Interface graphique

# Objectifs

Dans un premier temps, l’application étant utilisée sur un téléphone Android tactile, il doit être ergonomique (la taille et la position des touches doivent être adaptées à n’importe quel type de doigt), intuitif (on doit pouvoir, d’un seul coup d’œil, savoir où cliquer en fonction de ce que l’on veut faire) et fluide (l’application doit être la plus rapide possible).

Dans un second temps, sachant que nous développons une application gérant la musique, notre interface graphique doit contenir les éléments suivants : un bouton play/pause de lecture, musique suivante/précédente, une barre de progression synchronisée avec la musique, un accès à la liste des musiques stockées sur la sdcard et enfin tous les boutons des fonctionnalités à réaliser (boucle, equalizer, etc).

# Construction sous eclipse

Sous Android l'interface graphique est gérée dans des fichiers autres que les fichiers Java, l'interface se construit dans des fichiers XML présents dans le dossier "res " du projet.

L’unité de base pour les interfaces graphique sur la plateforme Android est la “View“ (composant de l’interface graphique avec lequel un utilisateur peut interagir). C’est une structure de données dont les propriétés stockent la présentation et le contenu d'une zone rectangulaire bien spécifique de l'écran, sa mise en page, dessin, les changements de focus (élément affiché qui recoit les ordres de l’utilisateur), le défilement... Tous les éléments de l’interface héritent de la classe “View“.

Grâce au plugin (logiciel qui complète un logiciel hôte pour lui apporter de nouvelles fonctionnalités) ADT installé sur Eclipse, nous disposons d’un outil simplifiant la construction de l’interface graphique. En effet, cet outil fournit une liste de “Views“ qu’il suffit de faire glisser sur une zone rectangulaire représentant l’écran du téléphone. L’outil génère ensuite automatiquement le code XML représentant cette insertion. On peut néanmoins modifier à volonté le fichier XML.

De nombreux éléments sont disponibles par défaut grâce à cet outil. On peut notamment trouver la TextView, EditText, Button, RadioButton, Checkbox, Seekbar, etc…

**(3.png : exemple d’éléments)**

Certains événements (action qui se produit par exemple lorsque l’on clique sur un bouton) peuvent être écrits en XML (changement d’icones quand on clique sur un bouton, animation, …) mais la plupart doivent être écrit en java dans la classe principale de notre projet. L’attribution d’une action sur un évènement se fait de la même manière que pour un projet java classique.

Le projet est également constitué de ressources et d’un fichier R.java qui sont utilisés pour la construction de l’interface graphique :

Les ressources sont des fichiers externes (fichiers non-code) qui sont utilisés par notre code et compilé dans notre application au moment de la compilation. Elles sont toutes stockées dans le répertoire “res“ du projet. Android supporte un certain nombre de différents types de fichiers de ressources, y compris XML, PNG et JPEG. Les fichiers XML sont des formats très différents en fonction de ce qu'ils décrivent. Les ressources sont externalisées du code source. Les fichiers XML sont compilés dans un format binaire pour des raisons d'efficacité. Les chaînes de caractères sont compressées dans une forme de stockage plus efficace.

Le fichier R.java est automatiquement générer, il indexe toutes les ressources de notre projet. On utilise cette classe dans notre code source comme une sorte de passage pour référencier toutes les ressources qu’on veut inclure dans notre projet. Par exemple, pour attribuer une animation à un élément de notre interface, on accèdera à la ressource (fichier xml qui décrit l’animation) par “R.anim.monanimation“ (monanimation.xml se trouvant dans le répertoire res/anim)

## Liste des ressources

* Animations 🡪 res/anim, accessible par R.anim
* Couleurs 🡪 res/color, accessible par R.color
* Images et variations d’images 🡪 res/drawable, accessible par R.drawable
* Mises en page de notre interface 🡪 res/layout, accessible par R.layout
* Menus (contenu des différents menus de notre projet) 🡪 res/menu, accessible par R.menu
* Textes et tableaux de textes 🡪 res/values, accessible par R.string, R.array
* Styles (apparence et format des élements de l’interface) 🡪 res/values, accessible par R.style
* Fichiers raw comme mp3s ou videos 🡪 res/raw, accessible par R.raw
* Autres tels que les booleans, entiers, dimensions, … 🡪 res/values, accessible chacun par R.bool, R.integer, R.dimen, etc.

