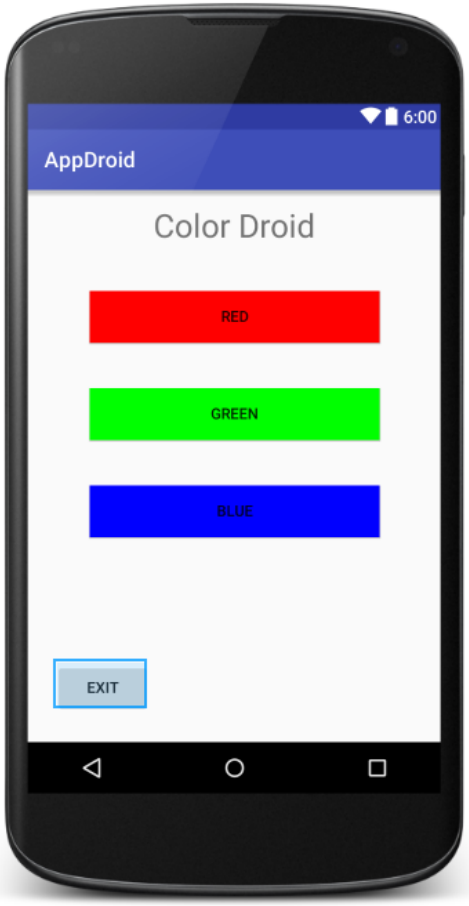
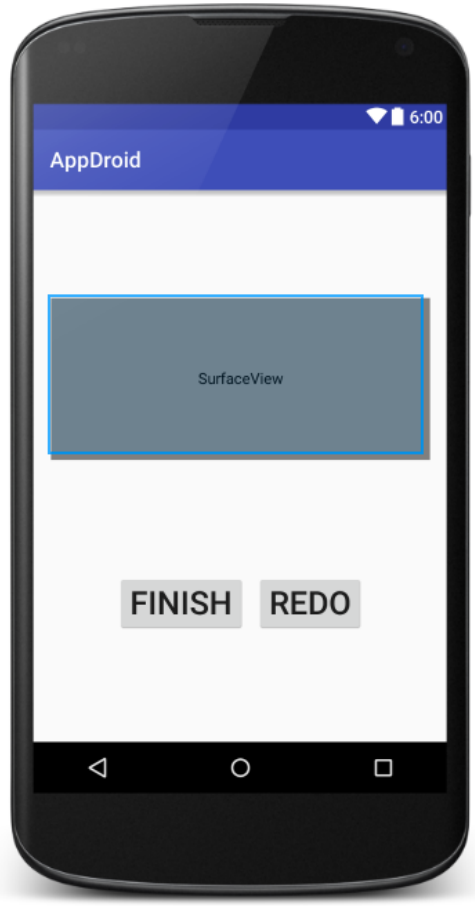
Deux activités dans une application



`MainActivity`



`GameActivity`

Un nouveau jeu

Le changement de couleur dépend de la position du toucher sur le surface.

+ de rouge	+ de vert	+ de bleu
- de rouge	- de vert	- de bleu

```
public interface IModel {
  int getColor();
  void resetColor();
  void changeColor(int c, boolean sign);
}
```

Plus un constructeur `GameModel(int color)`

AndroidManifest.xml

Déclarer la seconde activité

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

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

        <activity android:name=".GameActivity" />

    </application>
</manifest>
```

Balise orpheline (sans contenu)

activity_main.xml

Modifications : 3 boutons pour choisir le jeu (couleur de départ)

```
<RelativeLayout [..] >
    <TextView [..] />
    <LinearLayout [..] >
        <Button [..]
            android:id="@+id/redChoice"
            android:onClick="chooseGame" />
        <Button [..]
            android:id="@+id/greenChoice"
            android:onClick="chooseGame" />
        <Button [..]
            android:id="@+id/blueChoice"
            android:onClick="chooseGame" />
    </LinearLayout>
    <Button [..]
        android:onClick="onClickExit" />
</RelativeLayout>
```

Une méthode pour tous les choix

res/layout/activity_game.xml

Création : contient la surface de jeu et deux boutons (rejouer et finir)

```
<LinearLayout [..] >

    <android.appdroid.GameView [..] />

    <LinearLayout [..] >
        <Button [..]
            android:onClick="onClickFinish" />
        <Button [..]
            android:onClick="onClickRedo" />
    </LinearLayout>

</LinearLayout>
```

TheApplication

Initialiser une configuration de départ (couleur)

```
public class TheApplication extends Application {

    GameModel theGame;

