1. Application `Dessin`

Pour commencer, créez un projet appelé `Dessin`, puis téléchargez Dessin.zip, décompressez-le et importez-le dans Android Studio, dans `app/src` de façon à remplacer le `main` existant. Relire aussi le cours n° 7 : canvas et écouteur pour les actions utilisateur.

1.1. Description du projet

Ce projet est bâti sur quelques classes dans deux packages :

- `DessinActivity` : écran principal de l'application. En fait, elle ne contient qu'un menu et une seule vue, un `DessinView`.
- `DessinView` : c'est la sous-classe de `View` qui dessine les figures et qui gère les touchers de l'utilisateur. Les figures sont stockées dans une liste qui appartient au contexte d'application.
- `DessinApplication` : c'est le contexte d'application du projet. Elle gère une liste de `Figure`.
- `Figure` : c'est une sur-classe abstraite pour chaque type de figure : rectangle, cercle, ligne, etc. Une figure est définie par deux points (x,y), deux « coins » de la figure. Le premier, (x1,y1), appelé *référence*, est simplement le tout premier point posé et (x2,y2), appelé *coin*, est le dernier point défini. Alors il se peut parfaitement que la référence soit en haut à droite et le coin en bas à gauche. Il faudra prendre en compte ces éventualités. Une figure est aussi définie par une peinture.
- `Rectangle` : elle représente un rectangle. Remarquez comment les différents cas de positionnement des coins sont facilement gérés avec *min* et *max*. Pour un cercle ce sera un peu différent.

1.2. Affichage du DessinView

Pour voir quelque chose, il faut que `DessinActivity` affiche un `DessinView`. Vous avez deux possibilités pour ça :

- Soit vous complétez le layout, n'oubliez pas les propriétés de taille.
- Soit vous demandez à `DessinActivity` de n'afficher qu'un `DessinView`, donc sans aucun fichier layout.xml :

```
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    dessin = new DessinView(this, null);
    setContentView(dessin);
}
```

Vous avez le droit d'utiliser la seconde solution si vous comprenez qu'elle empêche d'ajouter autre chose sur l'écran, mais qu'elle convient parfaitement à notre problème : dessin en plein écran. Du coup, le layout est inutile, vous pouvez le supprimer.

À ce stade, vous devriez voir une multitude de cercles superposés, qui changent si on touche ou si on fait pivoter l'écran. C'est fait exprès pour vérifier que c'est bien le dessin qu'on voit.

1.3. Dessin des figures

Supprimez ou commentez toutes les lignes de la méthode `onDraw` et mettez à la place la boucle qu'il faut pour dessiner toutes les figures de la liste.

Cette fois, vous ne verrez que quelques rectangles superposés. D'autre part, ce dessin ne change pas si on pivote l'écran. Il change seulement d'une exécution à l'autre.

Enlevez les lignes qui créent ces rectangles dans `DessinApplication`.

2. Actions sur l'écran

2.1. Ajout d'une figure

On arrive au plus complexe, la gestion des touchers afin de créer des figures. C'est dans la méthode `onTouchEvent` de `DessinView`. Elle reçoit des événements contenant les coordonnées des touchers de l'écran. Les événements indiquent aussi si c'est un toucher, un glissé ou la fin. Vous allez devoir rajouter un aiguillage selon le type d'événement indiqué par `event.getAction()` :

- `MotionEvent.ACTION_DOWN` : premier toucher de l'écran. Vous devez créer une nouvelle figure du type et de la couleur courants à l'aide de `Figure.creer(typefigure, color)`, l'ajouter à la liste des figures, et lui donner les coordonnées (x,y) de son coin de référence.
- `MotionEvent.ACTION_MOVE` : déplacement du doigt, vous devez définir l'autre coin de la figure courante, c'est à dire la dernière de la liste.
- `MotionEvent.ACTION_UP` : le doigt quitte l'écran. Vous devez seulement appeler la méthode `vue.performClick()` afin de gérer d'autres types d'actions dans Android.

Normalement, à chaque fois que vous posez et glissez le doigt sur l'écran, une nouvelle figure doit apparaître.

Notez que la couleur est choisie aléatoirement à chaque nouvelle figure. Plus loin, il est proposé de créer un sélecteur de couleur.

2.2. Nouveaux types de figures

Rajoutez le dessin de lignes et de cercles et ce sera presque fini.

