Projet Android

À la création d'un *projet Android* (IDE AndroidStudio – *package* `l2i013.appdroid` – thème empty) tout un ensemble de répertoires et de fichiers sont engendrés. On en distingue 3 :

1. un fichier `AndroidManifest.xml` (dans le répertoire `manifests`) qui donne au système Android les caractéristiques de votre application ; dont, en particulier, la liste de ses *activités*.
2. un répertoire `java` où sont placés les codes JAVA de votre application.
3. un répertoire `res` contenant d'autres répertoires, dont `layout`. Dans ces répertoires sont placés des fichiers, au format XML, de description des ressources des interfaces graphiques de l'application.

XML et JAVA

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
 xmlns:android="http://schemas.android.com/apk/res/android"
 package="l2i013.appdroid">
 <application
  android:icon="@mipmap/ic_launcher"
  android:label="@string/app_name"
  android:theme="@style/AppTheme">
  <activity android:name=".MainActivity">
   <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
  </activity>
 </application>
</manifest>
```

MainActivity.java

Classe JAVA engendrée par défaut à la création d'un projet

```
package l2i013.appdroid;

import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity {

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

}
```

On ne définit ni n'utilise de constructeur d'activité

Nota Bene : oublier `AppCompatActivity` pour `Activity`

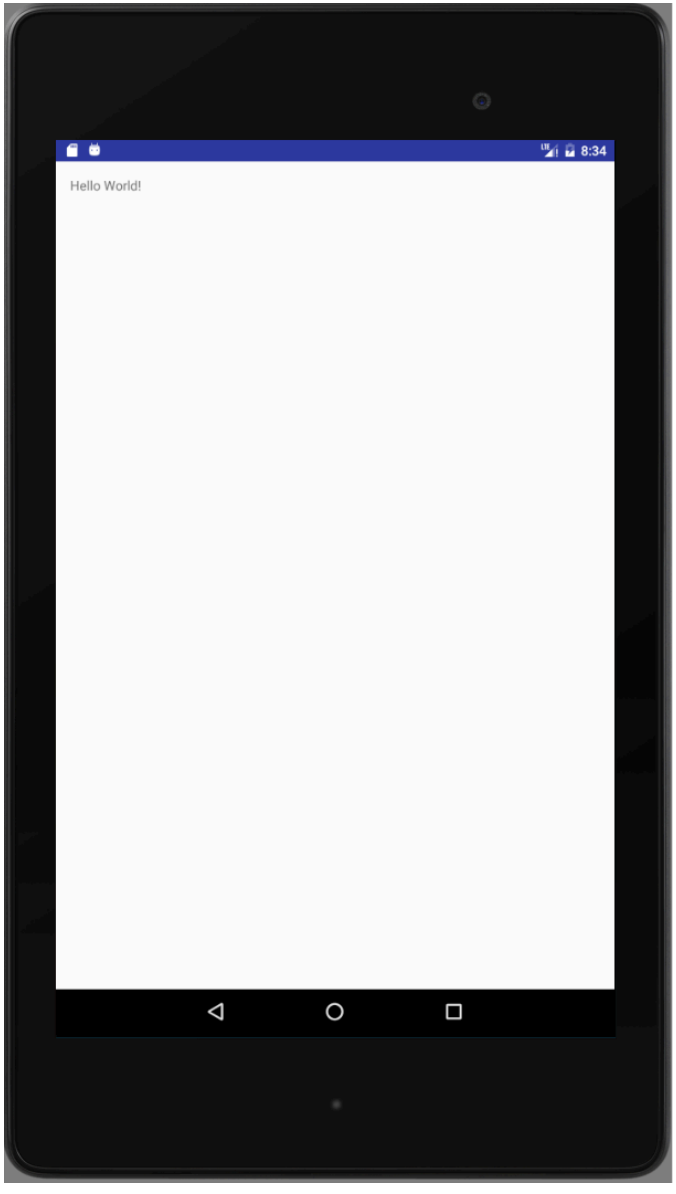
res/layout/activity_main.xml

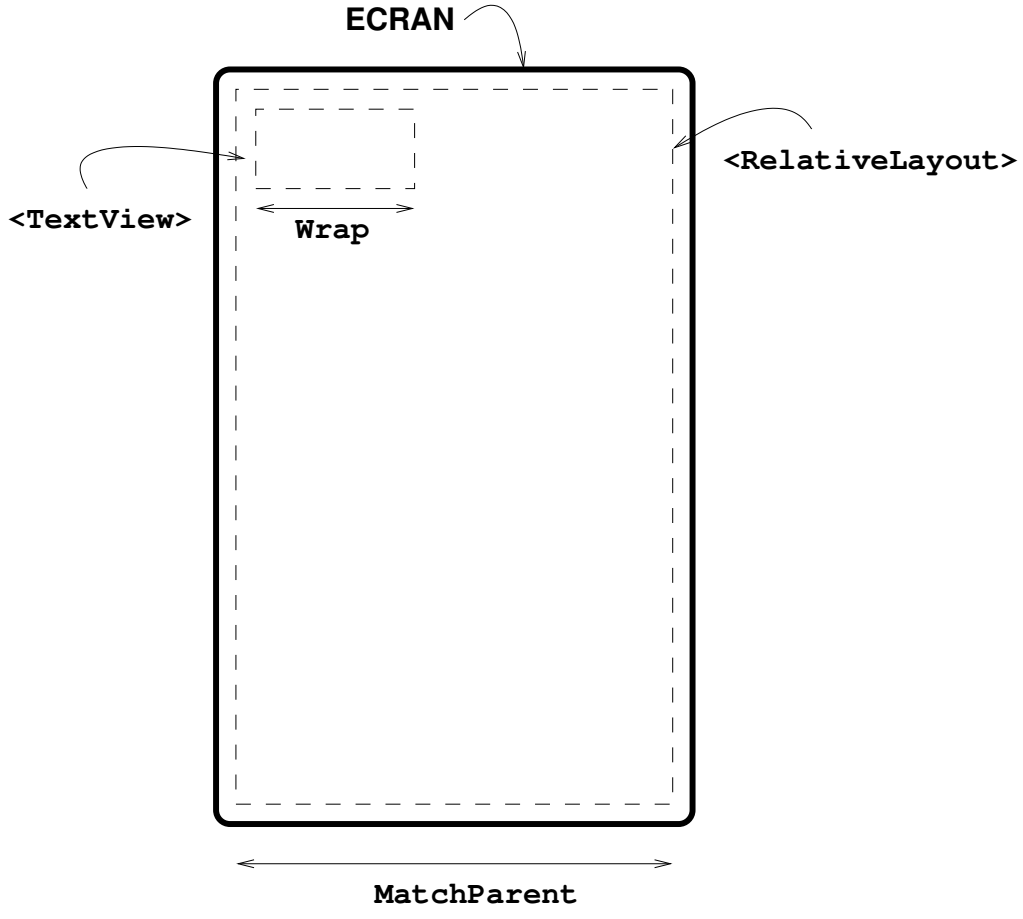
Fichier XML engendré à la création du projet

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="l2i0013.appdroid.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
</RelativeLayout>
```

