U.D. 5 Gràfics en Android

5.1 Classes per a gràfics en Android:

- 5.1.1 Canvas
- 5.1.2 Paint
- 5.1.3 Path
- 5.1.4 Drawable
  - 5.1.4.1 BitmapDrawable
  - 5.1.4.2 GradienDrawable
  - 5.1.4.3 TransitionDrawable
  - 5.1.4.4 ShapeDrawable
  - 5.1.4.5 AnimationDrawable.

5.2 Creació d'una vista en un fitxer independent.

5.3 Crear l'activitat principal de la nostra aplicació.

5.4 Representació de gràfics vectorials en la nostra aplicació.

5.5 Animacions: Animacions de vistes, Animacions de propietats

Android ens proporciona a través del seu API gràfica una potent i variada col·lecció de funcions que poden cobrir pràcticament qualsevol necessitat gràfica d'una aplicació. Podem destacar la manipulació d'imatges, gràfics vectorials, animacions, treball amb text o gràfics en 3D.

En aquest capítol s'introdueixen alguna de les característiques més significatives de l'API gràfica d'Android. Ens centrarem en l'estudi de les classes utilitzades per al desenvolupament de gràfics en 2D. En un capítol anterior hem descrit com s'utilitzen les vistes com a element constructiu per al disseny de la interfície d'usuari. Disposàvem d'una àmplia paleta de vistes. No obstant això, en moltes ocasions serà interessant dissenyar les nostres pròpies vistes. En aquest capítol veurem com fer-ho.

Tractarem d'aplicar allò après en un exemple concret, la representació de gràfics a Asteroides. S'utilitzaran dues tècniques alternatives: els gràfics en mapa de bits i en format vectorial. A la fi del capítol es descriuen les eines disponibles a Android per realitzar animacions. En concret es descriuran les animacions Tween i les animacions de propietats. Per descomptat, resultaria impossible abastar totes les seves funcions per a gràfics, de manera que es recomana al lector que consulti la documentació d'Android per a una descripció detallada.

5.1 Classes per a gràfics a Android

5.1.1 Canvas

La classe Canvas representa una superfície on podem dibuixar. Disposa d'una sèrie de mètodes que ens permeten representar línies, cercles, text, etc. Per dibuixar en un Canvas necessitarem un pinzell (Paint) on definirem el color, gruix de traç, transparència, etc. També podrem definir una matriu de 3x3 (Matrix) que ens permetrà transformar coordenades aplicant una translació, escala o rotació. Una altra opció consisteix a definir una àrea coneguda com Clip, de manera que els mètodes de dibuix afectin només a aquesta àrea.

Vegem a continuació alguns mètodes de la classe Canvas. No es tracta d'un llistat exhaustiu i molts d'aquests mètodes poden treballar amb altres paràmetres, per tant es recomana consultar la documentació de l'SDK per a una informació més detallada.

https://developer.android.com/reference/android/graphics/Canvas?hl=fr

Per dibuixar figures geomètriques:

```
drawCircle(float cx, float cy, float radio, Paint pinzell)
drawOval(RectF ovalo, Paint pinzell)
drawRect(RectF rect, Paint pinzell)
drawPoint(float x, float y, Paint pinzell)
drawPoints(float[] pts, Paint pinzell)
```

Per dibuixar línies i arcs:

```
drawLine(float iniX, float iniY, float finX, float finY, Paint
pinzell)
drawLines(float[] puntos, Paint pinzell)
drawArc(RectF ovalo, float iniAnglulo, float anglulo, boolean
usarCentro,Paint pinzell)
drawPath(Path cami, Paint pinzell)
```

Per dibuixar text:

```
drawText(String texto, float x, float y, Paint pinzell)
drawTextOnPath(String texto, Path cami, float desplazamHor, float
desplazamVert, Paint pinzell)
drawPosText(String texto, float[] posicion, Paint pinzell)
```

Per omplir tot el Canvas (a no ser que hi hagi un Clip definit):

```
drawColor(int color)
drawARGB(int alfa, int rojo, int verde, int azul)
drawPaint(Paint pinzell)
```

Per dibuixar imatges:

```
drawBitmap(Bitmap bitmap, Matrix matriz, Paint pinzell)
```

Si definim un Clip, només es dibuixarà a l'àrea indicada:

```
boolean clipRect(RectF rectangulo)
boolean clipRegion(Region region)
boolean clipPath(Path cami)
```

Definir una matriu de transformació (Matrix) ens permetrà transformar coordenades aplicant una translació, escala o rotació.

```
setMatrix(Matrix matriz)
Matrix getMatrix()
concat(Matrix matriz)
translate(float despazX, float despazY)
scale(float escalaX, float escalaY)
rotate(float grados, float centroX, float centroY)
skew(float despazX, float despazY)
```

Per saber la mida del Canvas:

```
int getHeight()
int getWidth()
```

Exemple

```
public class GraficsActivity extends AppCompatActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(new ExempleView(this));
    }

    public class ExempleView extends View {
        public ExempleView (Context context) {
            super(context);
        }
        @Override
        protected void onDraw(Canvas canvas) {
            //Dibuixar aquí
        }
    }
}
```

Comença amb la creació d'una Activity, però en aquest cas, l'objecte View que associem a l'activitat mitjançant el mètode setContentView () no està definit mitjançant XML. Per contra, és creat mitjançant codi a partir del constructor de la classe *ExempleView*.

La classe *ExempleView* estén *View*, modificant només el mètode *onDraw*() responsable de dibuixar la vista. Al llarg de el capítol anirem veient exemples de codi que poden ser escrits en aquest mètode.

Si executes l'aplicació no s'observarà res. Aprendrem a dibuixar en aquest *Canvas* utilitzant l'objecte Paint descrit en la següent secció.

5.1.2 Paint

Com acabem de veure, la majoria dels mètodes de la classe *Canvas* utilitzen un paràmetre de tipus *Paint*. Aquesta classe ens permet definir el color, estil o gruix del traçat d'un gràfic vectorial.

1. Escriu dins *OnDraw* de l'exercici anterior el codi següent:

```
Paint pinzell = new Paint();
pinzell.setColor(Color.BLUE);
pinzell.setStrokeWidth(8);
pinzell.setStyle(Paint.Style.STROKE);
canvas.drawCircle(150, 150, 100, pinzell);
```

Prova altres mètodes de dibuix, com *drawLine ()* o *drawPoint ()*.

Definició de colors

Android representa els colors utilitzant enters de 32 bits. Aquests bits són dividits en 4 camps de 8 bits: alfa, vermell, verd i blau (ARGB, usant les inicials en anglès). A l'estar format cada component per 8 bits, podrà prendre 256 valors diferents.

Els components vermell, verd i blau són utilitzats per definir el color, mentre que el component alfa defineix el grau de transparència pel que fa a el fons. Un valor de 255 significa un color opac i a mesura que anem reduint el valor el dibuix s'anirà fent transparent.

Per definir un color tenim les següents opcions:

```
int color;
color = Color.BLUE;                    //Blau opac
color = Color.argb(127, 0, 255, 0);    //Verd transparent
color = 0x7F00FF00;                    //Verd transparent
color = getResources().getColor(R.color.color_cercle);
```

Per aconseguir una adequada separació entre programació i disseny, es recomana utilitzar l'última opció. És a dir, no definir els colors directament en codi, sinó utilitzar el fitxer de recursos *res / values / colors.xml*:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<color name="color_cercle">#7fffff00</color>
</resources>
```

Exemple:

1. Modifica l'exercici anterior, afegint al final de *OnDraw* el codi següent:

```
pinzell.setColor(Color.argb(127,255,0,0));
canvas.drawCircle(150, 250, 100, pinzell);
```

2. Observa com el color vermell seleccionat es mescla amb el color de fons. Prova altres valors d'alfa.
3. Reemplaça la primera línia que acabes d'introduir per

```
pinzell.setColor(0x7FFF0000);
```

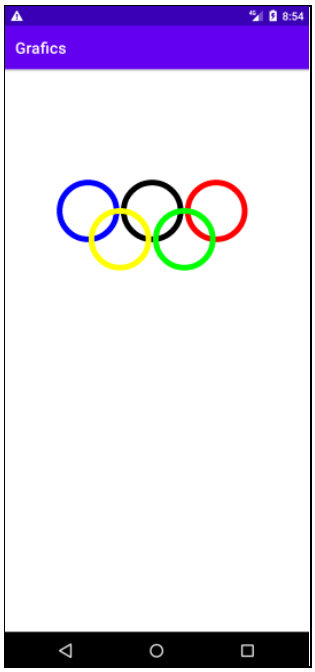
4. Observa com el resultat és idèntic.
5. Defineix en el fitxer *res / values / colors.xml* un nou color utilitzant el següent codi:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
        <color name="color_cercle">#7fffff00</color>
</resources>
```

6. Modifica l'exemple anterior perquè s'utilitzi aquest color definit en els recursos:

```
pinzell.setColor(getResources().getColor(R.color.color_cercle));
```

Exercici 5.1: Escriure el codi perquè a la pantalla del vostre mòbil es mostri la següent imatge:



5.1.3 Path

La classe Path permet definir un traçat a partir de segments de línia i corbes. Un cop definit pot ser dibuixat amb canvas.drawPath (Path, Paint). Un Path també pot ser utilitzat per a dibuixar un text sobre el traçat marcat.

