UPMC/Licence/Info/2I013
Flowdroid – Android

Janvier 2015

Exemple de mise en œuvre

Un jeu pour les enfants programmeurs

Une variante du *taquin*

Une grille 9x9 dont les cases contiennent les chiffres de 1 à 9 dans un ordre aléatoire. Remettre les nombres dans l'ordre en échangeant les cases contigües.

Actions de jeu :

1. poser le doigt sur une case
2. le glisser jusqu'à une case voisine
3. quand on lève le doigt, le contenu des deux cases sont inversés

Un modèle du jeu

```
public class GaminModel
```

Fin de partie (les chiffres sont dans l'ordre)

```
  public boolean isAchieved()
```

Sélection d'une première case

```
  public boolean actionSelect(int x, int y)
```

Inversion de la case(x,y) avec la case sélectionnée

```
  public boolean actionSwap(int x, int y)
```

Nouvelle distribution aléatoire

```
  public void reNew()
```

Contenu de la case(x,y)

```
  public int get(int x, int y)
```

Les classes pour Android

Contient le modèle du jeu : création et accès

```
public class TheApplication extends Application
```

L'activité principale (et unique) : présente l'écran du jeu, définit la réaction aux boutons `NEW` et `EXIT`

```
public class GaminActivity extends Activity
```

La grille interactive du jeu : définit l'affichage l'état de la grille, réagit aux événements tactiles, lié au *thread* d'affichage.

```
public class GameView extends SurfaceView
        implements SurfaceHolder.Callback,
                   View.OnTouchListener
```

Le *thread* d'afficage : actualise périodiquement le dessin de la grille

```
class GameViewThread extends Thread
```

Le fichier AndroidManifest.xml

Description des composants de l'application : engendré par l'IDE puis modifié

```
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="eleph.android.games.gamin">
  <application android:label="@string/app_name"
               android:icon="@drawable/ic_launcher"
               android:name="TheApplication">
      <activity android:name="GaminActivity">
          <intent-filter>
           <action android:name="android.intent.action.MAIN"/>
           <category
              android:name="android.intent.category.LAUNCHER"/>
          </intent-filter>
      </activity>
  </application>
</manifest>
```

Ajout : `android:name="TheApplication"`

Que dit le «manifeste» ?

Balise `manifest`
C'est un manifeste pour les *android packages* (applications Android)
`xmlns:android="http://schemas.android.com/apk/res/android"`
Les composants JAVA de l'application sont dans le *package* JAVA
`package="eleph.android.games.gamin"`

Balise `application`
L'application possède sa propre (sous)classe de `Application`
`android:name="TheApplication"`
Le nom publié de l'application («écran des applications») est défini dans les ressources `values/strings.xml`
`android:label="@string/app_name"`
L'icone de l'application est dans les ressources `drawable*`
`android:icon="@drawable/ic_launcher"`

Que dit le «manifeste» ?

Balise `activity`
L'application contient l'activité `GaminActivity`
`android:name="GaminActivity"`