## Construction détaillée d’une interface

Une fois le projet de type “Android Application“ crée sous Eclipse, un fichier représentant l’interface graphique est automatiquement créé et stocké dans le répertoire “res/layout“ du projet.

Pour que l’application comprenne que ce fichier correspond à celui qu’il faut lancer, la classe java de notre activité principale doit contenir :

**public** **class** tuto **extends** Activity {

/\*\* Appelle quand l’activité est créé pour la première fois \*/

@Override

**public** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(**R.layout.*main***);

}

}

“main“ étant le nom du fichier xml contenant notre interface.

L’“activity“ est le fichier java lié à notre interface. Il permet de gérer les évènements se produisant sur celle-ci.

Lorsqu’on ouvre le fichier XML de notre interface, grâce au plugin ADT, on a, au centre, une zone rectangulaire représentant l’écran du téléphone, à gauche, une liste d’éléments que l’on peut placer et, à droite, la hiérarchie des éléments présents sur notre interface. De plus, si on sélectionne un des éléments déjà placé, on peut modifier ses propriétés grâce à l’onglet “propriétés“ situé en bas de la fenêtre.

**(1.png : présentation de l’outil ADT d’éclipse pour la création de l’interface graphique)**

### les layouts

Avant de commencer à placer des éléments sur notre interface il faut définir l’allocation de l’espace de l’écran grâce à des “Layout“.

Ces derniers représentent l’agencement des différents éléments graphiques dans votre interface.  
Voici quelques exemples de layouts :

* LinearLayout : Il organise les différents éléments de l’interface sur une ligne ou sur une colonne.
* AbsoluteLayout : Cette mise en page laisse définir les coordonnées exactes des éléments.
* RelativeLayout : Il permet de définir la position des éléments en fonction de la position de leurs éléments parents ou d’autres elements (à droite de, au dessus de, …).
* TableLayout : il range les enfants sous forme de lignes ou de colonnes. Il est composé d’un ensemble de TableRow, chacun définissant une ligne ou une colonne. Les éléments sont ensuite insérés dans cette objet.

Une fois que l’on en a choisi un et qu’on l’a fait glissé au centre sur la zone rectangulaire simulant l’écran, on peut placer nos différents éléments tels que des zones de texte, des boutons, des boutons images, une barre de progression, etc … toujours en faisant glisser l’élément de la liste vers l’endroit souhaité sur l’écran. Une fois un élément placé, on peut modifier ses propriétés à volonté. Pour un bouton on peut modifier sa taille, le texte, le fond, l’espace avec les autres éléments, etc). Pour attribuer une image (on le nomme img.png pour notre exemple) à un élément “ImageButton“ il faudra noter dans la propriété “Background“ : @drawable/img (notre image étant placé dans le répertoire res/drawable de notre projet).

### Le fichier XML de notre interface

Lorsque l’on place les éléments que l’on veut dans la zone rectangulaire représentant l’écran du téléphone, tout ceci est en réalité écrit en XML.