Exemple: Ús de la classe Path

1. Substitueix dins *OnDraw* de l'exercici anterior el codi següent:

```
Path cami = new Path();
cami.addCircle(150, 150, 100, Path.Direction.CCW);
canvas.drawColor(Color.WHITE);
Paint pinzell = new Paint();
pinzell.setColor(Color.BLUE);
pinzell.setStrokeWidth(8);
pinzell.setStyle(Paint.Style.STROKE);
canvas.drawPath(cami, pinzell);
pinzell.setStrokeWidth(1);
pinzell.setStyle(Paint.Style.FILL);
pinzell.setTextSize(20);
pinzell.setTypeface(Typeface.SANS_SERIF);
canvas.drawTextOnPath("Desenvolupament d'aplicacions per a mòbils
amb Android", cami, 10, 40, pinzell);
```

2. El resultat ha de ser:

3. Modifica en la segona línia el paràmetre *Direction.CCW* (contrari a les agulles del rellotge) por *Direction.CW* (a favor de les agulles del rellotge). Observa el resultat.

4. Modifica els paràmetres de *canvas.drawTextOnPath ()* fins que comprenguis el seu significat.

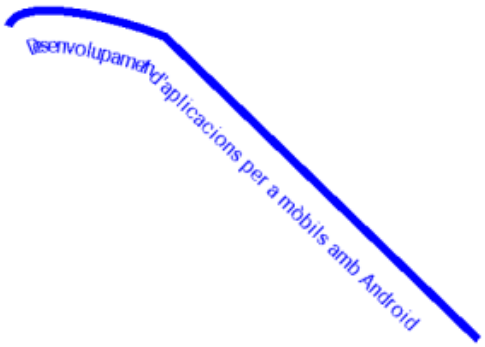
Exercici 5.2 Podries dibuixar el text en el mateix sentit que el de les agulles del rellotge, però per fora del cercle?

Exercici pas a pas: Un exemple de Path més complex

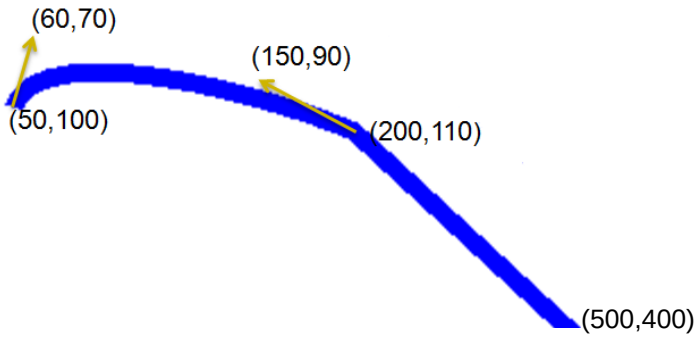
1. Substitueix les dues primeres línies de l'exemple anterior per:

```
Path  cami = new Path();
cami.moveTo(50, 100);
cami.cubicTo(60,70, 150,90, 200,110);
cami.lineTo(500,400);
```

2. El resultat obtingut ha de ser similar a:



El *camí* comença desplaçant-se a les coordenades (50,100). Després introdueix una corba cúbica o Bézier fins a la coordenada (200,110). Una corba Bézier introdueix dos punts de control, el primer (60,70) permet controlar com arrenca la direcció de començament de la corba i el segon (150,90) la direcció de la fi de la corba. Funciona de la següent manera, si traces una recta des del començament de la corba (50,100) fins al primer punt de control (60,70) la corba es traçarà tangencialment a aquesta recta. Finalment, s'afegeix una línia des de les coordenades (200,110) fins a (500,400).



5.1.4 Drawable

https://developer.android.com/guide/topics/resources/drawable-resource

La classe Drawable és una abstracció que representa "una cosa que pot ser dibuixat". Aquesta classe s'estén per definir gran varietat d'objectes gràfics més específics. Molts d'ells poden ser definits com a recursos usant fitxers XML. Entre ells tenim els següents:

- **BitmapDrawable**: Imatge basada en un fitxer gràfic (PNG o JPG). Etiqueta XML <*bitmap*>.

Dins un .xml podem introduir un <ImageView>

```
<ImageView
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:src="@drawable/myimage" />
```

I dins el MainActivity el podem anar a cercar utilitzant getResources():

```
Resources res = getResources();
Drawable drawable = ResourcesCompat.getDrawable(res,
R.drawable.myimage, null);
```

- **VectorDrawable**: Igual que *ShapeDrawable*, permet crear gràfics de forma vectorial. A diferència de *ShapeDrawable*, podem definir els gràfics en XML usant un format basat en SVG (Scalable Vector Graphics). Aquest *Drawable* està disponible des Android v5.0 (API 21), tot i això, s'ha inclòs a la llibreria de compatibilitat v7. Per tant, podrem utilitzar-lo des de l'API 7.
- **LayerDrawable**: Conté un array de *Drawable* que es visualitzen segons l'ordre de l'array. L'índex major de l'array és el que es representa a sobre. Cada *Drawable* pot situar-se en una posició determinada. Etiqueta XML <*layer-list*>.
- **StateListDrawable**: Similar a l'anterior però ara podem usar una màscara de bits i podem seleccionar els objectes visibles. Etiqueta XML <*selector*>.
- **GradientDrawable**: Degradat de color que pot ser usat en botons o fons. Etiqueta XML <*gradient*>.
- **TransitionDrawable**: Una extensió de *LayerDrawable* que permet un efecte de fusió entre la primera i la segona capa. Per iniciar la transició cal trucar a *startTransition (int temps)*. Per visualitzar la primera capa cal trucar a *resetTransition ()*. Etiqueta XML <*transition*>.

- **ShapeDrawable**: Permet realitzar un gràfic a partir de primitives vectorials, com a formes bàsiques (cercles, quadrats ...) o traçats (Path). Etiqueta XML <*shape*>. Crear un ShapeDrawable des XML està molt limitat. Les opcions de dibuix es limiten a cercles o quadrats. Per fer un gràfic més complex hem d'usar la classe *Path*, opció només possible des de codi.

- **AnimationDrawable**: Permet crear animacions frame a frame a partir d'una sèrie de objectes *Drawable*. Etiqueta XML <*animation-list*>

També pot ser interessant que facis servir la classe *Drawable* o un dels seus descendents com a base per crear les teves pròpies classes gràfiques.

A més de ser dibuixada, la classe *Drawable* proporciona una sèrie de **mecanismes genèrics** que permeten indicar com ha de ser dibuixada. No tot *Drawable* ha d'implementar tots els mecanismes. Vegem els més importants:

- El mètode ***setBounds (x1, y1, x2, y2)*** permet indicar el rectangle on ha de ser dibuixat. Tot Drawable ha de respectar la mida sol·licitat pel client, és a dir, ha de permetre l'escalat. Podem consultar la mida preferida d'un Drawable mitjançant els mètodes *getIntrinsicHeight ()* i *getIntrinsicWidth ()*.

- El mètode ***getPadding (Rect)*** proporciona informació sobre els marges recomanats per representar continguts. Per exemple, un *Drawable* que intenti ser un marc per a un botó, ha de tornar els marges correctes per localitzar les etiquetes, o altres continguts, a l'interior de el botó.

- El mètode ***setState (int [])*** permet indicar al Drawable en quin estat ha de ser dibuixat, per exemple "amb focus", "seleccionat", etc. Alguns *Drawable* canviaran la seva representació segons aquest estat.

- El mètode ***setLevel (int)*** alguns *Drawable* tenen un nivell associat. Amb aquest mètode podem canviar-ho. Per exemple, un nivell pot ser interpretat com una bateria de nivells o un nivell de progrés. Alguns *Drawable* modificaran la imatge basant-se en el nivell.

Un drawable pot realitzar animacions al ser cridat des de la interfície *Drawable.Callback*. Després implementar aquesta interfície, cal registrar un objecte de la classe creada cridant a *setCallback (Drawable.Callback)*).