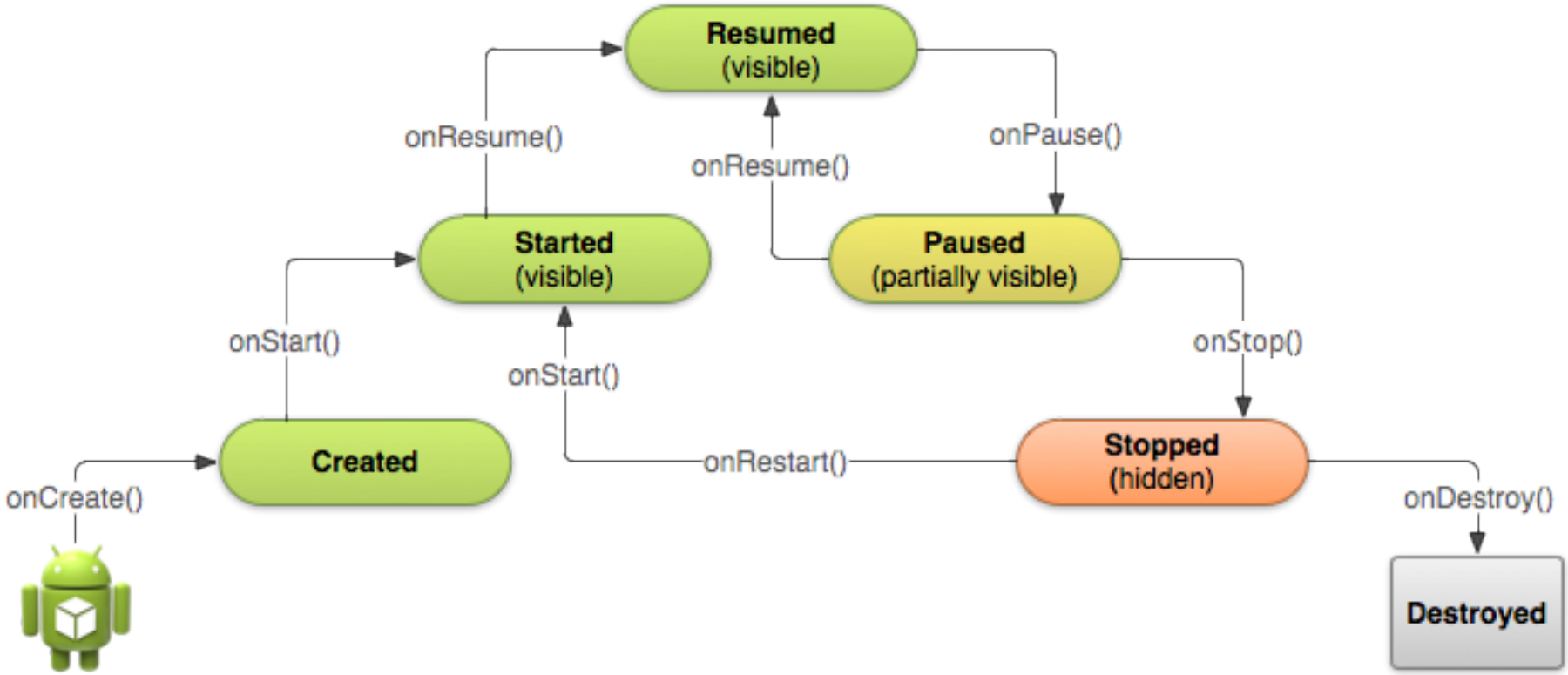
Balise intent-filter
Elle est déclarée comme étant l'activité principale `action`
`android:name="android.intent.action.MAIN"`
Elle est exécutable depuis «l'écran des applications» `category`
`android:name="android.intent.category.LAUNCHER"`

Lancement d'une application Android

Lorsque l'application est lancée (pression sur l'icône de l'application dans «l'écran des applications»)

1. Le fichier manifeste est lu par le système.
2. Il crée une instance de `TheApplication` et invoque sa méthode `onCreate`
   Cette instance persiste en mémoire tant que l'utilisateur ne quitte pas «explicitement» l'application ou que le système ne «récupère» pas la mémoire allouée.
3. Il crée une instance de l'activité principale `GaminActivity` et invoque sa méthode `onCreate`, puis sa méthode `onStart`.
   L'écran d'accueil définit pour l'activité devient visible.
   Cette instance est suspendue, arrêtée ou détruite lorsque l'application invoque une autre activité (écran) ou que l'utilisateur n'invoque pas une autre application.

