UNIVERSITE PARIS 8 VINCENNES-SAINT-DENIS

CONCEPTION SUR OS NOMADES

ANDROID

(rappels & conception d'un jeu)

INTRODUCTION A ANDROID

Plan du cours

Rappels:

- Qu'est ce qu'une application ou un jeu Android ?
- Les principales classes du SDK

Conception d'un projet de jeu:

- Organisation des ressources d'un projet Android
- Les fondamentaux du développement d'un jeu
- Exemple: un sokoban

UNIVERSITE PARIS 8 VINCENNES-SAINT-DENIS

CONCEPTION SUR OS NOMADES

INTRODUCTION A ANDROID

(RAPPELS)

Une application Android ?

DES COMPOSANTS, DES AUTORISATIONS, UNE CONFIGURATION & DES RESSOURCES

Les applications Android sont composées:

- D'un ou plusieurs composants
  - *Activity*
    (composant graphique correspondant à un écran)
  - *Services*
    (composant sans UI tournant en tâche de fond)
  - *Content Providers*
    (permet à l'application de partager des données)
  - *Broadcast Receivers*
    (répond aux notifications ou changement d'états)

- De ressources (images, textes, sons, fichiers xml, …).

UNIVERSITE PARIS 8 VINCENNES-SAINT-DENIS

Le fichier *AndroidManifest.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.storybird.bubblebuster"
      android:versionCode="4"
      android:versionName="1.3"
      android:screenOrientation="portrait" >
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.READ_CONTACTS"></uses-permission>
    <uses-permission android:name="android.permission.VIBRATE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.SEND_SMS" />



    <application android:icon="@drawable/icon" android:label="@string/app_name"
        android:launchMode="singleInstance" >
        <activity android:name=".BubbleBuster"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name="com.google.ads.AdActivity" android:configChanges="keyboard|keyboardHidden|orientation"/>

    </application>

 <uses-sdk android:minSdkVersion="7" />

<supports-screens
        android:largeScreens="true"
        android:normalScreens="true"
        android:smallScreens="true"
        android:anyDensity="true"
/>

</manifest>
```

LE MANIFEST DE L'APPLICATION

- **Description de l'application**
  - Package
  - Version
  - Orientation
- **Permissions**
- **Composants (Activity(s)…)**
- **Version minimum**
- **Ecrans supportés**

Les classes clés : *Activity*

LA CLASSE ACTIVITY

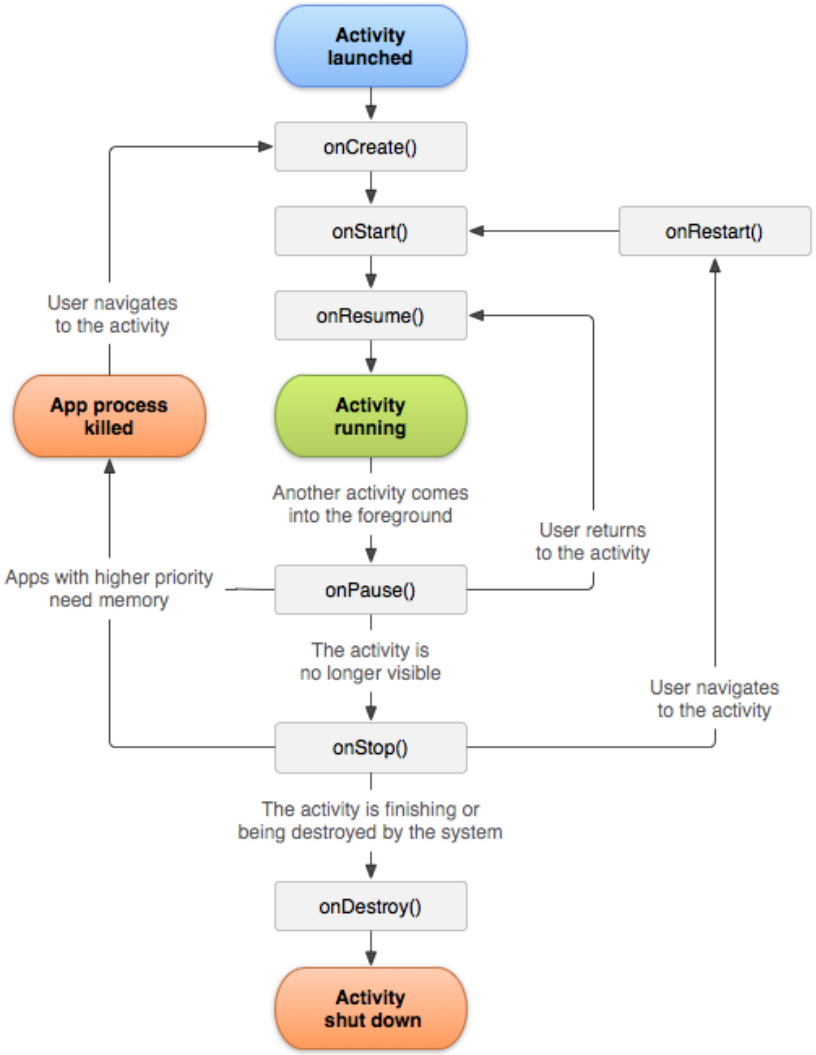
- **La classe *Activity* est l'unité d'exécution simple et visible.**
- **La classe *Activity* est en charge de la création d'un écran dans lequel vous pouvez placer votre UI avec *setContentView(int)*.**
- **2 méthodes doivent impérativement être implémentées :**
  - *onCreate(android.os.Bundle):* appelé lors de la création de l'*activity*. Vous y appellerez *setContentView(int)* avec un *Layout* définit pour votre UI.
  - *onPause:* appelé lorsque l'utilisateur quitte votre *activity*. Tous les changements réalisés par l'utilisateur sont à sauvegarder (en règle générale par le *ContentProvider*).

Les classes clés : *Activity*

CYCLE DE VIE

- Les *activity* au niveau système sont gérées sous la forme d'un pile. Quand une activity est démarrée elle est placée en haut de la pile, la précédente application suit derrière dans la pile.
- Une *activity* peut se trouver dans 4 états:
  - Active ou running (a le focus)
  - Paused (visible mais sans le focus)
  - Stopped (non visible)
  - Dropped (de la mémoire) doit être relancée.

