Naji Astier

Mathieu De Viti

Rapport de Projet Sudoku Android

Question 1 :

Le projet Snake est composé de quatre classes java :

- BackgroundView gère le Background
- TileView crée une variation d'une View utilisé pour gérer des tableau d'icônes ou d'images (drawables),
- SnakeView qui génère la View du jeu et permet de sauver l'état du jeu (lorsque l'on pause ou quitte le jeu), gère les coordonnées du Snake ou des pommes, et donc les déplacement du Snake.
- Snake qui hérite de Activity, c'est la main Activity de l'application. Elle gère la création du jeu, la pause, et les évènements lorsque l'écran est touché.

Il est composé d'une layout, de drawable, c'est-à-dire des images que l'on peut dessiner sur l'écran et des fihciers xml contenant les strings et les couleurs.

Question 2 :

La couleur du fond est spécifié par la ligne : `android:background="@color/black"`, la couleur black ayant été spécifiée dans le fichiers colors.xml par la ligne suivante `<color name="black">#000</color>`.

On ne fournit que le nombre de colonnes car c'est la `GridView` qui déterminera le nombre de lignes nécessaires pour remplir chaque cellule avec le nombres d'éléments que l'on veut. Dans notre cas, on a 81 chiffres à insérer, avec un seul chiffre par cellule et avec 9 colonnes. La `GridView` détermine automatiquement qu'il nous faut 9 lignes complètes.

Question 3 :

Pour avoir accès à la `GridView` et à Item_`Sudoku` définis en XML on leur donne un ID avec les lignes de code XML `android:id="@+id/item_sudoku"` et `android:id="@+id/gridView1"` et on y accède dans les Activity à l'aide de la fonction `findViewById(R.id.X)`, X étant l'ID définis dans le fichiers XML.

Question 4 :

L'adapter sert à fixer les éléments par lesquels remplir la `TextView` passé en paramètre dans son constructeur. La `TextView` ne pourra être rempli que par les `toString()` des éléments présents dans la List passé en paramètre du constructeur de l'Adapter.

Une `GridView` est une View qui affiche des éléments, présent dans le `ListAdapter` associé, dans une grille à deux dimensions. Tandis qu'une `GridLayout` est une Layout qui place ses enfants dans une grid rectangulaire.

Question 5 :

`android:layout_margin="0dp"` pour fixer les marge à 0.

android:layout_margin="0dp"

`android:background="@color/white"` fixe le background de l'`EditText` en blanc. Avec le background de la `GridView` fixé à noir, on a des cases blanches avec des bordures noires.

`android:gravity="center"` centre les numéros dans les cases.

`android:maxLength="1"` fixe à un le nombre de chiffres par case.

Question 6 :

Pour la fonction `onMeasure` de `SquareGridView` :

```
@Override
    public void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
        int min = Math.min(widthMeasureSpec, heightMeasureSpec);
        super.onMeasure(min, min);
    }
```

On utilise le minimum parce que l'on veut que la GridView soit carré peu importe l'orientation de l'écran, elle ne doit donc pas dépasser la largeur minimum de l'écran (soit la hauteur ou la largeur selon l'orientation).

De même pour la fontion onMeasure de SquareEditTExt on utilise la mê fonction mais en prenant le maximum dans ce cas, car le SquareEditText doit prendre toute la case de la GridView.

Question 7 :

La méthode `getView` permet de récupérer une View qui montre et nous permet de modifier les données aux positions spécifiées. Dans notre cas, elle permet aussi de récupérer les coordonnées de ses données et les sauvegarder dans la `SudokuGrid`. La View retournée sera celle affiché.

Question 8 :

Pour gérer le Layout lors du passage portrait-paysage nous avons créé une variation paysage du Layout content_main.xml, placée dans le dossier layout-land avec une propriété d'orientation différente : `android:orientation="horizontal"`. Pour le layout portrait cette propriété, dans le content_main.xml du dossier layout, est vertical : `android:orientation="vertical"`.

Question 9 :

Pour éviter le blocage de l'UI on a utilisé, comme conseillé dans le cours, un thread et un handler.

Question 10 :

A chaque génération de grille la fonction setSudokuGrid est appelée. Cette fonction vide la liste conflictPosition contenant la position des cases en conflits qui ne seront donc plus prises en compte après la génération de la grille.

Pour que les autres cases en conflit soient aussi passé en rouge, on stocke les positions dans la liste conflictPosition. Ces positions sont obtenues en convertissant les coordonnées obtenues avec checkValue grâce à la fonction getPosition et pour chaque position incluse dans conflictPosition on met le background en rouge.

Question bonus :

Afin de couper la grille en 9 secteurs avec des lignes plus épaisses, nous nous proposons de mettre une image de grille sudoku en background de la GridView afin d'obtenir le résultat souhaité. Afin de gérer l'affichage selon la résolution des différents écrans et ne pas avoir de décalage entre les lignes de la GridView et de l'image on peut utiliser la m