Développement Mobile *Labo 4*

La boule qui tombe, qui tombe….

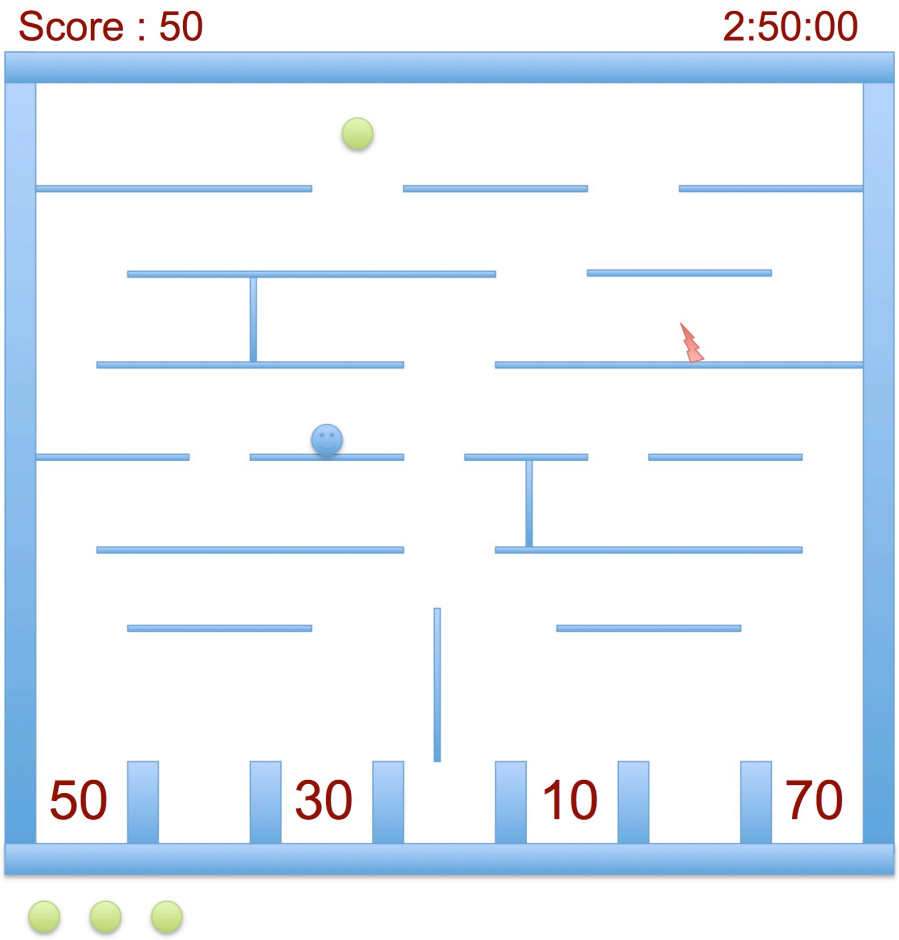
Stéphane Malandain - semestre d'hiver 2017

Objectifs

Ce laboratoire répond aux objectifs suivants:

1) Créer un canvas et dessiner avec Android
2) Créer un thread dédié pour le dessin
3) Utiliser un capteur (accéléromètre)
4) Faire un projet MVC complet

1. Introduction



Le but de ce tp est de réaliser un jeu comme l'illustre la figure ci-dessus.

Une boule apparaît en haut de l'écran. Elle tombe vers le bas sous l'effet de la gravité. Le joueur peut la diriger vers la gauche ou la droite, en inclinant la tablette vers l'une ou

l'autre de ces 2 directions. Le but est de la faire tomber dans une des cases du bas, la plus importante en point dans l'idéal.
Une fois dans la case, la boule disparaît pour réapparaître en haut et retomber. Le joueur marque le nombre de points prévu par la case dans laquelle est tombée la boule. Ceci recommence pendant un certain laps de temps (2 mn par exemple).

Les murs extérieurs du labyrinthe vont faire rebondir la boule. L'intérieur du labyrinthe est constitué de murs, peu nombreux, et surtout de plateformes et de trous qui la fond chuter vers le bas.

Une boîte de dialogue signale à l'utilisateur quand le jeu est terminé avec le nombre de points marqués.