Visualisation sur un terminal





Soigner la présentation

- center les éléments graphiques
- modifier la taille
- modifier le contenu du message

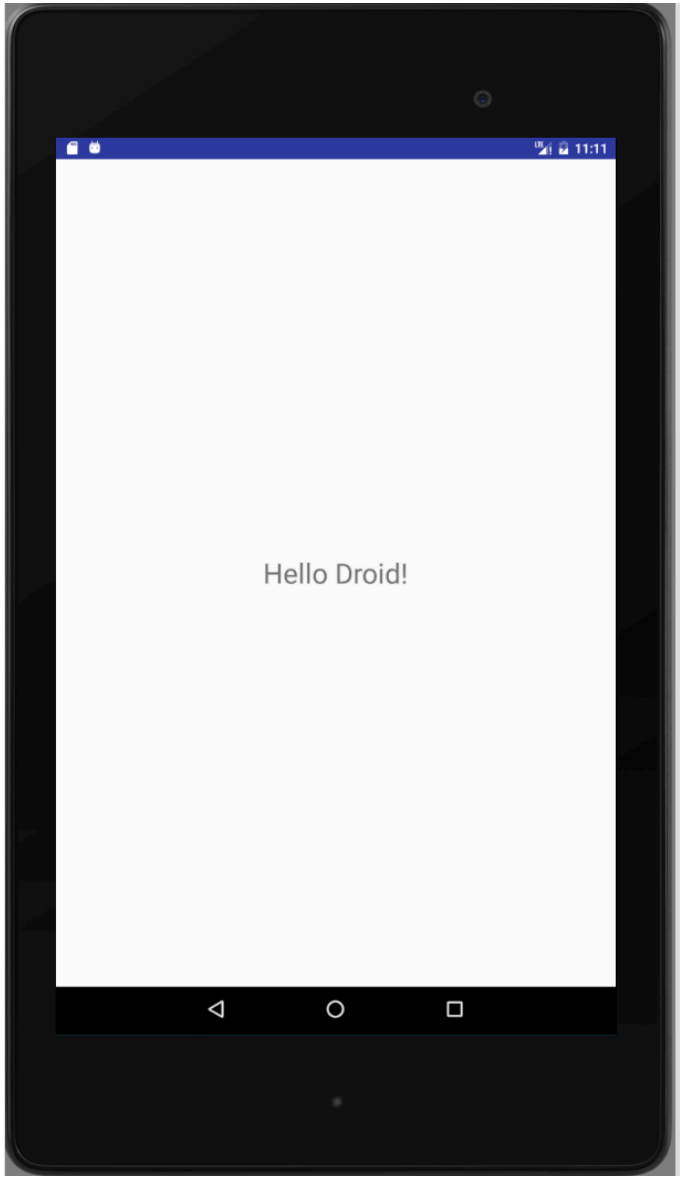
Editer `res/layout/activity_main.xml`, balise `TextView`

```
<RelativeLayout
    [...]
    android:gravity="center"
    tools:context="l2i0013.appdroid.MainActivity">

    <TextView
        android:id="@+id/title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="30dp"
        android:text="Hello Droid!" />
</RelativeLayout>
```

