obtenir fàcilment el sentit fent la resta entre el punt inicial i el final. Aquesta serà la implementació que farem a la nostra aplicació.

- **8-axis**: a més les direccions que disposem amb 4-axis, podem considerar les que es produeixen a partir d'una combinació dels dos eixos, és a dir, les posicions: dalt-dreta, baix-dreta, baix-esquerra i dalt-esquerra. Com que és molt complex realitzar un desplaçament de "0" en qualsevol eix, per tal d'implementar aquesta solució haurem de definir un valor mínim a partir del qual es considera que hi ha hagut un desplaçament en una determinada direcció. Si no afegim aquest valor mínim, ens trobaríem amb què tots els desplaçaments serien una combinació dels dos eixos i no hi hauria cap moviment dels 4 *axis* principals.

El codi de la funció `onTouch` serà el següent:

```
1  @Override
2      public boolean onTouch(View v, MotionEvent event) {
3
4          if (event.getAction() == MotionEvent.ACTION_DOWN) {
5
6              // Registrem l'event al TextView
7              accions.setText(String.valueOf(numAccio) + " - " + "ACTION_DOWN\n"
                   + accions.getText());
8              iniciX = event.getX();
9              iniciY = event.getY();
10
```

```
11
12          } else if (event.getAction() == MotionEvent.ACTION_UP) {
13
14              // Registrem l'event al TextView
15              accions.setText(String.valueOf(numAccio) + " - " + "ACTION_UP\n" +
                    accions.getText());
16              float finalX = event.getX();
17              float finalY = event.getY();
18
19              //Comprovem quin dels dos moviments ha estat major. Si és l'
                    horitzontal
20              if (Math.abs(finalX - iniciX) > Math.abs(finalY - iniciY)) {
21
22                  // Si la X final és major que la inicial ha fet "drag" cap a la
                        dreta
23                  if (finalX > iniciX) {
24                      direccio.setText("→");
25                  // Si no, cap a l'esquerra
26                  } else {
27                      direccio.setText("←");
28                  }
29              // Si és el vertical
30              } else {
31                  // Si la Y final és major que la inicial ha fet "drag" cap
                        avall
32                  if (finalY > iniciY) {
33                      direccio.setText("↓");
34                  // Si no, cap a dalt
35                  } else {
36                      direccio.setText("↑");
37                  }
38              }
39
40          } else if (event.getAction() == MotionEvent.ACTION_MOVE) {
41
42              // Registrem l'event al TextView
43              accions.setText(String.valueOf(numAccio) + " - " + "ACTION_MOVE\n"
                    + accions.getText());
44
45          }
46          // Incrementem el comptador d'acció
47          numAccio += 1;
48          return true;
49      }
```

Si l'acció realitzada (`event.getAction()`) ha estat `ACTION_DOWN` guardarem les coordenades X i Y de la pulsació i quan es produeixi `ACTION_UP` recollirem els valors finals.

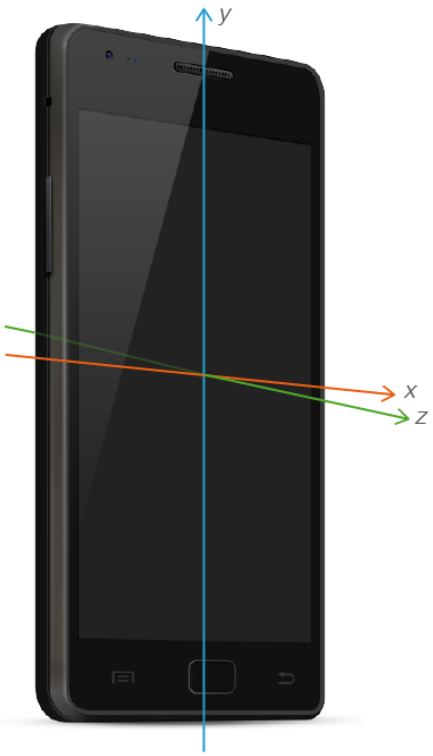
Com que controlarem els moviments en *4-axis* haurem de comprovar quin ha estat l'eix que més s'ha deplaçat amb la condició `Math.abs(finalX - iniciX) > Math.abs(finalY - iniciY)`, és a dir, mirem quina és la diferència més gran entre els punts inicials i finals a l'eix de les X i l'eix de les Y. Només ens quedarà saber quina ha estat la direcció comparant els punts final i inicial de l'eix.

1.2.2 Acceleròmetre

L'acceleròmetre és uns dels sensors que més informació ens dóna sobre l'usuari. Ens permet conèixer en tot moment en quina posició es troba el mòbil, quins

desplaçaments es produeixen i a quina velocitat. Aquest sensor és molt utilitzat als jocs (per controlar els girs o desplaçaments dels personatges) però també és utilitzat per a accions tan bàsiques com la de girar la pantalla quan l'usuari gira el telèfon. Podem controlar els canvis que es produeixen als eixos X, Y i Z tal com es pot veure a la figura 2.6

FIGURA 1.4. Eixos X, Y i Z en un dispositiu mòbil



A la plataforma Android, els sensors s'han d'enregistrar per poder-se fer servir i els hem d'alliberar quan no siguin necessaris (per evitar consums excessius de bateria). Els mètodes `onPause()` i `onResume()` són els més adequats per realitzar aquestes tasques: quan s'executa l'`onPause()` deixem de seguir els canvis del sensor i continuem llegint-los en tornar a l'aplicació (`onResume()`).

Podeu descarregar el projecte de la següent aplicació a l'apartat "MotionSensor" dels annexos.

Per tal d'introduir els conceptes bàsics de la gestió de l'acceleròmetre desenvoluparem una aplicació que ens mostrarà en tot moment l'estat dels eixos. Creeu un projecte anomenat **MotionSensor** amb el *Company Domain* **cat.xtec.ioc** i creeu una activitat buida deixant els valors per defecte.

El *layout* de la nostra aplicació estarà format per sis *TextView*, tres seran les etiquetes "X", "Y" i "Z" i els altres tres seran els valors dels sensors en cada moment.

El codi XML podria ser:

```
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"
```

```
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin" tools:context=".
        MainActivity">

    <TextView android:text="X: " android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textX" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:id="@+id/valueX"
        android:layout_alignParentTop="true"
        android:layout_toRightOf="@+id/textX"
        android:layout_toEndOf="@+id/textX" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:text="Y: "
        android:id="@+id/textY"
        android:layout_below="@+id/textX"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:id="@+id/valueY"
        android:layout_below="@+id/valueX"
        android:layout_alignLeft="@+id/valueX"
        android:layout_alignStart="@+id/valueX" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:text="Z: "
        android:id="@+id/textZ"
        android:layout_below="@+id/textY"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:id="@+id/valueZ"
        android:layout_below="@+id/textY"
        android:layout_toRightOf="@+id/textY"
        android:layout_toEndOf="@+id/textY" />

</RelativeLayout>
```

Pel que fa al codi de l'aplicació, el primer que hem de fer per controlar els canvis al sensor és implementar la classe `SensorEventListener` afegint a la capçalera:

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

Una vegada hem afegit l'`implements`, l'Android Studio ens informarà d'un error que solucionarem fent clic a *Implement methods*. Al final de la classe tindrem els mètodes:

```
@Override
    public void onSensorChanged(SensorEvent sensorEvent) {

    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int i) {

    }
```

Aquests mètodes seran cridats quan es produeixin canvis als valors dels sensors (`onSensorChanged`) o en la seva precisió (`onAccuracyChanged`).

En el nostre cas ens interessen els valors dels sensors, ho sigui que haurem d'afegir el codi corresponent per recollir els canvis produïts. Prepararem, però, abans les variables i inicialitzarem el sensor.

Afegirem les següents declaracions de variables just entre la declaració de la classe i el mètode `onCreate`:

```
TextView valueX, valueY, valueZ;
    SensorManager sensorMgr;
    Sensor sensor;
```

per després inicialitzar-les al mètode `onCreate`, els tres *TextView* ens permetran representar per pantalla l'estat dels sensors, el *SensorManager* ens permetrà assignar i enregistrar l'acceleròmetre, al qual podrem accedir a través de la variable *Sensor*.

Així, el contingut del mètode quedarà de la següent manera:

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

        valueX = (TextView) findViewById(R.id.valueX);
        valueY = (TextView) findViewById(R.id.valueY);
        valueZ = (TextView) findViewById(R.id.valueZ);

        sensorMgr = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        sensor = sensorMgr.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        sensorMgr.registerListener(this, sensor , SensorManager.
            SENSOR_DELAY_NORMAL);

    }
```

Per tal de deixar de controlar l'activitat dels sensors quan no estem a l'aplicació i evitar així els problemes de bateria descrits anteriorment, hem d'afegir el següent codi a la nostra activitat:

```
@Override
    protected void onPause() {
        super.onPause();
        sensorMgr.unregisterListener(this);
    }

    @Override
    protected void onResume() {
        super.onResume();
```

```
10          sensorMgr.registerListener(this,sensor, SensorManager.
                SENSOR_DELAY_NORMAL);
11      }
```

Quan la nostra activitat no estigui en primer pla (`onPause()`) deixem de seguir els canvis produïts al sensor (`sensorMgr.unregisterListener(this);`) i el tornem a enregistrar quan l'usuari torna a l'aplicació (`onResume()`).

Ja sols ens quedarà escriure el codi del mètode `onSensorChanged(SensorEvent sensorEvent)`, que actualitzarà els *TextView* amb la informació que ens proporciona el sensor.

El paràmetre `sensorEvent` ens permetrà accedir al sensor, la precisió d'aquest, els valors, etc. Així, el primer que comprovarem és si la informació que estem rebent és la del sensor correcte amb el condicional:

```
1   if(sensorEvent.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
```

És a dir, accedim al sensor i fem la crida al mètode `getType()`, si és l'acceleròmetre, llavors actualitzarem els valors dels *TextView*. Per llegir els valors hem de recollir la variable *values*, un *array* que conté la informació de l'eix de les "X", de les "Y" i de les "Z" a les posicions 0, 1 i 2 respectivament. Així, si consultem el `values[0]` obtindrem el valor de la "X".

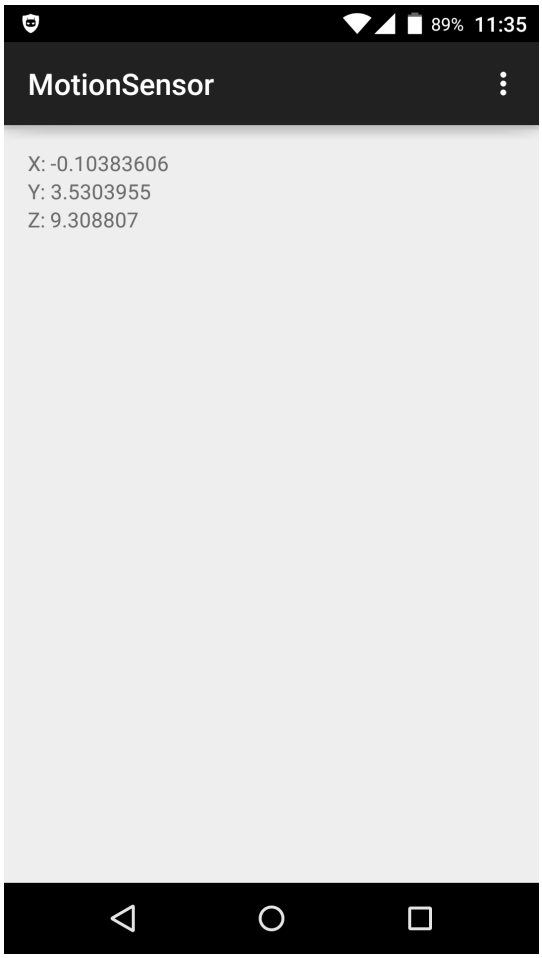
Una vegada comprovat que el sensor és el correcte, assignem als *TextView* els valors dels 3 eixos, passant-los prèviament a *string*.

```
1   @Override
2       public void onSensorChanged(SensorEvent sensorEvent) {
3
4           if(sensorEvent.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
5
6               valueX.setText(String.valueOf(sensorEvent.values[0]));
7               valueY.setText(String.valueOf(sensorEvent.values[1]));
8               valueZ.setText(String.valueOf(sensorEvent.values[2]));
9
10          }
11
12      }
```

Per tal de provar aquesta aplicació necessitarem fer-ho en un telèfon real ja que l'emulador d'Android no disposa d'aquesta opció. En cas de no tenir un dispositiu on provar-la, podeu mirar emuladors alternatius a l'oficial o instal·lar SensorSimulator i configurar-lo perquè treballi amb l'emulador.

El resultat serà el que podeu veure a la figura 2.7. Proveu a girar el telèfon en els distints eixos i observeu els canvis que es produeixen en els valors.

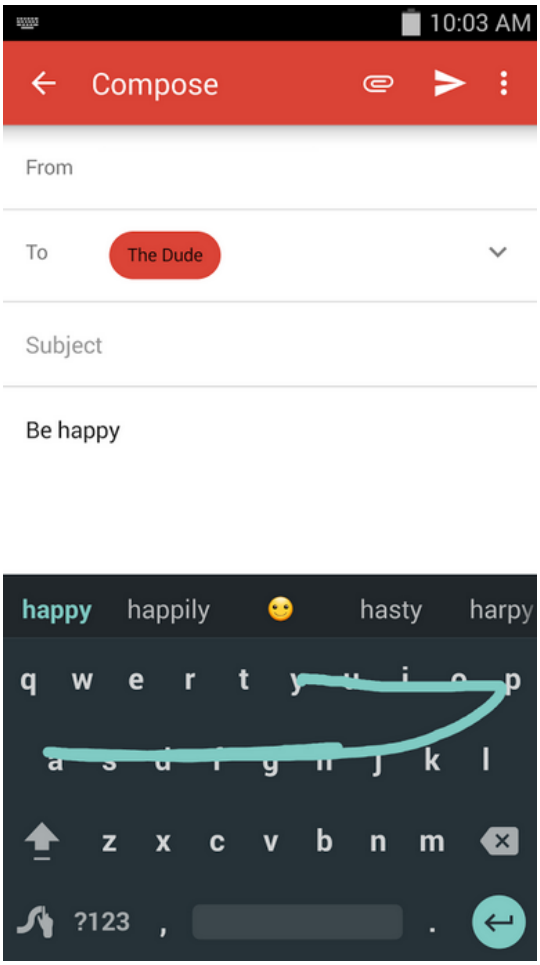
FIGURA 1.5. Projecte MotionSensor en funcionament



1.2.3 Teclat / Botons

Cada vegada és menys freqüent trobar dispositius mòbils amb teclat físic, els teclats virtuals (com per exemple el que veiem a la figura 2.8) han millorat molt pel que fa a entrada de text i prediccions i han substituït pràcticament els teclats físics, que disposen de limitacions quant a funcionalitats i no es poden adaptar al context de les aplicacions tal com fan els teclats en pantalla. No obstant això, no hem de deixar de banda la gestió dels *events* de tecles, a banda del reduït però present sector de dispositius amb teclat incorporat, hem de tenir en compte aquells portàtils amb sistema operatiu Android i més pensant en l'actual tendència de convergència entre sistemes operatius d'escriptori i sistemes operatius pensats per a dispositius mòbils.

FIGURA 1.6. Swype: exemple de teclat virtual en Android



Si volem gestionar aquest tipus d'*events*, de la mateixa manera que amb els de la pantalla tàctil, ho podem fer de dues maneres: gestionant qualsevol *event* que es produeixi a la nostra activitat o controlant aquells *events* dels *views* que nosaltres desitgem.

Gestió des de l'activitat

Per tal de gestionar els *events* de l'activitat, disposem de les funcions:

- *onKeyDown*: cridat en el moment de polsar la tecla.
- *onKeyUp*: cridat en finalitzar un *onKeyDown*, és a dir, quan deixem de pressionar la tecla.
- *onKeyLongPress*: s'executa quan deixem pressionada una tecla.
- *onKeyMultiple*: quan la mateixa tecla produeix diversos *events down* i *up* en un temps reduït.

Un exemple per controlar quan l'usuari prem la tecla de baixar volum seria el següent:

```
@Override
    public boolean onKeyUp(int keyCode, KeyEvent event) {

        if(keyCode == KeyEvent.KEYCODE_VOLUME_DOWN) {

            // Codi de baixar volum
            Toast.makeText(this, "Baixem volum!", Toast.LENGTH_SHORT).show();
            return true;

        }

        return super.onKeyUp(keyCode, event);
    }
```

El mètode `onKeyUp` s'executarà quan deixem de prémer la tecla de volum i rep com a paràmetres:

- `int keyCode`: enter que fa referència a constants definides a la classe KeyEvent. Ens indica quina tecla ha estat premuda.
- `KeyEvent event`: informació sobre l'*event* generat.

A l'exemple anterior comprovem que la tecla que s'ha pressionat és la corresponent a la variable KeyEvent.KEYCODE_VOLUME_DOWN, en cas de ser així controlem l'*event* i hem de retornar `true` en cas de voler gestionar-lo o `false` en cas de voler que sigui gestionat pel següent receptor.

Teniu una llista completa de les constants a http://developer.android.com/reference/android/view/KeyEvent.html així com també els mètodes i funcions disponibles de la classe `KeyEvent`.

Gestió d'un view en concret

Si la gestió dels *events* de tecles l'hem de fer des d'un *view* concret haurem de fer el corresponent *implements* de la classe:

```
public class MainActivity extends ActionBarActivity implements View.
    OnKeyListener {
```

Com sempre ens donarà error i haurem de seleccionar el suggeriment de l'Android Studio i escollir l'opció `Implement methods` que ens afegirà el codi del mètode `onKey(View view, int i, KeyEvent keyEvent)`. Els paràmetres són els mateixos que a la funció `onKeyUp` però a més ens informa del *view* que ha generat l'*event* amb el paràmetre `View view`.

Recordeu que per tal de controlar els *events* d'un *view*, a banda de fer l'*implements* hem d'assignar-li el *listener* amb `vista.setOnKeyListener(this);` on vista serà el *view* corresponent.

1.3 Motors de jocs

Els motors de jocs són *frameworks* dissenyats per a la creació de videojocs, és a dir, proporcionen recursos i metodologies per tal de facilitar-ne la programació. Un motor de joc ens ha de proporcionar una estructura sobre la qual començar a programar i crear la base del nostre joc d'una manera ràpida i senzilla. Aquest tipus de programari acostuma a ser modular, de manera que podem afegir aquelles funcionalitats que necessitem o bé programar-ne de noves sense afectar a la resta de components utilitzats. Alguns dels mòduls que podem trobar són: Tractament de físiques, tractament de col·lisions, intel·ligència artificial, gestió de recursos, etc.

> En general, i sempre que sigui possible, es recomana la utilització d'un motor de joc, d'aquesta manera reduirem el temps de dedicació i els possibles errors de disseny que podria implicar crear un projecte des de l'inici.

El sector dels dispositius mòbils ha experimentat un fort creixement els darrers anys i els sector dels videojocs ha estat un dels més importants. Si analitzem els motors de jocs disponibles trobarem una gran varietat pel que fa a llicències, llenguatges de programació, funcionalitats, plataformes, etc. Fem un repàs d'alguns dels motors de jocs disponibles actualment:

- Unity: desenvolupament de jocs multiplataforma (Android, iOS, Linux, OSX, PS4, Xbox One, etc.). Disposa d'una versió personal i una professional de pagament.
- Unreal engine: motor multiplataforma molt popular de la companyia *Epic Games*. La primera versió del motor es va implementar als *FPS*: *Unreal* i *Unreal Tournament*. És gratuït però s'ha de pagar un *royaltie* d'un 5% en cas de comercialitzar el producte i obtenir beneficis.
- AndEngine: motor conegut en el món dels jocs 2D en Android. És *open source* i gratuït.
- LibGDX: ens permet crear jocs 2D i 3D multiplataforma per a: Windows, Linux, OSX, Android, iOS, BlackBerry i HTML5. És *open source*, gratuït i es programa amb el llenguage Java.
- Godot Game Engine: motor multiplataforma i *open source*, disposa d'un llenguatge propi molt similar a *python*.

Per a la creació del joc en aquesta unitat hem decidit escollir **LigGDX** per les seves característiques i per tenir com a base el llenguatge de programació Java.

2. Desenvolupament d'un joc

Crear un joc partint d'un projecte buit pot ser una tasca molt complexa i que suposarà un repte durant tot el seu desenvolupament. Un error de disseny pot implicar grans canvis fins i tot en etapes inicials del desenvolupament. Per evitar aquests errors i simplificar aquesta tasca (tant en temps com en complexitat) el més adequat és fer servir eines que ja existeixin i que ens garanteixin que els components principals del joc ja estan implementats i que no ens n'haurem de preocupar.

Una vegada ja tenim clar que hem de recórrer a eines de tercers, haurem d'escollir la més adequada. La gran comunitat que hi ha al darrere, la documentació disponible i les tecnologies utilitzades (Java per a la programació i varietat de *plugins* de tercers disponibles) han fet que el motor de jocs escollit per aquest mòdul sigui **LibGDX**.

2.1 LibGDX

LibGDX és un *framework open source* (llicència Apache 2.0) per desenvolupar jocs en 2D i en 3D que ens permetrà generar fàcilment els fitxers binaris per poder-se executar, entre altres, a plataformes com ara:

- Android
- OSX i iOS
- Windows
- GNU/Linux
- HTML5
- BlackBerry

Disposarem, per tant, d'una part de programació comuna a totes les plataformes i una part específica en què configurarem la nostra aplicació per tal que s'executi correctament a la plataforma destí. A la pàgina oficial: http://libgdx.badlogicgames.com/ trobarem les novetats (*changelogs*) de cada versió, les característiques principals del *framework*, els *plugins* disponibles, etc. I especialment, com a programadors, ens interessaran les seccions de documentació i de comunitat, on podrem resoldre la major part dels nostres dubtes.

Estem davant d'un motor de jocs molt complet, que té una gran comunitat al darrere i que ens permetrà realitzar jocs de tot tipus amb resultats professionals sense haver de realitzar una despesa econòmica per utilitzar-lo.

Per seguir les explicacions, i per tal de tenir un exemple pràctic dels conceptes, desenvoluparem un joc de naus espacials: l'**SpaceRace**, on prendrem el rol del capità d'una nau espacial que haurem de dirigir per tal d'evitar col·lidir amb els asteroides que ens trobarem pel camí.

2.2 Començant amb LibGDX

LibGDX compta amb un assistent de configuració de projectes en el qual definirem quines propietats tindrà el nostre projecte. Aquest assistent s'encarregarà de descarregar tot allò necessari i de generar els fitxers que ens permetran importar el projecte al nostre IDE.

Per descarregar l'assistent, accedirem a la secció *Download* de la pàgina oficial: http://libgdx.badlogicgames.com/download.html. El fitxer descarregat serà el **gdx-setup.jar**.

Per executar-lo haurem d'obrir una terminal i executar la comanda `java -jar /ruta/al/fitxer/gdx-setup.jar`, per exemple:

```
1 java –jar ~/Descargas/gdx–setup.jar
```

Se'ns mostrarà una imatge com la que podem veure a la figura 2.1.

FIGURA 2.1. Assitent de projecte de LibGDX

L'assistent ens demana els següents atributs:

- *Name*: nom del projecte, en el nostre cas **spacerace**.
- *Package*: l'espai de noms de la nostra aplicació, per exemple **cat.xtec.ioc**.
- *Game class*: classe principal, serà del tipus `ApplicationListener`. La nostra classe s'anomenarà **SpaceRace**.
- *Destination*: directori al nostre sistema de fitxers on generarà el projecte.
- *Android SDK*: indiquem la ruta a l'SDK en cas que l'Android Studio no el tingui disponible.
- *LibGDX Version*: versió de libGDX, deixeu la proposada.
- *Sub projects*: indiquem les plataformes sobre les quals podrem executar el projecte i que necessiten configuració addicional. Deixem marcades **Desktop** i **Android**.
- *Extensions*: afegits al *framework* per a la gestió de col·lisions, intel·ligència artificial, fonts, etc. Podem deixar marcat **Box2d** (llibreria de físiques 2D) tal com ens proposa.
- *Third party Extensions*: extensions externes a LibGDX. No farem servir cap d'aquestes extensions.
- *Advanced*: opcions avançades. Podrem afegir *mirrors* per a les descàrregues, generació de fitxers per a IDEA o Eclipse o bé el mode *offline*. Ens serà molt útil marcar IDEA ja que l'Android Studio està basat en aquest IDE.

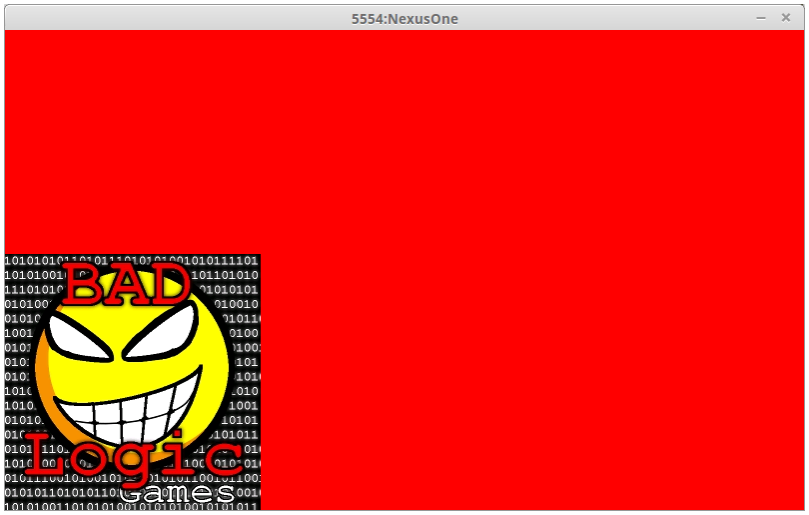
Una vegada emplenats tots els atributs, farem clic a *Generate* i començarà la descàrrega i configuració dels fitxers necessaris, si ens pregunta si volem continuar amb versions més recents de les *android build tools* o de les *Android API*, responeu que sí. Esperarem fins que ens surti el missatge: **BUILD SUCCESSFUL** i la informació sobre com importar el projecte. Podeu veure a la figura 2.2 l'assistent amb la configuració i el resultat final.

FIGURA 2.2. Assistent amb la configuració

Obriu l'Android Studio, feu clic a *File/Open* i seleccioneu el projecte a la ruta indicada a l'assistent. Una vegada obert, si feu clic a la pestanya *Project* de l'esquerra, podreu observar tres subprojectes: *android*, *core* i *desktop*; *android* i *desktop* ens serviran per configurar i personalitzar l'aplicació per a cadascuna de les dues plataformes i a *core* definirem els fitxers comuns, tota la implementació del joc.

Si executeu el projecte en un emulador, obtindreu el resultat que es mostra a la figura 2.3.

FIGURA 2.3. Aplicació executant-se a l'emulador



El logotip de *Bad Logic Games* sobre un fons vermell. Si obrim el fitxer SpaceRace.java del subprojecte *core* trobarem el següent codi:

```
public class SpaceRace extends ApplicationAdapter {
  SpriteBatch batch;
  Texture img;

  @Override
  public void create () {
    batch = new SpriteBatch();
    img = new Texture("badlogic.jpg");
  }

  @Override
  public void render () {
    Gdx.gl.glClearColor(1, 0, 0, 1);
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
    batch.begin();
    batch.draw(img, 0, 0);
    batch.end();
  }
}
```

L'`SpriteBatch` és l'encarregat de dibuixar objectes en 2D: genera un lot de comandes gràfiques i les optimitza per enviar-les a la GPU. El nostre `SpriteBatch` s'inicialitza al mètode `create()` i dibuixarà tot allò que estigui entre els mètodes `begin()` i `end()`, en aquest cas dibuixarà la imatge `img` i ho farà a la coordenada (0,0). A l'exemple la coordenada (0,0) representa la part inferior esquerra de la pantalla (sistema de coordenades *Y-Up*) però compte perquè no sempre és així, habitualment i al nostre joc, la coordenada (0,0) farà referència a la cantonada superior esquerra (sistema *Y-Down*).

El codi:

```
Gdx.gl.glClearColor(1, 0, 0, 1);
Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
```

Definirà el color del fons.

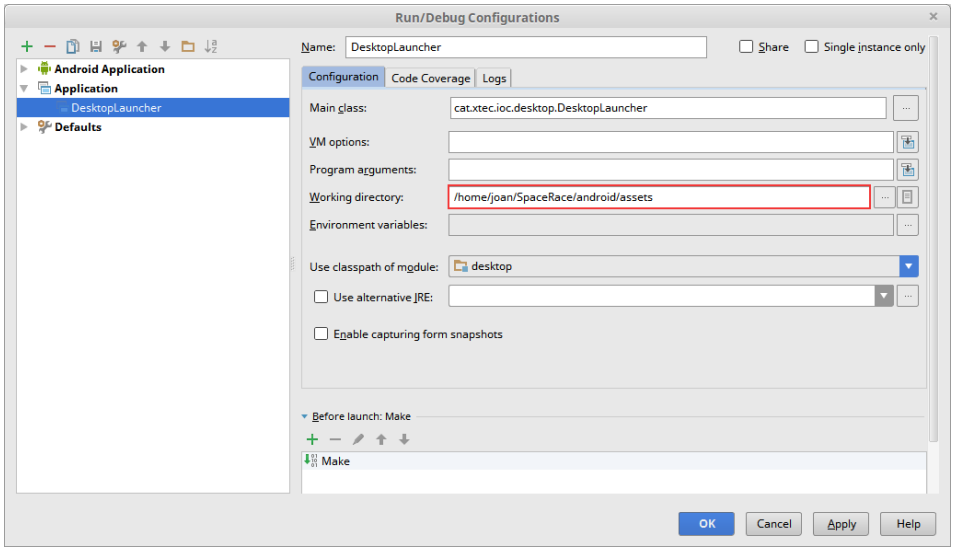
El mètode `glClearColor()` rep 4 paràmetres (del tipus `float` del 0.0 a l'1.0) que fan referència a la quantitat de vermell, de verd, de blau i a l'opacitat respectivament. Un valor 0.0 a l'opacitat indica que l'element és transparent i un 1.0 que és completament opac.

Al codi anterior, carrega el color vermell al *buffer* i el mètode `glClear(GL20.GL_COLOR_BUFFER_BIT)` indica que el *buffer* està habilitat per escriure colors.

El fitxer **badlogic.jpg** el podreu trobar al directori *assets* del subprojecte *android*, en aquest directori deixarem tots els fitxers gràfics que farem servir a l'aplicació i els enllaçarem des de la resta de plataformes. Si fem clic al botó dret sobre el fitxer *DesktopLauncher* del subprojecte *desktop* i seleccionem *Run DesktopLauncher main()* ens saltarà la següent excepció: `Exception in thread "LWJGL Application" com.badlogic.gdx.utils.GdxRuntimeException: Couldn't load file: badlogic.jpg`. Per solucionar-ho, seleccionem l'entrada del menú

Run/Edit Configurations..., seleccionem *Application/DesktopLauncher* i canviem el *working directori* a la ruta d'*assets* d'android (figura 2.4).

FIGURA 2.4. Configuració DesktopLauncher



Si torneu a executar el projecte se us obrirà una finestra amb el mateix contingut que abans: el logotip sobre un fons vermell. Ens serà molt útil poder treballar únicament amb l'ordinador sense haver de tenir un dispositiu físic sempre connectat o un emulador obert. No obstant això, haurem de provar la nostra aplicació cada cert temps a la plataforma Android per assegurar-nos que tot el que anem fent és compatible.

2.3 Desenvolupament del joc

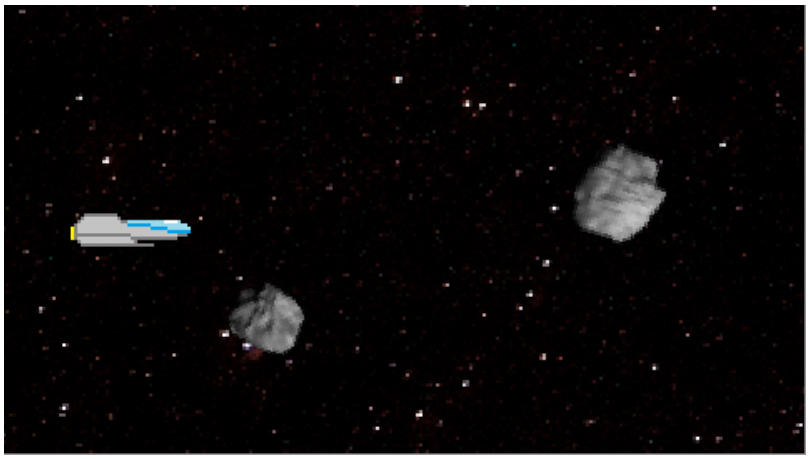
Una vegada tenim el projecte configurat i podent-se executar a l'ordinador, haurem de començar a desenvolupar el nostre joc. Començarem per les classes principals i afegirem els gràfics, els sons i la lògica de tots els components del joc.

Podeu descarregar els recursos de l'aplicació des dels annexos.

2.3.1 Classes del joc

El nostre joc estarà compost per diverses classes, cadascuna de les quals tindrà una funció específica. Cal, doncs, fer un repàs a cada classe del joc i anar-les implementant a poc a poc. A la figura 2.5 teniu una imatge del joc resultant que servirà d'ajuda per a la comprensió de les classes necessàries.

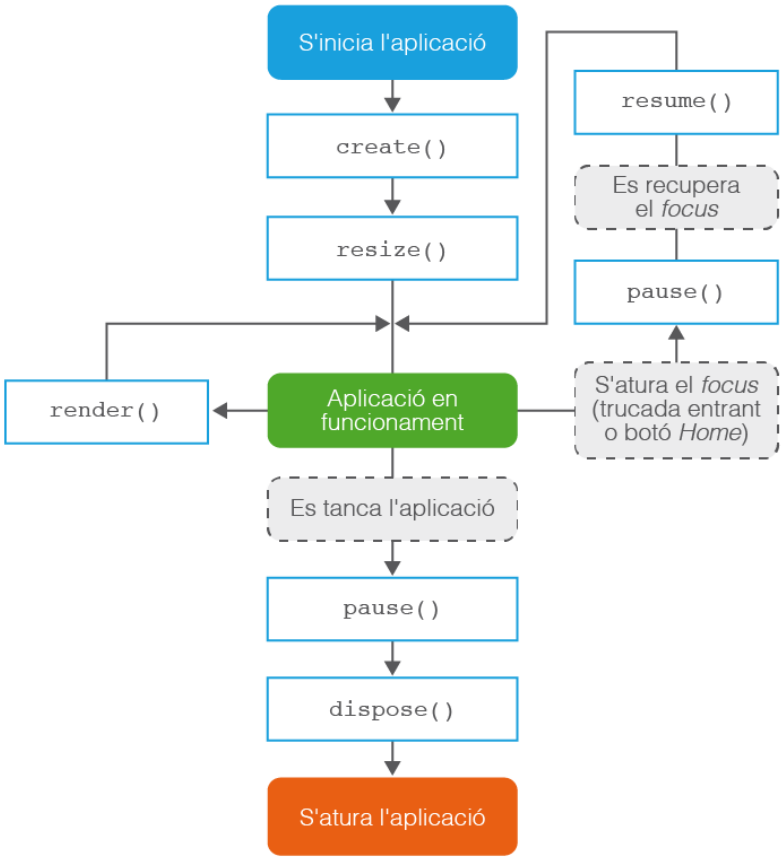
Figura 2.5. Joc resultant



- **SpaceRace** *(cat.xtec.ioc)*

Aquesta classe serà l'encarregada de carregar els recursos i controlar les pantalles del joc. Estendrà la classe `Game` i farem servir el mètode `setScreen(Screen screen)` per afegir la pantalla principal. Els `ApplicationListener`, que implementa la classe `Game`, tenen un cicle de vida similar al de les activitats d'Android com podem veure a la figura 2.6.

Figura 2.6. Cicle de vida d'ApplicationListener



Els mètodes disponibles són:

TAULA 2.1.

Mètode	Descripció
create()	Primer mètode que s'executa quan l'aplicació ha estat creada
resize(int width, int height)	Es crida una vegada després del mètode `create()` i cada vegada que es canvia la mida de l'aplicació i el joc no està a l'estat de pausa. Els paràmetres que rep són les noves dimensions de la pantalla
pause()	Quan l'aplicació perd el primer pla. S'executa també abans del mètode `dispose()`. Aquí haurem de guardar l'estat del joc
resume()	Quan l'aplicació torna a primer pla
render()	Mètode que es crida des del bucle principal de l'aplicació quan s'ha de representar la pantalla. Les actualitzacions de la lògica del joc s'habituen a fer en aquest mètode
dispose()	Darrer mètode que s'executa abans de tancar l'aplicació

- **GameScreen** *(cat.xtec.ioc.screens)*

Serà la pantalla principal del joc, encarregada de definir el gestor d'*events* d'entrada i d'actualitzar els components principals del joc: l'*stage* i els *actors*. Té un cicle de vida similar a la interfície `AplicationListener`, però a més afegeix els mètodes `show()` i `hide()`, quan una pantalla és afegida amb el mètode `setScreen()` i quan és canviada per una altra pantalla respectivament.

- **AssetManager** *(cat.xtec.ioc.helpers)*

En aquesta classe definirem tots els recursos bé siguin gràfics o sonors de la nostra aplicació.

- **InputHandler** *(cat.xtec.ioc.helpers)*

Gestió dels *events* de l'usuari, hereta d'`InputProcessor`. Podrem controlar els *events*: *keyDown*, *keyUp*, *keyTyped*, *touchDown*, *touchUp*, *touchDragged*, *mouseMoved* i *scrolled*, aquest dos últims exclusivament per les aplicacions d'escriptori i que fan referència al moviment del punter i de la roda de desplaçament respectivament.

- **Spacecraft** *(cat.xtec.ioc.objects)*

La nau del joc, serà un `Actor` del qual controlem la seva direcció segons els *events* rebuts per l'`InputHandler`.

- **Scrollable** *(cat.xtec.ioc.objects)*

Hereta de la classe `Actor`. Representa elements que disposen de desplaçament horitzontal al joc, en cada actualització els mourem segons la velocitat indicada. El fons de pantalla i els asteroides seran subclasses seva.

- **Asteroid** *(cat.xtec.ioc.objects)*

Classe que hereta d'`Scrollable`. Hi necessitarem controlar les possibles col·lisions amb la nau.

- **Background** *(cat.xtec.ioc.objects)*

Classe que hereta d'`Scrollable`, no té cap propietat ni mètode especial a banda de sobreescriure el mètode `draw`.

- **ScrollHandler** *(cat.xtec.ioc.objects)*

Classe encarregada de crear i actualitzar tots els elements que són `Scrollable`.

Ara que ja tenim una idea bàsica de les classes, cal veure quines són les implementacions per cada classe.

2.3.2 Implementació inicial

Començarem editant la classe `SpaceRace` que és l'única classe que tenim disponible en aquest moment. El primer que farem és modificar-ne la definició i fer que hereti de `Game` i no de `ApplicationAdapter`. Així, la definició de la classe quedarà de la següent manera:

```
1  public class SpaceRace extends Game {
```

Eliminem tot el contingut de `create()` i de `render()` així com les variables d'instància `batch` i `img`.

En aquesta classe iniciarem tots els recursos de l'aplicació i posarem la pantalla principal de l'aplicació. Abans de res, comprovarem el cicle de vida de l'aplicació amb el següent codi:

```
1  package cat.xtec.ioc;
2
3  import com.badlogic.gdx.Game;
4  import com.badlogic.gdx.Gdx;
5
6  public class SpaceRace extends Game {
7
8      @Override
9      public void create() {
10         Gdx.app.log("LifeCycle", "create()");
11     }
12
13     @Override
```

```
14      public void resize(int width, int height) {
15          super.resize(width, height);
16          Gdx.app.log("LifeCycle", "resize(" + Integer.toString(width) + ", " +
                Integer.toString(height) + ")");
17      }
18
19      @Override
20      public void pause() {
21          super.pause();
22          Gdx.app.log("LifeCycle", "pause()");
23      }
24
25      @Override
26      public void resume() {
27          super.resume();
28          Gdx.app.log("LifeCycle", "resume()");
29      }
30
31      @Override
32      public void render() {
33          super.render();
34          //Gdx.app.log("LifeCycle", "render()");
35      }
36
37      @Override
38      public void dispose() {
39          super.dispose();
40          Gdx.app.log("LifeCycle", "dispose()");
41
42      }
43  }
```

`Gdx.app.log(String tag, String msg)` és l'equivalent al `Log.d(String tag, String msg)` que estem acostumats a fer servir. Si executem l'aplicació i la tanquem obtindrem el següent resultat:

```
1  LifeCycle: create()
2  LifeCycle: resize(640, 480)
3  LifeCycle: pause()
4  LifeCycle: dispose()
```

Si descomenteu la línia `//Gdx.app.log(``LifeCycle'', ``render()'');` veureu com aquest mètode es crida contínuament entre els mètodes `resize()` i `pause()`. Modifiqueu la classe per tal que tingui el següent codi:

```
1  package cat.xtec.ioc;
2
3  import com.badlogic.gdx.Game;
4
5  import cat.xtec.ioc.helpers.AssetManager;
6  import cat.xtec.ioc.screens.GameScreen;
7
8  public class SpaceRace extends Game {
9
10     @Override
11     public void create() {
12
13         // A l'iniciar el joc carreguem els recursos
14         AssetManager.load();
15         // I definim la pantalla principal com a la pantalla
16         setScreen(new GameScreen());
17
18     }
19
20     // Cridem per descartar els recursos carregats.
21     @Override
```

```
22      public void dispose() {
23          super.dispose();
24          AssetManager.dispose();
25      }
26  }
```

En aquest moment l'aplicació ens donarà error ja que no tenim disponibles les classes `AssetManager` ni `GameScreen`.

Crearem la classe `GameScreen` però abans, crearem el paquet `screens` que emmagatzemarà les pantalles del joc: fem clic al botó dret sobre el paquet *cat.xtec.ioc* i fem clic a *New/Package* i posem de nom **screens**. Fem ara clic en el nou paquet i creem una nova classe que anomenarem **GameScreen**, que serà la pantalla principal del joc. Aquesta classe implementarà la interfície `com.badlogic.gdx.Screen`, afegiu l'`implements Screen` a la definició de la classe i solucioneu l'error que ens diu l'Android Studio amb *Implement methods*, així afegirem els mètodes necessaris per implementar la classe *Screen*. La classe quedarà de la següent manera:

```
1   package cat.xtec.ioc.screens;
2
3   import com.badlogic.gdx.Screen;
4
5   public class GameScreen implements Screen {
6
7
8       @Override
9       public void show() {
10
11      }
12
13      @Override
14      public void render(float delta) {
15
16      }
17
18      @Override
19      public void resize(int width, int height) {
20
21      }
22
23      @Override
24      public void pause() {
25
26      }
27
28      @Override
29      public void resume() {
30
31      }
32
33      @Override
34      public void hide() {
35
36      }
37
38      @Override
39      public void dispose() {
40
41      }
42
43  }
```

Hem creat una de les dues classes que necessitàvem, creem el *package* `cat.xtec.ioc.helpers` i creem la classe `AssetManager` per solucionar el segon error.

Necessitem afegir dos mètodes `static` a la classe: `load()` i `dispose()` obtenint com a resultat la següent classe:

```
1  package cat.xtec.ioc.helpers;
2
3
4  public class AssetManager {
5
6      public static void load() {
7      }
8
9      public static void dispose() {
10     }
11
12 }
```

I amb això ja serem capaços d'executar la nostra aplicació, si afegim a més les línies:

```
1  Gdx.gl.glClearColor(1, 0, 0, 1);
2  Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
```

El mètode `render()` s'executarà contínuament, sent un bon candidat per ser el bucle principal de l'aplicació.

Al mètode `render(float delta)` de la classe `GameScreen`, tindrem com a resultat una pantalla amb fons vermell.

Deixarem de banda la classe `AssetManager` fins que tractem els gràfics i els sons de l'aplicació i ens centrarem en la classe `GameScreen`.

> Un **stage** és una escena 2D que conté `Actors` i que gestiona els *events* d'entrada i *viewports* (que veurem a la secció **Gràfics i sons**).
> Un **actor** és un node de l'`stage` que té propietats i sobre el qual es poden realitzar accions.

`GameScreen` és la classe que s'encarregarà de crear l'*stage* i els actors, és a dir, l'encarregada de controlar els elements principals del joc. La nostra classe haurà de tenir les següents variables d'instància:

```
1  private Stage stage;
2      private Spacecraft spacecraft;
3      private ScrollHandler scrollHandler;
```

Les classes `Spacecraft` i `ScrollHandler` les crearem més endavant. Els tres objectes els iniciarem al constructor de la pantalla:

```
1  public GameScreen() {
2
3          // Creem l'stage
4          stage = new Stage();
5
6          // Creem la nau i la resta d'objectes
7          spacecraft = new Spacecraft();
8          scrollHandler = new ScrollHandler();
9
```

```
10          // Afegim els actors a l'stage
11          stage.addActor(scrollHandler);
12          stage.addActor(spacecraft);
13
14      }
```

Més endavant necessitarem afegir paràmetres en la creació dels objectes però de moment ho deixarem així.

La classe `stage` disposa dels mètodes `draw()` i `act(float delta)` que cridarà als mètodes `draw()` i `act(float delta)` de tots els seus `actors`. Com que a la pantalla principal haurem de controlar l'actualització dels elements del joc, modificarem el mètode `render(float delta)` per tal que tingui el següent codi:

```
1  @Override
2      public void render(float delta) {
3
4          // Dibuixem i actualitzem tots els actors de l'stage
5          stage.draw();
6          stage.act(delta);
7
8      }
```

El valor **delta** marcarà la diferència en segons respecte l'anterior crida a `render`, fent servir aquest valor aconseguirem fer aplicacions amb moviments independents del *frame-rate*. Si fem la divisió **1/delta** obtindrem els *fps* (*frames per second)* de l'aplicació.

El valor `delta` el podem obtenir des de qualsevol lloc de l'aplicació amb `Gdx.graphics.getDeltaTime()`.

Hem de destacar el paràmetre `float delta`, que passarem al mètode `act` per tal que en actualitzar-se ho faci independentment del *frame-rate* i l'aplicació vagi a la mateixa "velocitat" independentment del dispositiu en què es faci servir.

Afegirem també a la classe els *getters* dels elements. Feu clic a *Code/-Generate.../Getter* i seleccioneu els tres elements, l'`stage`, l'`spacecraft` i l'`scrollhandler`, que donaran com a resultat el següent codi:

```
1  public Spacecraft getSpacecraft() {
2          return spacecraft;
3      }
4
5      public Stage getStage() {
6          return stage;
7      }
8
9      public ScrollHandler getScrollHandler() {
10         return scrollHandler;
11     }
```

Passarem ara a definir els elements del joc i després modificarem aquestes classes.

2.3.3 Elements del joc

Per tal de tenir organitzats els elements del joc, el primer que farem és crear el paquet cat.xtec.ioc.objects. Aquí haurem de crear les classes per a la nau espacial (`Spacecraft`), els asteroides (`Asteroid`) i el fons de pantalla (`Background`). Aquests dos últims tindran una classe pare comuna que anomenarem `Scrollable`.

Spacecraft

FIGURA 2.7. Spritesheet de la nau

Per a la nau espacial, necessitarem conèixer la posició que ocupa, les dimensions (alt i ample) i la direcció (si es troba "recta", pujant o baixant, tal com podem veure a la figura 2.7).

La nostra nau espacial serà un `Actor` de l'`stage`. Crearem la classe `Spacecraft` i modificarem la definició de la classe per tal que hereti d'`actor`:

```
1  public class Spacecraft extends Actor {
```

Afegirem a la classe les variables d'instància corresponents i crearem el constructor. Crearem a més tres constants de classe per controlar la direcció. La classe quedarà de la següent manera:

```
1  package cat.xtec.ioc.objects;
2
3  import com.badlogic.gdx.math.Vector2;
4  import com.badlogic.gdx.scenes.scene2d.Actor;
5
6  public class Spacecraft extends Actor {
7
8      // Diferents posicions de l'Spacecraft: recta, pujant i baixant
9      public static final int SPACECRAFT_STRAIGHT = 0;
10     public static final int SPACECRAFT_UP = 1;
11     public static final int SPACECRAFT_DOWN = 2;
12
13         // Paràmetres de l'Spacecraft
14     private Vector2 position;
15     private int width, height;
16     private int direction;
17
18     public Spacecraft(float x, float y, int width, int height) {
19
20         // Inicialitzem els arguments segons la crida del constructor
21         this.width = width;
22         this.height = height;
23         position = new Vector2(x, y);
24
25         // Inicialitzem l'Spacecraft a l'estat normal
26         direction = SPACECRAFT_STRAIGHT;
27
```

```
28      }
29  }
```

A la wiki oficial de LibGDX podeu trobar informació sobre la classe Vector2D

Afegim els *getters* de les propietats i afegim els mètodes `goUp()`, `goDown()` i `goStraight()` que canviaran la direcció depenent de si la nau puja, baixa o va recta.

I ens quedarà el mètode més important, el mètode `act(float delta)` que es cridarà quan la nau estigui com a `actor` en un `stage` i aquest faci una crida al seu mètode `act`. Aquest mètode haurà d'actualitzar la nau canviant la seva posició segons la direcció i la velocitat.

El nostre joc tindrà un sistema de coordenades *Y-Down*, que vol dir que la coordenada 0 de la Y serà a la part superior de la pantalla, una Y positiva anirà en direcció descendent i una Y negativa serà ascendent. Per moure la nau, ens trobarem en dos casos:

- Nau ascendent: la posició de la Y l'haurem de disminuir multiplicant la velocitat pel valor `delta` (per aconseguir un moviment independent del *frame-rate*), i ho farem sempre que la nova posició no sigui menor a 0.
- Nau descendent: la posició de la Y l'haurem d'incrementar multiplicant la velocitat pel valor `delta`, i ho farem sempre que la nova posició més l'alçada de la nau no sigui major que l'alçada de la pantalla.

Per evitar que la nau surti de la pantalla mentre realitza un moviment descendent necessitem conèixer-ne l'alçada. Com que serà un valor que consultarem força sovint i que volem tenir disponible per canviar d'una manera senzilla, crearem una classe que anomenarem *Settings* en la qual emmagatzemarem constants que ens seran útils des de distintes classes. Crearem un nou paquet Java de nom **cat.xtec.ioc.utils** i una classe **Settings** que contindrà el següent codi:

```
1  package cat.xtec.ioc.utils;
2
3  public class Settings {
4
5      // Mida del joc, s'escalarà segons la necessitat
6      public static final int GAME_WIDTH = 240;
7      public static final int GAME_HEIGHT = 135;
8
9      // Propietats de la nau
10     public static final float SPACECRAFT_VELOCITY = 50;
11     public static final int SPACECRAFT_WIDTH = 36;
12     public static final int SPACECRAFT_HEIGHT = 15;
13     public static final float SPACECRAFT_STARTX = 20;
14     public static final float SPACECRAFT_STARTY = GAME_HEIGHT/2 -
           SPACECRAFT_HEIGHT/2;
15
16 }
```

Hem afegit també les constants que serviran per definir els paràmetres de la nau: velocitat, amplada, alçada i punt inicial. A 20 píxels de la vora esquerra i centrada en l'eix vertical (alçada del joc menys la meitat de l'alçada de la nau).

El mètode `act(float delta)` es cridarà sempre que es faci una crida a `stage.act(delta)`.

El codi del mètode encarregat d'actualitzar la nau podria ser el següent:

```
public void act(float delta) {

        // Movem l'Spacecraft depenent de la direcció controlant que no surti
            de la pantalla
        switch (direction) {
            case SPACECRAFT_UP:
                if (this.position.y - Settings.SPACECRAFT_VELOCITY * delta >=
                    0) {
                    this.position.y -= Settings.SPACECRAFT_VELOCITY * delta;
                }
                break;
            case SPACECRAFT_DOWN:
                if (this.position.y + height + Settings.SPACECRAFT_VELOCITY *
                    delta <= Settings.GAME_HEIGHT) {
                    this.position.y += Settings.SPACECRAFT_VELOCITY * delta;
                }
                break;
            case SPACECRAFT_STRAIGHT:
                break;
        }
    }
```

Si el moviment és cap a dalt, controlem que la coordenada Y, si li apliquem l'increment de posició (restant la velocitat multiplicada pel valor `delta`), no sigui mai menor de 0. En el cas de moviment descendent, hem de controlar que la Y més l'alçada de la nau i sumant-li el nou increment de posició (velocitat per `delta`) no superi els límits de la pantalla, és a dir, el `GAME_HEIGHT`.

El codi complet d'`Spacecraft` serà el següent:

```
package cat.xtec.ioc.objects;

import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.scenes.scene2d.Actor;

import cat.xtec.ioc.utils.Settings;

public class Spacecraft extends Actor {

    // Distintes posicions de l'Spacecraft: recta, pujant i baixant
    public static final int SPACECRAFT_STRAIGHT = 0;
    public static final int SPACECRAFT_UP = 1;
    public static final int SPACECRAFT_DOWN = 2;

    // Paràmetres de l'Spacecraft
    private Vector2 position;
    private int width, height;
    private int direction;

    public Spacecraft(float x, float y, int width, int height) {

        // Inicialitzem els arguments segons la crida del constructor
        this.width = width;
        this.height = height;
        position = new Vector2(x, y);

        // Inicialitzem l'Spacecraft a l'estat normal
        direction = SPACECRAFT_STRAIGHT;

    }

    public void act(float delta) {

        // Movem l'Spacecraft depenent de la direcció controlant que no surti
            de la pantalla
        switch (direction) {
```

```
36              case SPACECRAFT_UP:
37                  if (this.position.y - Settings.SPACECRAFT_VELOCITY * delta >=
                        0) {
38                      this.position.y -= Settings.SPACECRAFT_VELOCITY * delta;
39                  }
40                  break;
41              case SPACECRAFT_DOWN:
42                  if (this.position.y + height + Settings.SPACECRAFT_VELOCITY *
                        delta <= Settings.GAME_HEIGHT) {
43                      this.position.y += Settings.SPACECRAFT_VELOCITY * delta;
44                  }
45                  break;
46              case SPACECRAFT_STRAIGHT:
47                  break;
48          }
49      }
50
51      // Getters dels atributs principals
52      public float getX() {
53          return position.x;
54      }
55
56      public float getY() {
57          return position.y;
58      }
59
60      public float getWidth() {
61          return width;
62      }
63
64      public float getHeight() {
65          return height;
66      }
67
68      // Canviem la direcció de l'Spacecraft: Puja
69      public void goUp() {
70          direction = SPACECRAFT_UP;
71      }
72
73      // Canviem la direcció de l'Spacecraft: Baixa
74      public void goDown() {
75          direction = SPACECRAFT_DOWN;
76      }
77
78      // Posem l'Spacecraft al seu estat original
79      public void goStraight() {
80          direction = SPACECRAFT_STRAIGHT;
81      }
82
83  }
```

A la classe `GameScreen` havíem creat un nou objecte `Spacecraft` sense passar-li cap paràmetre i el constructor que hem definit necessita quatre paràmetres: coordenada X, coordenada Y, amplada i alçada. Tornem a la classe `GameScreen` i modifiquem del constructor la línia `spacecraft = new Spacecraft()` per:

```
1  spacecraft = new Spacecraft(Settings.SPACECRAFT_STARTX, Settings.
       SPACECRAFT_STARTY, Settings.SPACECRAFT_WIDTH, Settings.SPACECRAFT_HEIGHT);
```

Scrollables

Crearem, ara, els objectes que ens falten: els asteroides i el fons de pantalla. Tots dos tenen coses en comú: són descendents d'`actor`, tenen un desplaçament lateral

i es van reutilitzant cada vegada que desapareixen pel costat esquerre (valor de X=0). Crearem al paquet `cat.xtec.ioc.objects` una classe pare anomenada **scrollable** i dos descendents d'aquesta: **Asteroid** i **Background**.

La classe `scrollable` heretarà d'`actor` i necessitarem:

- `Vector2 position`: vector de posició.
- `float velocity`: `float` amb la velocitat de l'element. Serà variable segons l'element.
- `float width, height`: amplada i alçada dels elements.
- `boolean leftOfScreen`: per saber si l'element està fora de la pantalla, quan sobrepassa el 0 de l'eix d'X.

Creem la classe amb els atributs, el constructor, els *getters*, la funció `getTailX()` que retornarà la posició de l'element més la seva amplada i la funció `reset (float newX)` que assignarà una nova posició a l'element i posarà a `false` el booleà que indica que ha sortit de la pantalla.

La classe `scrollable` quedarà així:

```
1  package cat.xtec.ioc.objects;
2
3  import com.badlogic.gdx.math.Vector2;
4  import com.badlogic.gdx.scenes.scene2d.Actor;
5
6  public class Scrollable extends Actor {
7
8      protected Vector2 position;
9      protected float velocity;
10     protected float width;
11     protected float height;
12     protected boolean leftOfScreen;
13
14     public Scrollable(float x, float y, float width, float height, float
           velocity) {
15         position = new Vector2(x, y);
16         this.velocity = velocity;
17         this.width = width;
18         this.height = height;
19         leftOfScreen = false;
20
21     }
22
23     public void reset(float newX) {
24         position.x = newX;
25         leftOfScreen = false;
26     }
27
28     public boolean isLeftOfScreen() {
29         return leftOfScreen;
30     }
31
32     public float getTailX() {
33         return position.x + width;
34     }
35
36     public float getX() {
37         return position.x;
38     }
39
```

```
    public float getY() {
        return position.y;
    }

    public float getWidth() {
        return width;
    }

    public float getHeight() {
        return height;
    }


}
```

Faltarà implementar el mètode principal: `act(float delta)`. Aquest mètode s'encarregarà de desplaçar l'element en l'eix d'X. Controlarem a més el booleà `leftOfScreen`, que posarem a `true` quan la posició en l'eix X més l'amplada de l'objecte sigui menor de 0.

Així, el mètode quedarà de la següent manera:

```
public void act(float delta) {

        // Desplacem l'objecte en l'eix d'X
        position.x += velocity * delta;

        // Si es troba fora de la pantalla canviem la variable a true
        if (position.x + width < 0) {
            leftOfScreen = true;
        }
    }
```

Les dues classes filles, `asteroid` i `background`, de moment les deixarem amb la implementació mínima, quedant de la següent manera:

Classe Asteroid:

```
package cat.xtec.ioc.objects;

public class Asteroid extends Scrollable {
    public Asteroid(float x, float y, float width, float height, float velocity
        ) {
        super(x, y, width, height, velocity);
    }
}
```

Classe Background:

```
package cat.xtec.ioc.objects;

public class Background extends Scrollable {
    public Background(float x, float y, float width, float height, float
        velocity) {
        super(x, y, width, height, velocity);
    }
}
```

Ens quedarà per últim definir la classe **scrollHandler** que servirà per controlar tots els objectes del tipus `scrollable`. En aquesta classe necessitarem un mètode que ens retornarà un `float` aleatori entre dos valors. Crearem una classe on guardarem

mètodes que puguin ser útils des de qualsevol classe del joc. Creeu al paquet `cat.xtec.ioc.utils` la classe `methods` i que tindrà com a contingut el següent:

```
package cat.xtec.ioc.utils;

import java.util.Random;

public class Methods {

    // Mètode que torna un float aleatori entre un mínim i un màxim
    public static float randomFloat(float min, float max) {
        Random r = new Random();
        return r.nextFloat() * (max - min) + min;

    }
}
```

Haurem d'afegir també les següents constants a la classe `Settings`:

```
// Rang de valors per canviar la mida de l'asteroide
    public static final float MAX_ASTEROID = 1.5f;
    public static final float MIN_ASTEROID = 0.5f;

    // Configuració scrollable
    public static final int ASTEROID_SPEED = -150;
    public static final int ASTEROID_GAP = 75;
    public static final int BG_SPEED = -100;
```

La classe `scrollHandler` crearà els `asteroid` (que s'aniran reutilitzant) i dos `background` i serà l'encarregada d'actualitzar-los. Aquesta classe heretarà de `group` (`com.badlogic.gdx.scenes.scene2d.Group`), que és un conjunt d'`actors`.

El codi inicial de la classe serà el següent:

```
package cat.xtec.ioc.objects;

import com.badlogic.gdx.scenes.scene2d.Group;

import java.util.ArrayList;
import java.util.Random;

public class ScrollHandler extends Group {

    // Fons de pantalla
    Background bg, bg_back;

    // Asteroides
    int numAsteroids;
    ArrayList<Asteroid> asteroids;

    // Objecte random
    Random r;

    public ScrollHandler() {
    }
}
```

Un **group** és un node d'una escena 2D que conté actors. Els actors es representaran segons l'ordre d'inserció al grup i igual que aquests, disposen d'un mètode `act` i un mètode `draw`.

Començarem creant els dos fons que s'aniran concatenant per donar aquesta sensació de fons infinit. El seu codi, que afegirem al constructor d'`scrollHandler`, serà el següent:

```
//Creem els dos fons
        bg = new Background(0, 0, Settings.GAME_WIDTH * 2, Settings.GAME_HEIGHT
             , Settings.BG_SPEED);
        bg_back = new Background(bg.getTailX(), 0, Settings.GAME_WIDTH * 2,
             Settings.GAME_HEIGHT, Settings.BG_SPEED);

        //Afegim els fons (actors) al grup
        addActor(bg);
        addActor(bg_back);
```

El primer fons començarà en la coordenada (0,0) i tindrà una amplada el doble que la mida de la pantalla i la mateixa alçada. També li passarem la velocitat en què es mourà. Pel que fa al segons fons, començarà a la cua del fons principal i amb la Y=0, les dimensions i la velocitat seran les mateixes. Una vegada creats els afegim al `group` amb el mètode `addActor(Actor actor)`.

Pels que fa als `asteroid`, tindran una amplada i alçada aleatòria que serà entre 0.5 i 1.5 vegades la seva mida original (en aquest cas 34 píxels). Així, guardarem a un `float` la nova mida i la farem servir per calcular les propietats. Necessitarem calcular les coordenades X i Y segons la nova mida:

- Coordenada X: en el cas del primer `asteroid` serà l'ample del joc i pels altres la posició de la cua (`getTailX()`) de l'anterior asteroide més una separació que definirem a `settings` (`ASTEROID_GAP`).
- Coordenada Y: serà un valor aleatori entre 0 i l'alçada total de la pantalla menys l'alçada de l'asteroide que hem calculat.

Crearem un `private ArrayList<Asteroid>` anomenat **asteroids** que contindrà els `asteroid` que anirem generant. Inicialment crearem 3 asteroides però ho deixarem preparat per tal que afegir-ne de nous no sigui complex. El codi següent l'afegirem després de la creació dels fons:

```
// Creem l'objecte random
        r = new Random();

        // Comencem amb 3 asteroides
        numAsteroids = 3;

        // Creem l'ArrayList
        asteroids = new ArrayList<Asteroid>();

        // Definim una mida aleatòria entre el mínim i el màxim
        float newSize = Methods.randomFloat(Settings.MIN_ASTEROID, Settings.
             MAX_ASTEROID) * 34;

        // Afegim el primer asteroide a l'array i al grup
        Asteroid asteroid = new Asteroid(Settings.GAME_WIDTH, r.nextInt(
             Settings.GAME_HEIGHT - (int) newSize), newSize, newSize, Settings.
             ASTEROID_SPEED);
        asteroids.add(asteroid);
        addActor(asteroid);

        // Des del segon fins l'últim asteroide
        for (int i = 1; i < numAsteroids; i++) {
```

```
            // Creem la mida aleatòria
            newSize = Methods.randomFloat(Settings.MIN_ASTEROID, Settings.
                MAX_ASTEROID) * 34;
            // Afegim l'asteroide
            asteroid = new Asteroid(asteroids.get(asteroids.size() - 1).
                getTailX() + Settings.ASTEROID_GAP, r.nextInt(Settings.
                GAME_HEIGHT - (int) newSize), newSize, newSize, Settings.
                ASTEROID_SPEED);
            // Afegim l'asteroide a l'ArrayList
            asteroids.add(asteroid);
            // Afegim l'asteroide al grup d'actors
            addActor(asteroid);
        }
```

Cadascun dels asteroides s'haurà d'afegir a l'`ArrayList` i també al `group`.

Cada `group` té un mètode `act(float delta)` comú per a tots els `actor`, al mètode `act` de l'`scrollHandler`. Comprovarem si algun dels seus `actors` ha deixat de ser visible per la pantalla i en cas de ser així, el reiniciarem assignat-li una nova posició: en el cas del fons li assignarem la posició de la cua de l'altre fons i en el cas dels asteroides agafarem la cua de l'anterior asteroide (en el cas de que sigui el primer asteroide agafarem la cua del darrer).

Per generar el codi del mètode `act` feu clic a *Code/Override Methods...* i escolliu el mètode `act(delta:float):void`. Una vegada generat, afegiu el següent codi per fer les comprovacions dels objectes:

```
@Override
    public void act(float delta) {
        super.act(delta);
        // Si algun element es troba fora de la pantalla, fem un reset de l'
            element
        if (bg.isLeftOfScreen()) {
            bg.reset(bg_back.getTailX());

        } else if (bg_back.isLeftOfScreen()) {
            bg_back.reset(bg.getTailX());

        }

        for (int i = 0; i < asteroids.size(); i++) {

            Asteroid asteroid = asteroids.get(i);
            if (asteroid.isLeftOfScreen()) {
                if (i == 0) {
                    asteroid.reset(asteroids.get(asteroids.size() - 1).getTailX
                        () + Settings.ASTEROID_GAP);
                } else {
                    asteroid.reset(asteroids.get(i - 1).getTailX() + Settings.
                        ASTEROID_GAP);
                }
            }
        }
    }
```

i finalment afegirem el *getter* per obtenir l'`ArrayList`, quedant la classe de la següent manera:

```
package cat.xtec.ioc.objects;

import com.badlogic.gdx.scenes.scene2d.Group;

import java.util.ArrayList;
```

```
6  import java.util.Random;
7
8  import cat.xtec.ioc.utils.Methods;
9  import cat.xtec.ioc.utils.Settings;
10
11 public class ScrollHandler extends Group {
12
13     // Fons de pantalla
14     Background bg, bg_back;
15
16     // Asteroides
17     int numAsteroids;
18     ArrayList<Asteroid> asteroids;
19
20     // Objecte Random
21     Random r;
22
23     public ScrollHandler() {
24
25         // Creem els dos fons
26         bg = new Background(0, 0, Settings.GAME_WIDTH * 2, Settings.GAME_HEIGHT
               , Settings.BG_SPEED);
27         bg_back = new Background(bg.getTailX(), 0, Settings.GAME_WIDTH * 2,
               Settings.GAME_HEIGHT, Settings.BG_SPEED);
28
29         // Afegim els fons al grup
30         addActor(bg);
31         addActor(bg_back);
32
33         // Creem l'objecte random
34         r = new Random();
35
36         // Comencem amb 3 asteroides
37         numAsteroids = 3;
38
39         // Creem l'ArrayList
40         asteroids = new ArrayList<Asteroid>();
41
42         // Definim una mida aleatòria entre el mínim i el màxim
43         float newSize = Methods.randomFloat(Settings.MIN_ASTEROID, Settings.
               MAX_ASTEROID) * 34;
44
45         // Afegim el primer asteroide a l'array i al grup
46         Asteroid asteroid = new Asteroid(Settings.GAME_WIDTH, r.nextInt(
               Settings.GAME_HEIGHT - (int) newSize), newSize, newSize, Settings.
               ASTEROID_SPEED);
47         asteroids.add(asteroid);
48         addActor(asteroid);
49
50         // Des del segon fins l'últim asteroide
51         for (int i = 1; i < numAsteroids; i++) {
52             // Creem la mida aleatòria
53             newSize = Methods.randomFloat(Settings.MIN_ASTEROID, Settings.
                   MAX_ASTEROID) * 34;
54             // Afegim l'asteroide
55             asteroid = new Asteroid(asteroids.get(asteroids.size() - 1).
                   getTailX() + Settings.ASTEROID_GAP, r.nextInt(Settings.
                   GAME_HEIGHT - (int) newSize), newSize, newSize, Settings.
                   ASTEROID_SPEED);
56             // Afegim l'asteroide a l'ArrayList
57             asteroids.add(asteroid);
58             // Afegim l'asteroide al grup d'actors
59             addActor(asteroid);
60         }
61
62     }
63
64     @Override
65     public void act(float delta) {
66         super.act(delta);
```

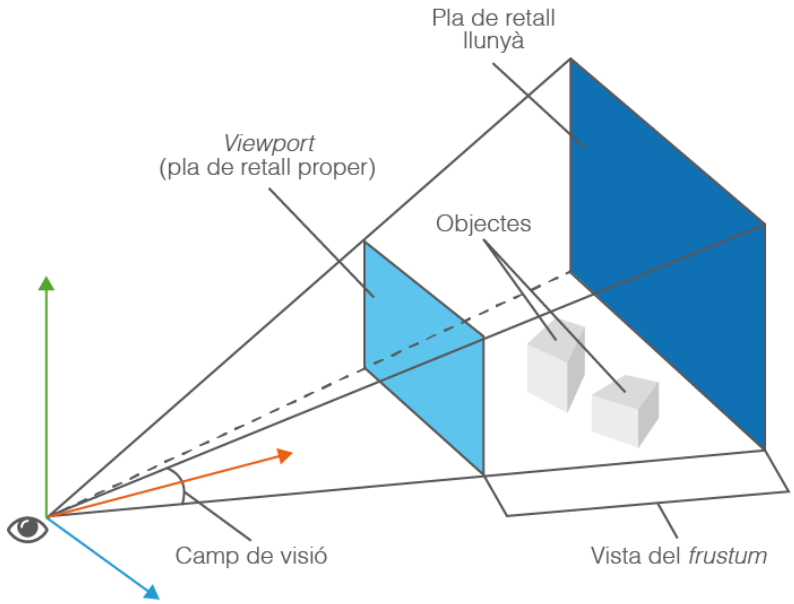
```
67          // Si algun element es troba fora de la pantalla, fem un reset de l'
                element
68          if (bg.isLeftOfScreen()) {
69              bg.reset(bg_back.getTailX());
70
71          } else if (bg_back.isLeftOfScreen()) {
72              bg_back.reset(bg.getTailX());
73
74          }
75
76          for (int i = 0; i < asteroids.size(); i++) {
77
78              Asteroid asteroid = asteroids.get(i);
79              if (asteroid.isLeftOfScreen()) {
80                  if (i == 0) {
81                      asteroid.reset(asteroids.get(asteroids.size() - 1).getTailX
                            () + Settings.ASTEROID_GAP);
82                  } else {
83                      asteroid.reset(asteroids.get(i - 1).getTailX() + Settings.
                            ASTEROID_GAP);
84                  }
85              }
86          }
87      }
88
89      public ArrayList<Asteroid> getAsteroids() {
90          return asteroids;
91      }
92  }
```

En aquest moment tenim tots els objectes preparats i si executem el projecte s'estaran actualitzant els valors tot i que no serem capaços de visualitzar-ho ja que no tenen cap representació gràfica.

2.3.4 Gràfics i sons

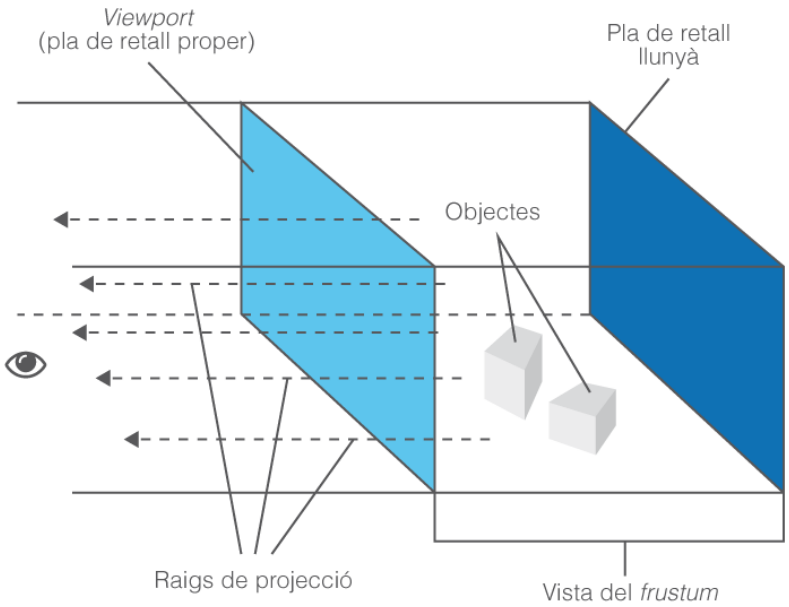
Totes les aplicacions que fan servir gràfics necessiten treballar amb la pantalla d'alguna manera, necessiten tenir definida una *camera* i un *viewport*. El *viewport* és una regió rectangular de la pantalla on es projecta una escena en 3D, és a dir, representa els objectes en 3D en un pla en 2D.

Figura 2.8. Escena 3D amb una projecció perspectiva



La ***OrthographicCamera*** és un tipus de càmera (determina els límits d'allò que pot veure el jugador) que fem servir en 2D per contra de la *PerspectiveCamera* que faríem servir en entorns 3D (on les dimensions i la profunditat dels objectes dependrà del camp de visió del jugador, tal i com veiem a la figura figura 2.8). En aquest tipus de càmera s'ignora la projecció dels objectes respecte el *viewport*, mantenint així la seva mida original (figura 2.9).

Figura 2.9. Projecció ortogonal



Prepararem, ara, els gràfics per tenir una representació visual del que està passant. Abans d'assignar els *sprites* a cada objecte, dibuixarem per pantalla les figures

geomètriques dels asteroides i de la nau per veure el resultat del que tenim fins ara. Ho podem fer de dues maneres: des de la classe `GameScreen` fent servir el mètode `render` o implementant els mètodes `draw()` de cadascun dels objectes `actor`. Per simplificar aquesta part ho farem des de la classe `GameScreen`. Aquesta, però, no serà la manera correcta, ja que s'hauria de fer al mètode `draw` de cada objecte, tal com farem per dibuixar els *sprites*.

Figures geomètriques

Per a la representació de figures geomètriques necessitarem un objecte `ShapeRenderer` i assignar-li una `OrthographicCamera` i un `viewport` a l'`stage`.

El `ShapeRenderer` serà l'encarregat de dibuixar per pantalla les figures geomètriques (rectangles i cercles) a la pantalla.

Obrim la classe `GameScreen` i definim les variables d'instància:

```
1 // Representació de figures geomètriques
2     private ShapeRenderer shapeRenderer;
3     // Per obtenir el batch de l'stage
4     private Batch batch;
```

L'objecte `batch` és l'element que ens permet dibuixar *sprites* a la pantalla. És un element molt 'pesat' del qual es recomana tenir una única instància. Quan creem un `stage` aquest ja disposa d'un `batch` que podrem recollir amb el mètode `stage.getBatch()` evitant-nos crear-ne un de nou cada cop que volem dibuixar *sprites* per pantalla.

Al constructor crearem la `OrthographicCamera` i el `viewport` per tal d'assignar-los a l'`stage`. Iniciarem el `ShapeRenderer` i recollirem l'objecte `batch` de l'`stage`:

- Inicialitzem el `ShapeRenderer`:

```
1 // Creem el ShapeRenderer
2         shapeRenderer = new ShapeRenderer();
```

El `FitViewport` respecta la relació d'aspecte entre les dimensions horitzontals i verticals a l'ampliar o reduir una aplicació, mentre que l'`StretchViewport` es redimensiona fins ocupar tota la pantalla, cosa que pot produir una deformació dels objectes ja que no manté la relació d'aspecte.

- Creem la càmera amb les dimensions del joc i la configurem perquè treballi amb el sistema de coordenades *Y-Down*:

```
1 // Creem la càmera de les dimensions del joc
2         OrthographicCamera camera = new OrthographicCamera(Settings.GAME_WIDTH,
              Settings.GAME_HEIGHT);
3         // Posant el paràmetre a true configurem la càmera perquè
4         // faci servir el sistema de coordenades Y_Down
5         camera.setToOrtho(true);
```

- Creem el *viewport*, en aquest cas, ens permetrà veure tot el que mostra la càmera (té les mateixes dimensions):

```
1 // Creem el viewport amb les mateixes dimensions que la càmera
2         StretchViewport viewport = new StretchViewport(Settings.GAME_WIDTH,
              Settings.GAME_HEIGHT , camera);
```

- Assignem el *Viewport* a l'`stage`:

```
// Creem l'stage i assignem el viewport
        stage = new Stage(viewport);
```

- Recuperem el `batch` per fer-lo servir posteriorment:

```
batch = stage.getBatch();
```

Amb aquests canvis, el constructor de `GameScreen` quedarà de la següent manera:

```
public GameScreen() {

        // Creem el ShapeRenderer
        shapeRenderer = new ShapeRenderer();

        // Creem la càmera de les dimensions del joc
        OrthographicCamera camera = new OrthographicCamera(Settings.GAME_WIDTH,
             Settings.GAME_HEIGHT);
        // Posant el paràmetre a true configurem la càmera perquè
        // faci servir el sistema de coordenades Y-Down
        camera.setToOrtho(true);

        // Creem el viewport amb les mateixes dimensions que la càmera
        StretchViewport viewport = new StretchViewport(Settings.GAME_WIDTH,
            Settings.GAME_HEIGHT , camera);

        // Creem l'stage i assignem el viewport
        stage = new Stage(viewport);

        batch = stage.getBatch();

        // Creem la nau i la resta d'objectes
        spacecraft = new Spacecraft(Settings.SPACECRAFT_STARTX, Settings.
            SPACECRAFT_STARTY, Settings.SPACECRAFT_WIDTH, Settings.
            SPACECRAFT_HEIGHT);
        scrollHandler = new ScrollHandler();

        // Afegim els actors a l'stage
        stage.addActor(scrollHandler);
        stage.addActor(spacecraft);


