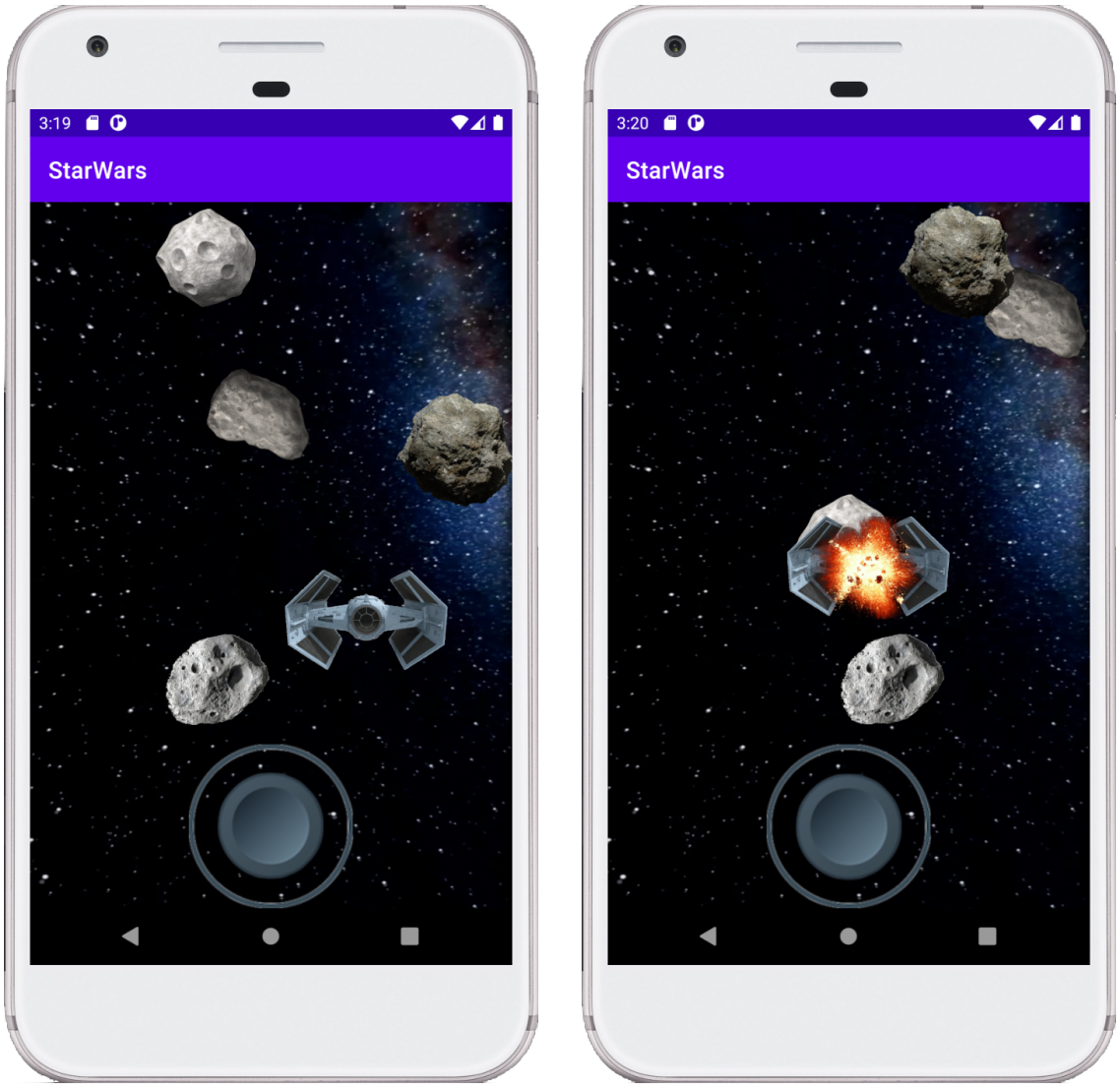
TP 2 et 3 Programmation sous Android

Codage d'un jeu

IMT Lille Douai

1 Objectifs du TP

Ce sujet de TP sera sur 2 séances. Il vous guidera dans le développement d'un jeu simple qui consiste à déplacer un vaisseau spatial de manière à lui faire éviter des astéroïdes se déplaçant à l'écran (voir screenshot ci-dessous). Deux modes de contrôle du vaisseau seront à développer : l'un utilisant un joystick 'tactile', l'autre utilisant l'accéléromètre du téléphone.