```
public class Activity extends ApplicationContext {
            protected void onCreate(Bundle savedInstanceState);
            protected void onStart();
            protected void onRestart();
            protected void onResume();
            protected void onPause();
            protected void onStop();
            protected void onDestroy();
}
```



Pour en savoir plus: http://www.youtube.com/watch?v=fL6gSd4ugSI

Les classes clés : *View*

LA CLASSE VIEW

- La classe *View* est la classe de base de l'interface graphique.
- Une *View* occupe une zone rectangulaire sur l'écran et est responsable du dessin et de la gestion des évènements d'interaction.
- Les *View* d'une fenêtre sont arrangées sous la forme d'un arbre.
- Vous pouvez ajouter des *View* depuis le code ou depuis un ou plusieurs fichiers XML Layout.
- Il existe de nombreuses classes spécialisées qui fournissent des services pour afficher du texte, des images …
- La classe *View* est une classe haut niveau.

Les classes clés : *View*

LA CLASSE VIEW : implémenter une View customisée

- **Pour implémenter une *View* customisée, vous allez surcharger des méthodes comme *onDraw(android.graphics.Canvas)* ou le constructeur.**
- **Evènements customisables :**
  - CREATION
    - *onFinishInflate()* Appelé après qu'une *View* et ses sous classes soient créées d'un XML.
  - LAYOUT
    - *onMeasure()* Appelé pour déterminer la taille d'une *View* et de ses enfants.
    - onLayout() Appelé quand une vue repositionne l'ensemble de ses fils.
    - onSizeChanged() Appelé quand la taille de la vue change.
  - EVENT PROCESSING
    - onKeyDown() Appelé quand une touche est pressée.
    - onKeyUp() Appelé quand une touche est relâchée.
    - onTrackballEvent() Appelé quand une action est réalisée sur la trackball.
    - onTouchEvent() Appelé quand une action est réalisée sur l'écran tactile.

Les classes clés : *View*

LA CLASSE VIEW : implémenter une View customisée

- **Evènements customisables (suite):**
  - FOCUS
    - *onFocusChanged()* — Appelé quand la *View* perd le focus.
    - *onWindowFocusChanged()* — Appelé quand la fenêtre contenant la *View* perd ou gagne le focus.
  - ATTACHING
    - onAttachedToWindow() — Appelé quand la *View* est attachée à la fenêtre.
    - onDetachedFromWindow() — Appelé quand la *View* est détachée à la fenêtre.
    - onWindowVisibilityChanged() — Appelé quand la visibilité de la fenêtre contenant la *View* a changé.
  - DRAWING
    - *onDraw()* — Appelé quand la *View* doit dessiner son contenu.

CONCEPTION SUR OS NOMADES

ANDROID

(Structure d'un projet)

Les IDE pour Android

Différents IDE plus ou moins matures

- Eclipse, le sdk android et le plugin adt
- Android studio en beta
- IntelliJ

Structure d'un projet Android

- **Fichier AndroidManifest.xml**
  - Décrit l'application et les composants
- **Répertoires**
  - bin: contient l'application compilée
  - src: vos sources .java organisés en package
  - gen: contient R.java fichier auto généré décrivant les ressources
  - res: contient les ressources organisées par spécialisation (drawable, layout, menu, binaires, chaînes)
  - android library: librairies utilisées dans le projet
  - assets: contient les fichiers statiques de l'application

Utilisation des ressources

Gérer les ressources de son application

- **Les ressources sont des données statiques stockées en dehors du code java.**
- **Dans un projet Android, les ressources sont stockées sous le répertoire res.**
- **Avec Android vous pouvez gérer simplement des ensembles de ressources par configuration (écran, langue, densité, interaction, …)**
- **Les différents types de ressources:**
  - raw: contient les ressources brutes (non analysées automatiquement par le système)
  - layout: contient les xml décrivant la mise en page
  - anim: contient les animations courtes de l'interface utilisateur
  - drawable: contient les images et icônes de l'application
  - values: contient les chaînes, les couleurs, les tableaux et les dimensions
  - xml: contient nos propres données au format xml
  - menu: contient la description des menus

Référence Androïd Developper: *http://developer.android.com/guide/topics/resources/index.html*

Utilisation des ressources

Les avantages du système Android

- **L'accès simplifié aux ressources**

  On accède aux ressources grâce à une constante générée dans le R.java.

  Ex: R.drawable.background pour l'image background.png.

- **Gestion simple de la spécialisation des ressources**

  La spécialisation se base sur le nom des répertoires.

  Le répertoire racine contient les ressources par défaut les branches puis les feuilles les ressources plus spécialisées.

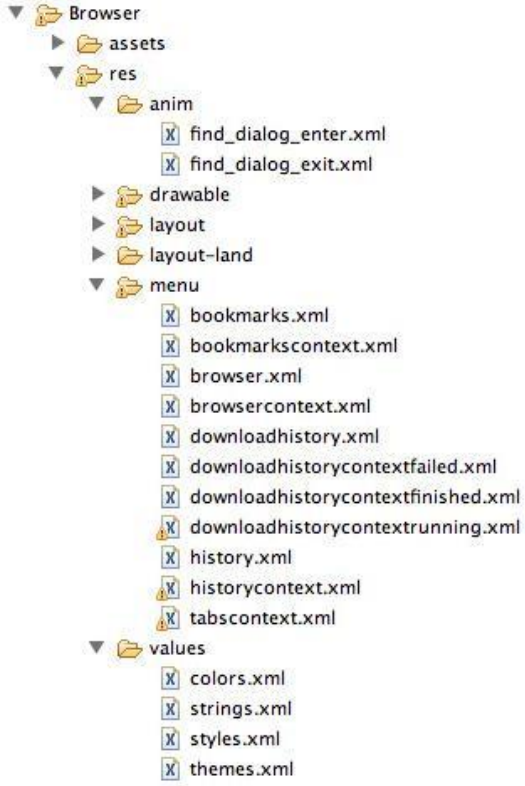
  Attention: toujours bien définir des ressources par défaut

- **Localisation**

  Vous pourrez facilement traduire votre application car les ressources sont gérées dans un fichier externe.

Utilisation des ressources

Gestion de la spécialisation des ressources



- Les paramètres de spécialisation sont séparés par des tirets.
- Le système va toujours chercher la ressource la plus spécialisée.

Les ressources de l'application

Externaliser la gestion des ressources

- **Externaliser la gestion des ressources vous permettra de gérer simplement les configurations spécifiques des terminaux Android (Taille d'écran…).**
- **Pour externaliser la gestion des ressources, vous devez organiser les ressources de votre projet dans le répertoire res/ et ses sous répertoires.**
- **Pour tous les types de ressources vous pouvez définir:**
  - Des ressources par défaut
  - Des ressources alternatives

Exemple:

- Layout par défaut sauvé dans res/layout/
- Layout en cas d'orientation paysage res/layout-land/
- Appplication automatique par le système de la bonne configuration

Figure 1. Two different devices, both using default resources.

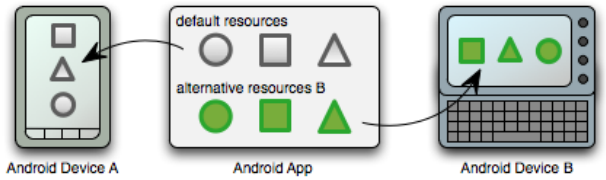


Figure 2. Two different devices, one using alternative resources.

Référence Androïd Developper: *http://developer.android.com/guide/topics/resources/index.html*

Les ressources de l'application

Externaliser la gestion des ressources

- **Fournir des ressources**

  Produire les ressources pour chaque configuration cible

- **Accéder aux ressources**

  Androïd propose un système d'accès aux ressources simplifié et centralisé

- **Gérer les changements à l'exécution**

  Androïd gérera le chargement des ressources durant le RunTime

  Le seul point d'attention: définir des configurations par défaut

- **Localisation**

  Simplement vous pourrez également localiser votre application

- **Types de ressources**

  Animation / Couleur / Images / Layout / Menu / Texte / Style …

Les ressources de l'application

Externaliser la gestion des ressources

- Arborescence de stockage des ressources -

Répertoire	Type de ressource
`anim/`	Fichiers XMLdéfinissant les animations.
`color/`	Fichiers XML définissant des liste de couleurs.
`drawable/`	Fichiers images
`layout/`	Fichiers XML définissant les layouts qui définissent un agencement de l'interface utilisateur
`menu/`	Fichiers XML qui définissent les menus de l'application (options, menu contextuel, sous-menu...)
`raw/`	Fichiers binaires
`values/`	Fichiers XML qui contiennent des valeurs simples, telles que les chaînes, entiers, tableaux ...
`xml/`	Divers fichiers de configuration XML (configuration de recherche, ...)

Les ressources de l'application

Produire votre gestion spécifique des ressources

- **Produire vos alternatives selon la nomenclature Androïd**

  Ex: drawable-port-hdpi

  Ressources graphiques en mode portrait

  pour les devices hdpi.

- **Les alias**

  Pour ne pas dupliquer vos ressources

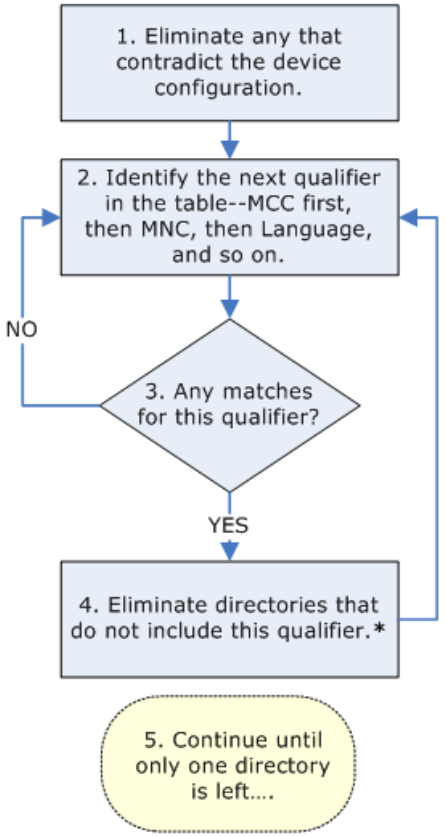
- !! Important !! **Ressources par défaut**

  → Toujours définir une ressource par défaut

  Langue / Image / Mode d'affichage (port / land)…

- **Fournir les meilleurs ressources**

  Itération du système pour déterminer le répertoire



* If the qualifier is the screen density, Android selects a "best" match and the process is done.

Les ressources de l'application

Accéder aux ressources

- Accédez aux ressources par leur identifiant et leur type

Ex: R.string.bonjourstr

→dans le code

@string/bonjourstr

→ dans le xml