    }
```

Crearem un mètode `drawElements()` en el qual dibuixarem els elements. El primer que haurem de fer és configurar el `shapeRenderer` perquè dibuixi les figures geomètriques amb les mateixes propietats que el *viewport*, és a dir, amb les mateixes dimensions i fent servir un sistema de coordenades *Y-Down*. Per configurar-lo cridarem al mètode `shapeRenderer.setProjectionMatrix(batch.getProjectionMatrix());` que assignarà la matriu de projecció de l'`stage` al `ShapeRenderer`:

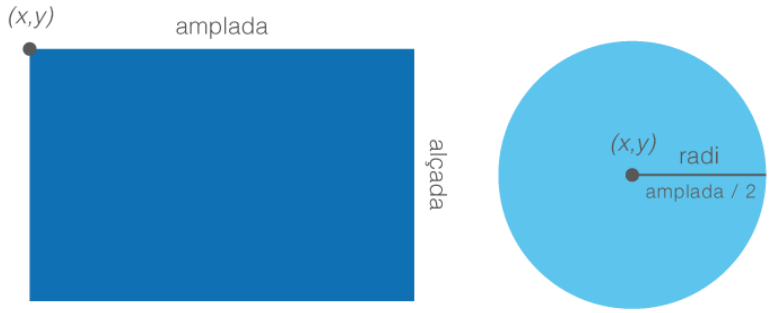
```
//Recollim les propietats del batch de l'stage
        shapeRenderer.setProjectionMatrix(batch.getProjectionMatrix());
```

Tot el que dibuixarem haurà d'anar entre els mètodes `shaperenderer.begin(ShapeType type)` i `shaperenderer.end()`. El `type` serà com volem dibuixar les figures geomètriques: `ShapeRenderer.ShapeType.Filled` per dibuixar l'interior o

`ShapeRenderer.ShapeType.Line` per dibuixar el contorn. Per configurar el color cridarem a `shaperenderer.color(Color color)`.

Per pintar els elements necessitarem un rectangle i diversos cercles, un per cada asteroide. Per dibuixar rectangles cridarem al mètode `rect(float x, float y, float width, float height)` i pels cercles el mètode `circle(float x, float y, float radius)` de l'objecte `ShapeRenderer`.

FIGURA 2.10. Propietats del rectangle i el cercle



Per pintar els rectangles necessitarem la posició de la cantonada superior esquerra i les mides d'ample i alt del rectangle, en aquest cas els valors X i Y de la nau, així com el seu ample i el seu alt; pels cercles necessitarem el punt central del cercle i la mida del radi. Aquests valors els podrem obtenir a partir de les característiques dels elements, tal com podem veure en la figura 2.10.

Els passos a seguir seran:

1. Per evitar l'efecte *flickering* necessitem esborrar la pantalla cada cop que dibuixem les figures geomètriques, ho farem pintant de negre la pantalla.
2. Configurem el `ShapeRenderer` i cridem al mètode `begin()`.
3. Definim el color i pintem la nau.
4. Recorrem tots els `asteroid` i pintem els tres primers de vermell, blau i groc, tots els que podem afegir després seran pintats de blanc.
5. Fem la crida a `end()` i s'enviarà tot el contingut per ser pintat.

El codi de `drawElements()` podria quedar de la següent manera:

```
1  private void drawElements(){
2
3          /* 1 */
4          // Pintem el fons de negre per evitar el "flickering"
5          Gdx.gl20.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
6          Gdx.gl20.glClear(GL20.GL_COLOR_BUFFER_BIT);
7
8          /* 2 */
9          // Recollim les propietats del Batch de l'Stage
10         shapeRenderer.setProjectionMatrix(batch.getProjectionMatrix());
11         // Inicialitzem el shaperenderer
12         shapeRenderer.begin(ShapeRenderer.ShapeType.Filled);
13
```

```
        /* 3 */
        // Definim el color (verd)
        shapeRenderer.setColor(new Color(0, 1, 0, 1));
        // Pintem la nau
        shapeRenderer.rect(spacecraft.getX(), spacecraft.getY(), spacecraft.
            getWidth(), spacecraft.getHeight());

        /* 4 */
        // Recollim tots els Asteroid
        ArrayList<Asteroid> asteroids = scrollHandler.getAsteroids();
        Asteroid asteroid;

        for (int i = 0; i < asteroids.size(); i++) {

            asteroid = asteroids.get(i);
            switch (i) {
                case 0:
                    shapeRenderer.setColor(1,0,0,1);
                    break;
                case 1:
                    shapeRenderer.setColor(0,0,1,1);
                    break;
                case 2:
                    shapeRenderer.setColor(1,1,0,1);
                    break;
                default:
                    shapeRenderer.setColor(1,1,1,1);
                    break;
            }
            shapeRenderer.circle(asteroid.getX() + asteroid.getWidth()/2,
                asteroid.getY() + asteroid.getWidth()/2, asteroid.getWidth()
                /2);
        }
        /* 5 */
        shapeRenderer.end();
    }
```

Haurem de cridar a `drawElements()` des del mètode `render(float delta)`, afegiu la línia `drawElements();` al final del mètode perquè quedi de la següent manera:

```
@Override
    public void render(float delta) {

        // Dibuixem i actualitzem tots els actors de l'stage
        stage.draw();
        stage.act(delta);

        drawElements();

    }
```

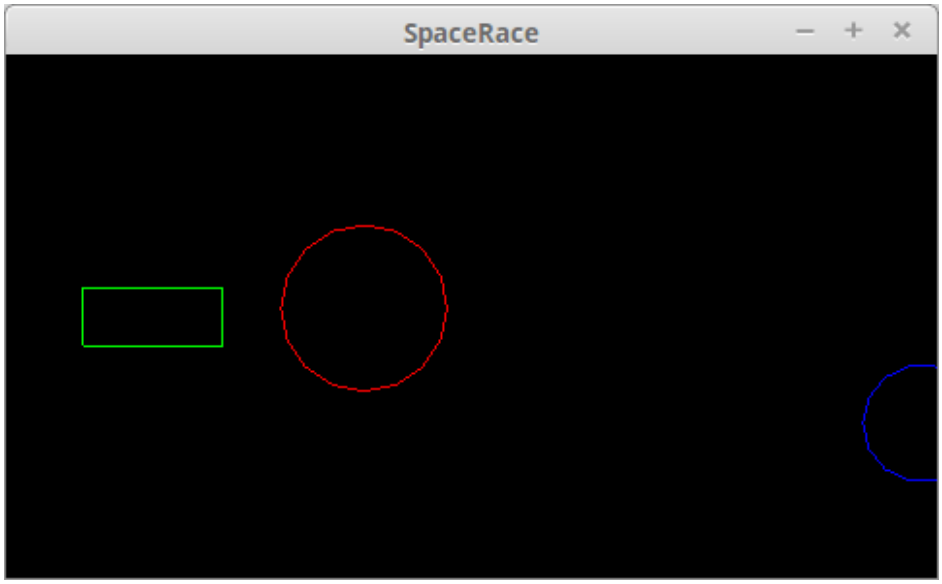
Abans d'executar el projecte i veure el resultat, cal configurar com s'executarà aquest treball com a aplicació d'escriptori. Obriu el fitxer *DesktopLauncher* del subprojecte *desktop* i modifiqueu el mètode `main`. Cal que definim una amplada i alçada de finestra dues vegades superior que com s'ha definit a `Settings`:

```
public static void main (String[] arg) {
    LwjglApplicationConfiguration config = new LwjglApplicationConfiguration();
    config.title = "SpaceRace";
    config.width = Settings.GAME_WIDTH * 2;
    config.height = Settings.GAME_HEIGHT * 2;
    new LwjglApplication(new SpaceRace(), config);
  }
```

Podeu canviar el gruix del `SapeType.Line` amb la propietat `Gdx.gl20.glLineWidth(int width)`.

Executeu el projecte i observeu el resultat, podeu veure la nau i els asteroides! Proveu de canviar la línia `shapeRenderer.begin(ShapeRenderer.ShapeType.Filled);` per `shapeRenderer.begin(ShapeRenderer.ShapeType.Line);`. El resultat serà similar a la figura 2.11.

FIGURA 2.11. ShapeRenderer dibuixant els objectes



La nostra aplicació funciona correctament però si ens hi fixem amb més detall veurem que la posició dels asteroides no canvia quan reiniciem. Haurem d'afegir el mètode `reset(float newX)` a la classe `asteroid` amb el següent contingut:

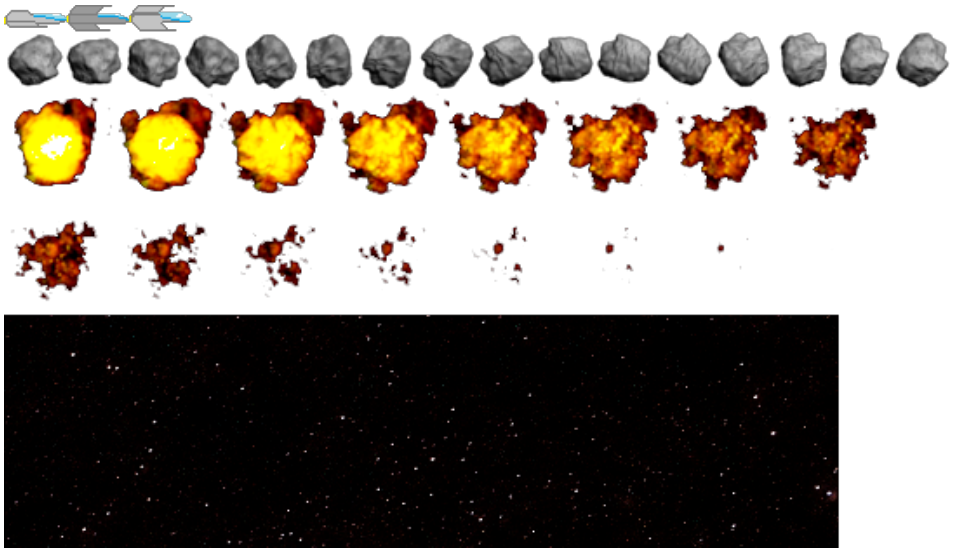
```
1  public void reset(float newX) {
2          super.reset(newX);
3          // Obtenim un número aleatori entre MIN i MAX
4          float newSize = Methods.randomFloat(Settings.MIN_ASTEROID, Settings.
               MAX_ASTEROID);
5          // Modificarem l'alçada i l'amplada segons l'aleatori anterior
6          width = height = 34 * newSize;
7          // La posició serà un valor aleatori entre 0 i l'alçada de l'aplicació
               menys l'alçada de l'asteroide
8          position.y =  new Random().nextInt(Settings.GAME_HEIGHT - (int) height)
               ;
9      }
```

Si tornem a executar el joc observarem com ara sí que es van canviant les dimensions i posicions dels asteroides de manera aleatòria.

Treball amb textures

Per fer més atractiva la nostra aplicació assignarem imatges als objectes. Per fer-ho partirem d'un objecte `texture`, que és una imatge carregada a la memòria de la GPU. Els jocs acostumen a fer servir diversos gràfics i es recomana tenir-los tots en un mateix fitxer (figura 2.12) i dividir-los en `TextureRegions`. Així, carregarem una sola imatge a la memòria i la dividirem en parts més petites que són les que representarem per pantalla.

FIGURA 2.12. 'Sprite sheet' del joc



Per fer-ho, cal obrir l'`AssetManager` i afegir els següents atributs de classe:

```
// Sprite Sheet
    public static Texture sheet;

    // Nau i fons
    public static TextureRegion spacecraft, spacecraftDown, spacecraftUp,
        background;

    // Asteroide
    public static TextureRegion[] asteroid;
    public static Animation asteroidAnim;

    // Explosió
    public static TextureRegion[] explosion;
    public static Animation explosionAnim;
```

Podem fer servir l'aplicació *TexturePacker* per tal d'obtenir tots els *sprites* a LibGDX fent servir un objecte `TextureAtlas`, facilitant-nos la generació i lectura d'*sprites*.

Haurem de definir l'*sprite sheet* i crearem `TextureRegion` per a les tres posicions de la nau, per al fons de pantalla i dos *arrays* per a l'asteroide i per a l'explosió, que a més, necessitaran un objecte del tipus `animation`.

El primer que farem és definir l'*sprite sheet*. Cal copiar el fitxer **sheet.png** al directori *assets* del subprojecte android i afegir les següents línies al mètode `load()`:

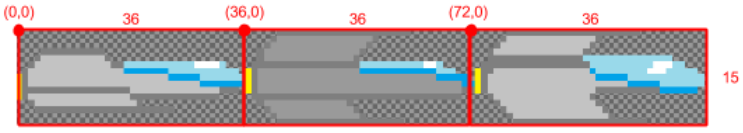
```
// Carreguem les textures i li apliquem el mètode d'escalat 'nearest'
        sheet = new Texture(Gdx.files.internal("sheet.png"));
        sheet.setFilter(Texture.TextureFilter.Nearest, Texture.TextureFilter.
            Nearest);
```

La primera línia definirà el `Texture` i la segona definirà els filtres d'escalat que s'aplicaran en cas de ser necessari: el primer paràmetre fa referència al filtre que s'aplicarà en cas que la imatge s'hagi de disminuir i el segon en cas de necessitar augmentar-se. A la figura 2.13 podeu veure una comparació dels `TextureFilter` *nearest* i *linear*.

Figura 2.13. Diferència entre filtres

Començarem generant els `TextureRegion` de les naus. Hem de calcular la coordenada inicial del rectangle (sempre a la cantonada superior esquerra), l'amplada i l'alçada de l'element, el codi de les naus. Si ens fixem en la figura figura 2.14, quedaria de la següent manera:

Figura 2.14. Càlcul dels TextureRegion




```
// Sprites de la nau
        spacecraft = new TextureRegion(sheet, 0, 0, 36, 15);
        spacecraft.flip(false, true);

        spacecraftUp = new TextureRegion(sheet, 36, 0, 36, 15);
        spacecraftUp.flip(false, true);

        spacecraftDown = new TextureRegion(sheet, 72, 0, 36, 15);
        spacecraftDown.flip(false, true);
```

Al mètode `flip(boolean x, boolean y)` indiquem mitjançant un `true` si volem voltejar la imatge en l'eix horitzontal (X) i en el vertical (Y). En el nostre cas, com que LibGDX fa servir un sistema de coordenades *Y-Up* i nosaltres estem fent servir *Y-Down*, hem de voltejar la imatge en l'eix de les Y fent la crida al mètode `flip(false, true)`.

En el cas de l'asteroide haurem de crear un *array* amb tots els *sprites* de l'animació. A l'*sprite sheet* podem veure com tenim tots els *sprites* en una sola filera i les dimensions de cada element són de 34x34 píxels. Així, haurem de fer 16 iteracions guardant cada *sprite* de l'asteroide. El bucle quedaria així:

```
// Carreguem els 16 estats de l'asteroide
        asteroid = new TextureRegion[16];
        for (int i = 0; i < asteroid.length; i++) {

            asteroid[i] = new TextureRegion(sheet, i * 34, 15, 34, 34);
            asteroid[i].flip(false, true);
```

```
7
8       }
```

Primerament inicialitzem l'*array* amb 16 posicions (de la 0 a la 15), generem el `for` perquè faci iteracions de 0 a 15 i en cadascun calculem la posició de l'*sprite*. Multipliquem el valor d'`i` per 34 per obtenir els valors d'X que seran: 0, 34, 68, 102... Mentre que la Y serà sempre 15 (on acabava la nau espacial). L'amplada i alçada sempre seran 34x34, les dimensions dels asteroides. Per cada iteració, a més, farem el `flip` corresponent per adaptar-la al nostre sistema de coordenades.

Una vegada tenim carregats els *sprites* a l'*array*, haurem de crear l'animació, el constructor ens demana un `float` amb els segons que durarà cada *frame* i l'*array* de *TextureRegion*. Així, amb les línies:

```
1 // Creem l'animació de l'asteroide i fem que s'executi contínuament en sentit
      antihorari
2         asteroidAnim = new Animation(0.05f, asteroid);
3         asteroidAnim.setPlayMode(Animation.PlayMode.LOOP_REVERSED);
```

Crearem i configurarem l'animació perquè es repeteixi de manera invertida (des del final cap al principi: `LOOP_REVERSED`) i així fer que l'asteroide giri en sentit antihorari.

Entre els `PlayMode` tenim:

- `Animation.PlayMode.NORMAL`: l'animació no es repetirà, reproduirà els *frames* del 0 fins el final de l'*array*.
- `Animation.PlayMode.REVERSED`: sense repetir-se, mostrarà els *frames* en ordre invers, des del final fins al *frame* 0.
- `Animation.PlayMode.LOOP`: quan acabi l'animació la tornarà a començar des de l'inici.
- `Animation.PlayMode.LOOP_REVERSED`: farà l'animació en ordre descendent i tornarà a repetir-la en acabar.
- `Animation.PlayMode.LOOP_PINGPONG`: comença l'animació en sentit ascendent i quan arriba al final la fa en sentit descendent i així successivament. Penseu en el joc del ping-pong, en com la pilota va del costat esquerre al dret, del costat dret torna al costat esquerre i tornem a començar de nou.
- `Animation.PlayMode.LOOP_RANDOM`: agafa imatges a l'atzar de l'*array* per crear l'animació.

Seguint els mateixos passos que l'animació dels asteroides, crearem l'animació de l'explosió. Hi ha, però, dues diferències: els *sprites* necessaris es troben en dues files de l'*sprite sheet* i l'animació de l'explosió sols s'executarà una vegada. El primer que hem de fer és pensar de quina manera podem obtenir tots els *sprites* en un *array*: necessitarem fer dos bucles. En el primer controlarem quina fila estem llegint, que serà o la 0 o la 1, i per cadascuna de les files haurem de llegir 8 *sprites*

(segon bucle). Abans de començar els bucles crearem un índex que s'incrementarà de 0 a 15 per guardar cadascun dels *sprites* en l'*array*.

El bucle per guardar els *sprites* de l'explosió seria:

```
// Creem els 16 estats de l'explosió
        explosion = new TextureRegion[16];

        // Carreguem els 16 estats de l'explosió
        int index = 0;
        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 8; j++) {
                explosion[index++] = new TextureRegion(sheet, j * 64,  i * 64 +
                     49, 64, 64);
                explosion[index-1].flip(false, true);
            }
        }
```

Les files es calcularan al tercer paràmetre del `TextureRegion`, `i * 64 + 49`: a la primera iteració la Y serà 49 (que era on acabaven els *sprites* dels asteroides) i a la segona iteració serà 113, és a dir, 49 dels asteroides més els 64 de la primera fila de l'explosió. En cadascuna de les files, la X anirà augmentant de 64 en 64, prenent els valors 0, 64, 128,... fins a 448.

Per guardar-les a l'*array* fem servir l'expressió `explosion[index++]`, que seria l'equivalent a realitzar dues accions: `explosion[index]` i després `index += 1`. El `flip` del `TextureRegion` l'haurem de fer al de la posició `index-1` ja que en actualitzar l'índex en la instrucció anterior aquest estarà apuntant al següent valor del bucle.

Finalment creem l'animació amb la línia `explosionAnim = new Animation(0.04f, explosion);` i en aquest cas la imatge no es repetirà així que no serà necessari configurar el `setPlayMode` (el mode per defecte és `PlayMode.NORMAL`).

Finalment, ja sols ens queda afegir el `TextureRegion` del fons de pantalla (480x135 píxels) que quedaria de la següent manera:

```
// Fons de pantalla
        background = new TextureRegion(sheet, 0, 177, 480, 135);
        background.flip(false, true);
```

Al mètode `dispose()` haurem de cridar al `dispose()` de l'element `texture`:

```
public static void dispose() {

        // Descartem els recursos
        sheet.dispose();