2 L'interface graphique

L'interface peut être décomposée en deux parties :
— une "zone de jeu" dans laquelle évolueront les éléments mobiles du jeu
— une zone en bas d'écran comportant le joystick.

Il est conseillé d'utiliser des *FrameLayout* pour faciliter le positionnement de la zone de jeu par rapport au joystick. Tous les éléments graphiques de l'interface (vaisseau, astéroïdes, joystick) seront affichés à l'aide d'*ImageView*. Vous trouverez sur MLS, une archive avec des images utilisables (vous pouvez également aller sur le web chercher vos propres images. Attention à prendre un format qui gère la transparence, le PNG par exemple.).

3 Animation graphique

Pour positionner vos *ImageView*, vous pouvez utiliser les méthodes *setX*, *setY*, *getX*, *getY*. Pour animer vos *ImageView*, vous pouvez utiliser les animations : `https://developer.android.com/training/animation/reposition-view`. Ce principe d'animations peut être très intéressant notamment pour la gestion du déplacement des astéroïdes ; il permet d'obtenir des déplacements plus riches que le simple déplacement en ligne droite. Ainsi, l'usage du déplacement en courbe `https://developer.android.com/training/animation/reposition-view#CurvedMotion`, vous permet de coder assez simplement des déplacements elliptiques :

```
1  path.arcTo(0f, 0f, 1000f, 1000f, 0f, -359f, true);
```

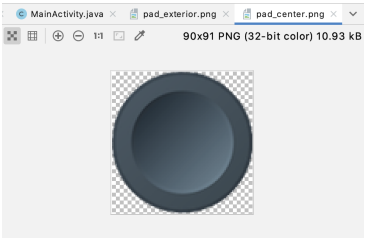
Ici la méthode *arcTo* décrit une trajectoire d'ellipse inscrite dans un rectangle (coin supérieur gauche à (0,0) et coin inférieur droit à (1000,1000)) en partant de la position initiale d'angle 0 degré. A partir d'un objet de type *ObjectAnimator*, cette trajectoire pourra éventuellement être rejouée à l'infini en utilisant la méthode *setRepeatCount*. Ci-dessous l'exemple complet :

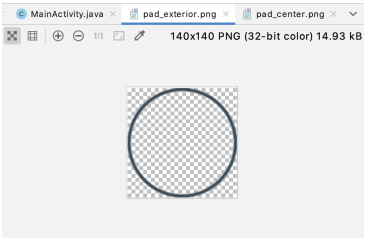
```
1  Path path = new Path();
2  path.arcTo(0f, 0f, 1000f, 1000f, 0f, -359f, true);
3  animatorsOfAsteroid = ObjectAnimator.ofFloat(imageViewOfAsteroid, View.X, View.Y, path);
4  animatorsOfAsteroid.setDuration(4000);
5  animatorsOfAsteroid.setRepeatCount(Animation.INFINITE);
6  animatorsOfAsteroid.start();
```

Toutes les animations peuvent se composer entre elles : déplacement en courbe, rotation, translation. N'hésitez pas à proposer vos propres trajectoires d'astéroïdes !

4 Gestion du joystick

Le joystick est composé de deux images que vous trouverez dans l'archive :





Vous devrez superposer les deux *ImageView* correspondantes et gérer le déplacement de l'image *pad_center.png*. Pour cela, associez à l'*ImageView* de cette dernière un écouteur d'évènements *View.OnTouchListener()*.