Ce sont les cercles les plus difficiles à dessiner. Ils doivent s'inscrire dans le rectangle (x1,y1), (x2,y2). Or attention, il n'y aucune relation d'ordre entre x1 et x2, y1 et y2. Déjà, mettez-vous d'accord avec vous-même pour savoir si ce sont des cercles bien ronds ou plus généralement des ellipses que vous préférez.

Dans le cas de cercles, `drawCircle` demande un centre et un rayon. Le centre (xc,yc) est le milieu entre (x1,y1) et (x2,y2). C'est le rayon qui est le plus pénible à déterminer. C'est la plus petite distance entre (xc,yc) et les bords. Par exemple *max(xc-min(x1,x2), yc-min(y1,y2))*, mais c'est à vous d'y réfléchir.

Sinon, il y a une autre possibilité, c'est de calculer le rayon comme étant la distance euclidienne entre (xc,yc) et (x2,y2). Dans ce cas, le cercle sort de la zone (x1,y1), (x2,y2) mais il touche le point (x2,y2) qui est défini par la position du doigt sur l'écran. C'est sans doute le plus naturel.

Rappel mathématique : $distance((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

Dans le cas d'ellipses, `drawOval` demande un rectangle `RectF` englobant. Pour créer un `RectF`, il faut fournir *min(x1,x2)*, *min(y1,y2)*, *max(x1,x2)*, *max(y1,y2)* au constructeur. Par contre, dans ce cas, il est plus difficile de calculer la véritable boîte englobante pour que l'ellipse touche le doigt de l'utilisateur.

2.3. Menus

Il serait utile de programmer les menus *Undo* et *Swap*. Les commentaire indiquent ce qu'il faut faire. Allez voir les méthodes de la classe `List`, telles que `pop`, `removeLast` et `add`.

NB: les icônes des menus ne sont pas tout à fait adéquats, mais j'avais pas le temps d'en dessiner de meilleurs.

3. Sélecteur de couleur

Nous allons maintenant construire un dialogue de sélection de couleur pour remplacer le tirage aléatoire des teintes.

3.1. Layout du sélecteur

Vous allez compléter le layout `colorpickerdialog`. Pour l'instant, il ne comprend qu'une ligne avec une vue colorée en rouge par `android:background="#F00"` et un `SeekBar` d'identifiant `sbRouge`. Il faut rajouter trois autres lignes de ce genre : vert, bleu et alpha (transparence).

3.2. Dialogue d'alerte

Complétez la méthode `onCreateDialog` de la classe `ColorPickerDialog` comme montré en cours, dans le transparent intitulé « Méthode `onCreateDialog` ».

3.3. Gestion des curseurs

Complétez le constructeur de la classe interne `ColorPickerView`. En fait, c'est très répétitif, mais il faut comprendre le principe pour le faire correctement. À chaque fois qu'on bouge un curseur, ça modifie la couleur dans `mCouleur` et la teinte de l'échantillon de couleur sous les curseurs, `ivCouleur`.

3.4. Appel du sélecteur

La dernière chose consiste à faire appeler le sélecteur dans le menu `Palette` de la classe `DessinActivity`. L'écouteur fait modifier la couleur courante dans le dessin afin de définir la prochaine figure. Il est utile de fournir la couleur de dessin courante en 2e paramètre du constructeur du dialogue.

4. Travail à rendre

Important : votre projet doit se compiler et se lancer sur un AVD. La note sera zéro si ce n'est pas le cas. Mettez donc en commentaire ce qui ne compile pas.

Avec le navigateur de fichiers, descendez dans le dossier `app/src` du projet Dessin du TP7.

Rajoutez un fichier appelé exactement `IMPORTANT.txt` dans le sous-dossier `main` si vous avez rencontré des problèmes techniques durant le TP : plantages, erreurs inexplicables, perte du travail, etc. mais pas les problèmes dus à un manque de travail ou de compréhension. Décrivez exactement ce qui s'est passé. Le correcteur pourra lire ce fichier au moment de la notation et compenser votre note.

Cliquez droit sur le dossier `main`, choisissez `Compresser...`, format `ZIP`, cliquez sur `Creer`. Déposez l'archive `main.zip` sur Moodle au bon endroit, voir sur la page Moodle Programmation Mobile.