```
public myView(Context context, AttributeSet attrs)
mContext  = context;
mRes                    = mContext.getResources();
block     = BitmapFactory.decodeResource(mRes, R.drawable.block);
```

- Le fichier R.java (auto généré)

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.

package p8.demo.p8sokoban;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int block=0x7f020000;
        public static final int diamant=0x7f020001;
        public static final int icon=0x7f020002;
        public static final int perso=0x7f020003;
        public static final int vide=0x7f020004;
        public static final int zone_01=0x7f020005;
        public static final int zone_02=0x7f020006;
        public static final int zone_03=0x7f020007;
        public static final int zone_04=0x7f020008;
    }
    public static final class id {
        public static final int SokobanView=0x7f050000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

CONCEPTION SUR OS NOMADES

ANDROID

(Conception d'un jeu)

Développer un jeu sous Androïd

Les points nécessaires à maîtriser pour développer un jeu 2D

- **Le chargement de ressources graphiques**
- **Le rendu graphique**
- **L'interaction avec le joueur**
  - Via le clavier
  - Via un écran tactile
- **L'animation**

La classe *Context*

Charger vos ressources graphiques

- La classe *Context* est une classe abstraite mise en œuvre par Androïd pour servir d'interface à l'environnement de l'application.
- *Context* permet d'accéder aux ressources spécifiques à l'application, en particulier les ressources graphiques (*Context.getResources()*).
- *Context* permet également la diffusion / réception d'*Intent* ainsi que le lancement d'*Activity*
- Vous accéderez grâce à *Context* aux ressources graphiques placées dans les répertoires « res ».

La classe *Resources*

Charger vos ressources graphiques

- **La classe *Resources* permet d'accéder aux ressources de l'application:**
  - Images
  - Layout
  - XML
  - Texte localisé
  - …

- **Vous utiliserez plus particulièrement la classe *Resources* pour charger vos images:**

