CCTP Jeu Android

Ando Randriamaro 37006102, et Maroson Brice 37005590 M1 informatique

November 6, 2017

Introduction

Ce rapport explique la conception de développement dún Jeu Android. Le jeu consiste à utiliser le gyroscope dún smartphone pour déplacer une boule à travers des pièges et atteindre une case pour avoir des points. Nous allons voir premièrement, plus en detail le mode fonctionnement du jeu, comment on joue, comment on obtient les points et quand le jeu sera terminé. Puis, on va expliquer la conception du code, la gestion des objets, des activités, et de l'affichage. On rappelle que le projet a été développé en Android studio

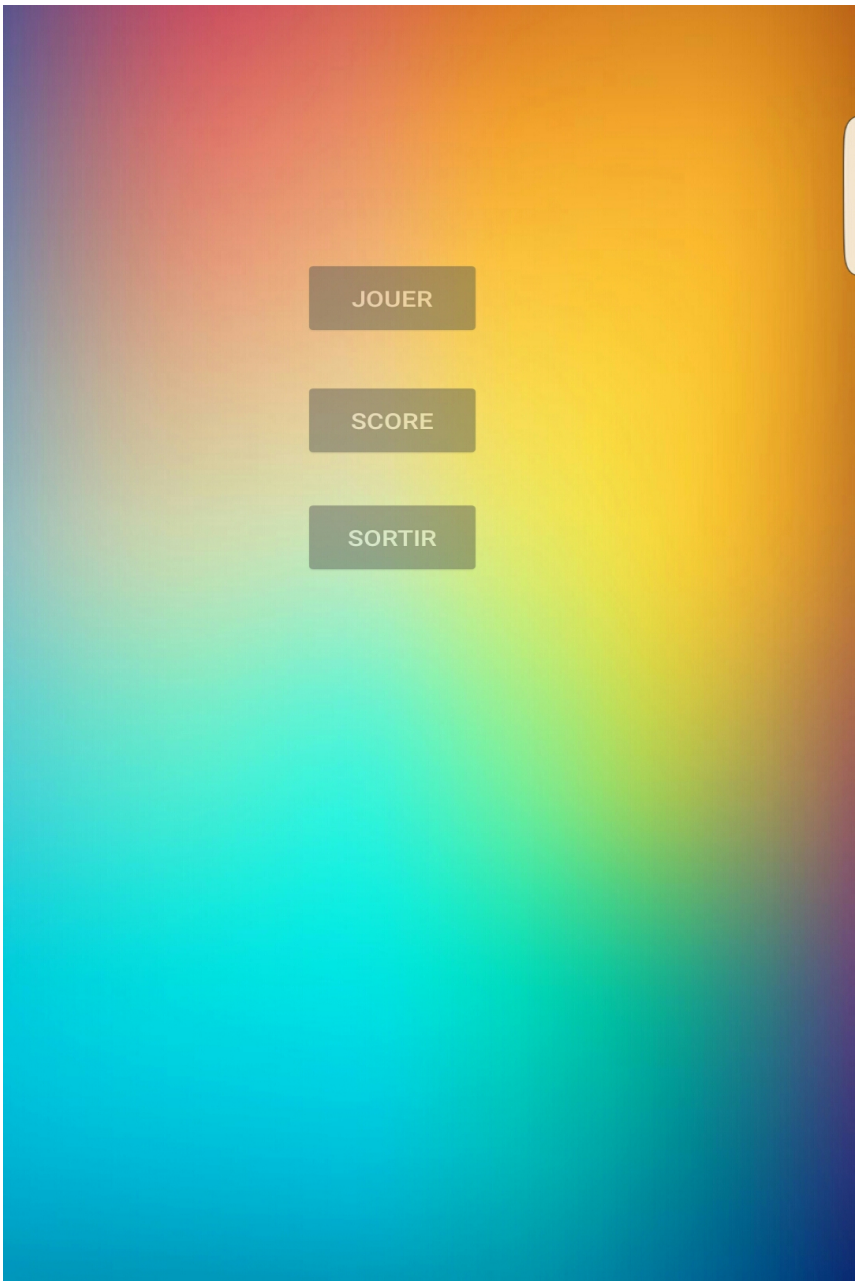
1 A propos du jeu

A l'aide de l'inclinaison du smartphone, on peut déplacer la boule dans la zone d'ecran pour atteindre une case en violet et eviter les