Vous implémenterez ensuite la méthode *onTouch*. A l'intérieur de celle-ci, vous pouvez décomposer le mouvement du doigt à l'écran grâce au paramètre de type *MotionEvent* :

```
public boolean onTouch(View v, MotionEvent event) {
        ...
            switch (event.getAction()) {
                case MotionEvent.ACTION_DOWN:
                    ...
                    break;
                case MotionEvent.ACTION_MOVE:
                    ...
                    break;
                case MotionEvent.ACTION_UP:
                    ...;
                    break;
                default:
                    ...
                    return false;
            }
            return true;
        }
```

ACTION_DOWN correspond au premier contact du doigt à l'écran, *ACTION_MOVE* correspond au déplacement du doigt après ce premier appui, et *ACTION_UP* correspond au retrait du doigt de la surface de l'écran. Dans le premier *case* du *switch*, il vous faut lancer le déplacement du vaisseau, qu'il faudra arrêter dans le dernier *case*. Pour cela il vous faut créer un thread dédié gérant le déplacement du vaisseau. Il existe plusieurs manières de coder des threads sous Android. Google déconseille de passer directement par la classe *Thread* et préconise plutôt l'usage des class *Runnable* et *Handler*. Dans l'exemple ci-dessous on crée un objet *Handler* dédié au déplacement du vaisseau et un objet *Runnable* au sein duquel le déplacement du vaisseau sera géré (code à placer dans la méthode *onCreate*) :

```
handlerForMovingTie = new Handler();

Runnable movingTie = new Runnable() {
        @Override
        public void run() {
            tieFighterImageView.setX(...
            tieFighterImageView.setY(...
            ...
            if (joystickIsPressed) {
                handlerForMovingTie.postDelayed(this, 10);
            }
        }
    };
```

Le booléen *joystickIsPressed* est un attribut de l'activité à mettre à jour à l'intérieur de la méthode *onTouch*.

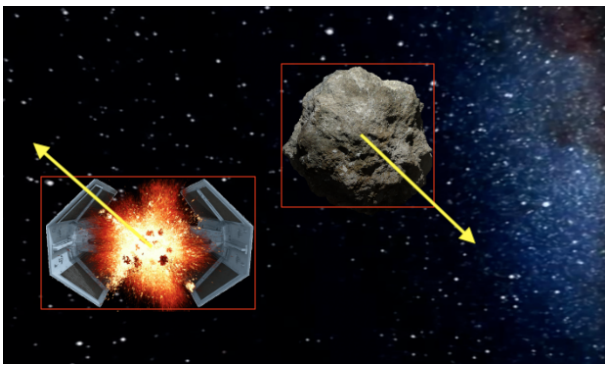
5 Détection des collisions entre *ImageView*

Pour faire apparaître l'explosion du vaisseau, il faut détecter le chevauchement entre deux *ImageView*. Pour cela, il faut extraire le périmètre des *ImageView* sous la forme d'un objet *Rect* puis utiliser la méthode *intersect*. Le code ci-dessous propose une implémentation simpliste de ce principe :

```
boolean sontEnCollision(ImageView firstView, ImageView secondView) {
    int []  firstPosition  = new int[2];
```

```
3           int [] secondPosition = new int[2];
4           firstView.getLocationOnScreen(firstPosition);
5           secondView.getLocationOnScreen(secondPosition);
6
7           Rect rectFirstView = new Rect(firstPosition[0], firstPosition[1],
8                   firstPosition[0] + firstView.getMeasuredWidth(), firstPosition[1] + firstView.getMeasuredHeight());
9           Rect rectSecondView = new Rect(secondPosition[0], secondPosition[1],
10                  secondPosition[0] + secondView.getMeasuredWidth(), secondPosition[1] + secondView.getMeasuredHeight());
11          return(rectFirstView.intersect(rectSecondView));
12      }
```

Pour un meilleur rendu visuel, il vous est conseillé d'améliorer cette méthode de manière à gérer les "fausses" collisions. Par exemple, dans l'exemple ci-dessous, la collision a été détectée car les rectangles rouges se sont intersectés (de très peu).



Les formes des images étant complexes, vous pouvez essayer de calculer l'aire de l'intersection entre les deux triangles et fixer un seuil à partir duquel cette aire est significative pour décider s'il y a collision entre les images.