    @Override
    public void onCreate() { [..] }

    void setGame(int color) {
        theGame = new GameModel(color);
    }

    GameModel getGame() { [..] }

}
```

MainActivity

Choisir la configuration de départ et lancement d'une activité

```
public void chooseGame(View button) {
    switch (((Button)button).getId()) {
        case R.id.blueChoice : {
            app.setGame(Color.BLUE); break;
        }
        case R.id.greenChoice : {
            app.setGame(Color.GREEN); break;
        }
        default : {
            app.setGame(Color.RED); break;
        }
    }
    Intent ChooseIntent =
       new Intent(this, GameActivity.class);
    startActivity(ChooseIntent);
}
```

GameActivity

La nouvelle activité

```
public class GameActivity extends Activity {
  TheApplication app;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_game);
  }
  public void onClickFinish(View view) {
     finish();
  }
  public void onClickRedo(View view) {
     app = (TheApplication)this.getApplication();
     app.getGame().resetColor();
     ((GameView)findViewById(R.id.gameView)).tryDrawing();
  }
}
```

GameView : réagir

Prendre en compte la position dans onTouchEvent

```
@Override
public boolean onTouchEvent(MotionEvent event) {
    int w = 350 / 3;  // UGLY CONSTANT
    int h = 150 / 2;  // UGLY CONSTANT
    int x = (int) event.getX();
    int y = (int) event.getY();
    int action = event.getAction();
    switch (action) {
        case MotionEvent.ACTION_DOWN: {
            int c;
            if (x < w) c = Color.RED;
            else if (x < 2*w) c = Color.GREEN;
            else c = Color.BLUE;
            app.getGame().changeColor(c, y<h);
            tryDrawing();
            return true;
        }
        [..]
```

Un *thread* d'affichage

Confier la mise à jour du rendu graphique de `gameView` à une *processus autonome*

```
class GameViewThread extends Thread {

    SurfaceHolder holder;
    GameView view;
    boolean running = false;

    public GameViewThread(SurfaceHolder holder, GameView view) {
        this.holder = holder;
        this.view = view;
    }

    public void setRunning(boolean b)
    {
        this.running = b;
    }

    [to be continues]
}
```

Un *thread* d'affichage (continued)

La boucle de dessin : méthode `run`

```
@Override
public void run() {
    Canvas c;
    while (this.running)
    {
        c = holder.lockCanvas();
        if (c != null) {
            this.view.onDraw(c);
            this.holder.unlockCanvasAndPost(c);
        }
    }
}
```

Début et fin du *thread* d'affichage

Dans la classe `GameView`

1) Un nouveau champ : `GameViewThread th;`

2) Création et démarrage

```
public void surfaceCreated(SurfaceHolder h) {
    th = new GameViewThread(getHolder(), this);
    th.setRunning(true);
    th.start();
}
```

3) Arrêt

```
public void surfaceDestroyed(SurfaceHolder h) {
    boolean retry = true;
    th.setRunning(false);
    while (retry) {
        try {
            th.join();
            retry = false;
        } catch (InterruptedException e) { }
    }
}
```

Mise-à-jour automatique du dessin

Dans `GameActivity`

```
public void onClickRedo(View view) {
    app = (TheApplication)this.getApplication();
    app.getGame().resetColor();
    //((GameView)findViewById(R.id.gameView)).tryDrawing();
}
```

Dans `GameView` : supprimer la définition de `tryDrawing()` et dans `onTouchEvent`

```
        case MotionEvent.ACTION_DOWN: {
            int c;
            if (x < w) c = Color.RED;
            else if (x < 2*w) c = Color.GREEN;
            else c = Color.BLUE;
            app.getGame().changeColor(c, y<h);
            // tryDrawing();
            return true;
        }
```

XML : analyse syntaxique

javax.xml.parsers.DocumentBuilderFactory et
javax.xml.parsers.DocumentBuilder

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

XML : DOM API

Interface générique : `Node`
Interfaces filles : `Document` `Element` `Attr`
Dans `Node` :

- `NodeList getChildsNode` listes des nœuds contenus

Dans `Document` :

- `Element getDocumentElement()` élément racine d'un noeud/document

XML : DOM API (suite)

Dans `Element` :

- `String getNodeName()` nom de l'élément (balise)
- `NamedNodeMap getAttributes` liste des attributs

Dans `NodeList` :

- `int getLength()` nombres de nœuds contenus
- `Node item(int i)` ième nœud

Dans `NamedNodeMap` :

- `int getLength()` nombre d'attributs contenus
- `Attr getNamedItem(String name)` attribut nommé `name`

Dans `Attr` :

- `String getValue()` valeur de l'attribut