Ainsi, par exemple pour une interface de cette sorte :

**(4.png : interface de test correspondant au code XML suivant)**

On aura le code XML suivant :

<?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"*>

<Button android:id=*"@+id/Button01"* android:layout\_height=*"wrap\_content"*

android:text=*"Play"*

android:textStyle=*"bold"*

android:textSize=*"30dp"* android:layout\_gravity=*"center\_horizontal"* android:layout\_width=*"fill\_parent"*

android:layout\_margin=*"10dp"*></Button>

<SeekBar android:id=*"@+id/SeekBar01"* android:layout\_height=*"wrap\_content"* android:layout\_width=*"match\_parent"* android:layout\_margin=*"10dp"*></SeekBar>

<ImageButton android:id=*"@+id/ImageButton01"* android:background=*"@drawable/image"* android:layout\_gravity=*"center\_horizontal"* android:layout\_height=*"150dp"*

android:layout\_width=*"150dp"* android:layout\_margin=*"10dp"*></ImageButton>

</LinearLayout>

LinearLayout : mise en place choisie pour notre interface

Dans cette mise en page on a placé trois éléments : un bouton, une SeekBar (barre de progression) et une image bouton.

Tous ce qui commence par “android :“ correspondent aux différentes propriétés que l’on a affecté à nos objets. On retrouve notamment pour chacun un id (identifiant unique pour chaque objet), layout\_height (taille en hauteur. Wrap\_content : taille de l’élément par défaut ou fill\_parent : taille du parent), mêmes caractéristiques pour layout\_width mais pour la largeur. On retrouve également layout\_margin qui correspond à l’espace libre que l’on veut autour de notre élément.

### les animations

On peut ensuite associer des animations, grâce à d’autres fichiers XML, sur nos éléments. On détaillera le fonctionnement de deux animations (celles utilisées dans notre projet) :

* Lorsque l’on clique sur un bouton image on veut que quand le bouton est pressé, l’image change et quand on retire le doigt, le bouton revient à son état d’origine.

Pour cela on créé un nouveau fichier xml que l’on place dans le répertoire drawable. Ce fichier sera constitué de cette manière :

<?xml version=*"1.0"* encoding=*"utf-8"*?>

<selector xmlns:android=*"http://schemas.android.com/apk/res/android"*>

<item android:state\_pressed=*"true"*

android:drawable=*"@drawable/cdclick"* /> <!-- pressed -->

<item android:drawable=*"@drawable/cd"* /> <!-- default -->

</selector>

“cdclick“ étant l’image au moment où le bouton est pressé et “cd“ l’image par défaut.

Pour affecté cette animation à un élément il suffit de modifier la propriété “Background“ et de lui mettre @drawable/nomdufichier

* On veut, lorsque l’on clique sur un bouton, afficher un autre élément par un effet de glissé du haut vers le bas.

Comme pour la description précédente, on va créer un nouveau fichier xml que l’on place dans le répertoire res/anim. Ce fichier sera consitué de cette manière :

<set

xmlns:android=*"http://schemas.android.com/apk/res/android"*

android:interpolator=*"@android:anim/accelerate\_interpolator"*>

<translate

android:fromYDelta=*"-100%"*

android:toYDelta=*"0%"*

android:duration=*"1000"*

/>

</set>

- Accelerate\_interpolator : Il correspond au type de l’animation. L’animation commence doucement puis s’accélère. Il existe de nombreux autres type comme RotateAnimation (contrôle la rotation de l’objet), linearInterpolator (le rythme du changement d’état est constant), etc.

- translate : animation qui joue sur la position de l’élément. Ici on le fait apparaître de haut vers le bas. “duration“ définie la durée de l’animation en millisecondes. Ici, on a fixé la durée à 1 seconde. On peut avoir aussi “Alpha“ (animation sur la transparence), scale (animation sur la taille), rotate (animation sur la rotation).

Pour affecter l’animation à un objet et pour qu’elle se lance à un moment précis, cela doit être définie dans la classe java de notre activité :

On commencer par récupérer l’objet que l’on veut animer. Tous les objets de notre interface sont récupérables par leur id

FrameLayout frame = (FrameLayout) findViewById(R.id.*fenetreAAfficher*);

On crée un objet “Animation“ en le liant au fichier que l’on vient de créer

Animation a = AnimationUtils.*loadAnimation*(AAP.*activity*, R.anim.*fichiernAnimation*);

On lui affecte ensuite l’animation et on la lance

frame.startAnimation(a);

### les évènements

Pour un évènement “clic“, tous les éléments disposent de la propriété “On click“. Il suffit de remplir cette propriété par le nom de la méthode qui doit être lancé lorsque l’on clique sur l’évènement. La méthode sera ensuite déclarée dans le fichier java de notre activité sous la forme :

**public** **void** methode(View v){ … }

Elle doit obligatoirement avoir comme attribut un objet “View“. Celui correspond à l’élément qui aura appelé la méthode.