Quant on passe d'une boule à l'autre, les plateformes, murs et points dans les cases du bas changent.

2. Spécifications techniques

Le jeu va être développé avec plusieurs composants :

a. Le 'moteur' graphique (vue) qui s'occupera de dessiner
b. Le composant physique (contrôleur) qui s'occupe de gérer les positions, les déplacements, les interactions entres les différents éléments (entre autre, les collisions), l'accéléromètre pour les déplacements lattéraux.
c. Le modèle qui définit l'ensemble des différents éléments qui compose le jeu (la boule, les différentes cases, la plateau de jeu, etc…

3. L'activitée principale

L'activité principale présente deux boutons. Le premier (Start) permet de lancer une nouvelle partie. Le second (Hi-Score) permet d'avoir accès aux meilleurs scores.

Cette activité définit également un menu qui permet de définir le niveau de difficulté souhaité (*Difficulty*) ou d'avoir des informations sur l'application (*About*). La difficulté fait varier la vitesse de chute de la boule. On définie 3 niveaux : Easy, Medium, Hard.

3.1 Définition du menu

Consulter Creating Menus (http://developer.android.com/guide/topics/ui/menus.html) pour la creation de menu sous Android.

Définir le menu et le choix du niveau de difficulté. On ne s'occupera pas de *About* dans un premier temps.

3.2 Activer le bouton Start

Activer le bouton *Start*. Il lance une nouvelle activité et lui transmet le niveau de difficulté choisi (niveau *Easy* par défaut). Pour transmettre des informations entre activités, on peut utiliser les méthodes putExtras() et getExtras() de *Intent* (Voir tps précédents).

Voir aussi Intents and Intent filters (http://developer.android.com/guide/components/intents-filters.html) et plus précisément, la rubrique Extras.

4. Les Modèles

4.1 La classe boule

La boule va se déplacer. Nous devons connaître sa position. Elle a une vitesse, sur les deux axes, ainsi qu'une vitesse max. En effet, pour donner cette impression de 'gravité' et d'accélération quand on penche la tablette à gauche ou à droite, la boule se déplace dans la direction correspondante, en accélérant, jusqu'à une vitesse maximale. Nous avons aussi besoin d'un attribut pour la couleur de la boule, et pour sa taille.

4.2 Les cases

Les cases sont de différents types : `Mur, Plateforme, Trou, Arrivée, bonus, malus`. Elles sont représentées par un rectangle carré de type `Rectf`, très pratique ensuite pour gérer les collisions avec la boule. En effet, la classe `Rectf` possède une méthode qui permet de détecter si deux `RectF` entrent en collision : `boolean intersect(RectF r);`

Attention, le rectangle passé en attribut est modifié par la méthode; il faut donc une copie du rectangle que l'on veut tester sinon celui-ci sera modifié.

Pour la gestion des collisions, un moyen simple sera d'ajouter un `RectF` autour de notre boule; La majorité des collisions ne pouvant se faire en diagonale, cette représentation est suffisante pour notre cas. Ajouter donc un attribut `Rectf` dans la classe Boule, à mettre à jour des que la boule bouge.

4.3 Le plateau

Tout d'abord, il doit prendre en compte la taille maximale de l'écran de la tablette et s'adapter en conséquence, ce qui veut dire adapter la surface de jeu, la taille des cases et de la boule.
Ensuite, il vous est demandé de trouver une structure de données appropriée pour représenter celui-ci.

5. Le moteur graphique

Pour effectuer le dessin de l'ensemble, on va utiliser un `SurfaceView`, manière la plus simple de dessiner avec de bonnes performances. Chaque élément devra être dessiné sur le `Canvas` du `SurfaceView`.

5.1 Comment dessiner ?

L'objet qui réalise un dessin est le `canvas`. Pour dessiner, le `canvas` a besoin d'un `Bitmap` (= une surface pour dessiner). C'est un peu comme si le `canvas` était le peintre,

et le `Bitmap` la toile. Il nous manque le pinceau. C'est ce que pourrait représenter l'objet `Paint` qui permet de définir la couleur du trait, sa taille, etc…

C'est au `canvas` que l'on envoi les ordres. Vous avez besoin d'un `Bitmap` si vous créez vous même un `canvas`.