Existeixen diverses alternatives per crear una instància de *Drawable*. Pots crear-a partir d'un fitxer d'imatge emmagatzemat en els recursos de el projecte, també pots crear-a partir de el disseny basat en XML o pots crear-a partir de codi.

Veurem amb més detall algunes de les subclasses de Drawable, més endavant.

5.1.4.1 BitmapDrawable

La forma més senzilla d'afegir gràfics a la teva aplicació és incloure'ls en la carpeta *res / drawable* de el projecte. El SDK d'Android suporta els formats PNG, JPG i GIF. El format aconsellat és PNG, encara que si el tipus de gràfic així ho recomana també pots utilitzar JPG. El format GIF està desaconsellat.

Cada gràfic d'aquesta carpeta és associat a un ID de recurs. Per exemple, per al fitxer *la_meva_imatge.png* es crearà el ID *la_meva_imatge*. Aquest ID et permetrà fer referència al gràfic des del codi o des de XML.

Exercici pas a pas: Dibuixar un BitmapDrawable dels recursos

Busca a Internet un fitxer gràfic en codificació png o jpg (els formats gràfics usats per defecte en Android).

1. Renomena el fitxer perquè es digui la_meva_imatge.png o la_meva_imatge.jpg i arrossega'l a *res / drawable*.

2. Declara la variable *laMevaImatge* a la classe *ExempleView* de l'exercici anterior:

```
private Drawable laMevaImatge;
```

3. Escriu les següents tres línies dins el constructor d'aquesta classe:

```
laMevaImatge=ContextCompat.getDrawable(GraficsActivity.this,R.drawable
.la_meva_imatge);
laMevaImatge.setBounds(30,30,400,400);
```

4. Afegeix aquesta instrucció dins onDraw

```
laMevaImatge.draw(canvas);
```

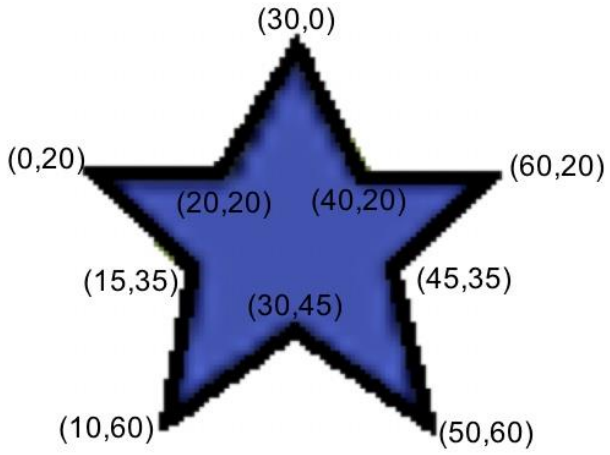
5.1.4.2 VectorDrawable

A partir d'Android Lollipop (v5.0, API 21) s'incorpora la possibilitat de definir *drawables* de forma vectorial utilitzant un format basat en SVG (Scalable Vector Graphics). També es disposa d'una llibreria de compatibilitat que ens permet usar gràfics vectorials en versions anteriors. El gran avantatge dels gràfics vectorials és que poden ser reescalats sense perdre definició. Només necessites un petit fitxer per definir línies i corbes, després podràs representar-lo en la mida desitjada. Resulta molt més senzill que dissenyar diferents imatges en mapa de bits per a diferents densitats.

Per definir un *VectorDrawable* amb XML crea un nou fitxer dins *res / drawable*. Per exemple:

```
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="60dp"
    android:height="60dp"
    android:viewportHeight="60"
    android:viewportWidth="60">
    <path
        android:fillColor="@color/design_default_color_primary"
        android:strokeColor="#000000"
        android:strokeWidth="2"
        android:pathData="M0 20 L20 20 L30 0 L40 20 L60 20 L45 35 L50
60 L30 45 L10 60 L15 35 Z" />
</vector>
```

Primer es defineix l'ample i alt amb que volem que aparegui el *drawable* i després l'ample i alt amb el que treballarem per definir-lo. Amb l'etiqueta <*path*> definim el seu traçat, començam definint el color amb el que l'omplirem, color de traç i gruix de traç. Finalment s'indica les comandes que permeten dibuixar l'estrella mostrada a continuació. M 0 20 mou el punter de dibuix fins a aquestes coordenades. L20 20 dibuixa una línia fins a aquestes coordenades, ... Finalment, Z tanca el traçat actual.



Per a més detalls sobre aquestes comandes SVG consultar

http://www.w3.org/TR/SVG11/paths.html#PathData

VectorDrawable no suporta algunes característiques de SVG, com els degradats. Consulta la documentació per veure totes les possibilitats tant en Java com a XML

https://developer.android.com/reference/android/graphics/drawable/VectorDrawable.html?hl=es

Afegir gràfics vectorials amb Vector Asset.

En el primer exercici d'aquesta secció hem vist com crear *VectorDrawable* mitjançant el seu codi XML. A la pràctica això no se sol així així. Quan necessitem un gràfic podem buscar-lo en una col·lecció, o si hem de generar nosaltres farem servir un editor de gràfics vectorials. Per ajudar-nos en aquest treball, Android Studio incorpora l'eina Vector Asset. Vegem com s'usa en aquest exercici.

Exercici 5.3: Utilitzar l'eina Vector Asset

1. A Android studio selecciona File / New / Vector Asset.
2. Selecciona *Clip Art*, i prem a la icona. Pots accedir a una col·lecció d'icones. Selecciona un de molt senzill, per exemple, *arrow downward*. Prem Next i després *Finish*.
3. Modifica l'exercici anterior perquè es visualitzi aquest gràfic.
4. Obre el fitxer XML que acabes d'afegir. Observa amb què poca informació s'ha definit el gràfic. Canvia el color i algun valor del path. Verifica que el resultat és l'esperat.
5. Busca a Internet i fitxer amb extensió SVG (Scalable Vector Graphics) o PSD (Adobe Photoshop).
6. Obre de nou Vector Asset i selecciona *Local file* (SVG, PSD). En el camp *Name* indica un nom de recurs i en *Path* el fitxer que acabes de descarregar. Si ho desitges també pots canviar la mida que tindrà per defecte la imatge i el grau d'opacitat.
7. El format XML usat en *VectorDrawable* no suporta totes les característiques disponibles en els formats SVG i PSD. En el quadre *Errors*, que trobaràs a la part inferior, es mostrarà una llista amb els problemes en la conversió.
8. Observa com converteix el fitxer al format utilitzat en Android.

9. El gran avantatge d'usar gràfics vectorials és que ocupen molt poc espai, podem canviar fàcilment els colors i no hem de preocupar-nos de preparar diferents recursos per a diferents resolucions. Com a inconvenient els gràfics vectorials poden costar més recursos per ser generats, especialment si són complexos i tenen corbes.

5.1.4.3 GradientDrawable

També podem definir en XML altre tipus de *Drawables* com *GradientDrawable*. Per exemple, el següent fitxer defineix un degradat des del color blanc (FFFFFF) a blau (0000ff):

```
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <gradient
        android:startColor="#FFFFFF"
        android:endColor="#0000FF"
        android:angle="270" />
</shape>
```

Exercici 5.4: Definir un *GradientDrawable* pel fons del menú principal d'Asteroides.

1. Obre el projecte Asteroides.

2. Crea el següent fitxer res / drawable / degradat.xml amb un degradat que t'agradi.

3. El més convenient és definir el fons d'una vista en el seu Layout en XML. Introdueix el següent atribut al Layout main.xml.

```
android:background="@drawable/degradat"
```

5.1.4.4 TransitionDrawable

Un *TransitionDrawable* és una extensió de *LayerDrawables* que permet un efecte de fusió entre la primera i la segona capa. Per iniciar la transició cal cridar a *startTransition (int temps)*. Per tornar a visualitzar la primera capa cal cridar a *resetTransition ()* .

Exercici pas a pas: Definir un *TransitionDrawable*

1. Crea un nou projecte amb nom *Transicio*.

2. Crea el següent recurs dins *res / drawable / transition.xml* amb el codi:

```
<?xml version="1.0" encoding="utf-8"?>
<transition
xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/asteroide1"/>
    <item android:drawable="@drawable/asteroide3"/>
</transition>
```

3. Copia els fitxers asteroide1.png i asteroide3.png a la carpeta res / drawable.

4. Reemplaça a l'activitat el mètode *onCreate* pel següent codi:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

```
    ImageView image = new ImageView(this);
    setContentView(image);
    TransitionDrawable transition = (TransitionDrawable)
            ContextCompat.getDrawable(this, R.drawable.transition);
    image.setImageDrawable(transition);
    transition.startTransition(2000);
}
```

5. Si tot funciona correctament, veuràs com la crida a *transition.startTransition (2000)* provoca que la primera imatge es transformi al llarg de dos segons en la segona imatge.