Cycle de vie d'une activité



http ://developer.android.com/training/basics/activity-lifecycle/starting.html

TheApplication

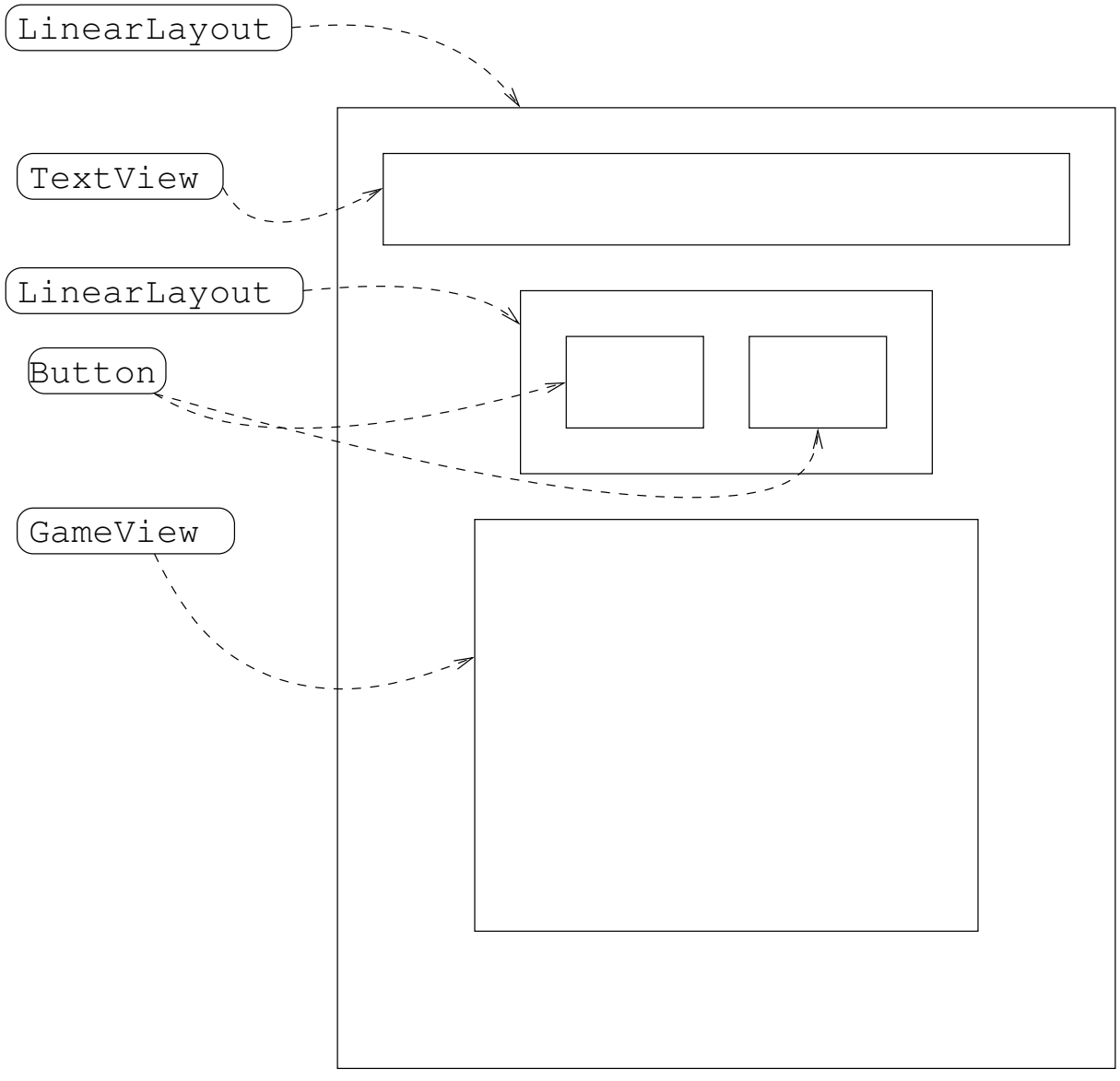
Redéfinition de onCreate et définition de getGame

```
public class TheApplication extends Application {
  GaminModel theGame;
  @Override
  public void onCreate() {
    super.onCreate();
    theGame = new GaminModel();
  }
  GaminModel getGame() {
    return theGame;
  }
}
```