Ex :
```
Bitmap b = Bitmap.createBitmap(128,128, Config.ARGB_8888);
Canvas c = new canvas(b);
```

5.2 Le pinceau (objet `Paint`)

L'objet `Paint` représente à la fois le pinceau et la palette; Pour des dessins plus net, on peut ajouter les fanions `Paint.ANTI_ALIAS_FLAG` et `Paint.DITHER_FLAG`.

Ex 1 :
```
Paint p=new Paint(Paint.ANTI_ALIAS_FLAG | Paint.DITHER_FLAG);
```

Ex 2 :
```
Paint p =  new Paint();
// dessiner à l'intérieur d'une figure
p.setstyle(Paint.Style.FILL);
// des contours
p.setStyle(Paint.Style.STROKE);
// les deux …
p.setStyle(Paint.Style.FILL_AND_STROKE);
```

5.3 Le peintre

Pour dessiner une figure, il suffit d'utiliser la méthode appropriée :

```
void drawColor(int color); // pour remplir la surface du Bitmap d'une couleur

void drawRect(Rect r, Paint paint); // pour dessiner un rectangle

void drawText(int color);
```

5.4 Afficher la toile

5.4.1 Sur une toile standard

Ce n'est pas la solution idéale pour un affichage rapide, mais nous nous en contenterons !

Il suffit d'utiliser la méthode `onDraw(Canvas Canvas)`. Il est cependant possible d'indiquer à la vue qu'elle doit se redessiner le plus vite possible en utilisant la méthode `void invalidate()`.

5.4.2 Sur une surface dédiée à ce travail

On utilise un thread dédié à la tâche d'affichage ce qui 'n'encombre' pas le thread UI. Au lieu d'attendre qu'Android déclare à notre vue qu'elle peut se redessiner, c'est un thread à part entière qui s'en occupe. Mieux, cette surface peut être prise en charge par OpenGL si besoin.

On va se servir de la classe `SurfaceView` en ne la manipulant pas directement, mais au travers d'une couche d'abstraction représentée par la classe `SurfaceHolder`.

Afin de récupérer un `SurfaceHolder` depuis un `SurfaceView`, il suffit d'appeler `SurfaceHolder getHolder()`. De plus, pour gérer le cycle de vie de notre `SurfaceView`, on devra implémenter l'interface `SurfaceHolder.Callback`, qui permet au `SurfaceView` de recevoir des informations sur les différentes phases et modifications qu'elle effectue.

Pour associer un `SurfaceView` à un `SurfaceHolder.Callback`, on utilise la méthode `void addCallBack(SurfaceHolder.Callback callback)` sur le `SurfaceHolder` associé au `SurfaceView`. Attention, cette opération doit toujours être effectuée dès la création du `SurfaceView`.

Exemple :

```
Public class ExSurfaceView extends SurfaceView implements
SurfaceHolder.Callback {

   private SurfaceHolder mHolder = null;

   public ExSurfaceView ( Context context) {
     super(context);
     mHolder = getHolder();
     mHolder.addCallback(this);
   }
}
```

Il faut ensuite implémenter les trois méthodes de Callback qui réagissent à trois évènements différents :

- `void surfaceChanged(SurfaceHolder holder, int format, int width, int height)` qui est appelée à chaque fois que la surface est modifiée. Cette méthode possède 4 paramètres. La largeur, `width`, la hauteur `height`, et le `PixelFormat format`. Le `SurfaceHolder holder` est une interface qui représente la surface sur laquelle dessiner. Mais c'est bien avec le `Canvas` que l'on dessine, de façon à ne pas manipuler directement le `SurfaceView`.

- `void surfaceCreated(SurfaceHolder holder)` appelée à la création de la surface. On peut donc commencer à dessiner ici.

- `Void surfaceDestroyed(SurfaceHolder holder)` appelée dès que la surface est détruite. Vous savez ainsi quand arrêter le thread. Une fois cette méthode invoquée, la surface n'est plus disponible du tout.