5.1.4.5 ShapeDrawable

Quan vulguis crear un gràfic dinàmicament mitjançant primitives vectorials, una bona opció pot ser utilitzar *ShapeDrawable*. Aquesta classe permet dibuixar gràfics a partir de formes bàsiques. Un *ShapeDrawable* és una extensió de *Drawable*, per tant pots utilitzar tot el que permet *Drawable*.

Exercici pas a pas: Definir un ShapeDrawable.

Vegem un exemple de com utilitzar un objecte ShapeDrawable per crear una vista a mida.

1. Obre el projecte Grafics.

2. A la classe *ExempleView* declara la següent variable:

```
private ShapeDrawable laMevaImatge;
```

3. Afegeix les següents tres línies dins el constructor d'aquesta classe:

```
laMevaImatge = new ShapeDrawable(new OvalShape());
laMevaImatge.getPaint().setColor(0xFF0000FF);
laMevaImatge.setBounds(10,10,310,60);
```

En el constructor, un objecte ShapeDrawable és definit com un oval. Se li assigna un color i es defineixen els seus límits.

4. Escriu la següent línia en el mètode *onDraw*:

```
laMevaImatge.draw(canvas);
```

5. Executa l'aplicació i observa el resultat.

Un ShapeDrawable també es pot definit utilitzant un fitxer.xml amb aquestes opcions:

```
<shape
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape=["rectangle" | "oval" | "line" | "ring"] >

      <corners
      android:radius="integer"
      android:topLeftRadius="integer"
      android:topRightRadius="integer"
      android:bottomLeftRadius="integer"
      android:bottomRightRadius="integer" />
```

```
    <gradient
    android:angle="integer"
    android:centerX="float"
    android:centerY="float"
    android:centerColor="integer"
    android:endColor="color"
    android:gradientRadius="integer"
    android:startColor="color"
    android:type=["linear" | "radial" | "sweep"]
        android:useLevel=["true" | "false"] />

    <padding
    android:left="integer"
    android:top="integer"
    android:right="integer"
    android:bottom="integer" />

    <size
    android:width="integer"
    android:height="integer" />

    <solid
    android:color="color" />

    <stroke
    android:width="integer"
    android:color="color"
    android:dashWidth="integer"
    android:dashGap="integer" />

</shape>
```

5.1.4.6 AnimationDrawable

Android ens proporciona diversos mecanismes per a crear animacions. Un avantatge a destacar és que aquestes animacions poden ser creades en fitxers XML. En aquest apartat veurem una de les animacions més senzilles, les creades a partir d'una sèrie de fotogrames. Per a això utilitzarem la classe AnimationDrawable.

Exercici 5.5: Ús de *AnimationDrawable*.

1. Crea un nou projecte amb nom Animació. Ha de crear-se una activitat inicial amb nom AnimationActivity.

2. Crea el següent recurs res / drawable / animacio.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list
    xmlns:android= "http://schemas.android.com/apk/res/android"
    android:oneshot= "false">
    <item android:drawable="@drawable/misil1"
        android:duration="200" />
    <item android:drawable="@drawable/misil2"
        android:duration="200" />
    <item android:drawable="@drawable/misil3"
        android:duration="200" />
</animation-list>
```

Copia els fitxers misil1.png, misil2.png i misil3.png a la carpeta *res / drawable*.

4. Reemplaça el codi de l'activitat per:

```
public class AnimacioActivity extends AppCompatActivity {
    AnimationDrawable animacio;
    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        animacio =
(AnimationDrawable)ContextCompat.getDrawable(this,R.drawable.animacio)
;
        ImageView vista = new ImageView(this);
        vista.setBackgroundColor(Color.WHITE);
        vista.setImageDrawable(animacio);
        vista.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                animacio.start();
            }
        });
        setContentView(vista);
    }
}
```

En aquest exemple es comença declarant un objecte animació de la classe *AnimationDrawable*. S'inicialitza utilitzant el fitxer XML inclòs en els recursos. Es crea una nova vista de la clase *ImageView* per ser representada per l'activitat i es posa com a imatge d'aquesta vista. Finalment es crea un escoltador d'event *onClick* perquè quan es premi sobre la vista es posi en marxa l'animació.

5.2 Creació d'una vista en un fitxer independent.

Com hem vist en els exemples anteriors, per poder dibuixar a Android hem hagut de crear una nova classe ExempleView, descendent de View. Aquesta classe era creada dins de GraficsActivity pel que solament podia ser utilitzada des d'aquesta classe. Resultarà molt més interessant crear aquesta classe en un fitxer independent. D'aquesta manera podrem utilitzar-la des de qualsevol part del nostre projecte, o per altres projectes. Fins i tot estarà disponible a la paleta de vistes de l'editor visual. El següent exercici, et permet verificar aquest concepte.

Exercici pas a pas: Creació d'una nova vista independent

En aquest exercici posarem la classe *ExampleView* en un fitxer independent perquè pugui ser utilitzada des de qualsevol part.

1. Obre el projecte Grafics.

2. A la classe *GraficsActivity* talla tot el text corresponent a la definició de la classe *ExampleView* i aferra'l a una nova classe anomenada *ExampleView*.

3. Verifica que a l'executar l'aplicació el resultat és idèntic a l'obtingut en la secció anterior.

En aquest apartat aprofitam per introduir altres aspectes que hauràs de tenir en compte per crear noves vistes. Quan vols crear una nova vista, hauràs de estendre la classe *View*, escriure un constructor i com a mínim sobreescriure els mètodes *onSizeChanged ()* i *onDraw ()*. És a dir, hauràs de seguir l'esquema següent:

```
public class ExempleView extends View {

    public ExempleView(Context context, AttributeSet attrs) {
        super(context, attrs);
        // Inicialitza la vista
        // Encara no es coneixen les seves dimensions
    }

    @Override protected void onSizeChanged(int ample, int alt,
                                           int ample_anterior, int
alt_anterior){
        //Informen de l'amplada i l'alt
    }

    @Override protected void onDraw(Canvas canvas) {
        //Dibuixa aquí la vista
    }
}
```

Observa com el constructor utilitzat té dos paràmetres: El primer de tipus *Context* et permetrà accedir al context d'aplicació, per exemple per utilitzar recursos d'aquesta aplicació. El segon, de tipus *AttributeSet*, et permetrà accedir als atributs d'aquesta vista, quan sigui creada des de XML. El constructor és el lloc adequat per crear tots els components de la teva vista, però en compte, en aquest punt encara no coneix les dimensions que tindrà.

Android realitza un procés de diverses passades per determinar l'ample i alt de cada vista dins d'un Layout. Quan finalment ha establert les dimensions d'una vista cridarà al seu mètode *onSizeChanged ()*. Ens indica com a paràmetres l'ample i alt assignat. En cas de tractar-se d'un reajustament de mides, per exemple una de les vistes del *Layout* desapareix i la resta ha d'ocupar el seu espai, se'ns passarà l'ample i alt anterior. Si és la primera vegada que es crida al mètode aquests paràmetres valdran 0.

L'últim mètode que sempre hauràs de reescriure és *onDraw ()*. És aquí on hauràs de dibuixar la vista.

Exercici 5.6: Creació d'una vista que pugui ser dissenyada des de XML

Anem a modificar la vista anterior perquè pugui ser utilitzada fent servir un disseny en XML:

1. Modifica el codi de la classe *ExempleView* perquè coincideixi amb el següent:

```
public class ExempleView extends View {
    private ShapeDrawable laMevaImatge;

    public ExempleView(Context context, AttributeSet attrs) {
        super(context, attrs);
        laMevaImatge = new ShapeDrawable(new OvalShape());
        laMevaImatge.getPaint().setColor(0xff0000ff);
    }
    @Override protected void onSizeChanged(int ample, int alt,
                                           int ample_anterior, int
alt_anterior){
        laMevaImatge.setBounds(0, 0, ample, alt);
    }

    @Override protected void onDraw(Canvas canvas) {
        laMevaImatge.draw(canvas);
```