    }
```

I amb això podem deixar de banda la classe `AssetManager` per centrar-nos en els objectes `actor`. Començarem per dibuixar el fons de pantalla i la nau espacial.

L'objecte encarregat de dibuixar els *sprites* serà un `batch`. Tots els objectes descendents d'`actor` disposen del mètode `draw(Batch batch, float parentAlpha)` que rep com a argument el `batch` necessari. El mètode `draw()`

s'executarà sempre que s'executi el mètode `draw` de l'`Stage` que els conté així que serà el millor lloc per dibuixar els objectes. Si traiem un `actor` de l'`stage` automàticament deixarà de dibuixar-se de la pantalla i si el tornem a afegir ho tornarà a fer. Això ens estalviarà moltes comprovacions i booleans innecessaris.

Per dibuixar el fons de pantalla haurem d'anar a la classe `Background` i fer l'`Override` del mètode `draw` (*Code/Override Methods...*). Des d'aquí podrem cridar al mètode `batch.draw(Texture texture, float x, float y, float width, float height)` per dibuixar qualsevol textura. Com que volem dibuixar el fons de pantalla, haurem de seleccionar la textura `AssetManager.background` i passem les propietats de l'element:

```
1  public void draw(Batch batch, float parentAlpha) {
2          super.draw(batch, parentAlpha);
3          batch.disableBlending();
4          batch.draw(AssetManager.background, position.x, position.y, width,
               height);
5          batch.enableBlending();
6      }
```

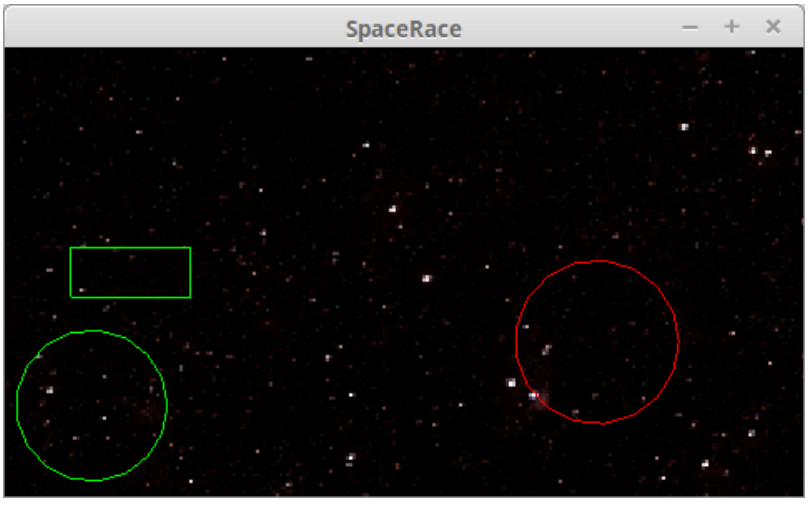
Abans de cridar al mètode `batch.draw()`, executarem `batch.disableBlending()` que ens dibuixarà imatges sense transparència d'una manera més eficient (el `blending` està activat per defecte). Una vegada dibuixat el fons el tornarem a habilitar per no alterar la resta d'objectes `actor` que potser sí que necessiten transparència.

Si comentem les línies:

```
1  //Gdx.gl20.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
2          //Gdx.gl20.glClear(GL20.GL_COLOR_BUFFER_BIT);
```

Del mètode `drawElements()` de la classe `GameScreen` (les encarregades de pintar el fons negre), enexecutar el joc tindrem com a resultat el que podem veure a la figura 2.15.

Figura 2.15. Aplicació amb el fons pintat



Per dibuixar la nau, haurem de fer el mateix. Cal anar a la classe `Spacecraft` i fem l'`Override` del mètode `draw`:

```
@Override
    public void draw(Batch batch, float parentAlpha) {
        super.draw(batch, parentAlpha);
        batch.draw(AssetManager.spacecraft, position.x, position.y, width,
            height);
    }
```

En el cas de la nau espacial no deshabilitem el `blending` ja que necessitem transparència.

Sols ens falta representar els asteroides, però abans haurem de fer unes modificacions en la classe `asteroid` per tal de calcular el *frame* que s'ha de mostrar en cada crida del mètode `draw`. Per fer això necessitem una variable que es vagi incrementant, que anomenarem `runTime`. Cada `asteroid` tindrà un `runTime` que s'actualizarà al seu mètode `act()`.

Obriu la classe `asteroid` i afegiu l'atribut `private float runTime` (amb el corresponent *getter*) que inicialitzarem al constructor amb un valor aleatori entre 0 i 1. Feu un `Override` dels mètodes `draw` i `act`, en `act` actualitzarem el valor del `runTime` incrementant-lo en `delta`. Això ens deixarà una classe com la següent:

```
package cat.xtec.ioc.objects;

import com.badlogic.gdx.graphics.g2d.Batch;

import java.util.Random;

import cat.xtec.ioc.utils.Methods;
import cat.xtec.ioc.utils.Settings;

public class Asteroid extends Scrollable {

    private float runTime;

    public Asteroid(float x, float y, float width, float height, float velocity
        ) {
        super(x, y, width, height, velocity);
        runTime = Methods.randomFloat(0,1);
    }

    @Override
    public void act(float delta) {
        super.act(delta);
        runTime += delta;
    }

    @Override
    public void reset(float newX) {
        super.reset(newX);
        // Obtenim un número al·leatori entre MIN i MAX
        float newSize = Methods.randomFloat(Settings.MIN_ASTEROID, Settings.
            MAX_ASTEROID);
        // Modificarem l'alçada i l'amplada segons l'al·leatori anterior
        width = height = 34 * newSize;
        // La posició serà un valor aleatòri entre 0 i l'alçada de l'aplicació
            menys l'alçada
        position.y =  new Random().nextInt(Settings.GAME_HEIGHT - (int) height)
            ;

    }

    // Getter pel runTime
    public float getRunTime() {

```

```
        return runTime;
    }

    @Override
    public void draw(Batch batch, float parentAlpha) {
        super.draw(batch, parentAlpha);
    }
}
```

Per obtenir l'*sprite* de l'animació de cada asteroide, haurem de cridar a `AssetManager.asteroidAnim.getKeyFrame(runTime)`, és a dir, canviarem el *frame* segons la variable `runTime` que s'anirà incrementant. Si afegim el codi al mètode `draw`:

```
@Override
    public void draw(Batch batch, float parentAlpha) {
        super.draw(batch, parentAlpha);
        batch.draw(AssetManager.asteroidAnim.getKeyFrame(runTime), position.x,
            position.y, width, height);
    }
```

Veiem com dibuixarà els asteroides que aniran animant-se a la velocitat definida per l'objecte `asteroidAnim`.

Comenteu la crida a `drawElements()` del mètode `render` de la classe `GameScreen`, executeu el joc a l'ordinador i proveu-lo també a un dispositiu físic si podeu. Ja tenim els gràfics llestos i només ens quedarà provar l'animació de l'explosió que afegirem en l'apartat de col·lisions.

Sons

Treballar amb sons i música en LibGDX és molt senzill. Veurem, ara, un exemple de cada tipus i modificarem el projecte per posar música de fons. El so de l'explosió l'implementarem més endavant en l'apartat de col·lisions.

Per tal d'afegir un so, ho farem mitjançant un objecte `sound`, i cridant el mètode `Gdx.audio.newSound`, per exemple, per afegir el so de l'explosió, que es troba a *android/assets/sounds/explosion.wav* ho farem mitjançant la següent instrucció:

```
explosionSound = Gdx.audio.newSound(Gdx.files.internal("sounds/explosion.wav"))
    ;
```

Cada cop que fem `play()` del so, ens retornarà un identificador (`long`) que necessitarem per realitzar tasques com ara canviar el volum, posar-lo en pausa, reprendre'l, etc. Alguns dels principals mètodes per treballar amb sons són:

- `dispose()`: allibera tots els recursos.
- `loop()`: comença la reproducció del so amb la repetició activada.
- `play()`: comença la reproducció del so, retorna un identificador que ens servirà per controlar la instància del so.
- `pause()`: aturem la reproducció per continuar-la des del mateix punt on l'hem aturat de totes les instàncies del so.

- `pause(long soundId)`: aturem la reproducció per continuar-la des del mateix punt on l'hem aturat d'una instància concreta del so.
- `resume()`: reprèn la reproducció de totes les instàncies del so.
- `resume(long soundId)`: reprèn la reproducció d'una instància del so.
- `stop()`: atura la reproducció de totes les instàncies del so.
- `stop(long soundId)`: atura la reproducció d'una instància del so.
- `setLooping(long soundId, boolean isLooping)`: reprodueix de manera infinita amb `true` i amb una sola reproducció amb `false` d'una instància del so.
- `setVolume(long soundId, float volume)`: ens permet modificar el volum d'una instància concreta del so (valor entre 0.0f i 1.0f).

L'objecte `sound` ens servirà per a sons curts que no necessiten *streaming* i que, a més, puguin tenir diverses instàncies. Si volem afegir un fitxer de més durada i que ocupa més espai al disc, hem de fer servir un objecte `Music`. Per exemple, si volem carregar el fitxer *android/assets/sounds/Afterburner.ogg* hauríem de fer-ho amb:

```
1 music = Gdx.audio.newMusic(Gdx.files.internal("sounds/Afterburner.ogg"));
```

Els objectes `music` només tenen una instància, al contrari que els objectes `sound`. Alguns dels mètodes que podrem fer servir als objectes `music` són:

- `dispose()`: quan l'objecte no es necessita.
- `play()`: comença la reproducció. Si estava pausada la reprèn.
- `pause()`: atura la reproducció per continuar-la des del mateix punt on l'hem aturat.
- `stop()`: atura la reproducció de la música.
- `setLooping(boolean isLooping)`: reprodueix de manera infinita amb `true` i amb una sola reproducció amb `false`.
- `isLooping()`: retorna un booleà indicant si el fitxer de música està o no configurat amb la reproducció infinita.
- `isPlaying()`: retorna un booleà per saber si s'està reproduint o no un fitxer de música.
- `getPosition()`: retorna la posició en segons de la reproducció.
- `setVolume(float volume)`: ens permet modificar el volum de la música (valor entre 0.0f i 1.0f).

Afegirem, ara, música al nostre joc i prepararem el so de l'explosió per més endavant. Per fer-ho, obriu el fitxer `AssetManager` i afegiu les variables de classe `explosionSound` i `music`:

```
// Sons
    public static Sound explosionSound;
    public static Music music;
```

Aquestes variables les inicialitzaren al final del mètode `load()` amb les següents línies:

```
/****************************** Sounds ***********************************/
        // Explosió
        explosionSound = Gdx.audio.newSound(Gdx.files.internal("sounds/
            explosion.wav"));

        // Música del joc
        music = Gdx.audio.newMusic(Gdx.files.internal("sounds/Afterburner.ogg")
            );
        music.setVolume(0.2f);
        music.setLooping(true);
```

Carreguem els dos fitxers del directori *assets/sounds* i configurem la música baixant-li el volum i fent que es repeteixi en acabar. Haurem d'afegir en el mètode `dispose()` les crides a `dispose` dels dos recursos sonors:

```
public static void dispose() {

        // Descartem els recursos
        sheet.dispose();
        explosionSound.dispose();
        music.dispose();
}
```

Per reproduir la música anem a la classe `GameScreen` i afegim al constructor la següent línia:

```
// Iniciem la música
        AssetManager.music.play();
```

2.3.5 Control de la nau

El control de la nau el farem mitjançant la pantalla tàctil: un desplaçament cap a baix farà que la nau incrementi el valor de la Y i farà el contrari en cas d'un desplaçament cap a dalt.

Per gestionar l'entrada de l'usuari anem a *Crea una classe* al paquet `cat.xtec.ioc.helpers`. La classe ha de ser `InputHandler` que implementarà la interfície `InputProcessor`. Per fer-ho, haurem d'implementar els mètodes:

- `keyDown`: s'executa quan es prem una tecla.
- `keyUp`: s'executa quan deixem anar una tecla després d'haver-la premut.
- `keyTyped`: s'executa quan s'escriu un caràcter a un element que té el focus del teclat.

- `touchDown`: s'executa quan l'usuari pressiona en algun lloc de la pantalla.
- `touchUp`: codi que s'executarà quan l'usuari deixi anar el dit de la pantalla.
- `touchDragged`: s'executa quan es produeix un arrossegament del dit per la pantalla.
- `mouseMoved`: exclusiva de les aplicacions d'escriptori, s'executa quan es mou el ratolí sobre la superfície de l'aplicació.
- `scrolled`: exclusiva de les aplicacions d'escriptori, respon a un desplaçament de la roda del ratolí.

Una vegada enllestida la classe tindrà el següent codi:

```
1  package cat.xtec.ioc.helpers;
2
3  import com.badlogic.gdx.InputProcessor;
4
5  public class InputHandler implements InputProcessor {
6
7      @Override
8      public boolean keyDown(int keycode) {
9          return false;
10     }
11
12     @Override
13     public boolean keyUp(int keycode) {
14         return false;
15     }
16
17     @Override
18     public boolean keyTyped(char character) {
19         return false;
20     }
21
22     @Override
23     public boolean touchDown(int screenX, int screenY, int pointer, int button)
           {
24         return false;
25     }
26
27     @Override
28     public boolean touchUp(int screenX, int screenY, int pointer, int button) {
29         return false;
30     }
31
32     @Override
33     public boolean touchDragged(int screenX, int screenY, int pointer) {
34         return false;
35     }
36
37     @Override
38     public boolean mouseMoved(int screenX, int screenY) {
39         return false;
40     }
41
42     @Override
43     public boolean scrolled(int amount) {
44         return false;
45     }
46 }
```

Els tres *events* que ens interessen en aquest moment són: `touchDown()`, `touchUp()` i `touchDragged()`. Quan l'usuari arrossegui el dit canviarem la

direcció de la nau i quan l'aixequi la nau deixarà de moure's. Necessitarem l'objecte `spacecraft` i l'objecte `GameScreen`, que el rebrem com a paràmetre del constructor, des del qual podrem accedir a la nau.

Afegirem a la classe les variables d'instància i crearem el constructor. També caldrà afegir un enter que ens permetrà saber en quina direcció s'ha fet un moviment de *drag*:

```
// Objectes necessaris
    private Spacecraft spacecraft;
    private GameScreen screen;

    // Enter per a la gestió del moviment d'arrossegament
    int previousY = 0;

    public InputHandler(GameScreen screen) {

        // Obtenim tots els elements necessaris
        this.screen = screen;
        spacecraft = screen.getSpacecraft();

    }
```

Ja ho tenim tot preparat per controlar quan l'usuari arrossega un dit. El mètode `touchDragged` rep tres paràmetres:

- `int screenX`: coordenada horitzontal (X) del dit.
- `int screenY`: coordenada vertical (Y) del dit.
- `int pointer`: identificador del punter actiu.

Com que volem desplaçar la nau en l'eix d'Y, necessitarem utilitzar únicament el paràmetre `screenY`: si `screenY` és major que l'anterior Y vol dir que s'està arrossegant el dit cap avall (sistema *Y-Down*) i per tant la nau haurà de baixar. En cas contrari haurà de pujar. Guardarem la posició quan l'usuari faci clic a la pantalla i caldrà definir un llindar per acceptar un moviment i evitar que tenint el dit quiet la nau es mogui cap a un o altre costat. Posarem un llindar de 2 píxels.

El codi corresponent als mètodes `touchDragged` i `touchDown` serà el següent:

```
@Override
    public boolean touchDown(int screenX, int screenY, int pointer, int button)
         {
        previousY = screenY;
        return true;
    }

    @Override
    public boolean touchDragged(int screenX, int screenY, int pointer) {
        // Posem un llindar per evitar gestionar events quan el dit està quiet
        if (Math.abs(previousY - screenY) > 2)

            // Si la Y és major que la que tenim
            // guardada és que va cap avall
            if (previousY < screenY) {
                spacecraft.goDown();
            } else {
                // En cas contrari cap amunt
                spacecraft.goUp();
```

```
            }
        // Guardem la posició de la Y
        previousY = screenY;
        return true;
    }
```

Hem d'assignar la classe `InputHandler` com a `InputProcessor` de la classe `GameScreen`. Cal anar al constructor de `GameScreen` i afegir al final les línies:

```
// Assignem com a gestor d'entrada la classe InputHandler
        Gdx.input.setInputProcessor(new InputHandler(this));
```

Si executem l'aplicació podrem desplaçar la nau en vertical però no la podrem aturar. Per fer-ho hauríem d'implementar el mètode `touchUp` i fer que la nau torni al seu estat natural quan s'aixequi el dit:

```
@Override
    public boolean touchUp(int screenX, int screenY, int pointer, int button) {

        // Quan deixem anar el dit acabem un moviment
        // i posem la nau a l'estat normal
        spacecraft.goStraight();
        return true;
    }
```

Amb aquests mètodes ja tenim la funcionalitat implementada, però la representació de la nau no és la correcta, hem de fer que mostri l'*sprite* corresponent quan la nau està movent-se. Anem a la classe `spacecraft` i afegim un mètode que retornarà el `TextureRegion` corresponent segons la direcció de la nau, i farem la crida a aquest mètode quan anem a dibuixar-la.

El mètode `getSpacecraftTexture()` serà el següent:

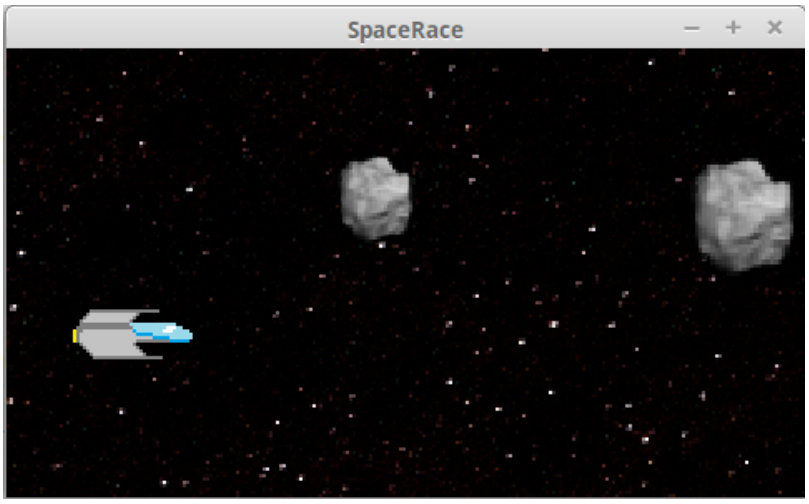
```
// Obtenim el TextureRegion depenent de la posició de l'spacecraft
    public TextureRegion getSpacecraftTexture() {

        switch (direction) {

            case SPACECRAFT_STRAIGHT:
                return AssetManager.spacecraft;
            case SPACECRAFT_UP:
                return AssetManager.spacecraftUp;
            case SPACECRAFT_DOWN:
                return AssetManager.spacecraftDown;
            default:
                return AssetManager.spacecraft;
        }
    }
```

I el mètode `draw` el modificarem perquè quedi de la següent manera:

```
@Override
    public void draw(Batch batch, float parentAlpha) {
        super.draw(batch, parentAlpha);
        batch.draw(getSpacecraftTexture(), position.x, position.y, width,
            height);
    }
```

Si executem l'aplicació ja tindrem el moviment de la nau i la representació correctament implementats, tal com podem veure a la figura 2.16.

FIGURA 2.16. Nau desplaçant-se



2.3.6 Col·lisions

Hi ha diverses maneres de gestionar les col·lisions. Existeixen llibreries externes com per exemple `Box2D` que disposa d'un gestor de col·lisions. També és possible fer-ho comprovant la intersecció entre figures geomètriques. Pel nostre joc, podrem comprovar les col·lisions entre la nau (rectangle) i els asteroides (cercles) amb el mètode `Intersector.overlaps` que accepta diverses figures geomètriques, com per exemple:

- `Intersector.overlaps(Circle c, Recangle r)`: retorna `true` si es sobreposen un cercle i un rectangle.
- `Intersector.overlaps(Circle c1, Circle c2)`: retorna `true` si es sobreposen dos cercles.
- `Intersector.overlaps(Rectangle r1, Recangle r2)`: retorna `true` si es sobreposen dos rectangles.
- `Intersector.overlapConvexPolygons(Polygon p1, Polygon p2)`: retorna `true` si es sobreposen dos polígons.

Necessitem llavors un rectangle per a la nau i cercles per a cadascun dels asteroides.

Anem a la classe `Spacecraft` i creem una variable d'instància `private Rectangle collisionRect;` que serà un `Rectangle` de `com.badlogic.gdx.math.Rectangle`. Aquest `Rectangle` l'haurem d'anar actualitzant segons la posició de la nau. Creem el rectangle i actualitzem la seva posició en el mètode `act` obtenint com a resultat el codi del constructor i del mètode `act` de la següent manera:

```
1 public Spacecraft(float x, float y, int width, int height) {
2
3         // Inicialitzem els arguments segons la crida del constructor
```

```
        this.width = width;
        this.height = height;
        position = new Vector2(x, y);

        // Inicialitzem l'spacecraft a l'estat normal
        direction = SPACECRAFT_STRAIGHT;

        // Creem el rectangle de col·lisions
        collisionRect = new Rectangle();

    }

    public void act(float delta) {

        // Movem l'spacecraft depenent de la direcció controlant que no surti
             de la pantalla
        switch (direction) {
            case SPACECRAFT_UP:
                if (this.position.y - Settings.SPACECRAFT_VELOCITY * delta >=
                     0) {
                    this.position.y -= Settings.SPACECRAFT_VELOCITY * delta;
                }
                break;
            case SPACECRAFT_DOWN:
                if (this.position.y + height + Settings.SPACECRAFT_VELOCITY *
                     delta <= Settings.GAME_HEIGHT) {
                    this.position.y += Settings.SPACECRAFT_VELOCITY * delta;
                }
                break;
            case SPACECRAFT_STRAIGHT:
                break;
        }

        collisionRect.set(position.x, position.y + 3, width, 10);

    }
```

En el codi anterior s'ha ajustat el rectangle 3 píxels en vertical i hem modificat l'alçada per evitar els espais transparents del `TextureRegion`.

Afegim també el corresponent *getter*.