pièges ou case en rose. A chaque fois que la boule touche son objectif, le joueur gagne 1 point et la case objectif se positionne aléatoirement dans la zone de jeu et le nombre de piège augmente.

1.1 Ecran d'accueil

On a 3 options sur l'ecran d'accueil, le bouton jouer pour acceder au jeu, le bouton score pour regarer la melleiur score. C'est l'écran principal de l'application, l'activité qui est lancé en premier. Après avoir appuyer sur le bouton jouer ou score, l'application lance une autre activité pour l'affichage suivante

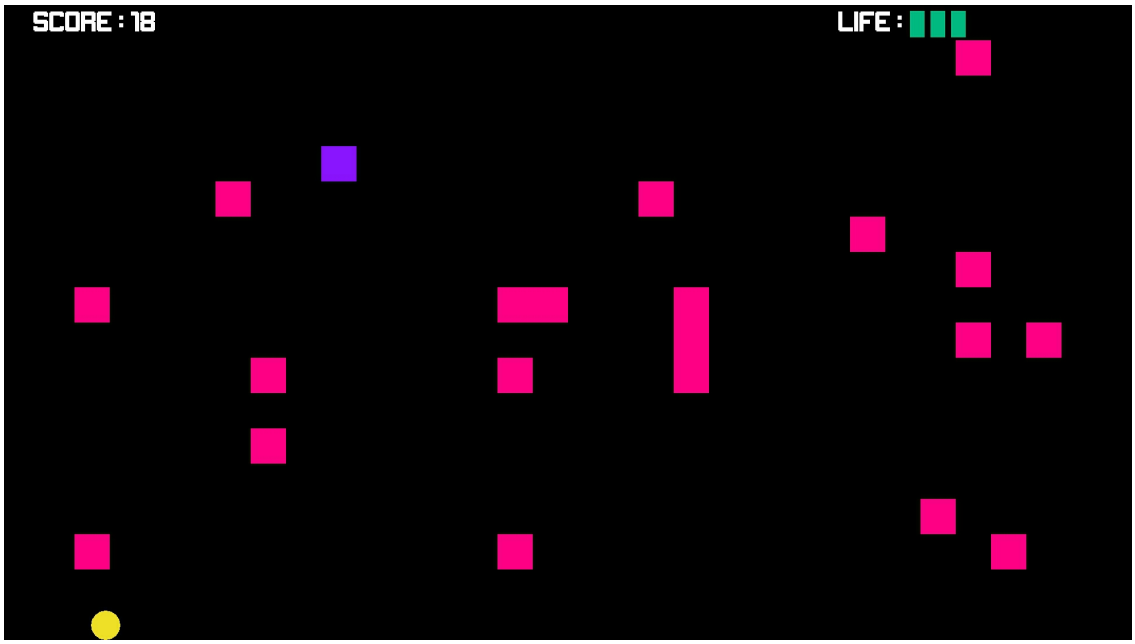
```
private View.OnClickListener jouerListener = new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            // playSound.start();
            Intent intent = new Intent(MainActivity.this, MyActivity.class);
            startActivity(intent);
        }
    };
```