```
        }
    }
```

La primera diferència és la utilització d'un constructor amb el paràmetre *AttributeSet*. Aquest constructor és imprescindible si vols poder definir la vista en XML. També s'ha afegit el mètode *onSizeChanged*, perquè l'òval es dibuixi sempre ajustat a la mida de la vista.

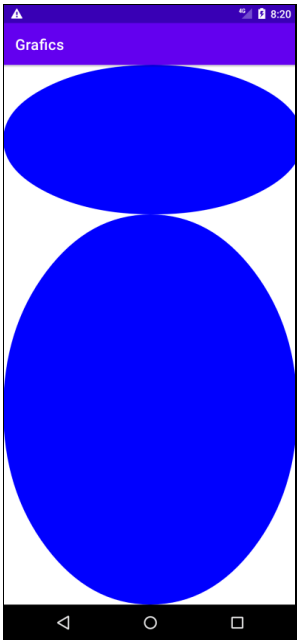
2. Obre el fitxer *activity_main.xml* i reemplaça el seu contingut pel següent:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <com.example.grafics.ExempleView
        android:id="@+id/exempleView1"
        android:layout_width="400dp"
        android:layout_height="200dp" />

    <com.example.grafics.ExempleView
        android:id="@+id/exempleView2"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```

NOTA: No utilitzis el valor "wrap_content". La nostra vista no ha estat programada per a suportar aquest tipus de valor.

3. A la classe GraficsActivity reemplaça *setContentView (new ExemploView (this))* per *setContentView (R.layout.activity_main)*.

4. Executa l'aplicació. El resultat ha de ser similar a:



Aquesta nova vista també pot ser inserida en un layout utilitzant l'editor visual. Prem a la llengüeta *Design* per passar a aquesta manera d'edició. A la paleta de vistes, busca la secció *Project* i selecciona ExempleView.

5 Selecciona la llengüeta Text per observar el codi introduït

5.3 Crear l'activitat principal de la nostra aplicació.

Una vegada que coneixem els rudiments que ens permeten utilitzar gràfics en Android anem a aplicar-los al nostre exemple.

Exercici 5.7: Crear l'activitat principal a Asteroides. GameActivity

El primer que necessitem és crear una activitat que controli la pantalla el joc pròpiament dit. A aquesta activitat li direm GameActivity.

1. Obre el projecte Asteroides.

2. Crea la classe GameActivity amb el següent codi:

```
public class GameActivity extends AppCompatActivity {

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

3. Necessitarem un Layout per a la pantalla de joc. Crea el fitxer res / layout / activity_game.xml amb el següent contingut:

```
<?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" >
    <com.example.asteroides.AsteroidsView
        android:id="@+id/AsteroidsView"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:focusable="true"
        android:background="@drawable/background" />
</LinearLayout>
```

Com pots observar, quan dissenyam un Layout en XML, no estem limitats a escollir els elements que tenim a la paleta de vistes; utilitzarem vistes creades per nosaltres. En aquest exemple utilitzarem la vista *com.example.asteroides.AsteroidsView* que serà descrita més endavant. Serà aquesta vista la que porti el pes de l'aplicació.

4. Observa que s'ha indicat el paràmetre *android: background* associat a el recurs *@ drawable / background*. Per tant, haurem de posar el recurs *background.jpg* a la carpeta corresponent. Còpia també els gràfics corresponents als asteroides, la nau i el míssil a la carpeta res / drawable. Aquests gràfics seran utilitzats en els següents apartats. Els pots trobar al classroom.

5. De moment no podrem executar l'aplicació fins haver definit la classe *AsteroidsView*.

5.3.1 La classe AsteroidsGraphic

El joc que estem desenvolupant desplaçarà molts tipus de gràfics per pantalla: asteroides, nau, míssils, etc. Atès que el comportament de tots aquests elements és molt similar, per tal de reutilitzar el codi i millorar la seva comprensió, crearem una classe que representi un gràfic a desplaçar per pantalla. A aquesta nova classe li direm *AsteroidsGraphic* i presentarà les següents característiques:

- L'element a dibuixar serà representat en un objecte Drawable. Com hem vist, aquesta classe presenta una gran versatilitat, el que ens permetrà treballar amb gràfics en bitmap (BitmapDrawable), vectorials (ShapeDrawable), gràfics amb diferents representacions (StateListDrawable), gràfics animats (AnimationDrawable), etc.
- Un Gràfic disposarà de **posició** (cenX, cenY), **velocitat de desplaçament (**incX, incY**)**, **angle de rotació (**rotAngle**)** i **velocitat de rotació (**rotSpeed**)**.
- Per finalitzar, un gràfic que surti per un dels extrems de la pantalla reapareixerà per l'extrem contrari, tal com passava en el joc original d'Asteroides.

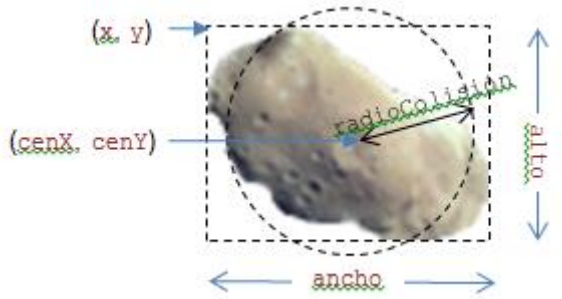
Exercici 5.8: La classe AsteroidsGraphic

1. Crea una nova classe AsteroidsGraphic en el projecte Asteroides i copia el següent codi:

```
class AsteroidsGraphic {
    private Drawable drawable;    //Imatge que dibuixarem
    private double cenX, cenY;    //Posició del centre de la imatge
    private double incX, incY;    //Velocitat desplaçament
    private int rotAngle, rotSpeed;//Àngle i velocitat de rotació
    private int width, height;     //Dimensions de la imatge
    private int collisionRadius;   //Per determinar colisió
    //On dibuixam el gràfic (emprada a view.ivalidate)
    private View view;
    // Per determinar l'espai a borrar (view.ivalidate)
    public static final int MAX_VELOCITY = 20;

    public AsteroidsGraphic(View view, Drawable drawable){
        this.view = view;
        this.drawable = drawable;
        width = drawable.getIntrinsicWidth();
        height = drawable.getIntrinsicHeight();
        collisionRadius = (height + width)/4;
    }
    public void drawGraphic(Canvas canvas){
        canvas.save();
        int x=(int) (cenX + width /2);
        int y=(int) (cenY + height /2);
        canvas.rotate((float) rotAngle,(float) x,(float) y);
        drawable.setBounds((int) cenX, (int) cenY,
                (int) cenX + width, (int) cenY + height);
        drawable.draw(canvas);
        canvas.restore();
        int rInval = (int) Math.hypot(width, height)/2 + MAX_VELOCITY;
        view.invalidate(x-rInval, y-rInval, x+rInval, y+rInval);
    }
    public void updatePos(double factor){
```

```
            cenX +=incX * factor;
            // Si sortim de la pantalla, corregim posició
            if(cenX <-width /2) {cenX =view.getWidth()- width /2;}
            if(cenX >view.getWidth()- width /2) {cenX =-width /2;}
            cenY +=incY * factor;
            if(cenY <-height /2) {cenY =view.getHeight()- height /2;}
            if(cenY >view.getHeight()- height /2) {cenY =-height /2;}
            rotAngle += rotSpeed * factor; //Actualitzam angle
        }
        public double distance(AsteroidsGraphic g) {
            return Math.hypot(cenX -g.cenX, cenY -g.cenY);
        }

        public boolean checkCollision(AsteroidsGraphic g) {
            return(distance(g) < (collisionRadius +g.collisionRadius));
        }
}
```

Cada objecte de la classe Gràfic es caracteritza per situar-se en unes coordenades centrals (cenX, cenY). Per tant la seva part superior dreta estarà en les coordenades (x, y) = (cenX - ample / 2, cenY - height / 2). Encara que un gràfics poden ser allargats (si l'alt és diferent a l'ample), a efecte de les col·lisions, considerarem que són cercles. El radi d'aquest cercle es podria calcular com width / 2 o height / 2, segons prenguéssim el radi major o el radi menor. El que fem és prendre una mitjana d'aquests dos possibles radis: (width / 2 + height / 2) / 2 = (width + height) / 4. El mètode *checkCollision ()* comprova si hem topat amb un altre gràfic. Per a això es comprova si la distància a l'altre objecte *AsteroidGraphic* es menor a la suma dels dos radis de col·lisió.



El mètode *drawGrafic()* s'encarrega de dibuixar el *Drawable* del *AsteroidGraphic* en un *Canvas*. Comença indicant els límits on es situarà el *Drawable*, utilitzant *setBounds ()*. Després, guarda la matriu de transformació del *Canvas*. A continuació, aplica una transformació de rotació segons el que indica la variable *rotAngle* utilitzant com a eix de rotació (cenX, cenY). Es dibuixa el *Drawable* al *Canvas* i es recupera la matriu de transformació, perquè la rotació introduïda no s'apliqui a futures operacions amb aquest Canvas.