Pour les autres types d’évènements tels que le pressé/relevé sur un bouton, la modification par glissé de la position du curseur d’une barre de progression, … doivent être déclarés entièrement en java comme un projet classique.

Par exemple pour le pressé/relevé sur un bouton :

((ImageButton) findViewById(R.id.*monBouton*)).setOnTouchListener( **new** OnTouchListener() {

@Override

**public** **boolean** onTouch(View v, MotionEvent event) { … }

}

On récupère l’objet et on lui affecte un écouteur sur un évènement.

# Evolution de l’interface graphique

## Démarrage

N’ayant jamais crée d’application pour téléphone Android auparavant, les premières étapes ont consisté à créer une interface graphique basique, comportant les éléments principaux, tout en découvrant le fonctionnement et les possibilités de l’outil de développement Android de eclipse. L’interface a donc évolué en fonction de notre découverte et de notre perfectionnement à cet outil.

On décomposera l’évolution de l’interface graphique en cinq grandes étapes majeures.

## première étape

Elle contient tous les boutons des fonctionnalités que nous voulons réaliser. La structure de l’interface a été définie et ne changera plus dans les prochaines versions :

* titre, auteur de la chanson, accès a la liste des musiques en haut de l’écran
* fonction de base d’un lecteur de musique en bas de l’écran (barre de progression, bouton play/pause, musique suivante/précédente)

Aucun bouton n’est encore fonctionnel mais elle dispose déjà de la lecture d’une musique et d’une barre de progression synchronisée avec celle-ci. La musique est stockée en tant que ressource dans le répertoire “res/raw“ de notre projet.

On a également eu l’idée de rajouter la possibilité d’afficher les paroles et les partitions dans notre application. Le centre de l’écran a donc été réservé à cet effet.

**(2.png : première version de notre interface)**

## deuxième étape

La présentation n’a pas subit de grandes modifications. On a cependant ajouté l’accès à la liste des musiques stockées sur la sdcard et la fonctionnalité “equalizer“.

On a également commencé à intégrer à l’interface graphique le réglage de début et de fin de boucle.

On voulait ajouter deux curseurs sur la barre de progression de la musique. On pourrait cliquer directement dessus et la faire glisser pour modifier les valeurs de la boucle.

**(7.png : objectif final pour la barre de progression)**

Après de nombreuses recherches, nous n’avons pas réussi à trouver le moyen de rajouter deux nouveaux curseurs que l’on puisse déplacer comme le curseur par défaut de la barre de progression. Nous avons donc commencé, en attendant de trouver une autre solution, par afficher une fenêtre en pop-up (nouvelle fenêtre s’ouvrant au dessus de la fenêtre actuelle). La position de début et de fin de boucle est alors rentrée en saisissant directement les valeurs au clavier.

**(6.png : première version du réglage de la boucle)**

## troisième étape

L’interface a subit un changement complet du graphisme, tout en gardant toujours la même structure définie à la première étape. En effet, après de nombreuses recherches sur la construction d’une interface graphique avec notre outil de développement Android, on n’a été en mesure d’améliorer ce qui avait été fait au départ.

On a également amélioré le réglage des valeurs de la boucle en supprimant le pop-up et en mettant à la place des boutons “+“ et “–“ de réglage. On a également réussi à placer des curseurs au même niveau que la barre de progression. Ces curseurs sont en réalité deux autres barres de progressions placées derrière la barre de progression de la musique et que l’on a personnalisé en leur définissant un style de telle sorte qu’un utilisateur lambda comprenne directement qu’il s’agit des positions de début et de fin de la boucle.