menu principale

1.2 Ecran de jeu

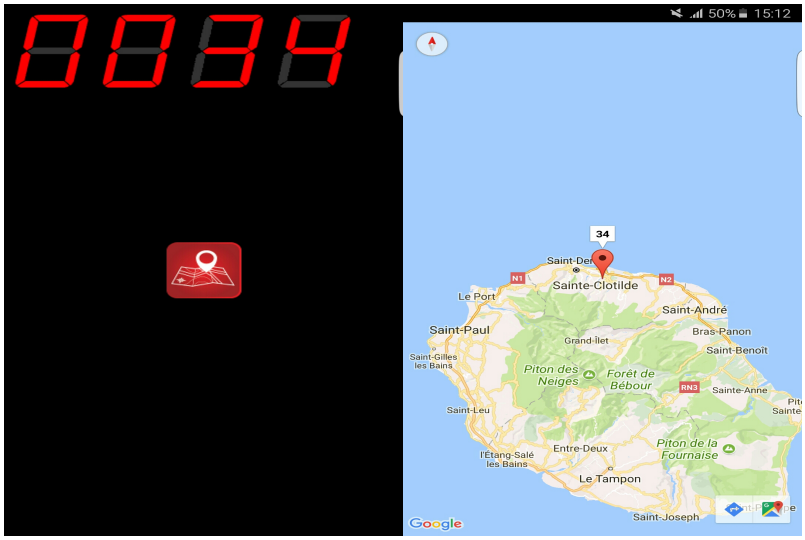
La boule se lance et se déplace selon l'inclinaison du smartphone. Les pièges sont des cases colorées en rose fixé à 2 cases au début du jeu. L'objectif est d'atteindre la case colorée en violet. Quand la boule l'atteint, le score augmente de 1 point, l'objectif change de place aléatoirement dans la zone de jeu, et les pièges s'incrémentent aussi aléatoirement sur l'écran afin de monter le niveau du jeu. Quand la boule se heurte avec l'un des pièges, on peut entendre une sorte de son, et une sorte d'explosion sur la coalisions, la position de la boule se réinitialise. La vie diminue de 1, la vie est initialisé à 3 soit quand la boule se heurte 3 fois à l'un des obstacles, le jeu sera terminer et le score enregistrer. Un music de fond est enclenché à chaque parti du jeu.



capture du jeu

1.3 L'ecran du score

En appuyant sur ce bouton, on va accéder au score maximal atteint parmi les parties effectuées. Un autre bouton de type localisation se trouve dans l'écran. En cliquant dessus, on peut trouver la position via GPS où le joueur a eu le score max.



score max et géolocalisation du position où le joueur a eu ce score

2 Conception du code

On définit une classe boule.java, qui caractérise les comportements de la boule dans le jeu : sa dimension, la vitesse dont elle va parcourir la zone du jeu, et aussi le code qui le permet de rebondir en touchant la fin de la zone d'écran.

```
public class Boule {
```

```
    // Vitesse maximale autorisée pour la boule
    private static final float MAX_SPEED = 20.0f;
    // Permet à la boule d'accélérer moins vite
    private static final float COMPENSATEUR = 8.0f;
    // Utilisé pour compenser les rebonds
    private static final float REBOND = 1.3f; // reglage difficulté (valeur bas, vitesse élevé)

    // Coordonnées en x
    private float mX = 1;
    // Coordonnées en y
    private float mY = 1;
    // Vitesse sur l'axe x
    private float mSpeedX = 0;
    // Vitesse sur l'axe y
    private float mSpeedY = 0;
    // Taille de l'écran en hauteur
    private int mHeight = 0;

if (mX < RAYON) {
        mX = RAYON;
        // Rebondir, c'est changer la direction de la balle
        mSpeedY = -mSpeedY / REBOND;//-
    } else if (mX > mWidth - RAYON) {
        mX = mWidth - RAYON;
        mSpeedY = -mSpeedY / REBOND;// reglage du rebond quand la boule touche extremite largeur
    }
}
 ainsi de suite pour mY
```

Pour les pièges et la case à atteindre, on les a mis dans un objet de type énumération, donc de même dimension. Pour la dimension de la boule, des pièges, et de l'objectif, on a utilisé la fonction RectF, qui permet de construire un objet de forme rectangulaire qu'on va manipuler dans le jeu. RectF est caractérisé par sa position sur l'axe x et y et sa taille sur l'axe x et y de l'écran soit : RectF(int x, inty, height, width).