Per finalitzar fem dues cridades al mètode *invalidate ()* de la vista on estem dibuixant el *Gràfic*. Amb aquest mètode informam a la vista que ha de ser redibuixada. Per millorar l'eficiència indicarem només el rectangle que hem modificat. D'aquesta manera, la vista no haurà de redibuixar-se íntegrament. En un primer moment podríem pensar que el rectangle d'invalidació coincideix amb l'utilitzat en *setBounds ()*. Però, cal recordar que hem fet una rotació sobre el *Drawable* i, com es pot veure en la il·lustració de l'esquerra, possiblement el *Drawable* se surti d'aquest rectangle. Per resoldre aquest problema augmentarem l'àrea d'invalidació a un quadrat amb la meitat del seu costat igual a la meitat de la diagonal del gràfic. Aquest valor és precalculat amb la variable *rInval*.

Cal tenir en compte que hem desplaçat el *Drawable* des d'una posició anterior. Per tant, també cal indicar a la vista que redibuixi l'àrea on estava abans el *Grafic*. Amb aquesta finalitat utilitzarem les variables *x* i *y*.

Un altre mètode interessant és *updatePos ()* que s'utilitza per a modificar la posició i angle del *Gràfic* segons la velocitat de translació (*incX, incY*) i la velocitat de rotació (*rotSpeed*). Aquest mètode té el paràmetre, *factor*, que permet ajustar aquesta velocitat. Amb un valor igual a 1, tindrem una velocitat normal; si val 2, el gràfic es mou al doble de velocitat. En el joc original d'Asteroides, si un gràfic sortia d'una banda de la pantalla, apareixia de nou pel costat oposat. Aquest comportament és implementat en les últimes línies del mètode.

Al principi de la classe hem definit diversos camps amb el modificador private. Necessitarem accedir a aquests camps des de fora de la classe, pel que resulta necessari **declarar els mètodes get i set corresponents**. Anem a realitzar aquesta tasca de forma automàtica utilitzant una eina d'Android Studio. Situa el cursor a la fi de la classe (just abans de l'última clau) i prem amb el botó dret. Selecciona al menú desplegable Generate... / Getters and setters. A la finestra de diàleg marca tots els camps i prem OK. d'Android Studio farà la feina per nosaltres.

5.3.2 La classe AsteroidsView

Descrivim ara la creació de *AsteroidsView*, que com hem indicat és la responsable de l'execució el joc. En una primera versió només es representaran els asteroides de forma estàtica.

Exercici 5.9: La classe AsteroidsView

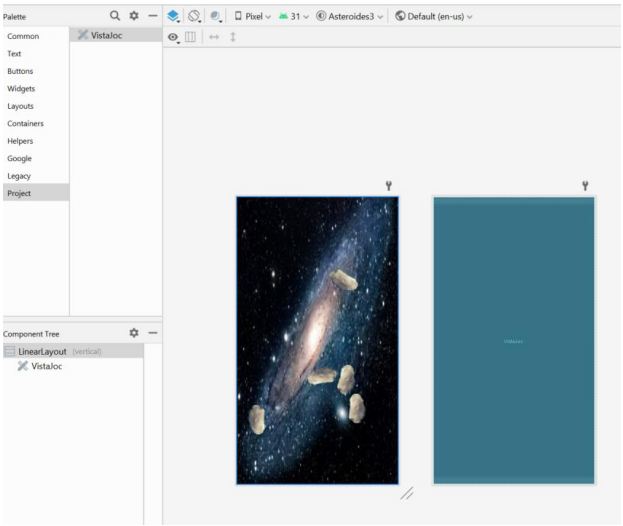
1. Crea una nova classe anomenada AsteroidsView amb el següent codi:

```
public class AsteroidsView extends View {
    // //// ASTEROIDS //////
    private List<AsteroidsGraphic> asteroids; // Vector amb els
Asteroides
    private int numAsteroids = 5; // Número inicial d'asteroides
    private int numFragments = 3; // Fragments en que es divideix

    public AsteroidsView(Context context, AttributeSet attrs) {
        super(context, attrs);
        Drawable drawableShip, drawableAsteroid, drawableMissile;
        drawableAsteroid = context.getResources().getDrawable(
                R.drawable.asteroide1);
        asteroids = new ArrayList<AsteroidsGraphic>();
        for (int i = 0; i < numAsteroids; i++) {
            AsteroidsGraphic asteroid = new AsteroidsGraphic(this,
drawableAsteroid);
            asteroid.setIncY(Math.random() * 4 - 2);
```

```
            asteroid.setIncX(Math.random() * 4 - 2);
            asteroid.setRotAngle((int) (Math.random() * 360));
            asteroid.setRotSpeed((int) (Math.random() * 8 - 4));
            asteroids.add(asteroid);
        }
    }
    @Override protected void onSizeChanged(int width, int height,
                                           int prevWidth, int
prevHeight) {
        super.onSizeChanged(width, height, prevWidth, prevHeight);
        // Un cop coneixem el nostre ample i alt.
        for (AsteroidsGraphic asteroid: asteroids) {

            asteroid.setCenX((int)(Math.random() * width));
            asteroid.setCenY((int)(Math.random() * height));

        }
    }
    @Override protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        for (AsteroidsGraphic asteroid: asteroids) {
            asteroid.drawGraphic(canvas);
        }
    }
}
```

Com veus s'han declarat tres mètodes. En el constructor cream els asteroides i inicialitzem la seva velocitat, angle i rotació. No obstant això, no és possible iniciar la seva posició, atès que no coneixem l'alt i ample de la pantalla. Aquesta informació serà coneguda quan es cridi a *onSizeChanged()*. Observa com en aquesta funció els asteroides estan situats de forma aleatòria. L'últim mètode, *onDraw()*, és el més important de la classe *View*, atès que és el responsable de dibuixar la vista.

2. Hem creat una vista personalitzada. No tindria gaire sentit, però podrà ser utilitzada en qualsevol altra aplicació que desenvolupis. Visualitza el Layout *activity_game.xml* i observa com la nova vista s'integra perfectament en l'entorn de desenvolupament.



3. Registra l'activitat *AsteroidsGame* a *AndroidManifest.xml*.

4. Assigna al boto ‘Jugar’ un *OnClickListener* per a que obri la *GameActivity*.

5. Executa l'aplicació. Has de veure cinc asteroides repartits a l'atzar per la pantalla.

5.3.3 Introduint la nau dins AsteroidsView

El següent pas consisteix a dibuixar la nau que controlarà l'usuari per destruir els asteroides.

Exercici 5.10: Introduir la nau dins la classe *AsteroidsView*

1. Declara les següents variables al començament de la classe *AsteroidsView*:

```
/////// SPACESHIP //////
private AsteroidsGraphic ship; // Gràfic de la nau
private int angleShip; // Angle de gir de la nau
private float accelShip; // Augment de velocitat
private static final double SHIP_MAX_SPEED = 50;
// Increment estàndar de gir i acceleració
private static final int STEPSIZE_ROT_SHIP = 5;
private static final float STEPSIZE_ACCEL_SHIP = 0.5f;
```

2. En el constructor de la classe instància la variable *drawableShip* de manera similar com s'ha fet a *drawableAsteroide*.
3. Inicialitza també al constructor la variable *ship* de manera similar a com s'han inicialitzat els asteroides
4. En el mètode *onSizeChange ()* posiciona la nau just al centre de la vista.
5. En el mètode *onDraw ()* dibuixa la nau al Canvas.
6. Executa l'aplicació. La nau ha d'aparèixer centrada:

7. Si quan situem els asteroides, algun coincideix amb la posició de la nau, el jugador no tindrà cap opció de sobreviure. Seria més interessant assegurar-nos que al posicionar els asteroides aquests es troben a una distància adequada a la nau, i en cas contrari tractar d'obtenir una altra posició. Per aconseguir-ho pots utilitzar un bucle do-while que enmarqui les dues línies que estableixen la posició vertical i horitzontal *setCenX* i *setCenY*, dins el mètode *onSizeChanged*. La condició del while ha de ser:

```
asteroid.distance(ship) < (width+height)/5
```

Exercici 5.11: Evitant que *AsteroidView* canviï la seva representació amb el dispositiu en horitzontal i en vertical

1. Executa l'aplicació
2. Canvia d'orientació la pantalla del dispositiu.
3. Observa com cada vegada, es reinicialitza la vista, regenerant els asteroides. Això ens impediria jugar de forma adequada. Per solucionar-ho edita *AndroidManifet.xml*. A la l'activitat *GameActivity* afegeix la propietat

```
android:screenOrientation="landscape"
```