GaminActivity

```
public class GaminActivity extends Activity {
    // référence sur (l'instance de) l'application
    TheApplication app;
    // visualisation et m-à-j réf. application
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main); // Visualisation
        app = (TheApplication)this.getApplication();
    }
    // Réactions aux boutons de l'activité
    public void onClickEXIT(View view) {
      finish();
    }
    public void onClickNEW(View view) {
      app.getGame().reNew();
    }
}
```

Composants de l'interface graphique



Fichier `main.xml`

Structure XML $\equiv$ Structure d'emboîtements graphiques

```
<LinearLayout [...] >

   <TextView [...] />

   <LinearLayout [...] >

     <Button [...] /> <Button [...] />

   </LinearLayout>

   <eleph.android.games.gamin.GameView [...] />

</LinearLayout>
```

Balises XML $\equiv$ Sous classes JAVA (`View`)

Composant principal

Contient tous les autres objets graphiques *alignés verticalement*
Emplit la totalité de l'écran en largeur et en hauteur

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

Composant texte

Ajusté en largeur et en hauteur autour de son contenu
Centré dans son conteneur
Le texte est défini dans les ressources `values/string.xml`

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:text="@string/main_message"
    />
```

Une zone pour les boutons

Contient des objets alignés horizontalement
Centré dans son conteneur
Ajusté en largeur et en hauteur autour de son contenu

```
<LinearLayout
    android:orientation="horizontal"
    android:layout_gravity="center"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    >
```

Un bouton

Identifié dans l'application par `new_button`
S'affiche avec l'intitulé `NEW`
Ajusté autour de son contenu (texte)
Centré dans son conteneur
Marges internes et externes
Couleur de fond : vert
Géré par la méthode `onClickNEW`

```
<Button
    android:id="@+id/new_button"
    android:text="NEW"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:padding="6dp"
    android:layout_margin="12dp"
    android:background="#0f0"
    android:onClick="onClickNEW" />
```

La grille de jeu

Composant définit par le programmeur sous le nom `GameView`
(*package* `eleph.android.games.gamin`)
Identifié dans l'application par `gameView`
(l'identifiant est engendré à la compilation)
Centré dans son conteneur
De la largeur et hauteur fixées

```
<eleph.android.games.gamin.GameView
    android:id="@+id/gameView"
    android:layout_gravity="center"
    android:layout_width="150dp"
    android:layout_height="150dp" />
```

La «surface» de jeu

Une sous-classe de `SurfaceView`

- qui implémente `View.OnTouchListener` pour réagir aux événements tactile
- qui implémente `SurfaceHolder.CallBack` pour interagir avec un *thread* d'affichage

```
public class GameView extends SurfaceView
        implements SurfaceHolder.Callback,
                   View.OnTouchListener
```

Ressource de la surface de jeu

Variables d'instances

- `app` : référence vers l'application
- `gameViewThread` : le *thread* pour l'affichage
- `paint` : contexte graphique (*style* et *couleur*) pour le dessin de la grille
- `canvasWidth` : largeur (en pixels) de la surface d'affichage
- `cellWidth` largeur (en pixels) de chaque case de la grille de jeu

```
TheApplication app;
GameViewThread gameViewThread;
Paint paint = new Paint();
int canvasWidth;
int cellSize;
```

Création de la surface de jeu

Constructueur

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

- `getHolder().addCallback(this)` est nécessaire pour que l'objet soit avisé de la création effective de sa surface d'affichage à l'écran
- `getApp` donne initialise la référence de l'application (voir ci-dessous)

```
final void getApp(Context context) {
  app = (TheApplication) (context.getApplicationContext()
}
```

NB : «`final`» car appelée dans un constructeur.

Création de la surface de jeu : post-traitement

Méthode invoquée lorsque la surface a été effectivement allouée sur l'écran : on peut alors commencer à dessiner et interagir tactilement