NOTA BENE : nommage du composant (`android:id`)

Visualisation



Ajouter un composant

Un *bouton EXIT* placé en colonne sous le titre pour quitter l'application.

1. ajouter le composant graphique à l'interface : modifier `activity_main.xml`.
2. définir l'action associée à l'activation du bouton : modifier `MainActivity`.

activity_main.xml

```
<RelativeLayout [...] >
    <TextView [...] />
    <Button
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:layout_below="@id/title"
          android:background="#ff0000"
          android:textSize="30dp"
          android:text="EXIT"
          android:onClick="onClickExit"/>
</RelativeLayout>
```

- `RelativeLayout` : placement relatif
  `layout_below="@id/title"`
- couleur de fond rouge : `background="#ff0000"` (RGB)
- action : `onClick=onClickExit` (c.f. classe `MainActivity`)

Visualisation

MainActivity

JAVA - XML

Dans la classe `MainActivity` : définir la méthode `onClickExit`

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        [..]
    }

    public void onClickExit(View v) {
        finish();
    }
}
```

Paramètre `View v` : origine de l'invocation.

Jouer un fichier MIDI – interface

Ajouter un bouton `PLAY` à `activity_main.xml`

```
<RelativeLayout [...] >
   <TextView [...] />
   <Button
        android:id="@+id/play"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/title"
        android:background="#00ff00"
        android:textSize="30dp"
        android:text="PLAY"
        android:onClick="onClickPlay"/>
   <Button [...]
        android:layout_below="@id/play" />
</RelativeLayout>
```

Positionner `EXIT` après `PLAY`

Jouer un fichier MIDI – code JAVA

Ajouter à `MainActivity` la méthode

```
public void onClickPlay(View v)
```

1. crée une `Partition`
2. l'enregistre dans le fichier `tmp.mid`
3. lit et joue le fichier

Ressources JAVA (entre autres)

```
package l2i013.musidroid;
import android.app.Application;
import android.media.MediaPlayer;
import java.io.File;
import l2i013.musidroid.model.Partition;
import l2i013.musidroid.util.*;
[..]
public class MainActivity extends Activity {
    [..]
    public void onClickPlay(View v) {
    }
}
```

Jouer un fichier MIDI – code de la méthode (1 et 2)

Créer une `Partition` (simple ici)

```
Partition p = new Partition(90);
int ip1 = p.addPart(InstrumentName.TRUMPET,6) ;
p.addNote(ip1, 0, NoteName.DO, 1);
p.addNote(ip1, 1, NoteName.FA, 1);
p.addNote(ip1, 2, NoteName.FA, 1);
p.addNote(ip1, 3, NoteName.FA, 1);
p.addNote(ip1, 4, NoteName.SOL, 1);
p.addNote(ip1, 5, NoteName.FA, 2);
```

Écrire le fichier `tmp.mid`

```
Application app = (Application) getApplicationContext();
MidiFile2IO13.write(new File(app.getFilesDir(),"tmp.mid"),p);
```

Jouer un fichier MIDI – code de la méthode (3)

Lire et faire jouer par un MediaPlayer

```
MediaPlayer mPlayer = new MediaPlayer();
MediaPlayer.OnPreparedListener mPrepared =
 new MediaPlayer.OnPreparedListener() {
  @Override
  public void onPrepared(MediaPlayer playerM) { }
}
mPlayer.setOnPreparedListener(mPrepared);

try {
  File f = new File(app.getFilesDir(), "tmp.mid");
  mPlayer.setDataSource(f.getPath());
  mPlayer.setLooping(false);
  mPlayer.prepare();

 if (!mPlayer.isPlaying()) { mPlayer.start(); }

} catch (Exception e) {
}
```

Une application : 2 activités

Ajouter une deuxième *activité* à l'application.
L'activité est lancée par un bouton depuis l'activité principale

1. déclarer l'activité comme ressource de l'application : modifier `AndroidManifest.xml`
2. ajouter un bouton `TOUCH` à l'activité principale :
   2.1 modifier `activity_main.xml` : nouveau `Button`
   2.2 modifier `MainActivity` : nouvelle méthode `onClickTouch`
3. créer l'interface graphique de l'activité de jeu : nouveau fichier `activity_touch.xml`
4. créer la classe `TouchActivity`

AndroidManifest