```
public myView(Context context, AttributeSet attrs)
mContext      = context;
mRes          = mContext.getResources();
block         = BitmapFactory.decodeResource(mRes, R.drawable.block);
```

La classe *BitmapFactory*

Charger vos ressources graphiques

- La classe *BitmapFactory* est une classe basée sur la pattern objet Factory.
  Elle propose uniquement des méthodes statiques retournant des *Bitmap*.

- La classe *BitmapFactory* vous permet de créer vos images à partir:
  - De fichiers:
    decodeFile(String PathName)
  - De flux:
    decodeStream(InputStream is)
  - De tableaux de bytes:
    decodeByteArray(byte[] data, int offset, int length)
  - De ressource:
    decodeResource(Resources res, int id, BitmapFactory.Options opts)

La classe *SurfaceView*

- le rendu graphique -

- La classe *SurfaceView* dérive de *View* et offre l'accès à une surface qui vous permet de contrôler le rendu à l'écran (*draw()*).

- L'accès à la surface sous-jacente est fourni via l'interface *SurfaceHolder* en appelant *getHolder()*.

- La *SurfaceView* vous permet d'avoir la certitude que votre vue ne sera pas masquée et sera affichée à la position souhaitée:

  SetVisibility(), SetZOrderOnTop()

- NOTE: Pour la 3D vous avez *GLSurfaceView* qui dérive de *SurfaceView*.

La classe *SurfaceView*

Rappel: l'héritage de la classe *View* -

Category	Methods	Description
Creation	Constructors	There is a form of the constructor that are called when the view is created from code and a form that is called when the view is inflated from a layout file. The second form should parse and apply any attributes defined in the layout file.
	onFinishInflate()	Called after a view and all of its children has been inflated from XML.
Layout	onMeasure(int, int)	Called to determine the size requirements for this view and all of its children.
	onLayout(boolean, int, int, int, int)	Called when this view should assign a size and position to all of its children.
	onSizeChanged(int, int, int, int)	Called when the size of this view has changed.
Drawing	onDraw(Canvas)	Called when the view should render its content.
Event processing	onKeyDown(int, KeyEvent)	Called when a new key event occurs.
	onKeyUp(int, KeyEvent)	Called when a key up event occurs.
	onTrackballEvent (MotionEvent)	Called when a trackball motion event occurs.
	onTouchEvent(MotionEvent)	Called when a touch screen motion event occurs.
Focus	onFocusChanged(boolean, int, Rect)	Called when the view gains or loses focus.
	onWindowFocusChanged (boolean)	Called when the window containing the view gains or loses focus.
Attaching	onAttachedToWindow()	Called when the view is attached to a window.
	onDetachedFromWindow()	Called when the view is detached from its window.
	onWindowVisibilityChanged (int)	Called when the visibility of the window containing the view has changed.

La classe *SurfaceView*

- le rendu graphique -

- La surface de *SurfaceView* est créée dès que la fenêtre est visible.

- Pour être notifié du cycle de vie de la surface, il faut implémenter les callback suivants:
  - *surfaceCreated(SurfaceHolder)*
  - *surfaceDestroyed(SurfaceHolder)*

- Un des services de la classe *SurfaceView* est de fournir une Surface dans laquelle un second thread pourra produire le rendu graphique.

  Attention, les callbacks sont levés via le thread principal de l'application.

→Soyez attentifs à la synchronisation entre le thread principal et votre thread.

La classe *SurfaceHolder*

- l'interface à une surface d'affichage -

- **L'interface *SurfaceHolder* permet:**
  - De suivre les changements de la surface
  - De contrôler la taille et le format de la surface
  - D'éditer des pixels

- **Si vous utilisez un autre thread pour le rendu graphique, vous pourrez sécuriser vos actions de dessin avec:**
  - *lockCanvas()*
  - *unlockCanvasAndPost()*

- **Veillez à ce que la surface soit instanciée ! (*surfaceCreated()*)**

La classe *Canvas*

L'espace du rendu graphique

- La classe *Canvas* offre l'ensemble des services pour effectuer des traitements de dessin:
  - Clipping
  - Dessin d'images
  - Dessin de formes
  - Opacité
  - …
- Lorsque vous invoquez *lockCanvas()* via le *SurfaceHolder*, le *Canvas* est utilisé pour dessiner dans la *Bitmap* de la *Surface*.
- Attention: redessinez bien l'ensemble de la surface.

La classe *Paint*

Les paramètres de dessin

- La classe *Paint* vous permet définir le style et les couleurs de dessin.

- Elle sert de support aux méthodes de dessin en transmettant les paramètres de dessin de formes géométriques, de textes, d'images.

```
paint = new Paint();
paint.setColor(0xff0000);

paint.setDither(true);
paint.setColor(0xFFFFFF00);
paint.setStyle(Paint.Style.STROKE);
paint.setStrokeJoin(Paint.Join.ROUND);
paint.setStrokeCap(Paint.Cap.ROUND);
paint.setStrokeWidth(3);
paint.setTextAlign(Paint.Align.LEFT);
```

Le clipping

Réduire la zone du canvas impactée par les opérations de dessin

- La classe *Canvas* permet d'effectuer des opérations pour réduire sa zone de dessin impactée. Cela permet par exemple d'effectuer des animations en clippant chacune des frames d'un sprite.

- Android permet de sauvegarder l'etat du clip courant et de le restaurer.

```
canvas.save();
canvas.clipRect(
        xAnchBilles+j*Bille_TileSize,
        yAnchBilles+i*Bille_TileSize,
        xAnchBilles+(j+1)*Bille_TileSize,
        yAnchBilles+(i+1)*Bille_TileSize
        );
canvas.drawBitmap(
        PoolBitmaps[IMG_contours],
        xAnchBilles+(j-(getContourSelInt(i, j)-1))*Bille_TileSize ,
        yAnchBilles+i*Bille_TileSize-2*(Math.abs(billes[i][j])-2)*Bille_TileSize,
        pt2);
canvas.restore();
```

L'interaction clavier

Récupérer les actions utilisateurs sur le clavier ou les boutons

- Une *View* vous permet d'être notifié des actions clavier quand une touche est pressée (*onKeyDown*) ou relâchée (*onKeyUp*)
- Le callback indique le keycode de la touche concernée
- Il indique également un (*event* de type *KeyEvent*) contenant le contexte du bouton (pression multiple, …).
- La classe *KeyEvent* contient les constantes de référence (numéro, back, …)