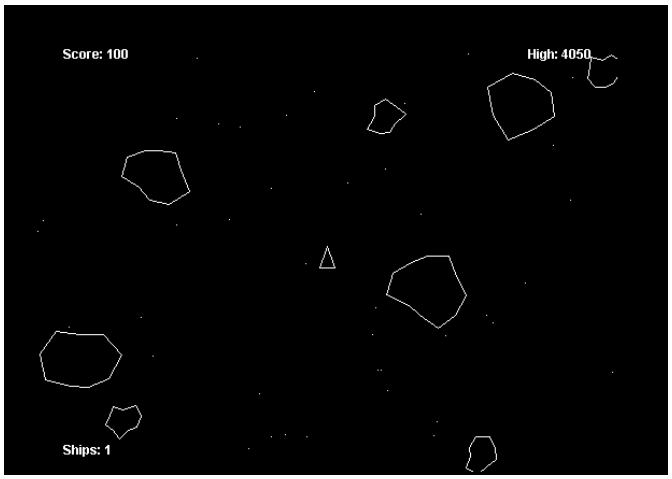
4. Executa de nou l'aplicació. Observa com l'activitat *GameActivity* serà sempre representada en mode horitzontal, de forma independent a la posició de el telèfon.
5. Afegeix també la següent propietat. Aquest tema visualitzarà la vista ocupant tota la pantalla, sense la barra de notificacions ni el nom de l'aplicació.

```
android:theme="@style/Theme.AppCompat.DayNight.NoActionBar"
```

6. Executa l'aplicació i verifica el resultat.

5.4 Representació de gràfics vectorials en la nostra aplicació.

La versió original del joc Asteroides s'executava sobre ordinadors amb poca potència gràfica. Tal com podeu veure a continuació, la nostra nau es representava amb un simple triangle i els asteroides, com polígons irregulars.



Atès que quan hem dissenyat la classe *AsteroidsGraphic*, la representació del mateix l'hem delegada en un objecte *Drawable*, resultarà molt fàcil canviar els gràfics de l'aplicació perquè

tinguin una aparença més retro. Simplement utilitzant la subclasse de *Drawable*, *ShapeDrawable*, en lloc de *BitmapDrawable* per canviar la manera de dibuixar els gràfics.

Exercici 5.12: Representació vectorial dels Asteroides

1. Obre la classe *AsteroidsView*.

2. Al constructor reemplaça la línia:

```
drawableAsteroid = context.getResources().getDrawable(
                R.drawable.asteroide1);
```

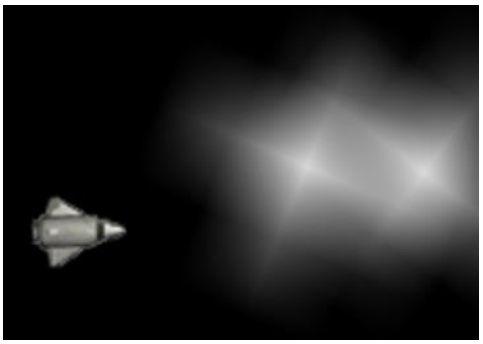
pel codi

```
SharedPreferences pref =
PreferenceManager.getDefaultSharedPreferences(getContext());


if (pref.getString("grafics", "1").equals("0")) {
    setLayerType(View.LAYER_TYPE_SOFTWARE, null);
    Path pathAsteroid = new Path();
    pathAsteroid.moveTo((float) 0.3, (float) 0.0);
    pathAsteroid.lineTo((float) 0.6, (float) 0.0);
    pathAsteroid.lineTo((float) 0.6, (float) 0.3);
    pathAsteroid.lineTo((float) 0.8, (float) 0.2);
    pathAsteroid.lineTo((float) 1.0, (float) 0.4);
    pathAsteroid.lineTo((float) 0.8, (float) 0.6);
    pathAsteroid.lineTo((float) 0.9, (float) 0.9);
    pathAsteroid.lineTo((float) 0.8, (float) 1.0);
    pathAsteroid.lineTo((float) 0.4, (float) 1.0);
    pathAsteroid.lineTo((float) 0.0, (float) 0.6);
    pathAsteroid.lineTo((float) 0.0, (float) 0.2);
    pathAsteroid.lineTo((float) 0.3, (float) 0.0);
    ShapeDrawable dAsteroid = new ShapeDrawable(new
PathShape(pathAsteroid, 1, 1));
    dAsteroid.getPaint().setColor(Color.WHITE);
    dAsteroid.getPaint().setStyle(Paint.Style.STROKE);
    dAsteroid.setIntrinsicWidth(50);
    dAsteroid.setIntrinsicHeight(50);
    drawableAsteroid = dAsteroid;
    setBackgroundColor(Color.BLACK);
} else {
    setLayerType(View.LAYER_TYPE_HARDWARE, null);
    drawableAsteroid = context.getResources().getDrawable(
                R.drawable.asteroide1);

}
```

El primer que fa aquest codi és consultar les preferències per veure si l'usuari ha escollit gràfics vectorials. En cas negatiu es farà la mateixa inicialització de *drawableAsteroid* que teníem abans. En cas afirmatiu comencem creant la variable *pathAsteroid* de la classe Path. En aquest objecte s'introdueixen totes les ordres de dibuix necessàries per dibuixar un asteroide. Després es crea la variable *dAsteroid* de la classe *ShapeDrawable* per crear un drawable a partir del path. Els darrers dos paràmetres (...,1,1) signifiquen el valor d'escala aplicat a l'eix x i a l'eix y. Després s'indica el color i l'estil del pinzell i indiquem l'alt i l'ample per defecte del drawable. Finalment assignem l'objecte creat a drawableAsteroid.

3. Executa l'aplicació i selecciona el tipus de gràfics adequat a les preferències.

NOTA: En alguns dispositius físics, quan s'activa l'acceleració gràfica per maquinari, els asteroides es poden representar de manera estranya:



Per evitar aquest problema, pot ser interessant desactivar l'acceleració gràfica quan treballem amb gràfics vectorials i activar-la en cas contrari. Per això, s'afegeix dins de l'if del codi anterior

```
setLayerType(View.LAYER_TYPE_SOFTWARE, null);
```

I dins l'else

```
setLayerType(View.LAYER_TYPE_HARDWARE, null);
```

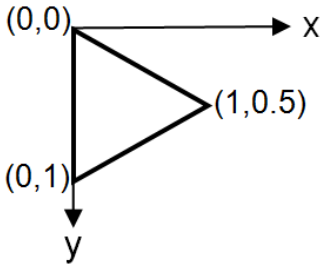
Una altra possible solució consisteix a desactivar a AndroidManifest.xml l'acceleració gràfica per maquinari. Per fer-ho a l'etiqueta <activity> corresponent a l'activitat Joc afegeix l'atribut:

```
android:hardwareAccelerated="false"
```

Exercici 5.13: Representació vectorial de la nau

Com haureu comprovat a l'exercici anterior la nau es representa sempre utilitzant un fitxer png. En aquesta pràctica has d'intentar que també es pugui representar vectorialment.

1. Crea un nou objecte de la classe *Path* per representar la nau dins de la secció *if* introduïda a l'exercici anterior. Com pots veure a la il·lustració següent ha de ser un simple triangle. Com que l'angle de rotació inicial és zero, la nau ha de mirar a la dreta.



2. Crea un nou *ShapeDrawable* a partir del *Path* anterior.

3. Inicialitza la variable *drawableShip* de forma adequada.

5.5 Animacions: Animacions de vistes, Animacions de propietats

L'entorn de programació Android incorpora tres mecanismes per crear animacions a les nostres aplicacions:

- **La classe *AnimationDrawable***: Permet crear *drawables* que reprodueixen una animació fotograma a fotograma. Se n'ha descrit l'ús a la secció de *Drawables*.
- **Animacions *Tween***: Permeten crear efectes de translació, rotació, zoom i alfa a qualsevol vista de la nostra aplicació.
- **Animacions de propietats**: Nou mecanisme incorporat a Android 3.0. Permet animar qualsevol propietat de qualsevol objecte, sigui una vista o no. A més modifica l'objecte en si, no només canvia la seva representació en pantalla com passa en una animació *Tween*.

5.5.1 Animacions Tween

Una animació *tween* pot realitzar sèries de transformacions simples (posició, mida, rotació i transparència) en el contingut d'un objecte *View*. Per exemple, si tens un objecte *TextView* el pots moure, rotar, augmentar, disminuir o canviar la transparència al text.

La seqüència d'ordres que defineix l'animació *tween* pot estar escrita mitjançant xml o codi, però és recomanable el fitxer *xml*, en ser més llegible, reutilitzable i intercanviable.