```
<manifest [..]>

    <application
        [..] >
        <activity android:name=".MainActivity">
            [..]
        </activity>
        <activity android:name=".TouchActivity" />
    </application>

</manifest>
```

Ajouter le bouton TOUCH

Dans `activity_main.xml`

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/touchButton"
    android:layout_below="@id/play"
    android:textSize="30dp"
    android:text="TOUCH"
    android:onClick="onClickTouch"/>
```

Dans `MainActivity` pour démarrer l'activité de jeu

```
public void onClickTouch(View v) {
    Intent playIntent = new Intent(this, TouchActivity.class
    startActivity(playIntent);
 }
```

La 2éme activité

Interface graphique : créer `res/layout/activity_touch.xml`

```
<LinearLayout [..] >
    <Button
        [..]
        android:onClick="onClickQuit"/>
</LinearLayout>
```

Nota : une autre classe de *layout*

L'activité : créer `TouchActivity.java`

```
public class TouchActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_touch);
    }
    public void onClickQuit(View v) {
        finish();
    }
}
```

Une aire de jeu

Surface de dessin réactive

JAVA+XML

1. définir une nouvelle classe JAVA : `TouchBoard`
   - qui hérite de `SurfaceView` pour le dessin et l'interactivité ;
   - et qui implémente l'interface `SurfaceHolder.Callback` pour le contrôle de l'affichage.
2. intégrer la surface de jeu à l'interface graphique de l'activité
   - une nouvelle balise XML dans `activity_touch`

Comportement attendu : afficher un point (petit cercle) au touché de la surface

TouchBoard : JAVA

Schéma

```
public class TouchBoard extends SurfaceView
                        implements SurfaceHolder.Callback {
    public TouchBoard(Context c) { [..] }
    public TouchBoard(Context c, AttributeSet as) { [..] }
    public void reDraw() { [..] }
    @Override
    public void onDraw(Canvas c) { [..] }
    @Override
    public void surfaceCreated(SurfaceHolder holder) { [..] }
    @Override
    public void surfaceChanged
      (SurfaceHolder holder, int format, int width, int height)
      { [..] }
    @Override
    public void surfaceDestroyed(SurfaceHolder holder) { [..] }
    @Override
    public boolean onTouchEvent(MotionEvent event) { [..] }
}
```

TouchBoard : XML

Ajouter à `activity_touch.xml`

```
<LinearLayout [..] >

<l2i013.appdroid.TouchBoard
    android:layout_width="match_parent"
    android:layout_height="256dp"
    android:id="@+id/boardSurface"
    />

    <Button [..] />

</LinearLayout>
```

- nouvelle balise : package+classe
- ATTENTION à la taille fixe : il faudra faire mieux...

TouchBoard : JAVA

Les constructeurs

```
public TouchBoard(Context context) {
    super(context);
    getHolder().addCallback(this);
}
public TouchBoard(Context context, AttributeSet attrs) {
    super(context, attrs);
    getHolder().addCallback(this);
}
```

`getHolder` donne le contrôleur de la surface
`addCallback(this)` lui signifie qu'il peut adresser des messages à la surface elle-même aux moments clés de sa vie : création, changement, destruction

Pourquoi 2 constructeurs ? Mystère ...

TouchBoard : JAVA

(Re)dessiner

Concurrence, section critique

```
void reDraw() {
    Canvas c = getHolder().lockCanvas();
    if (c != null) {
        this.onDraw(c);
        getHolder().unlockCanvasAndPost(c);
    }
}
```

Le dessin proprement dit : un simple fond gris (pour l'instant)

```
@Override
public void onDraw(Canvas c) {
    c.drawColor(Color.LTGRAY);
}
```

TouchBoard : JAVA

les *«call back»*

Synchronisation : se dessiner au bon moment

```
@Override
public void surfaceCreated(SurfaceHolder sh) {
    // rien
}
@Override
public void
  surfaceChanged(SurfaceHolder sh, int f, int w, int h) {
   reDraw();
}
@Override
public void surfaceDestroyed(SurfaceHolder sh) {
    // rien
}
```

`SurfaceHolder.Callback` : `surfaceChanged` n'est invoquée que lorsque la surface de dessin (*canvas*) est effective.
Nota : on connaît alors ses dimensions (`w` et `h`)

Un jeu et son interface

Un jeu idiot : afficher un point à chaque endroit touché
Une réalisation plus délicate :

- mémoriser l'ensemble des points touchés
- garder cette information persistante
- partager cette information entre différentes composantes ou méthodes

Une solution : un modèle de jeu logé au niveau de *l'application* qui peut être partagée par l'ensemble de composants de l'application (activités et composants graphiques)