```
/* ********************************************************** */
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if (keyCode == KeyEvent.KEYCODE_BACK) {
        switch (CurrentStep) {
            case STEP_CONTACTSMS:
            case STEP_STARTGAMEANIM:
            case STEP_MOREGAMES:
            case STEP_GAME:
                ChangeStep(STEP_MENU_MAIN);
                break;
            case STEP_MENU_MAIN:
            case STEP_LOADING:
                in = false;
                stopSound();
                activityparent.finish();
                break;
        }
        return true;
    }
    return super.onKeyDown(keyCode,event);
}
/* ********************************************************** */
```

Référence Androïd Developper: *http://developer.android.com/reference/android/view/KeyEvent.html*

L'interaction tactile

Récupérer les actions utilisateurs sur l'écran

- Une *View* vous permet d'être notifié des actions effectuées sur l'écran tactile (*onTouchEvent(MotionEvent event)*).

- Le *MotionEvent* transmis permet d'identifier le type d'action et les paramètres associés:
  - DOWN: je pose le doigt
  - UP: je relève le doigt
  - MOVE: je déplace le doigt

```
/* ********************************************************** */
public boolean onTouchEvent (MotionEvent event) {
    if (!inPack) {
    final int action = event.getAction();
    myx = 1*event.getX();
    myy = 1*event.getY();

    myx = myx / ((float) getWidth()/(float) myWidth());
    myy = myy / ((float) getHeight()/(float) myHeight());

    switch (action) {
        case MotionEvent.ACTION_MOVE:
            onTouchEventMove(event);
            return true;
        case MotionEvent.ACTION_UP:
            onTouchEventUp(event);
            return true;
        case MotionEvent.ACTION_DOWN:
            onTouchEventDown(event);
            return true;
        }
    }
    return super.onTouchEvent(event);
}
/* ********************************************************** */
```

Référence Androïd Developper: *http://developer.android.com/reference/android/view/MotionEvent.html*

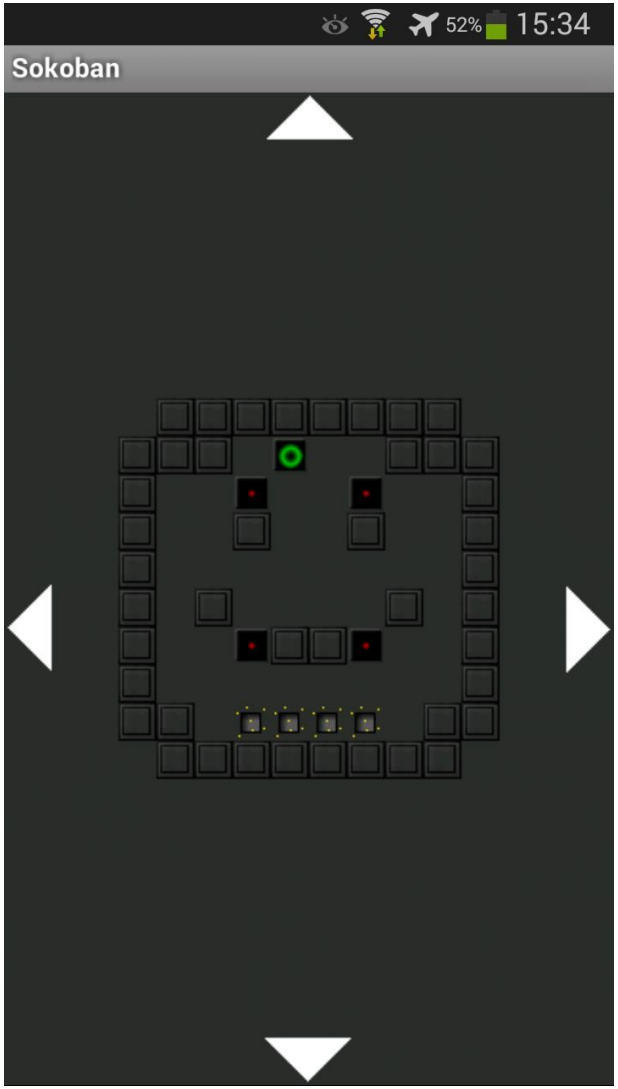
CONCEPTION SUR OS NOMADES

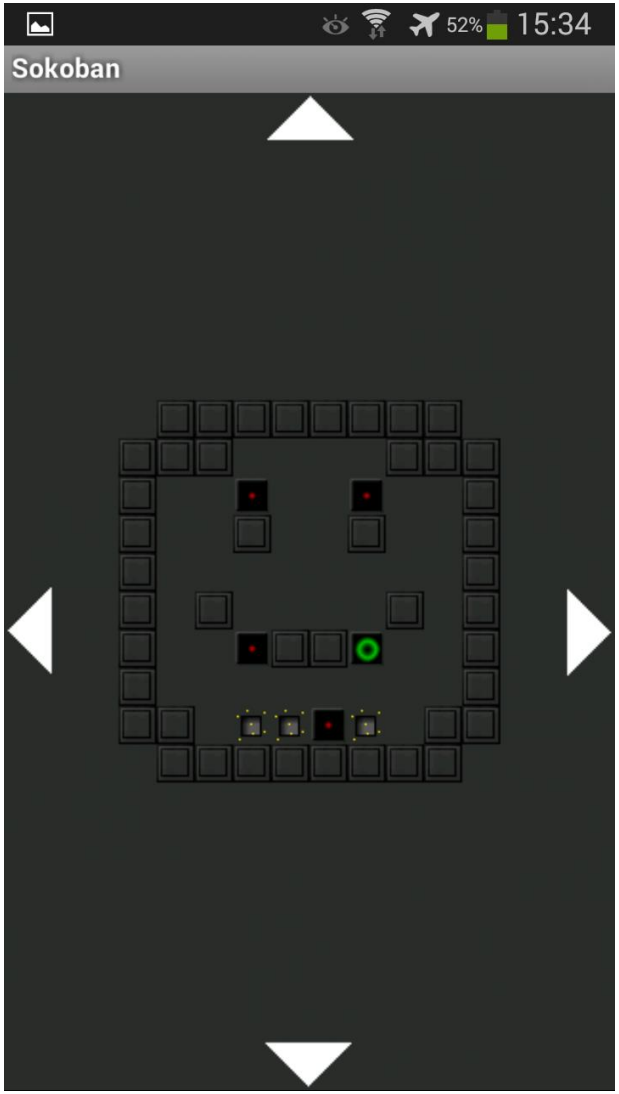
ANDROID

(EXEMPLE DU SOKOBAN)

Un exemple de jeu

Le sokoban





UNIVERSITE PARIS 8 VINCENNES-SAINT-DENIS

CONCEPTION SUR OS NOMADES

ANDROID

(ENRICHIR L'EXPERIENCE DANS LE JEU)

La classe *MediaPlayer*

Jouer des sons grâce au MediaPlayer

- On instancie un *MediaPlayer* en chargeant la ressource sonore via le contexte de notre application.
- On stocke les fichiers sonores dans le répertoire raw.
- On set le volume gauche et droite ainsi que la lecture en boucle ou non.