Le code du jeu se base sur 2 grandes classes : MoteurdeJeu.java et Moteur et Paneldejeu.

2.1 La classe PaneldeJeu

Cette classe s'occupe de l'affichage sur l'écran de toutes les composantes du jeu : la boule, les pièges, l'objectif, le score, et la vie. Le plus important est que cette classe hérite de SurfaceView, une superclasse qui permet d'utiliser et de manipuler un objet dessiné, ou de type image,...

2.2 La classe Moteur de Jeu

Comme son nom l'indique, c'est cette classe qui gère les composants du jeu, c'est-à-dire, l'interaction des objets, quelles objets doit on afficher, les méthodes pour chaque circonstances.

D'abords, on initialise les objets du jeu, et les capteurs du smartphone

```
public class MoteurdeJeu {

private Boule mBoule = null;
```

```
    private List<Bloc> mBlocks = null;
    private List<Bloc> mObstacles = null;
    private Bloc mTarget = null;

private MyActivity mActivity = null;
    private SensorManager mManager = null;
    private Sensor mAccelerometre = null;


}
```

on gère le type de capteur utilisé

```
public MoteurDeJeu(MyActivity pView) {
        mActivity = pView;
        mManager = (SensorManager) mActivity.getBaseContext().getSystemService(Service.SENSOR_SERVIC
        mAccelerometre = mManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        initBoard();
    }
```

on crée ensuite les fonctions de controle du capteur *verbatim* :

```
stop() pour arreter
resume() pour redemarrer
```

on crée aussi quelques fonction de controle sur la boule et l'objectif :

```
repositTarget() repositionne la boule lorsqu'elle a heurter l'un des obstacles.
```

on met en place en suite les obstacles comme suit :
mis en place de 2 cases de position aléatoires

```
private void List<Bloc> initObstacles (int n){
}
```

fonction qui permet de réactualiser les 2 cases d'obstacles à chaque nouvelle partie

```
private void reinitObstacle()
```

fonction qui permet d'incrementer le nombre d'obstacle à chaque point gagné.

```
private void addNewObstacle()
```

on gère ensuite les méthode à utiliser lors des interactions entre les objets

```
SensorEventListener mSensorEventListener = new SensorEventListener() {

 @Override
  public void onSensorChanged(SensorEvent pEvent) {
    float x = pEvent.values[0];
    float y = pEvent.values[1];

if (mBoule != null) {
       // On met à jour les coordonnées de la boule
RectF hitBox = mBoule.putXAndY(x, y);
       // Pour tous les blocs, iteraction de la collection
          for (Bloc block : mBlocks) {
       // On crée un nouveau rectangle pour ne pas modifier celui du bloc
```

```
                RectF inter = new RectF(block.getRectangle());
                 if (inter.intersect(hitBox)) {
                 // On agit différement en fonction du type de bloc
                    switch (block.getType()) {
                       case TROU:
                         mActivity.getView().startExplosion((int) mBoule.getX(),
(int)mBoule.getY());
                                  mBoule.reposit();
                                  if (mBoule.getLife() > 0) {
                                      mBoule.decrementLife();
                                      soundPlayer.playFreq(330);
                                  }
                                  if (mBoule.getLife() == 0) {
                                      mActivity.showDialog(MyActivity.DEFEAT_DIALOG);
                                      stop();
                                  }

                                  break;

                              case TARGET:
                                  mBoule.incrementScore();
                                  repositTarget();
                                  addNewObstacle();
                                  break;
                          }
                          break;
                      }
                  }
              }
          }
```

3 Les ressources

Pour gerer les ressources, on copie les fichiers images dans le dossier drawable du ressources et les fichiers audio dans raw.