Le modèle

Le bête modèle du jeu idiot.

```
public class Model {
    ArrayList<Position> xys;
    Model() {
        xys = new ArrayList<Position>();
    }
    void add(int x, int y) {
        xys.add(new Position(x,y));
    }
    ListIterator<Position> getAll() {
        return xys.listIterator();
    }
}
```

avec

```
public class Position {
    int x, y;
    Position(int x, int y) { this.x = x; this.y = y; }
    Integer getX() { return x; }
    Integer getY() { return y; }

}
```

Partage et persistance du modèle

Le modèle est détenu par *l'application*

⇒ personnaliser la classe `Application` d'Android :

```
public class TheApplication extends Application {
    Model m;
    @Override
    public void onCreate() {
        super.onCreate();
        m = new Model();
    }
    Model getModel() {
            return m;
    }
}
```

Déclarer (nommer) l'application au système : `AndroidManifest`

```
<manifest [..] >
    <application android:name="TheApplication" [..] >
        [..]
    </application>
</manifest>
```

Partager

Accéder à l'application depuis une activité :

```
public class MainActivity extends Activity {
  TheApplication app;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    [..]
    app = (TheApplication)(this.getApplication());
  }
```

Depuis un composant graphique de l'activité

```
public class TouchBoard extends SurfaceView
                        implements SurfaceHolder.Callback {
  TheApplication app;
  public TouchBoard(Context context, AttributeSet attrs) {
    [..]
    app = (TheApplication) (context.getApplicationContext());
  }
```

Dessiner l'état du jeu

Affiner onDraw dans TouchBoard

```
@Override
public void onDraw(Canvas c) {
    Model m = app.getModel();
    Paint p = new Paint();
    ListIterator<Position> it = m.getAll();
    c.drawColor(Color.LTGRAY);
    while(it.hasNext()) {
        Position xy = it.next();
        p.setColor(Color.DKGRAY);
        c.drawCircle(xy.getX(), xy.getY(),13,p); // ATTENTIO
    }
}
```

Notez l'utilisation de l'itérateur

Réagir au toucher

1. notifier au modèle
2. notifier au dessin

Dans `TouchBoard` (re)définir `onTouchEvent`

```
@Override
public boolean onTouchEvent(MotionEvent event) {
    int x = (int) event.getX();
    int y = (int) event.getY();
    int action = event.getAction();
    switch (action) {
        case MotionEvent.ACTION_DOWN: {
            app.getModel().add(x,y);
            reDraw();
            return true;
        }
        default:
            return false;
    }
}
```

Jouer encore

Ajouter à l'interface de jeu la possibilité de réinitialiser l'état du jeu.

Réinitialiser l'état du jeu (`Model`)

```
void reset() { xys.clear() }
```

Un bouton *reset* dans l'interface (`activity_play.xml`)

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="RESET"
    android:onClick="onClickReset"
    />
```

Réagir à la demande de réinitialisation (`PlayActivity`)

```
public void onClickReset(View v) {
   app.getMoel().reset();
   ((TouchBoard)findViewById(R.id.boardSurface)).reDraw();
}
```

Notez `findViewById`

Menu déroulant

Ajouter un menu déroulant pour choisir une forme

Ce qu'il faut faire

1. ajouter les formes au modèle
2. personnaliser et ajouter le composant à la vue de l'activité
3. ajouter le traitement du choix à l'activité
   - 3.1 rendre l'activité réactive au choix
   - 3.2 utiliser l'information pour le dessin

Modèle

Un type énuméré pour les formes

```
public enum ShapeName {
    CIRCLE, SQUARE, TRIANGLE;
}
```

La forme et ses méthodes d'accés et de modification

```
ShapeName shape;
[..]
ShapeName getShape() {
    return shape;
}

void setShape(ShapeName n) {
    shape = n;
}
```

Spinner

Ajouter le composant à la vue : `activity_touch.xml`

```
[..]
<Spinner
    android:id="@+id/shapeChoice"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
[..]
```

Personnaliser

Ressource XML `layout/spinner_item.xml`

```
<TextView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="50dp"
    android:textColor="#ffff00"/>
```

Ressource XML `layout/spinner_dropdown_item.xml`

```
<CheckedTextView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="50dp"
    android:textColor="#ffff00"/>
```

cf. infra :

Traitement du choix

L'activité `TouchActivity` doit réagir aux évènments du menu (sélection) : une *interface* pour `TouchActivity`

```
[..] implements AdapterView.OnItemSelectedListener
```

Deux méthodes :
Informer le modèle de la sélection

```
@Override
public void onItemSelected(AdapterView<?> parent,
                           View view, int position, long id) {
   app.getModel().setShape(ShapeName.values()[position]);
}
```

Traitement en l'absence de sélection (ici : rien)