```
public void launchSound(int soundType, boolean loop){
    stopSound();
    try {
        switch (soundType) {
            case 0:
                mMediaPlayer = MediaPlayer.create(mContext, R.raw.son);
                break;
            case 1:
                mMediaPlayer = MediaPlayer.create(mContext, R.raw.pack);
                break;
            case 2:
                mMediaPlayer = MediaPlayer.create(mContext, R.raw.menu);
                break;
            case 3:
                mMediaPlayer = MediaPlayer.create(mContext, R.raw.perfect);
                break;
            case 4:
                mMediaPlayer = MediaPlayer.create(mContext, R.raw.gameover);
                break;
        }
    } catch (Exception e) {
        mMediaPlayer    = null;
    }
    if (mMediaPlayer != null) {
        mMediaPlayer.setVolume((1), (1));
        mMediaPlayer.setLooping(loop);
        mMediaPlayer.start();
    }
}

public void stopSound(){
    if (mMediaPlayer != null) {
        mMediaPlayer.stop();
    }
}
```

Référence Androïd Developper: *http://developer.android.com/reference/android/media/MediaPlayer.html*

Utiliser le vibreur

Vibrer sur une période ou en via une séquence

- On accède au vibreur via le context de notre application
- On joue une séquence de vibration ou une vibration simple
- Il faut déclarer l'usage du vibreur dans le androidmanifest.xml

```
// Déclaration du vibreur
private Vibrator    vibrator;
// Instanciation via notre context
vibrator        = (Vibrator)activityparent.getSystemService(Context.VIBRATOR_SERVICE);
// appel simple
vibrator.vibrate(200);
```

Public Methods	
void	cancel () Turn the vibrator off.
boolean	hasVibrator () Check whether the hardware has a vibrator.
void	vibrate (long[] pattern, int repeat) Vibrate with a given pattern.
void	vibrate (long milliseconds) Turn the vibrator on.

Persistance: *SharedPreferences*

Ecrire des données dans les préférences

Sous forme de clé/valeur

```
/* ********************************************************** */
public void saveDatas() {
    if (activityparent != null) {
        SharedPreferences settings = activityparent.getSharedPreferences(PrefName, 0);
        SharedPreferences.Editor editor = settings.edit();
        editor.putInt("Version", Version);

        editor.putBoolean("SoundAvailable", SoundActivated);
        editor.putInt("highscore", highscore);
        editor.putInt("skinType", skinType);
        editor.putBoolean("gameInProgress", gameInProgress);

        for(int i = 0; i <= Nb_Row_Bille - 1; i++) {
            for(int j = 0; j <= Nb_Col_Bille - 1; j++) {
                editor.putInt("b"+(100*i)+j, billes[i][j]);
            }
        }

        editor.putInt("score", score);
        editor.putInt("highscoreLevel", highscoreLevel);
        editor.putInt("gameMode", gameMode);
        editor.putInt("currentLevel", currentLevel);

        for(int i = 0; i < nbSkins; i++) {
            editor.putBoolean("skin"+i, skinAccess[i]);
        }

        editor.putInt("modePack", newPackMode);

        for(int i = 0; i < 2 ; i++) {
            for(int j = 0; j < 4; j++) {
                editor.putLong("stats"+(100*i)+j, stats[i][j]);
            }
        }


        editor.commit();
    }
}
/* ********************************************************** */
```

Persistance: *SharedPreferences*

Lecture des données dans les préférences

Sous forme de clé/valeur

```
/* ********************************************************** */
public void loadDatas() {
    if (activityparent != null) {
        SharedPreferences settings = activityparent.getSharedPreferences(PrefName, 0);
        Version         = settings.getInt("Version", 0);
        SoundActivated  = settings.getBoolean("SoundAvailable", false);
        highscore       = settings.getInt("highscore", 0);
        skinType        = settings.getInt("skinType", 0);
        gameInProgress  = settings.getBoolean("gameInProgress", false);

        for(int i = 0; i <= Nb_Row_Bille - 1; i++) {
            for(int j = 0; j <= Nb_Col_Bille - 1; j++) {
                billes[i][j]    = settings.getInt("b"+(100*i)+j, 0);
            }
        }

        score           = settings.getInt("score", 0);

        highscoreLevel  = settings.getInt("highscoreLevel", 0);
        gameMode        = settings.getInt("gameMode", 0);
        currentLevel    = settings.getInt("currentLevel", 0);

        for(int i = 0; i < nbSkins; i++) {
            skinAccess[i]   = settings.getBoolean("skin"+i, false);
        }
        skinAccess[0]   = true;
        highscoreStr    = String.valueOf(highscore);
        highscoreLvlStr = String.valueOf(highscoreLevel);
        modePack        = settings.getInt("modePack", 0);
        newPackMode     = modePack;

        for(int i = 0; i < 2 ; i++) {
            for(int j = 0; j < 3; j++) {
                stats[i][j] = settings.getLong("stats"+(100*i)+j, 0);
            }
        }

    }
}
/* ********************************************************** */
```

CONCEPTION SUR OS NOMADES

ANDROID

(ENRICHIR LES ECHANGES AVEC L'UTILISATEUR)

Toast: afficher un msg à l'utilisateur

LES TOASTS: REMONTER UNE INFORMATION A L'UTILSATEUR

- Un *Toast* vous permettra d'afficher un texte à l'utilisateur par-dessus votre application.

- Cette affichage est momentané et de courte durée, donc idéal pour notifier l'utilisateur et non bloquant pour les actions de l'utilisateur. Le principe est un peu similaire aux tooltips.

- Exemple d'utilisation de toast pour indiquer l'utilisateur n'a pas accès:

```
Toast l_toast = Toast.makeText(
        mContext,
        R.string.skinblocked,
        Toast.LENGTH_LONG
        );
l_toast.show();
```

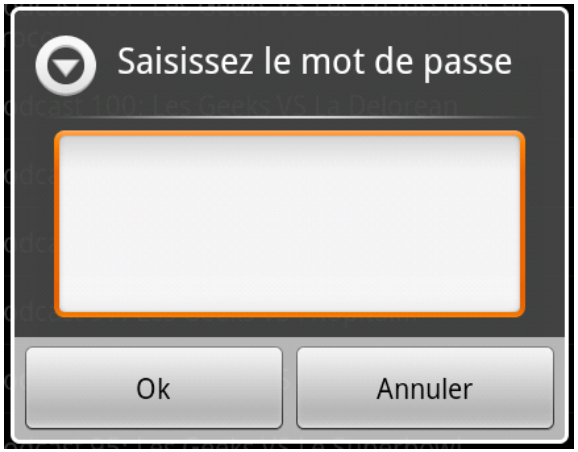
- Si vous threadez votre application, il n'est pas possible de lancer un toast depuis le thread, il vous faudra repasser par le thread principal.

```
activityparent.runOnUiThread(new Runnable() {
    public void run() {
        Toast l_toast = Toast.makeText(
                mContext,
                R.string.skinunlock,
                Toast.LENGTH_SHORT
                );
        l_toast.show();

        }
});
```

Layout & View: interface utilisateur

Exemple de création de vue haut niveau à la volée

```
/* ********************************************************** */
private void showAbout() {
    AlertDialog.Builder about = new AlertDialog.Builder(activityparent);
    about.setTitle(
        Html.fromHtml(
            "<b>  BUBBLE BUSTER</b>")
    );
    about.setIcon(R.drawable.icon);

    TextView l_viewabout    = new TextView(mContext);
    l_viewabout.setLayoutParams(new LayoutParams(LayoutParams.FILL_PARENT, LayoutParams.FILL_PARENT));
    l_viewabout.setPadding(20, 10, 20, 10);
    l_viewabout.setTextSize(20);
    l_viewabout.setText(
            Html.fromHtml(
                    "<small>Bubble Buster - &#169;Copyright 2011</small>"+
                    "<br/>"+"<br/>"+
                    "<b>Developped by:</b>"+
                    "<br/>"+
                    "<small>- Julien ROCCA</small>"+
                    "<br/>"+
                    "<small>- Ludovic GREMY</small>"+
                    "<br/>"+"<br/>"+
                    "<b>Published by:</b>"+
                    "<br/>"+
                    "<small>StoryBird: <a href=\"http://storybird.mobi\">storybird.mobi</a></small>"+
                    "<br/>"+"<br/>"+
                    "<b>Meet us on </b><a href=\"http://www.facebook.com/pages/Storybird/194195437285078\">Facebook&#0153;</a>"
                    )
    );
    l_viewabout.setMovementMethod(LinkMovementMethod.getInstance());

    about.setView(l_viewabout);
    about.setPositiveButton(
            "OK",
            new android.content.DialogInterface.OnClickListener() {

                @Override
                public void onClick(DialogInterface dialog, int which) {
                }
            }
            );

    about.setOnCancelListener(new android.content.DialogInterface.OnCancelListener() {

        @Override
        public void onCancel(DialogInterface dialog) {

        }
    }
    );

    about.show();
}
/* ********************************************************** */
```



MENU SYSTEMES ET CALLBACKS ASSOCIES

- Le menu système est accessible depuis le bouton home. Votre activity peut définir son propre menu système.

- *onCreateOptionsMenu:* appelé 1 seule fois pour permettre de construire le menu.
- *onPrepareOptionsMenu:* appelé une fois créé avant l'affichage du menu.
- *onOptionsItemSelected:* appelé lorsque l'utilisateur clique sur un item du menu.

```
/* ********************************************************* */
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    mMenu   = menu;
    return super.onCreateOptionsMenu(menu);
}
/* ********************************************************* */

/* ********************************************************* */
@Override
public boolean onPrepareOptionsMenu(Menu menu) {
    if (mMainView != null) {
        mMainView.RefreshOptionsMenu();
        return true;
    } else return false;
}
/* ********************************************************* */

/* ********************************************************* */
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    if (mMainView != null) {
        mMainView.OptionsItemSelected(item);
        return true;
    } else return false;
}
/* ********************************************************* */
```

Menus systèmes

MENU SYSTEMES ET CALLBACKS ASSOCIES

```
/* ********************************************************** */
public void RefreshOptionsMenu() {
    if (activityparent.mMenu != null) {
        activityparent.mMenu.clear();
        CreateOptionsMenu(activityparent.mMenu, CurrentStep);
    }
}
/* ********************************************************** */

/* ********************************************************** */
private boolean CreateOptionsMenu(Menu menu, int p_step) {
    MenuInflater inflater = activityparent.getMenuInflater();
    switch (p_step) {
        case STEP_MENU_MAIN:
            inflater.inflate(R.menu.main_menu, menu);
            return true;
        case STEP_GAME:
            inflater.inflate(R.menu.game_menu, menu);
            activityparent.mMenu.findItem(R.id.undo).setEnabled(undoAvailable);
            return true;
        default:
            return false;
    }
}
/* ********************************************************** */

/* ********************************************************** */
public boolean OptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.about:
            showAbout();
            return true;
        case R.id.help:
            showHelp();
            return true;
        case R.id.stats:
            showStats();
            return true;
        case R.id.options:
            showOptionsPack();
            return true;
        case R.id.undo:
            undo();
            return true;
        default:
            return false;
    }