4 Demarrage du jeu

C'est la classe MyActivity qui est l'activité principal du jeu.
On initialise d'abords les 2 classes de bases

```
// Le moteur physique du jeu
    private MoteurDeJeu mEngine = null;

    // Le moteur graphique du jeu
    private PanelDeJeu mView = null;
```

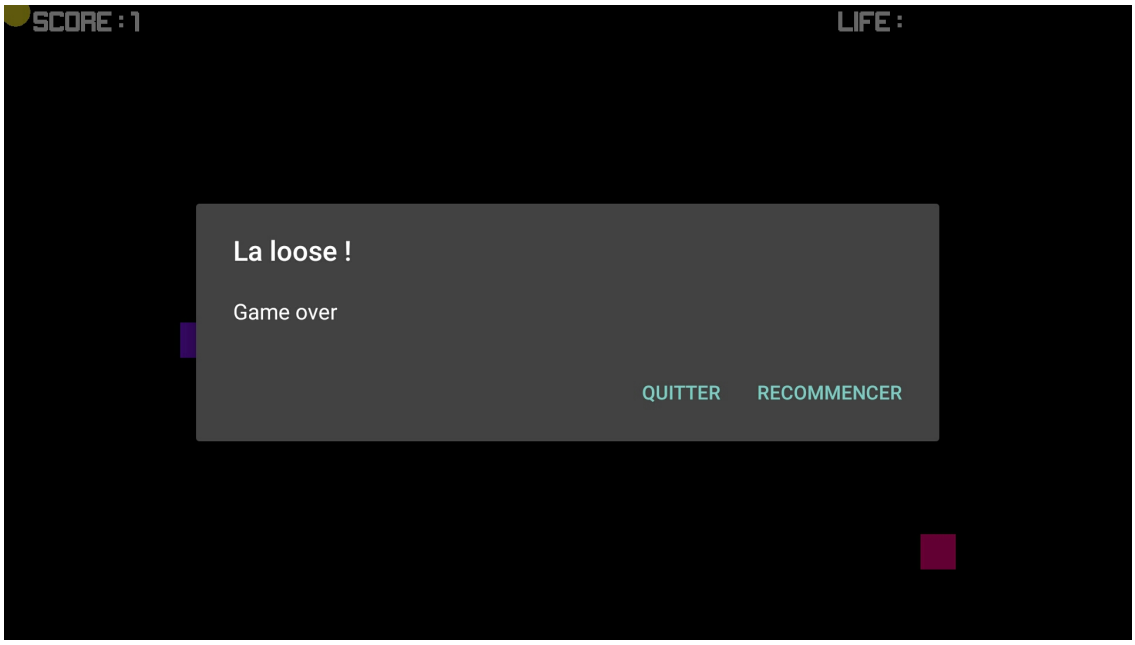
On defini ensuite le PanelDeJeu comme vue au lieu d'un xml On attribut un objet de type boule au moteur de jeu mEngine

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

```
        //initialisation

mView = new PanelDeJeu(this);
        setContentView(mView);

        mEngine = new MoteurDeJeu(this);
        Boule b = new Boule();
        mView.setBoule(b);
        mEngine.setBoule(b);

       }
```

On définit ensuite les methode onResume, onPause, onDestroy pour controler l'activité via le moteur mEngine.

Le jeu ne s'interrompe que lorsque le joueur n'a plus de vie donc il a le choix de recommencer ou de quitter



boite de dialogue de fin de jeu

Résumé

Pour conclure, la conception de ce projet est plus ou moins flexible mais très adaptable pour des éventuelles améliorations et extensions. De plus, Android Studio est un outil très optimal pour un développement d'application android qui offre beaucoup d'option mais il faut aussi bien maitriser les fonctions utilisées pour éviter de créer une application lourde et gourmande en termes de ressource.