Les instruccions de l'animació defineixen les transformacions que voleu que passin, quan passaran i quan temps triguen a completar-se. Les transformacions poden ser seqüencials o simultànies. Cada tipus de transformació té uns paràmetres específics, i també hi ha uns paràmetres comuns a totes les transformacions, com el temps que duraran i el temps d'inici.

El fitxer XML que defineix l'animació ha de pertànyer al directori *res/anim/* al teu projecte Android. El fitxer ha de tenir només un únic element arrel: aquest ha de ser un dels següents: *<translate>, <rotate>, <scale>, <alpha>* o element *<set>* que pot contenir grups d'aquests elements (a més d'un altre <set>). Per defecte, totes les instruccions d'animació tenen lloc a partir de l'instant inicial. Si voleu que una animació comenci més tard heu d'especificar l'atribut *startOffset*.

Exercici pas a pas: Creació d'una animació *Tween* per animar una vista

1. Crea un nou projecte amb nom AnimacioTween.
2. Crea el fitxer res/anim/animacio.xml i enganxa el codi següent:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <scale
        android:duration="2000"
        android:fromXScale="3.0"
        android:fromYScale="3.0"
        android:toXScale="1.0"
        android:toYScale="1.0" />
    <rotate
        android:duration="2000"
        android:fromDegrees="0"
        android:toDegrees="720"
        android:pivotX="50%"
        android:pivotY="50%"/>
    <translate
        android:startOffset="4000"
        android:duration="2000"
        android:fromXDelta="0"
        android:fromYDelta="0"
        android:toXDelta="50"
        android:toYDelta="100" />
    <alpha
        android:startOffset="4000"
        android:duration="2000"
```

```
        android:fromAlpha="1"
        android:toAlpha="0" />
</set>
```

3. Obre el fitxer res/layout/activity_main.xml i afegeix el següent atribut a la vista de tipus *TextView*:

```
android:id="@+id/textView"
```

4. Obre l'activitat del projecte i modifica *OnCreate()* afegint les línies següents.

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

    TextView text = (TextView) findViewById(R.id.textView);
    Animation animacio = AnimationUtils.loadAnimation(this,
            R.anim.animacio);
    text.startAnimation(animacio);
}
```

5. Executa l'aplicació.

Com podreu veure, el *TextView* comença fent-se més petit (etiqueta <scale>), després gira sobre si mateix (etiqueta <rotate>) i finalment, es desplaça (etiqueta <translate>) alhora que es fa transparent (etiqueta <alpha >). En finalitzar l'animació, torna a la seva posició i estat inicial, sense importar on ni com hagi acabat.

Llista d'etiquetes de les animacions tween i els seus atributs

Els atributs següents són aplicables a totes les transformacions:

- ***startOffset*** – Instant inicial de la transformació en mil·lisegons.
- ***duration*** – durada de la transformació en mil·lisegons.
- ***repeatCount*** - nombre de repeticions addicionals de l'animació
- ***interpolator***– en lloc de fer una transformació lineal s'aplica algun tipus d'interpolació. Algun dels valors possibles són:
    - *accelerate_decelerate_interpolator, accelerate_interpolator, anticipate_interpolator, anticipate_overshoot_interpolator, bounce_interpolator, cycle_interpolator, decelerate_interpolator, linear_interpolator, overshoot_interpolator*

Llista de les transformacions amb els seus atributs específics:

<translate> – Mou la vista

- *fromXDelta, toXDelta* – Valor inicial i final del desplaçament en l'eix X
- *fromYDelta, toYDelta* – Valor inicial i final del desplaçament en l'eix Y.

<rotate> – Rotació de la vista.

- *fromDegrees, toDegrees* – Valor inicial i final en graus de la rotació en graus. Si vols un gir complet en sentit antihorari posa de 0 a 360 i si ho vols en sentit horari, de 360 a 0 o de 0 a -360. Si voleu dos girs poseu de 0 a 720.
- *pivotX, pivotY*– Punt sobre el qual es farà el gir. Aquest quedarà fix a la pantalla.

<scale> – Canvia la mida de la vista

- *fromXScale,toXScale* – Valor inicial i final per a l'escala de l'eix X (0.5=50%, 1=100%)
- *fromYScale, toYScale* – Valor inicial i final per a l'escala de l'eix Y
- *pivotX, pivotY*– Punt sobre el qual es farà el zoom. Aquest quedarà fix a la pantalla.

<alpha> – Canvia l'opacitat de la vista

- *fromAlpha, toAlpha* – Valor inicial i final de l'opacitat

Exercici 5.14: Animacions dins el projecte d'Asteroides.

En aquesta pràctica has d'aconseguir que els diferents elements del Layout inicial d'Asteroides vagin apareixent un darrere l'altre amb diferents efectes.

1. Obre el projecte Asteroides.
2. Crea una animació nova amb nom *gir_amb_zoom.xml*. Ha de durar dos segons i simultàniament ha de fer un zoom d'escala 3 a 1 i un gir de dues voltes (720º). El punt d'ancoratge de la rotació i el zoom ha de ser el centre de la vista.
3. Selecciona el layout activity_main.xml i posa un id al *TextView* corresponent al títol (Asteroides).
4. A l'activitat inicial d'Asteroides (*ActivityMain.java*), creeu un objecte corresponent a aquest *TextView* i apliqueu-li l'animació anterior.
5. Crea una animació nova amb el nom *apareixer.xml*. Que comenci als dos segons, que duri un segon i que modifiqui el valor d'alpha de 0 fins a 1.
6. Aplica aquesta animació al primer botó.
7. Crea una animació nova amb nom *despl_dreta.xml*. Ha de començar al cap de tres segons, durar un segon i modificar el valor de desplaçament x de 400 fins a 0. Prova també algun tipus d'interpolació.
8. Aplica aquesta animació al segon botó.
9. Crea dues noves animacions al teu gust i aplica-les al tercer i quart botó.
10. Aplica l'animació *gir_amb_zoom.xml* al botó *Sobre* quan sigui premut. Observa com en llançar la nova activitat *Sobre*, l'activitat principal continua executant-se.

5.5.2 Animacions de propietats

A partir de la versió 3.0 d'Android (nivell API 11) es van incorporar un nou tipus d'animacions. A diferència de les animacions *Tween* que només és aplicable a la vista, una animació de propietats pot animar qualsevol tipus d'objectes. A més, no està restringit a les quatre transformacions

abans vistes, podem animar qualsevol propietat de l'objecte. Per exemple, podem fer una animació que canviï progressivament el color de fons d'una vista.

Una altra diferència pel que fa a les animacions *Tween*, és que aquestes només modifiquen la manera com la vista es representa, però no les seves propietats. Per exemple, si apliqueu una animació *Tween* perquè un text es desplaci per la pantalla, es visualitzarà correctament, però en acabar l'animació el text tornarà a ser al lloc inicial. Això us obligarà a implementar la vostra pròpia lògica per manejar aquest canvi de posició. En una animació de propietats estarà canviant l'objecte en si, no sols com es representa.

Desavantatges de les animacions Tween:

- Només podem animar objectes de la classe View.
- Està limitat a aquestes quatre transformacions i no es pot aplicar a altres aspectes com canviar el color de fons.
- Només modifica la manera com la vista és representada, però no les seves propietats en si.

Desavantatges de les animacions de propietats:

- Requereix més temps a inicialitzar-se i cal escriure més codi

En aquest apartat d'Android Developers podreu trobar tota la informació sobre l'animació de propietats:

https://developer.android.com/guide/topics/graphics/prop-animation

Resum exercicis:

Exercici 5.1: Ars olímpics

Exercici 5.2: Dibuixar un text sobre un cercle.

Exercici 5.3: Utilitzar l'eina Vector Asset

Exercici 5.5: Ús de *AnimationDrawable*.

Exercici 5.6: Creació d'una vista que pugui ser dissenyada des de XML

Exercicis asteroides:

Exercici 5.4: Definir un *GradientDrawable* pel fons del menú principal d'Asteroides.

Exercici 5.7: Crear l'activitat principal a Asteroides. GameActivity.

Exercici 5.8: La classe AsteroidsGraphic

Exercici 5.9: La classe AsteroidsView

Exercici 5.10: Introduir la nau dins la classe *AsteroidsView*

Exercici 5.11: Evitant que *AsteroidView* canviï la seva representació amb el dispositiu en horitzontal i en vertical

Exercici 5.12: Representació vectorial dels Asteroides

Exercici 5.13: Representació vectorial de la nau

Exercici 5.14: Animacions dins el projecte d'Asteroides.