6 Gestion de l'accéléromètre pour le déplacement du vaisseau

Pour pouvoir également déplacer le vaisseau avec les mouvements du téléphone (par exemple, pencher le téléphone vers l'avant pour faire monter le vaisseau vers le haut de l'écran), il faut pouvoir accéder aux valeurs renvoyées par l'accéléromètre du téléphone. Le code ci-dessous crée un objet *SensorManager* (ligne 14) permettant d'accéder à ce capteur (ligne 16) et de définir un écouteur sur celui-ci (ligne 18). À chaque changement d'état de l'accéléromètre, un évènement est déclenché et récupéré dans la méthode *onSensorChanged* (ligne 26). Trois valeurs sont ainsi obtenues (ligne 27). Elles correspondent respectivement à l'accélération en $\vec{x}$, en $\vec{y}$ et en $\vec{z}$.

```
1  import android.hardware.Sensor;
2  import android.hardware.SensorEvent;
3  import android.hardware.SensorEventListener;
4  import android.hardware.SensorManager;
5
6  public class MainActivity extends Activity implements SensorEventListener {
7      private SensorManager mSensorManager;
8      private Sensor accelerometer;
9
10     @Override
11     protected void onCreate(Bundle savedInstanceState) {
12         super.onCreate(savedInstanceState);
```

```
13          setContentView(R.layout.activity_main);
14          mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
15          // on recupere l accelerometre a partir du SensorManager
16          accelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
17          // on associe l ecouteur d'evenements au SensorManager
18          mSensorManager.registerListener(this, accelerometer, SensorManager.SENSOR_DELAY_NORMAL);
19      }
20
21      @Override
22      public void onAccuracyChanged(Sensor sensor, int accuracy) {// TODO Auto-generated method stub
23      }
24
25      @Override
26      public void onSensorChanged(SensorEvent event) {// TODO Auto-generated method stub
27          float gammaX = event.values[0], gammaY = event.values[1], gammaZ = event.values[2];
28          Log.d("Valeurs accelerometre",gammaX+","+gammaY+","+gammaZ);
29      }
30
31  }
```

Le déplacement du vaisseau est à mettre dans la méthode *onSensorChanged*. Observez avec le code ci-dessous les variations des valeurs *gammaX* et *gammaY* lorsque le téléphone est incliné vers l'avant et l'arrière pour en déduire comme déplacer le vaisseau.

7 Pour aller plus loin

7.1 Quelques idées pour améliorer votre TP

- améliorer l'apparition de l'explosion (voir les possibilités offertes par *alpha*, *scaleX*, *scaleY* dans les objets d'animation)
- ajouter un écran d'accueil avec une autre activité (qui devient l'activité principale)
- ajouter une gestion des scores (voir la section suivante)
- ajouter une gestion de la difficulté (en faisant par exemple varier le nombre d'astéroïdes, leur taille, leur trajectoire, leur vitesse)
- ajouter des bruitages son (voir *SoundPool* dans la doc Android)

7.2 Code pour l'enregistrement des scores

Une manière simple de gérer l'enregistrement des scores pour un jeu est d'utiliser les *SharedPreferences*. Elles permettent de sauvegarder et de charger des valeurs pour des types primitifs : *double*, *long*, *String*.

Le code suivant utilise des objets *SharedPreferences* (lignes 10 et 21) pour charger et sauvegarder le contenu d'une chaîne de caractères (ligne 3). Le chargement d'une valeur persistante se fait directement depuis un objet *SharedPreferences* en utilisant les méthodes *load* fournies (ligne 11). L'enregistrement d'une valeur nécessite l'utilisation d'un objet *SharedPreferences.Editor* (ligne 22). L'enregistrement se fait en deux temps : écriture de la chaîne dans l'objet *SharedPreferences.Editor* (ligne 23) puis écriture du *SharedPreferences.Editor* sur la flash du téléphone (appel de la méthode *commit()* ligne 24).

```
1  public class MainActivity extends Activity {
2
3      String s;
4
5      @Override
```

```
6       protected void onCreate(Bundle state){
7           super.onCreate(state);
8           ...
9
10          SharedPreferences load = getSharedPreferences("monFichier", 0);
11          s = load.getString("maChaine", "xxxx");
12
13          ...
14      }
15
16      @Override
17      protected void onStop(){
18          super.onStop();
19          ...
20
21          SharedPreferences save = getSharedPreferences("monFichier", 0);
22          SharedPreferences.Editor editor = save.edit();
23          editor.putString("maChaine", s);
24          editor.commit();
25      }
26  }
```