On cherche à développer une application qui dessine des cercles colorés de rayon croissant en suivant le toucher de l'utilisatrice ou de l'utilisateur.

1 Dessin

Question 1 Créez une nouvelle classe `Cercle` permettant de représenter un cercle colorés dans le plan. Les coordonnées du centre, ainsi que le rayon, seront représentés sous forme de `float`. La couleur n'est autre qu'un `int`. On dotera cette classe d'accesseurs en lecture, ainsi que du constructeur idoine.

Question 2 Créez une nouvelle classe `Dessin` qui hérite de `View`. Cette classe a comme attributs
- une `ArrayList` de `Cercle`
- une couleur (de type `int`).

On dotera cette classe d'un accesseur en écriture sur sa couleur, ainsi que d'un constructeur qui crée une `ArrayList` vide et qui choisit la couleur `Color.BLACK` par défaut.

Question 3 Réimplémentez la méthode `Dessin::onDraw()` héritée de `View` de sorte qu'un `Dessin` affiche chaque `Cercle` contenu dans son `ArrayList` à la position désirée et à la couleur de ce `Cercle`.

On rappelle qu'en plus du `Canvas` fourni en argument de `onDraw()`, il nous faudra créer un `Paint`, dont la couleur changera en fonction de celle du `Cercle` courant, et dont le style sera `Paint.Style.STROKE` (pour dessiner des cercles et non des disques).

Question 4 Faites en sorte que le `ConstraintLayout` contienne une unique `View`, qui est un `Dessin`. Vérifiez ensuite que tout fonctionne, en créant par exemple un `Cercle` noir de rayon `100f` aux coordonnées `(200f,200f)`.

Question 5 On veut maintenant que le tracé des cercles suive le doigt de l'utilisateur ou de l'utilisatrice. Pour cela, on va utiliser la méthode `onTouchEvent()` :

```
public boolean onTouchEvent(MotionEvent event) {
  switch (event.getAction()) {
    case MotionEvent.ACTION_DOWN:
     ...
    case MotionEvent.ACTION_UP:
     ...
    case MotionEvent.ACTION_MOVE:
     ...
  }
  invalidate ();
  return true;
}
```

de sorte à obtenir le comportement illustré en Figure 1.

Précisément, on souhaite qu'à chaque position du doigt, un `Cercle`, centré sur les coordonnées détectées, et d'un rayon de `1f` supérieur au précédent, soit inséré dans le `ArrayList`. La couleur de ce `Cercle` devra être celle de l'attribut du `Dessin`.

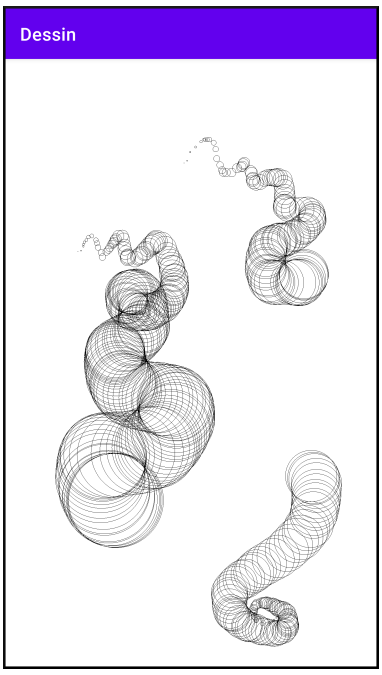

FIGURE 1 – Exemple de tracé, avec deux levés de doigt.

On souhaite également qu'à chaque lever du doigt, le rayon des `Cercle` soit réinitialisé à `1f`.

On se souviendra que l'appel à `invalidate()` permet de forcer une `View` à se redessiner - et donc, ici, à redessiner tous les `Cercle` présents dans le `ArrayList`.

2 Boîte de dialogue et choix d'une couleur

On souhaite maintenant offrir la posibilité à l'utilisatrice ou l'utilisateur de choisir une couleur. On va pour cela ouvrir un `Dialog`, cf. Figure 2.

Question 6 Créez une nouvelle classe `ColorDialog` qui hérite de `DialogFragment`. Vous devrez implémenter la méthode `onCreateDialog()` :

```
public Dialog onCreateDialog(Bundle saveInstanceState) {
  AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
  ...
  return builder.create();
}
```

Le `builder` pourra être paramétré grâce aux méthodes `AlertDialog.Builder::setTitle()` et `AlertDialog.Builder::setItems()`.

En particulier, `setItems()` prend deux arguments : une `CharSequence[]` correspondant aux différentes options proposées par la boîte de dialogue (ici, les couleurs), ainsi qu'une implémentation de l'interface `DialogInterface.OnClickListener`, qui implémente le callback

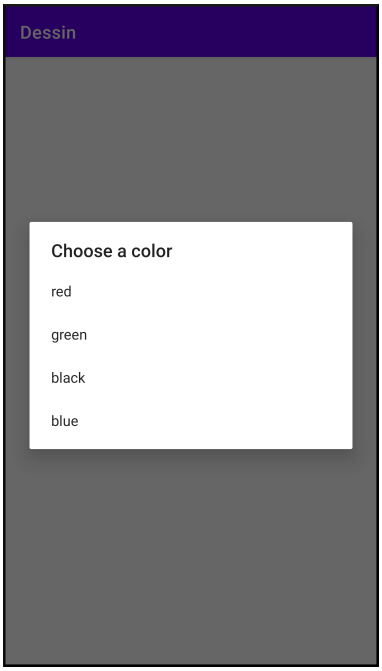

FIGURE 2 – Boîte de dialogue, ouverte à la création de `MainActivity`.

```
onClick(DialogInterface dialogInterface , int which)
```

On prêtera uniquement attention au deuxième argument, qui spécifie le numéro de l'option choisie. Dans ce callback, on modifiera la couleur du `Dessin`, ce qui impactera la couleur des `Cercle` à venir. Pour accéder à l'`Activity` (et donc au `Dessin`) depuis le `ColorDialog`, on pourra utiliser la méthode `getActivity()`.

Question 7 Faites en sorte qu'un `ColorDialog` s'ouvre à la création de `MainActivity`.

Pour cela, après la création d'un nouveau `ColorDialog`, on appellera

```
colorDialog.show(getSupportFragmentManager(), ‘‘color dialog’’);
```

Vérifiez que le choix de la couleur fonctionne comme désiré.

Question 8 On veut maintenant pouvoir changer de couleur au fil de l'eau, en plus de la choisir au démarrage de l'application. Ajoutez au layout un `Button` qui lance le `ColorDialog` à chaque clic.

3 Sauvegarde de l'état courant (☠)

Question 9 Tournez votre appareil après avoir dessiné quelques cercles. Que se passe-t-il ? Corrigez cela en utilisant notamment la méthode `Bundle::putParcelableArray()` (il faudra donc faire hériter `Cercle` de `Parcelable`).