```
@Override
public void onNothingSelected(AdapterView<?> parent) {

}
```

Installation du menu

À la création (onCreate)

1) mettre l'activité à l'écoute des choix

```
Spinnner spinnerShape =
    (Spinner) findViewById(R.id.shapeChoice);
spinnerShape.setOnItemSelectedListener(this);
```

2) renseigner la liste de choix (avec vue personnalisée des items)

```
List<ShapeName> list = Arrays.asList(ShapeName.values());
ArrayAdapter<ShapeName> dataAdapter =
     new ArrayAdapter<ShapeName>(this,R.layout.spinner_item,
                                 list);
dataAdapter.setDropDownViewResource
                (R.layout.spinner_dropdown_item);
spinnerShape.setAdapter(dataAdapter);
```

3) choix par défaut

```
spinnerShape.setSelection(0);
```

Affichage

Utiliser la forme sélectionnée pour le dessin (`TouchBoard.onDraw`)

```
ShapeName shape = m.getShape();

if (shape==ShapeName.CIRCLE)
    c.drawCircle(x, y,w,p);
else if (shape == ShapeName.SQUARE)
    c.drawRect(x-w,y-w,x+w,y+w,p);
else  // TRIANGLE
    Path path = new Path();
    path.moveTo(x-w,y+w);
    path.lineTo(x+w,y+w);
    path.lineTo(x,y-w);
    path.close();
    p.setStyle(Paint.Style.FILL_AND_STROKE);
    c.drawPath(path,p);
```

Un curseur

Ajouter un curseur pour régler la couleur

Ce qu'il faut faire

1. ajouter une information de couleur au modèle
2. ajouter un composant graphique texte+curseur
3. créer un gestionnaire pour le curseur
4. lier ce gestionnaire au composant graphique
5. utiliser l'information de couleur à l'affichage

Le modèle

Un champ et deux méthodes

```
int color;

int getColor() {
    return color;
}

void setColor(int c) {
    color = c;
}
```

Composants graphiques

Un `TextView` et un `SeekBar` (curseur) dans `activity_touch.xml`

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
    <TextView
        android:id="@+id/colorText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="35dp"
        android:background = "#ffffff"
        android:padding="5dp"
        android:text="(..)"/>
    <SeekBar
        android:id="@+id/changeColor"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#00ffff"/>
</LinearLayout>
```

Gestion du curseur

Implémenter l'interface `SeekBar.OnSeekBarChangeListener`
Trois méthodes

`onStartTrackingTouch` ce qu'il faut faire quand on touche le curseur :
initialiser le texte

`onProgressChanged` ce qu'il faut faire quand on fait glisser de curseur :
mettre à jour le texte et la couleur du fond

`onStopTrackingTouch` ce qu'il faut faire quand on relache le curseur : :
notifier la valeur au modèle

Classe ColorChangeListener

Entête de la classe

```
public class ColorChangeListener
    implements SeekBar.OnSeekBarChangeListener {
```

Variables d'instances

```
    TheApplication app;
    TextView colorText;
    int color = 0;  // arbitrary default value
```

Le constructeur

```
  public ColorChangeListener(TheApplication app, TextView ct) {
        super();
        this.app = app;
        this.colorText = ct;
    }
```

Notez les paramètres du constructeur.

Les méthodes de .OnSeekBarChangeListener

```
public void onProgressChanged (SeekBar sb, int n, boolean b) {
    color = n;
    colorText.setText("(" + color + ")");
    colorText.setBackgroundColor(app.computeColor(color));
}

public void onStartTrackingTouch(SeekBar seekBar) {
    colorText.setText("(" + color + ")");
}

public void onStopTrackingTouch(SeekBar seekBar) {
    app.getModel().setColor(color);
}
```

Avec computeColor :

blog.vermot.net/2011/11/03/generer-un-degrade-en-arc-en-ciel-en-fonction-d-une-valeur-programmatio/

Lier le gestionnaire au composant SeekBar

Dans `TouchActivity.onCreate`

```
TextView colorText = (TextView)findViewById(R.id.colorText);
colorText.setText("("+app.getModel().getColor()+")");
SeekBar colorSeekBar=(SeekBar)findViewById(R.id.changeColor);
ColorChangeListener colorListener =
    new ColorChangeListener(app, colorText);
colorSeekBar.setOnSeekBarChangeListener(colorListener);
```