}
/* ********************************************************** */
```

UNIVERSITE PARIS 8 VINCENNES-SAINT-DENIS

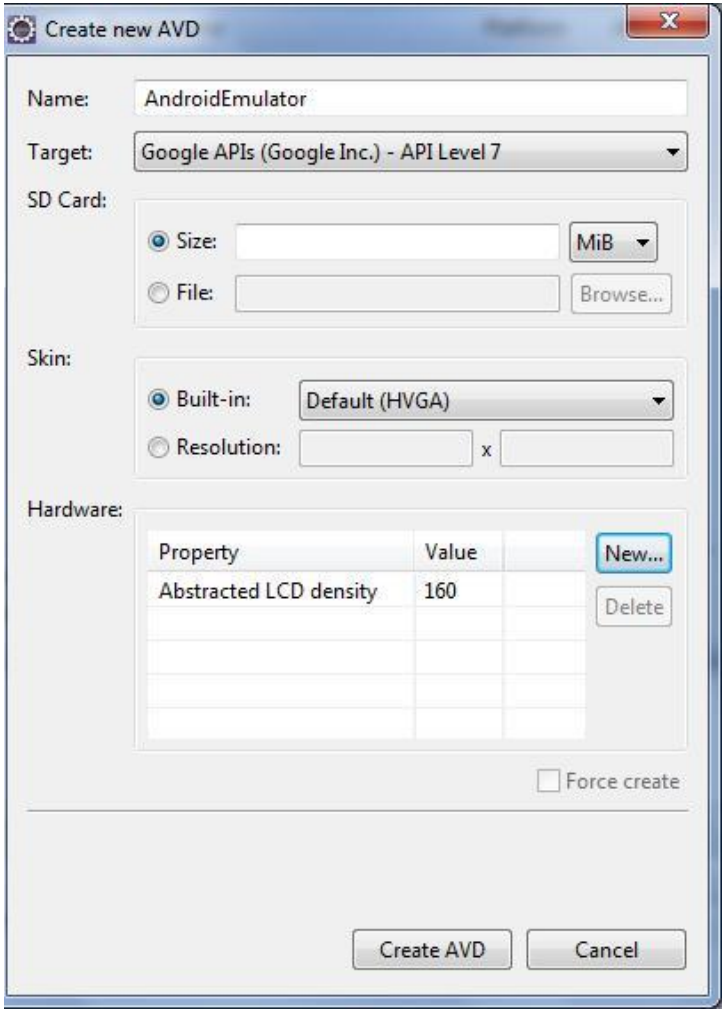
CONCEPTION SUR OS NOMADES

ANDROID

(POUR BIEN TRAVAILLER)

Créer son émulateur

Créer votre propre émulateur Android



- *Name:* Nom de votre choix (pas d'espace)
- *Target:* Version du sdk android utilisée par l'émulateur
- *SD Card:* configuration de la mémoire externe
- *Skins:* résolution de l'émulateur (pré configuré ou personnalisée)
- Hardware: pour customiser l'émulateur (clavier, GPS, accéléromètre, densité,…)

Directement sur le mobile

Connecter son mobile à Eclipse

- Passer le mobile en mode debug:

  Bouton Home > Settings > Applications > USB Debugging
- Le connecter en USB à la machine
- Passer le projet en choix manuel de l'environnement d'execution

  clic droit sur le projet > Run/Debug Settings > editer ou créer une configuration de lancement > target > sélectionner manuel
- Au prochain lancement de l'application depuis eclipse une popup apparaîtra pour choisir l'environnement d'exécution.
- Sur le mobile il est possible de faire du pas à pas et de profiter de tout le confort de debug (log, point d'arrêt, thread, …).

Comment débugger ?

- Le Logcat -

- Logcat vous permettra d'avoir toutes les informations sur l'exécution de l'émulateur.

- Logcat vous permet de définir plusieurs niveaux de logs qui seront filtrables:
  - *Verbose*
  - *Debug*
  - *Info*
  - *Warning*
  - *Error*

- Ajouter Logcat dans votre vue java: Window>ShowView>Others>Logcat

- Appel: *Log.e(« groupe », « msg »)* ou *Log.i(« groupe », « msg »)*

CONCEPTION SUR OS NOMADES

ANDROID

(SQL LITE)

Persistance: *SQLite*

Une base de données évoluée et simplifiée: SQLite

- Android ne fournit aucune base de données. Si vous voulez utiliser *SQLite*, vous devez créer votre base et la remplir.
- Pour créer et ouvrir une base de données, la meilleure solution consiste à créer une sousclasse de *SQLiteOpenHelper*. Cette classe enveloppe tout ce qui est nécessaire à la manipulation d'une base.
- Cette sous-classe aura besoin de trois méthodes :
  - Un constructeur qui appelle celui de sa classe parente et prend en paramètre le *Context*, le nom de la base, une éventuelle fabrique de curseur (le plus souvent, ce paramètre vaudra null) et un entier représentant la version du schéma de la base.
  - *onCreate()*, à laquelle vous passerez l'objet *SQLiteDatabase* que vous devrez remplir avec les tables et les données initiales que vous souhaitez.
  - *onUpgrade()*, à laquelle vous passerez un objet *SQLiteDatabase* ainsi que l'ancien et le nouveau numéro de version. Pour convertir une base d'un ancien schéma à un nouveau, l'approche la plus simple consiste à supprimer les anciennes tables et à en créer de nouvelles.

Persistance: *SQLite*

Une base de données évoluée et simplifiée: SQLite

- Créer une instance de base de données:
  - Pour utiliser votre sous-classe, créez une instance et appelez *getReadableDatabase()* ou *getWriteableDatabase()*
  - Ex: *db = (new DatabaseHelper(getContext())).getWritableDatabase();*
  - Cet appel renverra une instance de SQLiteDatabase qui vous servira ensuite à interroger ou à modifier la base de données.
  - Lorsque vous avez fini de travailler sur cette base, il suffit d'appeler la méthode close() de cette instance pour libérer votre connexion.

- Créer une table dans une instance de base de données:
  - On utilise execSQL pour executer notre requête de création.
  - *db.execSQL*("CREATE TABLE constantes (_id INTEGER PRIMARY KEY AUTOINCREMENT, titre TEXT, valeur REAL);");

Persistance: *SQLite*

Une base de données évoluée et simplifiée: SQLite

- Ajouter des données dans la base de données:
  - Via *execSQL*:

```
db.execSQL("INSERT INTO widgets (name, inventory)"+"VALUES ('Sprocket', 5)");
```

  - Via les *ContentValues* :

```
ContentValues cv=new ContentValues();
cv.put(Constantes.TITRE, "Gravity, Death Star I");
cv.put(Constantes.VALEUR, SensorManager.GRAVITY_DEATH_STAR_I);
db.insert("constantes", getNullColumnHack(), cv);
```

- Lire des données dans la base de données:
  - On utilise un Cursor:

```
Cursor c=db.rawQuery("SELECT name FROM sqlite_master WHERE type='table' AND
name='constantes'", null);
```