```
public Rectangle getCollisionRect() {
        return collisionRect;
    }
```

Pel que fa als `asteroid`, haurem de crear un cercle i actualitzar-lo cada vegada que es mou l'objecte. Per fer-ho cal anar a la classe`asteroid` i crear la variable d'instància `private Circle collisionCircle;` que serà un `com.badlogic.gdx.math.Circle`. L'iniciem i actualitzem deixant el constructor i el mètode `act` de la següent manera:

```
public Asteroid(float x, float y, float width, float height, float velocity) {
        super(x, y, width, height, velocity);
        runTime = Methods.randomFloat(0,1);

        // Creem el cercle
        collisionCircle = new Circle();
    }

    @Override
    public void act(float delta) {
        super.act(delta);
        runTime += delta;

```

```
        // Actualitzem el cercle de col·lisions (punt central de l'asteroide i
            del radi).
        collisionCircle.set(position.x + width / 2.0f, position.y + width / 2.0
            f, width / 2.0f);

    }
```

Des del mètode `render` de la classe `GameScreen`, anirem comprovant si els objectes `Scrollable` col·lideixen amb la nau. Per fer-ho hem de cridar al mètode `scrollhandler.collides(spacecraft)`: si el mètode citat torna `true`, llavors farem explotar i desaparèixer la nau:

```
@Override
    public void render(float delta) {

        // Dibuixem i actualitzem tots els actors de l'stage
        stage.draw();
        stage.act(delta);

        if (scrollHandler.collides(spacecraft)) {

            // La nau explota i desapareix
            Gdx.app.log("App", "Explosió");

        }

        //drawElements();

    }
```

Haurem de crear el mètode `collides` a la classe `ScrollHandler`. Aquest mètode comprovarà si algun dels asteroides col·lideixen amb la nau i en cas de que algun ho faci retornarem `true`, afegim el mètodes `collides`:

```
public boolean collides(Spacecraft nau) {

        // Comprovem les col·lisions entre cada asteroide i la nau
        for (Asteroid asteroid : asteroids) {
            if (asteroid.collides(nau)) {
                return true;
            }
        }
        return false;
    }
```

I sols ens faltarà el mètode `collides` de la classe `asteroid` on comprovarem si el cercle de l'asteroide se sobreposa al rectangle de la nau sempre que l'asteroide es troba a l'alçada de la nau:

```
// Retorna true si hi ha col·lisió
    public boolean collides(Spacecraft nau) {

        if (position.x <= nau.getX() + nau.getWidth()) {
            // Comprovem si han col·lisionat sempre que l'asteroide es trobi a
                la mateixa alçada que l'spacecraft
            return (Intersector.overlaps(collisionCircle, nau.getCollisionRect
                ()));
        }
        return false;
    }
```

Executem l'aplicació i veurem com cada vegada que es produeix una col·lisió apareixen missatges en el *log* de l'Android Studio.

Si volem controlar quan s'ha de deixar de comprovar les col·lisions, necessitarem un booleà `gameOver` inicialment a `false`. Si `gameOver` és `false` i s'ha produït una col·lisió:

- Reproduïm el so de l'explosió: `AssetManager.explosionSound.play();`
- Eliminem l'`actor` de l'`stage` perquè no continuï actualitzant-se ni dibuixant-se per pantalla. Per fer-ho, necessitarem fer dos passos:
  1. Posar-li nom a l'`actor` després d'afegir-lo a l'`stage` amb la instrucció `spacecraft.setName("spacecraft")`.
  2. Cercar-lo entre els actors de l'`stage` amb `stage.getRoot().findActor("spacecraft")` i fent la crida al mètode `remove()`.
- Posem la variable `gameOver` a `false`.

Si `gameOver` és `true`:

- Dibuixem l'explosió.
- Incrementem la variable `explosionTime` (prèviament declarada i inicialitzada a 0) sumant-li el valor `delta`.

El mètode `render` quedarà de la següent manera:

```
@Override
    public void render(float delta) {

        // Dibuixem i actualitzem tots els actors de l'stage
        stage.draw();
        stage.act(delta);

        if (!gameOver) {
            if (scrollHandler.collides(spacecraft)) {
                // Si hi ha hagut col·lisió: Reproduïm l'explosió
                AssetManager.explosionSound.play();
                stage.getRoot().findActor("spacecraft").remove();
                gameOver = true;
            }
        } else {
            batch.begin();
            // Si hi ha hagut col·lisió: reproduïm l'explosió
            batch.draw(AssetManager.explosionAnim.getKeyFrame(explosionTime,
                 false), (spacecraft.getX() + spacecraft.getWidth() / 2) - 32,
                 spacecraft.getY() + spacecraft.getHeight() / 2 - 32, 64, 64);
            batch.end();

            explosionTime += delta;
        }

        //drawElements();

    }
```

I la classe `GameScreen`:

```
package cat.xtec.ioc.screens;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Screen;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.utils.viewport.StretchViewport;

import java.util.ArrayList;

import cat.xtec.ioc.helpers.AssetManager;
import cat.xtec.ioc.helpers.InputHandler;
import cat.xtec.ioc.objects.Asteroid;
import cat.xtec.ioc.objects.ScrollHandler;
import cat.xtec.ioc.objects.Spacecraft;
import cat.xtec.ioc.utils.Settings;

public class GameScreen implements Screen {

    // Per controlar el gameover
    Boolean gameOver = false;

    // Objectes necessaris
    private Stage stage;
    private Spacecraft spacecraft;
    private ScrollHandler scrollHandler;

    // Encarregats de dibuixar elements per pantalla
    private ShapeRenderer shapeRenderer;
    private Batch batch;

    // Per controlar l'animació de l'explosió
    private float explosionTime = 0;

    public GameScreen() {

        // Iniciem la música
        AssetManager.music.play();

        // Creem el ShapeRenderer
        shapeRenderer = new ShapeRenderer();

        // Creem la càmera de les dimensions del joc
        OrthographicCamera camera = new OrthographicCamera(Settings.GAME_WIDTH,
                Settings.GAME_HEIGHT);
        // Posant el paràmetre a true configurem la càmera perquè
        // faci servir el sistema de coordenades Y-Down
        camera.setToOrtho(true);

        // Creem el viewport amb les mateixes dimensions que la càmera
        StretchViewport viewport = new StretchViewport(Settings.GAME_WIDTH,
            Settings.GAME_HEIGHT, camera);

        // Creem l'stage i assginem el viewport
        stage = new Stage(viewport);

        batch = stage.getBatch();

        // Creem la nau i la resta d'objectes
        spacecraft = new Spacecraft(Settings.SPACECRAFT_STARTX, Settings.
            SPACECRAFT_STARTY, Settings.SPACECRAFT_WIDTH, Settings.
            SPACECRAFT_HEIGHT);
        scrollHandler = new ScrollHandler();

        // Afegim els actors a l'stage
```

```
65          stage.addActor(scrollHandler);
66          stage.addActor(spacecraft);
67          // Donem nom a l'Actor
68          spacecraft.setName("spacecraft");
69
70          // Assignem com a gestor d'entrada la classe InputHandler
71          Gdx.input.setInputProcessor(new InputHandler(this));
72
73      }
74
75      private void drawElements() {
76
77          // Recollim les propietats del batch de l'stage
78          shapeRenderer.setProjectionMatrix(batch.getProjectionMatrix());
79
80          // Pintem el fons de negre per evitar el "flickering"
81          //Gdx.gl20.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
82          //Gdx.gl20.glClear(GL20.GL_COLOR_BUFFER_BIT);
83
84          // Inicialitzem el shaperenderer
85          shapeRenderer.begin(ShapeRenderer.ShapeType.Line);
86
87          // Definim el color (verd)
88          shapeRenderer.setColor(new Color(0, 1, 0, 1));
89
90          // Pintem la nau
91          shapeRenderer.rect(spacecraft.getX(), spacecraft.getY(), spacecraft.
                getWidth(), spacecraft.getHeight());
92
93          // Recollim tots els Asteroid
94          ArrayList<Asteroid> asteroids = scrollHandler.getAsteroids();
95          Asteroid asteroid;
96
97          for (int i = 0; i < asteroids.size(); i++) {
98
99              asteroid = asteroids.get(i);
100             switch (i) {
101                 case 0:
102                     shapeRenderer.setColor(1, 0, 0, 1);
103                     break;
104                 case 1:
105                     shapeRenderer.setColor(0, 0, 1, 1);
106                     break;
107                 case 2:
108                     shapeRenderer.setColor(1, 1, 0, 1);
109                     break;
110                 default:
111                     shapeRenderer.setColor(1, 1, 1, 1);
112                     break;
113             }
114             shapeRenderer.circle(asteroid.getX() + asteroid.getWidth() / 2,
                    asteroid.getY() + asteroid.getWidth() / 2, asteroid.getWidth()
                     / 2);
115         }
116         shapeRenderer.end();
117     }
118
119
120     @Override
121     public void show() {
122
123     }
124
125     @Override
126     public void render(float delta) {
127
128         // Dibuixem i actualizem tots els actors de l'stage
129         stage.draw();
130         stage.act(delta);
131
```

```
132         if (!gameOver) {
133             if (scrollHandler.collides(spacecraft)) {
134                 // Si hi ha hagut col·lisió: reproduïm l'explosió
135                 AssetManager.explosionSound.play();
136                 stage.getRoot().findActor("spacecraft").remove();
137                 gameOver = true;
138             }
139         } else {
140             batch.begin();
141             // Si hi ha hagut col·lisió: reproduïm l'explosió
142             batch.draw(AssetManager.explosionAnim.getKeyFrame(explosionTime,
                        false), (spacecraft.getX() + spacecraft.getWidth() / 2) - 32,
                        spacecraft.getY() + spacecraft.getHeight() / 2 - 32, 64, 64);
143             batch.end();
144 
145             explosionTime += delta;
146         }
147 
148         //drawElements();
149 
150     }
151 
152     @Override
153     public void resize(int width, int height) {
154 
155     }
156 
157     @Override
158     public void pause() {
159 
160     }
161 
162     @Override
163     public void resume() {
164 
165     }
166 
167     @Override
168     public void hide() {
169 
170     }
171 
172     @Override
173     public void dispose() {
174 
175     }
176 
177     public Spacecraft getSpacecraft() {
178         return spacecraft;
179     }
180 
181     public Stage getStage() {
182         return stage;
183     }
184 
185     public ScrollHandler getScrollHandler() {
186         return scrollHandler;
187     }
188 }
```

2.3.7 Text

Un dels elements més importants en els jocs és el text, bé sigui per donar instruccions de com es juga, per avisar el jugador de quan ha acabat el joc, de quina puntuació té, per representar diàlegs de personatges, etc.

LibGDX ens permet generar textos amb la font predeterminada o afegir les nostres pròpies fonts d'una manera senzilla. A més, tenim diverses maneres d'introduir el text per la pantalla, a través del mètode `draw` de la font o bé fent ús de les etiquetes (*label*). Anem a veure exemples dels dos mètodes.

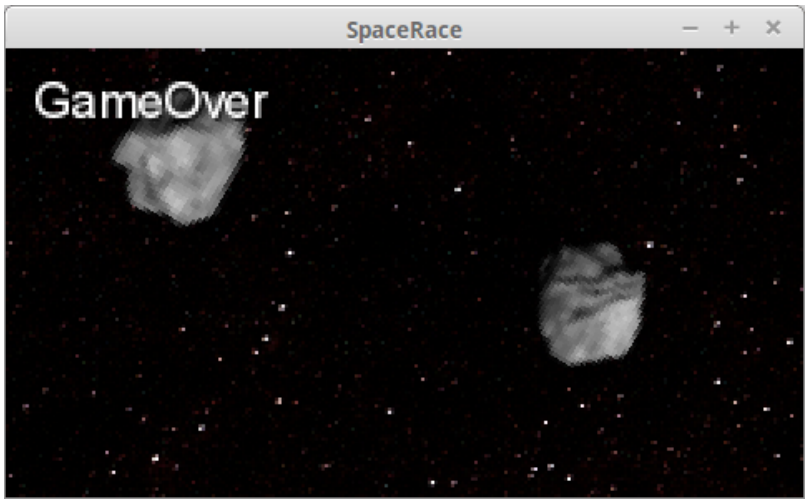
Mètode draw de font

Si volem generar un text molt bàsic, podem crear un nou `BitmapFont(Boolean flip)` i fer la crida al seu mètode `draw(Batch batch, String text, int x, int y)`, i proporcionar-li el `batch`, el text i la coordenada on es mostrarà. Un exemple podria ser:

```
1 BitmapFont font = new BitmapFont(true);
2 font.draw(batch, "GameOver", 10, 10);
```

El paràmetre `true` indica que estem fent servir un sistema *Y-down* i volem girar el text. Si posem aquest text entre `batch.begin()` i `batch.end()` del mètode `render` de la classe `GameScreen` apareixerà el missatge "GameOver" quan col·lideixen contra un asteroide (figura 2.17).

Figura 2.17. Text GameOver en col·lidir amb un asteroide



Podem modificar els paràmetres del text amb mètodes de l'objecte `BitmapFont`. Per exemple, podem canviar el color amb `setColor(Color color)` o `setColor(float r, float g, float b, float a)` i canviar-li la mida al text amb `getData().setScale(float xy)` o `getData().scale(float xy)`. Cal anar amb compte perquè els dos mètodes d'escalat no són iguals, `setScale` defineix la mida, i `scale` aplica una transformació sobre l'escala actual. Per exemple, si posem `setScale(0.25f)` l'objecte serà un 25% de la seva mida original, és a dir, més petit. En canvi, si fem un `scale(0.25f)` estarem fent l'objecte un 25% més gran respecte la mida que tenia.

Per exemple, el codi:

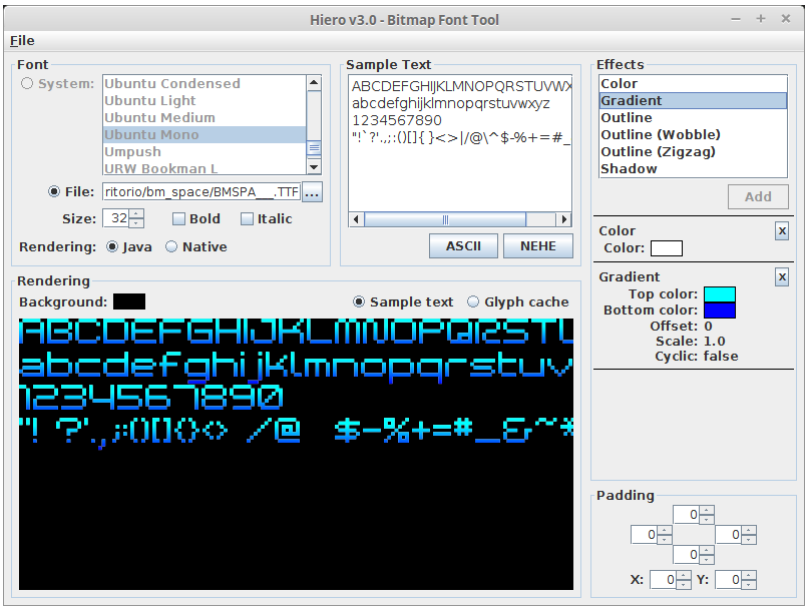
```
1 font.getData().setScale(0.75f);
2 font.setColor(205f/255f, 220f/255f, 57f/255f, 1);
```

Aquest codi fa que el text sigui un 75% de la seva mida original i defineix el color hexadecimal `#CDDC39`, que és: `CD` de vermell (205 en decimal), `DC` de verd (220 en decimal) i `39` de blau (57 en decimal) sense cap tipus de transparència. Dividim entre `FF` (el número màxim amb 2 bits d'hexadecimal que és 255 en decimal) per obtenir el percentatge de cada color.

Si volem personalitzar la font, des de LibGDX ens proporcionen l'eina *Hiero*. La podem instal·lar des de l'assistent de *setup* del projecte (dins del paquet *tools*) o la podem descarregar de: https://github.com/libgdx/libgdx/wiki/Hiero. Anem al directori de descàrrega i executem la següent comanda:

```
ubuntu@ubuntu ~/Descargas$ java –jar hiero.jar
```

Figura 2.18. Eina Hiero configurada



Des de l'aplicació podem seleccionar qualsevol font del sistema o seleccionar un fitxer TTF. Als recursos disposareu de la font *BM_space* per descarregar. Seleccioneu-la des de l'aplicació i proveu-ne diferents configuracions. Per a l'exemple afegirem un gradient fent clic en *Gradient* i a *Add*. Podeu personalitzar els colors del gradient al vostre gust. Podeu veure la captura de l'aplicació en la figura figura 2.18

Una vegada configurada la font, fem clic a *File/Save BMFont Files(text)...* i la guardem en el directori *assets/fonts/* del projecte amb el nom de **space.fnt**.

Després, cal carregar i configurar la font des de la classe `AssetManager`, obrir-la i afegir-hi una variable estàtica per a la font:

```
// Font
    public static BitmapFont font;
```

I al final del mètode load() el següent codi:

```
/****************************** Text ***********************************/
        // Font space
```

```
3         FileHandle fontFile = Gdx.files.internal("fonts/space.fnt");
4         font = new BitmapFont(fontFile, true);
5         font.getData().setScale(0.4f);
```

Si eliminem el text "GameOver" que havíem escrit abans i afegim la següent línia:

```
1 AssetManager.font.draw(batch, "GameOver", 10, 10);
```

S'escriurà el text amb la nova font, però mesurant la meitat que l'original.

El problema el tenim en intentar centrar el text. Podríem pensar que la línia `AssetManager.font.draw(batch, ''GameOver'', Settings.GAME_WIDTH/2, Settings.GAME_HEIGHT/2)` mostrarà el text centrat però no és així, ja que amb les coordenades estem especificant la cantonada superior esquerra d'allà on anirà el text. Per calcular les coordenades necessitem conèixer l'amplada i l'alçada del text. En versions anteriors de LibGDX estava disponible el mètode `BitmapFont.getBounds(String text)` que ens permetia saber les dimensions d'un text però ara no el trobem disponible. Si volem saber les dimensions d'un text haurem de recórrer a un objecte `GlyphLayout`.

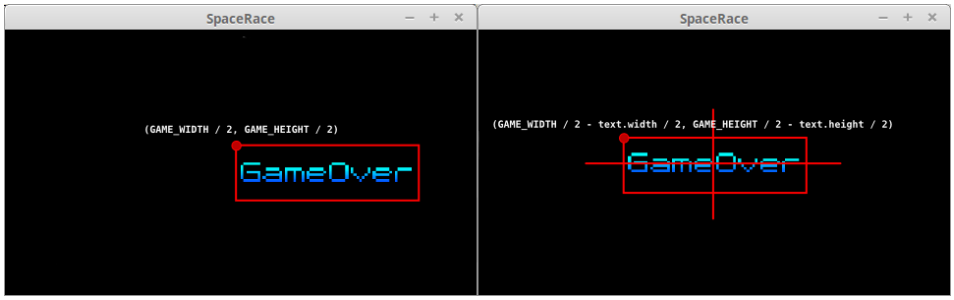
De `GlyphLayout` ens interessa el mètode `setText` que ens demanarà dos paràmetres: la font del text per poder-ne calcular les dimensions i el text. Una vegada crear el *layout* i una vegada li hem assignat el text podrem accedir a les propietats `width` i `height` per saber les dimensions del text.

Després cal centrar el text "GameOver". Per a això hem de crear a `GameScreen` la variable `private GlyphLayout textLayout;` que iniciarem al constructor amb `textLayout = new GlyphLayout();` i assignarem el text "GameOver" amb: `textLayout.setText(AssetManager.font, ''GameOver'');`. Reemplacem la línia `AssetManager.font.draw(batch, ''GameOver'', Settings.GAME_WIDTH/2, Settings.GAME_HEIGHT/2);` de `render` per:

```
1 AssetManager.font.draw(batch, textLayout, Settings.GAME_WIDTH/2 - textLayout.
      width/2, Settings.GAME_HEIGHT/2 - textLayout.height/2);
```

Obtenim el punt central de la pantalla i restem la meitat de l'amplada del text i la meitat de l'alçada per tal que el text quedi centrat.

FIGURA 2.19. Configuracions de posició del text



Sempre que vulguem modificar el contingut del text no hem de fer més que cridar al mètode `setText`.

Objecte label

Els objectes `com.badlogic.gdx.scenes.scene2d.ui.Label` són descendents de `com.badlogic.gdx.scenes.scene2d.ui.Widget` que a la vegada són descendents de `com.badlogic.gdx.scenes.scene2d.Actor`, és a dir, els objectes `label` són a la vegada objectes `actor`.

El fet que siguin `actor` fa que puguem afegir-los a un `stage` i modificar-los de la mateixa manera que fèiem amb la resta d'objectes.

Per a la creació d'un objecte `label` necessitem disposar d'un objecte `LabelStyle` amb la `BitmapFont` i el `color` en què es mostrarà l'etiqueta. Per exemple:

```
1 Label.LabelStyle textStyle = new Label.LabelStyle(AssetManager.font, null);
```

Posem el color a `null` per no modificar-lo. Una vegada tenim l'estil ja podem crear el `label` (text i estil). Per mostrar-lo per pantalla només haurem d'afegir-lo com a `actor` en l'objecte `stage`:

```
1 Label textLbl = new Label("SpaceRace", textStyle);
2 textLbl.setName("lbl");
3 textLbl.setPosition(Settings.GAME_WIDTH/2 - textLbl.getWidth()/2,  Settings.
      GAME_HEIGHT/2 - textLbl.getHeight()/2);
4 stage.addActor(textLbl);
```

Aquest codi crearia el text "SpaceRace" i el posicionaria al centre de la pantalla. Per treure'l hauríem d'eliminar l'actor de l'`stage` amb:

```
1 stage.getRoot().findActor("lbl").remove();
```

2.3.8 Preferències

Tant en les aplicacions d'escriptori com en les de qualsevol plataforma necessitem guardar informació de manera persistent. La manera d'emmagatzemar informació amb LibGDX és molt similar a com es pot fer amb Android.

Per a la gestió de les preferències es necessita un objecte del tipus `com.badlogic.gdx.Preferences`: `Preferences prefs;` per exemple. Aquest objecte l'inicialitzarem indicant un nom de preferències, per exemple: `prefs = Gdx.app.getPreferences(‘‘prefs’’);`. A partir d'aquest moment ja podrem guardar dades fent servir els mètodes `putString`, `pubInteger`, `putBoolean`, `putFloat` i `putLong` o recollir-les amb els corresponents `getString`, `getInteger`, `getBoolean`, `getFloat` o `getLong`.

> Si volem que la informació sigui persistent entre diverses execucions del programa hem de cridar al mètode `flush()` de les preferències.

2.3.9 Actions

Els objectes `actor` són elements molt importants en la programació de jocs amb LibGDX ja que qualsevol objecte pot ser un `actor` de la nostra escena i, per tant, tindrà una sèrie de propietats i mètodes disponibles. A més, LibGDX ens permetrà aplicar `actions` a tots els objectes descendents d'`actor`. Aquestes accions ens proporcionaran una gran versatilitat i ens permetran canviar propietats de l'element d'una forma senzilla.

> Per tal d'aplicar accions a un objecte `label`, tot i ser descendent de la classe `actor`, aquest ha d'estar a dins d'un objecte `table` (si hi ha més elements) o `containner` (si volem aplicar-les solament a l'etiqueta) i les propietats s'han d'aplicar sobre aquest objecte pare. L'estratègia a seguir serà la següent: creem l'etiqueta, creem el contenidor, creem les accions i les apliquem sobre el contenidor. Un `container` hereta de la classe `WidgetGroup`, que a la vegada hereta de `group` i aquesta de la classe `actor`.

El llistat de totes les accions possibles el podeu trobar a la documentació de Package com.badlogic.gdx.scenes.scene2d.actions. Entre les accions més destacades trobem:

- `AlphaAction`: modifica el valor d'*alpha* de l'actor.
- `ColorAction`: canvia el color de l'actor.
- `MoveToAction`: mou l'actor a una determinada posició.
- `MoveByAction`: mou l'actor uns valors determinats respecte la posició actual.
- `RotateToAction`: gira l'actor a un angle concret.
- `RotateByAction`: gira l'actor un angle determinat respecte l'angle actual.
- `ScaleToAction`: escala l'objecte a unes mides concretes.
- `ScaleByAction`: escala l'objecte unes mides determinades respecte la mida actual.
- etc.

I podem utilitzar accions a les pròpies accions, i permetre:

- `ParallelAction`: permet executar accions de manera paral·lela.
- `SequenceAction`: s'executen les accions una rere l'altra.
- `RepeatAction`: per repetir les accions diverses vegades. Serà útil la variable `RepeatAction.FOREVER`.

- `DelayAction`: endarrereix l'execució de l'acció un determinat temps.

> Per poder aplicar una acció a un `actor` que té una classe pròpia és necessari que el mètode `act(float delta)` faci una crida a `super.act(delta)`.

Canviarem l'animació dels asteroides per fer-la mitjançant accions. Així, necessitarem definir una acció per canviar l'angle de l'asteroide i que aquesta acció es faci de manera infinita: `RotateByAction` serà l'encarregada de fer la rotació i `RepeatAction` farà que aquesta es repeteixi infinitament.

Comprovem, ara, que tant les classes `ScrollHandler`, `scrollable` i `asteroid` tenen la sentència `super.act(delta)` en el seu mètode `act()` i en cas de no ser així les afegim. Una vegada fet, en el constructor de l'asteroide crearem les accions i les aplicarem a l'actor.

Afegim l'acció de rotació:

```
1 // Rotació
2         RotateByAction rotateAction = new RotateByAction();
3         rotateAction.setAmount(-90f);
4         rotateAction.setDuration(0.2f);
```

Creem l'acció i diem que giri 90º en sentit antihorari en 0.2 segons. Si volem que es faci de manera infinita, haurem de crear l'acció `RepeatAction`:

```
1 // Accio de repetició
2         RepeatAction repeat = new RepeatAction();
3         repeat.setAction(rotateAction);
4         repeat.setCount(RepeatAction.FOREVER);
```

Creem l'acció, l'assignem a l'acció de rotació i li diem que es repeteixi per sempre. Ja sols ens queda aplicar-la a l'objecte `actor` amb:

```
1 this.addAction(repeat);
```

Li assignem a l'objecte actual (qualsevol `asteroid` que creem) l'acció de repetició. Per dibuixar ara l'element haurem de canviar la línia del mètode `draw()` perquè dibuixi un sol `TextureRegion` amb les seves propietats:

```
1 batch.draw(AssetManager.asteroid[0], position.x, position.y, this.getOriginX(),
      this.getOriginY(), width, height, this.getScaleX(), this.getScaleY(),
     this.getRotation());
```

Si executem l'aplicació veurem que passa una cosa estranya, els asteroides no giren com esperàvem. Això és perquè el seu punt d'origen està definit en la cantonada superior-esquerra. Per solucionar aquest problema haurem de definir l'origen i posar-lo en la part central de la imatge, això ho farem amb `this.setOrigin(width/2, height/2);`. Cada cop que canviem les dimensions dels asteroides cal fer també aquest canvi, per tant crearem un mètode `setOrigin()` que realitzarà l'acció. A més, corregim ara un petit defecte que fa que sembli que la rotació dels *sprites* no és perfecta sumant-li 1 a `width/2` .

Afegim el mètode:

```
public void setOrigin() {

        this.setOrigin(width/2 + 1, height/2);

    }
```

I el cridem des del constructor i des del mètode `reset` ja que s'haurà de recalcular el punt central a causa del canvi de dimensions.

Eliminem tota referència a `runTime` perquè no són necessàries i definim una variable aleatòria (`assetAsteroid`) que agafi els diferents *sprites* de l'*array* d'asteroides. El codi de la classe quedarà així:

```
package cat.xtec.ioc.objects;

import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.math.Circle;
import com.badlogic.gdx.math.Intersector;
import com.badlogic.gdx.scenes.scene2d.actions.RepeatAction;
import com.badlogic.gdx.scenes.scene2d.actions.RotateByAction;

import java.util.Random;

import cat.xtec.ioc.helpers.AssetManager;
import cat.xtec.ioc.utils.Methods;
import cat.xtec.ioc.utils.Settings;

public class Asteroid extends Scrollable {

    private Circle collisionCircle;

    Random r;

    int assetAsteroid;

    public Asteroid(float x, float y, float width, float height, float velocity
        ) {
        super(x, y, width, height, velocity);

        // Creem el cercle
        collisionCircle = new Circle();

        /* Accions */
        r = new Random();
        assetAsteroid = r.nextInt(15);

        setOrigin();

        // Rotació
        RotateByAction rotateAction = new RotateByAction();
        rotateAction.setAmount(-90f);
        rotateAction.setDuration(0.2f);

        // Acció de repetició
        RepeatAction repeat = new RepeatAction();
        repeat.setAction(rotateAction);
        repeat.setCount(RepeatAction.FOREVER);

        this.addAction(repeat);

    }

    public void setOrigin() {

        this.setOrigin(width/2 + 1, height/2);
```

```
52
53      }
54
55      @Override
56      public void act(float delta) {
57          super.act(delta);
58
59          // Actualitzem el cercle de col·lisions (punt central de l'asteroide i
                el radi.
60          collisionCircle.set(position.x + width / 2.0f, position.y + width / 2.0
                f, width / 2.0f);
61
62
63      }
64
65      @Override
66      public void reset(float newX) {
67          super.reset(newX);
68          // Obtenim un número aleatori entre MIN i MAX
69          float newSize = Methods.randomFloat(Settings.MIN_ASTEROID, Settings.
                MAX_ASTEROID);
70          // Modificarem l'alçada i l'amplada segons l'aleatori anterior
71          width = height = 34 * newSize;
72          // La posició serà un valor aleatori entre 0 i l'alçada de l'aplicació
                menys l'alçada de l'asteroide
73          position.y =  new Random().nextInt(Settings.GAME_HEIGHT - (int) height)
                ;
74          setOrigin();
75      }
76
77
78      @Override
79      public void draw(Batch batch, float parentAlpha) {
80          super.draw(batch, parentAlpha);
81          batch.draw(AssetManager.asteroid[assetAsteroid], position.x, position.y
                , this.getOriginX(), this.getOriginY(), width, height, this.
                getScaleX(), this.getScaleY(), this.getRotation());
82      }
83
84      // Retorna true si hi ha col·lisió
85      public boolean collides(Spacecraft nau) {
86
87          if (position.x <= nau.getX() + nau.getWidth()) {
88              // Comprovem si han col·lisionat sempre que l'asteroide estigui a
                    la mateixa alçada que l'spacecraft
89              return (Intersector.overlaps(collisionCircle, nau.getCollisionRect
                    ()));
90          }
91          return false;
92      }
93  }
```

Podem crear i concatenar instruccions fent servir la classe `com.badlogic.gdx.scenes.scene2d.actions.Actions`, la documentació de la qual podeu trobar a https://libgdx.badlogicgames.com/nightlies/docs/api/com/badlogic/gdx/scenes/scene2d/actions/Actions.html. Això ens permetrà crear fàcilment accions.

La línia:

```
1   this.addAction(Actions.repeat(RepeatAction.FOREVER, Actions.rotateBy(-90f, 0.2f
        )));
```

és l'equivalent al codi anterior:

```
// Rotació
        RotateByAction rotateAction = new RotateByAction();
        rotateAction.setAmount(-90f);
        rotateAction.setDuration(0.2f);

        // Acció de repetició
        RepeatAction repeat = new RepeatAction();
        repeat.setAction(rotateAction);
        repeat.setCount(RepeatAction.FOREVER);

        this.addAction(repeat);
```

Podem concatenar les accions i fer coses com la següent:

```
spacecraft.addAction(Actions.repeat(RepeatAction.FOREVER, Actions.sequence(
     Actions.moveTo(0, 0), Actions.moveTo(50, 100, 2),Actions.moveTo(100, 50,
     5),Actions.moveTo(0, 0, 1), Actions.delay(2))));
```

Aquest codi faria de forma ininterrompuda (`RepeatAction.FOREVER`) la següent seqüència:

1. Posaria l'objecte en la posició (0,0).
2. Desplaçaria l'objecte durant 2 segons fins a la posició (50,100).
3. Després el desplaçaria durant 5 segons a la posició (100,50).
4. El tornaria a desplaçar a la posició (0,0) en 1 segon.
5. Esperaria 2 segons i tornaria a començar des del punt 1.

Cal consultar la documentació, hi ha molts mètodes útils que ens facilitaran la creació d'accions minimitzant la complexitat del codi.

2.3.10 Hit

Tal com tenim definit l'`InputProcessor` (`InputHandler`) no ens permet identificar sobre quins actors hem fet clic. Aquesta funcionalitat ens seria molt útil per tal d'afegir funcions als objectes o per crear elements amb els quals l'usuari pot interaccionar com per exemple botons (un botó seria un actor igual que la nau o els asteroides).

Per controlar aquesta possibilitat, haurem de tenir en compte que:

- Hem de definir els límits (*bounds* en anglès) de l'objecte cada cop que es mou.
- Hem de definir l'`actor` com a `touchable`.
- Quan fem clic sobre la pantalla, comprovarem si s'ha tocat algun element.

Per poder controlar quan es fa clic sobre la nau, anem al constructor d'`spacecraft` i afegim el codi:

```
// Per a la gestió de hit
        setBounds(position.x, position.y, width, height);
        setTouchable(Touchable.enabled);
```

I al final del mètode `act`:

```
setBounds(position.x, position.y, width, height);
```

Una vegada configurat l'objecte `spacecraft` cal anar a la classe `InputHandler` per controlar quan fa clic l'usuari. Creem les variables d'instància `private Vector2 stageCoord;` i `private Stage stage;`. Obtenim l'`stage` en el constructor amb `stage = screen.getStage();` i en el mètode `touchDown` afegim el següent codi:

```
stageCoord = stage.screenToStageCoordinates(new Vector2(screenX, screenY));
        Actor actorHit = stage.hit(stageCoord.x, stageCoord.y, true);
        if (actorHit != null)
            Gdx.app.log("HIT", actorHit.getName());
```

Convertim les coordenades de la pantalla a les de l'`stage` ja que per exemple a l'aplicació d'escriptori li hem duplicat les dimensions i l'aplicació de dispositius mòbils es redimensionarà per adaptar-se a la pantalla del dispositiu. Guardem l'`actor` que ha estat pressionat en les coordenades de la pantalla i si no és `null` mostrem un missatge de *log* amb el seu nom en cas de tenir-lo. El darrer paràmetre del mètode `hit` defineix si es respecta la `touchability` de l'objecte, és a dir, només mirarà actors que tinguin el `setTouchable` en `Touchable.enabled`.

Si executem l'aplicació i fem clic en la nau veurem missatges "HIT: spacecraft" en el *log* de l'Android Studio.

2.3.11 Configuració de l'aplicació d'Android

Si volem reduir el consum energètic generat pels sensors d'Android i millorar l'experiència de l'usuari, cal fer alguns canvis. La configuració de l'aplicació d'Android la farem des del fitxer del subprojecte *android/java/cat.xtec.ioc.android/AndroidLauncher*.

En la nostra aplicació utilitzarem ni l'acceleròmetre ni la brúixola així que afegirem entre `AndroidApplicationConfiguration config = new AndroidApplicationConfiguration();` i `initialize(new SpaceRace(), config);` les línies:

```
config.useAccelerometer = false;
config.useCompass = false;
```

D'aquesta manera evitarem el consum generat per aquests sensors.

Per millorar l'experiència de l'usuari en sistemes Android, configurarem l'aplicació per tal que mai s'apagui la pantalla i també ocultarem (en cas de tenir-los disponibles) els botons *software* de *home*, *back* i *recents* posant l'aplicació en el mode *immersive*.

El codi de la classe `AndroidLauncher` quedarà de la següent manera:

```
1  package cat.xtec.ioc.android;
2
3  import android.os.Bundle;
4
5  import com.badlogic.gdx.backends.android.AndroidApplication;
6  import com.badlogic.gdx.backends.android.AndroidApplicationConfiguration;
7  import cat.xtec.ioc.SpaceRace;
8
9  public class AndroidLauncher extends AndroidApplication {
10   @Override
11   protected void onCreate (Bundle savedInstanceState) {
12     super.onCreate(savedInstanceState);
13     AndroidApplicationConfiguration config = new
           AndroidApplicationConfiguration();
14     // Deshabilitem els sensors que hem de fer servir
15     config.useAccelerometer = false;
16     config.useCompass = false;
17     // Impedim que s'apagui la pantalla
18     config.useWakelock = true;
19     // Posem el mode immersive per ocultar botons software
20     config.useImmersiveMode = true;
21
22     // Apliquem la configuració
23     initialize(new SpaceRace(), config);
24   }
25 }
```