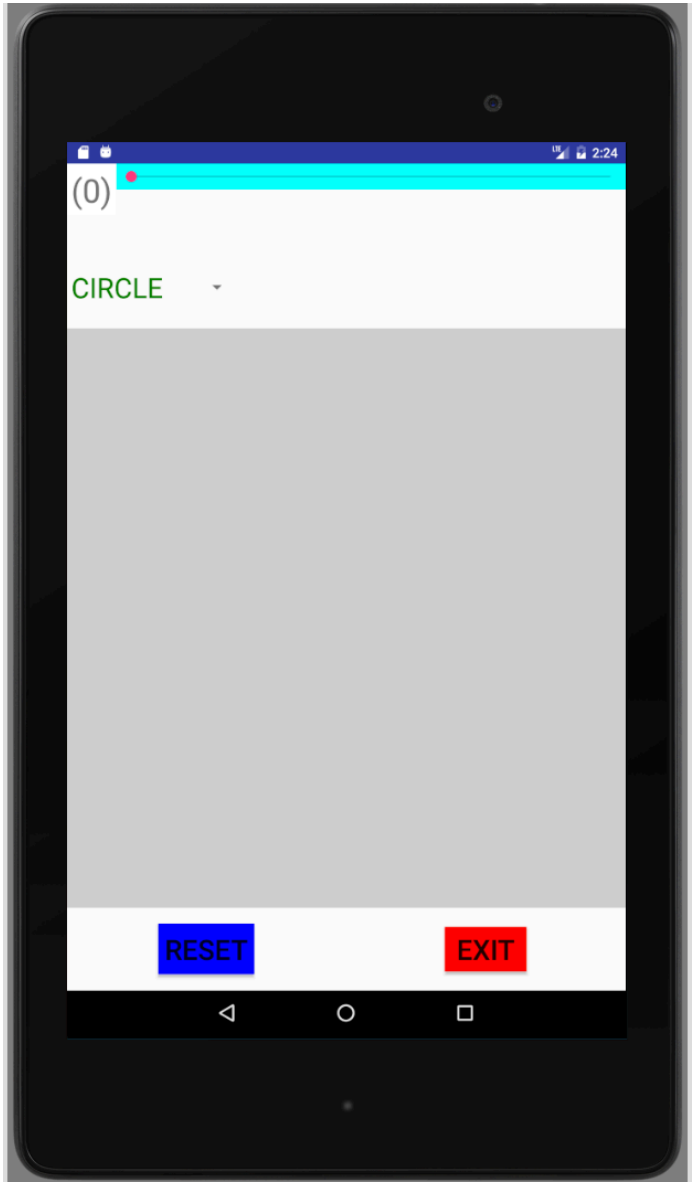
Utiliser la couleur

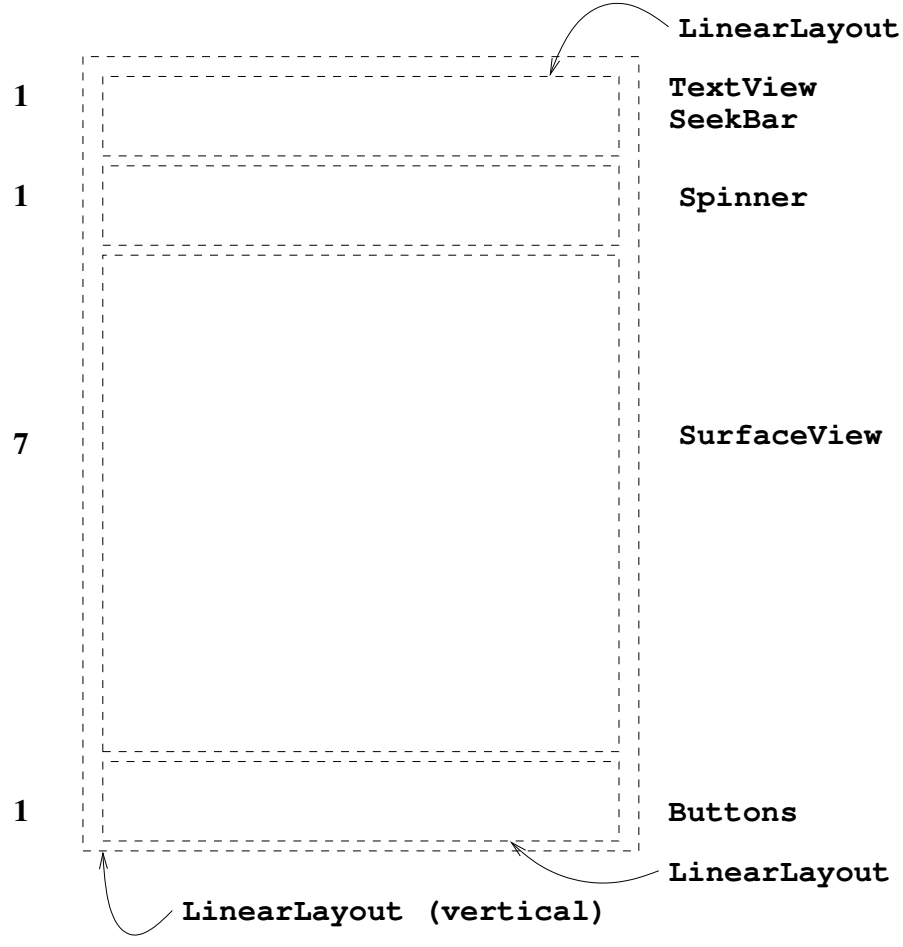
Dans `TouchBoard.onDraw`

```
p.setColor(app.computeColor(m.getColor()));
```