Un bouton stop a également été ajouté. Au premier clic la musique revient à la position correspondant au début de la boucle et au second clic, elle revient au début de la chanson.

On a ensuite affiché le titre et l’artiste de la chanson en fonction de celle qu’on a choisit dans la liste de lecture.

Pour finir, on a maintenant la possibilité de quitter l’application en cliquant sur le bouton “menu“ du téléphone.

**(8.png : deuxième version de notre interface)**

**(9.png : quitter l’application par le bouton “MENU“)**

## quatrième étape

L’amélioration principale de cette étape est la mise en œuvre d’une solution finale pour le réglage de la boucle. En effet, le point négatif de la précédente solution était que si on voulait avoir un boucle qui commence vers la fin de la chanson il fallait rester appuyer un certain temps sur le bouton “+“. Ce n’était donc pas une solution optimale. Nous avons donc créé une nouvelle barre de progression juste au dessus de celle de la musique. Celle-ci permet, par un simple glissé, de modifier les valeurs de début et de fin de la boucle. Ces valeurs sont également affichés de chaque coté de la barre.

Cette étape dispose également des fonctionnalités “musique suivante/précédente“ et de la sélection d’une musique par genre, album ou artiste.

**(10.png : version finale du réglage de la boucle)**

## cinquième étape

Il s’agit de l’étape finale du projet. Nous avons mit notre application en pleine écran pour récupérer un peu de place.

Liste complète des fonctionnalités réalisées :

* Boucle
* Enregistrement des paramètres de la boucle
* Chargement des paramètres de boucle
* Equalizer
* Affichage des paroles
* Affichage des partitions
* Enregistrement d’un son (pas encore fonctionnel)
* Fonctionnalités de base d’un lecteur de musique (sélection d’une musique dans une liste, musique suivante, musique précédente, play/pause, stop, modification curseur de la musique)

Dans les nouvelles fonctionnalités citées nous avons le chargement des paramètres de la boucle. Il est maintenant possible de sauvegarder les paramètres de la boucle pour une chanson et de les charger ensuite. Nous avons donc fait un menu contextuel (désigne un menu qui s’ouvre lorsqu’un utilisateur clique sur un objet de l’interface, offrant ainsi une liste d’options qui varient selon l’objet) qui s’ouvre lorsqu’on clique sur l’icône de chargement d’une sauvegarde.

**(11.png : menu de sélection d’une sauvegarde à charger)**

**(14.png : bouton chargement d’une sauvegarde)**

Ce menu contient toutes les sauvegardes disponibles correspondant à la chanson en cours. Il suffit alors d’en sélectionner une dans la liste et les curseurs de début et de fin de boucle de l’interface se mettent automatiquement à jour. (L’image du menu contextuel est temporaire en attendant que la fonctionnalité marche et qu’on ait un exemple réel)

Ayant ajoutés de nombreuses fonctionnalités il était impossible de faire tenir tous les icones sur une même ligne comme dans les versions précédentes de notre interface. On a donc placé tous les icones dans une fenêtre que l’on fait apparaître/disparaître (avec une animation : la fenêtre bouge de haut vers le bas et inversement) en cliquant sur “fonctionnalités“.

**(12.png : version finale de notre interface)**

**(13.png : aperçue lorsqu’on affiche les fonctionnalités disponibles)**

(+ illustration affichage page web avec paroles et partitions et gestion de l’enregistrement d’un son si on le fait)

# Sources

<http://android.cyrilmottier.com/> : AndroidDevBlog, Blog francophone sur le développement Android

<http://developer.android.com/index.html> : Android Developpers, site contenant toutes les informations concernant le développement pour téléphone Android

<http://android-france.fr/category/tutoriel/> : Android France, liste de tutoriels pour aider à développer

<http://wiki.frandroid.com/wiki/Menus_Android> : frAndroid, aide pour la création des menus de l’interface