```
public void surfaceCreated(SurfaceHolder holder) {
  gameViewThread = new GameViewThread(getHolder(), this);
  gameViewThread.setRunning(true);
  gameViewThread.start();

  setOnTouchListener(this);
}
```

1. le *thread* `gameViewThread` est créé et «lancé»
   c'est un *processus* parallèle au *processus* de l'application qui appelle périodiquement la méthode de dessin de l'instance de `GameView` qui l'a créé et lancé
2. l'instance «s'abonne» aux événements tactiles

Destruction de la surface de jeu, pré-traitement

Arrêter le *thread* de dessin avant la destruction

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

Signale au `thread` sa fin et attend qu'il ait effectivement terminé.

Interactions de jeu

Méthode de traitement des événements tactiles : par cas, selon la position de l'événement

```
public boolean onTouch(View view, MotionEvent event) {
  int x = (int) event.getX();
  int y = (int) event.getY();
  int action = event.getAction();
  GaminModel theGame = app.getGame();
  switch (action) {
    case MotionEvent.ACTION_DOWN: {
      return theGame.actionSelect(x/cellSize, y/cellSize);
    case MotionEvent.ACTION_MOVE: {
      return true; } // on pourrait faire des trucs ici :)
    case MotionEvent.ACTION_UP: {
      return theGame.actionSwap(x/cellSize, y/cellSize); }
    default:
      return false;
  }
}
```

Définition de dessin de l'affichage

Méthode `public void onDraw(Canvas canvas)`

1. initialisations
2. couleur de fond (jaune si partie finie, gris, sinon)
3. traits de séparations des cases
4. placement des chiffres

Dessin 1 : initialisations

- largeur de la surface de dessin
- on en déduit la largeur des cases
- réinitialisation contexte de dessin
- état du modèle du jeu

```
canvasWidth = canvas.getWidth();
cellSize = canvasWidth / 3; //! Constante
paint.reset();
GaminModel theGame = app.getGame();
```

Dessin 2 : couleur de fond

Méthodes de dessin de la classe `Canvas`

```
if (theGame.isAchieved())
  canvas.drawColor(Color.YELLOW);
else
  canvas.drawColor(Color.GRAY);
```

Dessin 3 : traits de séparation

Définition de la couleur des traits, boucle pour les verticales, boucle pour les horizontales

```
paint.setColor(Color.BLACK);
for (int x = 0; x < canvasWidth; x += cellSize) {
  canvas.drawLine(x, 0, x, canvasWidth, paint);
}
for (int y = 0; y < canvasWidth; y += cellSize) {
  canvas.drawLine(0, y, canvasWidth, y, paint);
}
```

Dessin 4 : les chiffres

Taille des caractères et «*anti-aliasing*», boucle de dessin des chiffres contenu dans le modèle (boucle sur les indices du modèle, calcul de positionnement en mileu de case)

```
//! Constantes un peu partout
 paint.setTextSize(50);
 paint.setFlags(Paint.ANTI_ALIAS_FLAG);
 for (int x = 0; x < 3; x++) {
   for (int y = 0; y < 3; y++) {
     canvas.drawText(Integer.toString(theGame.get(x, y)),
                     (x * cellSize) + 11,
                     (cellSize + y * cellSize) - 6,
                     paint);
   }
 }
```

Réalisation du dessin : GameViewThread

Le dessin est réalisé par appel de la méthode `onDraw` de `GameView`.
L'appel de la méthode `onDraw` est déléguée à un processus autonome (*thraed*) lancé lorsque la surface d'affiche est créée.
L'appel à `onDraw` est placé dans la mathode `run` du *thread*.

```
@Override // version simplifiée
public void run() {
  Canvas c = null;
  while(running) {
    try {
      c = this.surfaceHolder.lockCanvas(null);
      synchronized (this.surfaceHolder) {
        if ((c != null) && (running))
           this.theGame.onDraw(c); }
    }
    finally  {
      if (c != null)
        this.surfaceHolder.unlockCanvasAndPost(c); }
  }
```