Partager l'espace d'affichage

Accorder un *poids* à chaque composant : `layout_weight`





layout_weight 1

```
<LinearLayout [..]
    android:orientation="vertical">
    <LinearLayout [..]
        android:layout_height="0dp"
        android:layout_weight="1">
        <TextView [..] />
        <SeekBar [..] />
    </LinearLayout>
    <Spinner [..]
        android:layout_height="0dp"
        android:layout_weight="1" />
    <l2i013.appdroid2018.TouchBoard [..]
        android:layout_height="0dp"
        android:layout_weight="7"/>
```

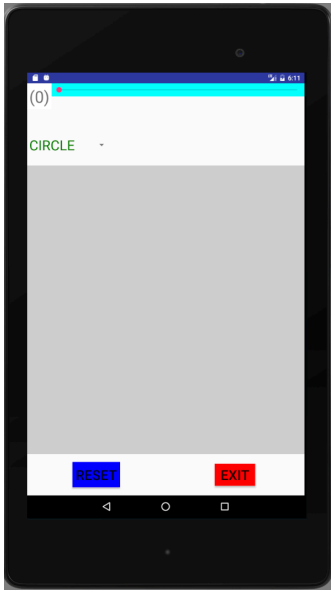
To be continued...

layout_weight 2

```
<LinearLayout [..]
    android:layout_height="0dp"
    android:layout_weight = "1"
    android:orientation="horizontal">
    <LinearLayout
        android:layout_width="0dp"
        android:layout_weight="1"
        android:layout_height="match_parent"
        android:gravity="center">
        <Button [..] />
    </LinearLayout>
    <LinearLayout
        android:layout_width="0dp"
        android:layout_weight="1"
        android:layout_height="match_parent"
        android:gravity="center">
        <Button [..] />
    </LinearLayout>
</LinearLayout>

</LinearLayout>
```

Orientation



layout land

Android studio

- layout-clic droit
- New - New ressource file
- Orientation » Landscape

Document XML

«Baliser» la structure d'une partition (exemple)

```
<partition tempo="...">
  <instrument_part name="..." octave="...">
    <note name="..." instant="..." duration="..."/>
    ⋮
  </instrument_part>
  ⋮
</partition>
```

DTD

```
<!DOCTYPE partition [
<!ELEMENT partition (instrument_part)+>
<!ATTLIST partition tempo CDATA #REQUIRED>
<!ELEMENT instrument_part (note)+>
<!ATTLIST instrument_part name CDATA #REQUIRED>
<!ATTLIST instrument_part octave CDATA #REQUIRED
<!ATTLIST note name CDATA #REQUIRED>
<!ATTLIST note instant CDATA #REQUIRED>
<!ATTLIST note duration CDATA #REQUIRED>
]>
```

Fichiers en lecture et écritures

Données dynamiques propres à l'application

Méthodes d'une instance d'`Application` ou de `Activity` (héritées de `Context`)

- Ouverture en écriture :
  `FileOutpuStream openOutputFile(String name, int mode)`
  Deux modes :
  - `MODE_PRIVATE` : création (où réinitialisation)
  - `MODE_APPEND` : pour écriture en fin de fichier (ajout)
- Ouverture en lecture :
  `FileInputStream openInputFile(String name)`

Écriture

Fichier texte

```
public void saveScore(String fname) {
    FileOutputStream oc = null;
    try {
      oc = openFileOutput(fname, Context.MODE_PRIVATE);
      //Parcours de la partition et écriture des balises
      oc.close();
    }
    catch (FileNotFoundException e) { }
    catch (IOException e) {
       try { oc.close(); } catch (IOException ee) {}
    }
}
```

Lecture

Lire et *analyser* le texte XML : construire une structure `Document`

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

  [..]

public Document readXMLFile(String fname) {
  try {
    DocumentBuilderFactory factory =
        DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory.newDocumentBuilder();
    return builder.parse(fname);
  } catch (Exception ex) {
    return null;
  }
}
```

Utiliser l'API DOM pour exploiter la structure `Document`

API DOM

```
import org.w3c.dom.*;
```

Élement racine : `theDoc.getDocumentElement()`
Descendants immédiats : `elt.getChildrenNodes()` (`NodeList`)
Nom de la balise : `elt.getNodeName()`
Liste de attributs : `elt.getAttributes()` (`NameNodeMap`)
Longueur des listes : `elts.getLength()`
Élement d'une `NodeList` : `elts.item(i)`
Valeur d'un attribut :
`((Attr)(elts.getNamedItem(attName))).getValue()`