  On va dessiner à l'aide d'un `canvas`, sachant qu'il à déjà un `Bitmap` attribué. Il faut d'abord le bloquer pour immobiliser l'image actuelle en invoquant la méthode `canvas lockCanvas()`. Une fois le dessin effectué, le remettre 'en route' avec la méthode `void unlockCanvasAndPost(Canvas canvas)`. C'est indispensable, sinon le device reste bloqué.

Attention, la surface sur laquelle se fait le dessin sera supprimée à chaque fois que l'activité se met en pause, et recrée des le retour de celle-ci. Il faut donc interrompre le dessin à ces moments là.

Dans le code qui suit, remarquez que l'on instaure une pause dans la boucle principale du thread. Pour l'économiser un peu, on se synchronise ainsi à environ 50 images par secondes ce qui est largement suffisant pour l'œil humain.

5.5 Exemple de code

Voici un programme qui fonctionne, dessine 2 rectangles et un texte 'hello' sur plusieurs lignes. Celui-ci est à tester et à comprendre; Il doit vous servir de base pour le développement du moteur graphique de ce projet.

```
package com.example.examplesurfaceview;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.RectF;
import android.util.AttributeSet;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public class ExampleSurfaceView extends SurfaceView implements SurfaceHolder.Callback {

	// Le holder
	SurfaceHolder mSurfaceHolder;
	// Le thread dans lequel le dessin se fera DrawingThread mThread;
	DrawingThread mThread;
	Paint mPaint;
	RectF mRect = new RectF(10,10, 780,1100);
	RectF mRect2 = new RectF(100,100, 580, 800);

	public ExampleSurfaceView (Context context) {
		super(context);
		mSurfaceHolder = getHolder();
		mSurfaceHolder.addCallback(this);
		mThread = new DrawingThread();
		mPaint = new Paint();
		mPaint.setStyle(Paint.Style.FILL);
	}
	@Override
	protected void onDraw(Canvas pCanvas) {
		// Dessinez ici !
		int x = 200;
		int y = 100;
		pCanvas.drawColor(Color.RED);
		mPaint.setColor(Color.WHITE);
		mPaint.setTextSize(30);
		pCanvas.drawRect(mRect, mPaint);
		mPaint.setColor(Color.GREEN);
		pCanvas.drawRect(mRect2, mPaint);
		mPaint.setColor(Color.BLUE);
		pCanvas.drawText("hello !", 400, 300, mPaint);
		for (int i=0; i<200;i++)  {
			y += 2* i;
			pCanvas.drawText("hello !", x, y, mPaint);
		}
	}
	@Override
	public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {
```

```
	// Que faire quand le surface change ? (Si l'utilisateur tourne son téléphone par exemple)
	}
	@Override
	public void surfaceCreated(SurfaceHolder holder) {
		mThread.keepDrawing = true;
		mThread.start();
	}
	@Override
	public void surfaceDestroyed(SurfaceHolder holder) {
		mThread.keepDrawing = false;
		boolean joined = false;
		while (!joined) {
			try {
				mThread.join();
				joined = true;
			} catch (InterruptedException e) {}
		}
	}

	private class DrawingThread extends Thread {
	// pour arreter le dessin quand il le faut
	boolean keepDrawing = true;
		public void run() {
			while (keepDrawing) {
				Canvas canvas = null;
				try {
					// on chope le canvas pour dessiner dessus
					canvas = mSurfaceHolder.lockCanvas();
					// on s'assure qu'aucun autre thread n'y accède
					synchronized (mSurfaceHolder) {
					// et on dessine
						onDraw(canvas);
					}
				} finally {
					// le dessin terminé on relâche
					// le canvas pour que le dessin s'affiche
					if (canvas != null)
						mSurfaceHolder.unlockCanvasAndPost(canvas);
				}
				// pour dessiner a 50 fps;
				try {
					Thread.sleep(20);
				} catch (InterruptedException e) {}

			}
		}

	   }
	}
```

et le Main activity :

```
package com.example.examplesurfaceview;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;

public class MainActivity extends Activity {
	private ExampleSurfaceView viewTest = null;

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);

		viewTest = new ExampleSurfaceView(this);
		setContentView(viewTest);
```

```
	}

	@Override
	public boolean onCreateOptionsMenu(Menu menu) {
		// Inflate the menu; this adds items to the action bar if it is present.
		getMenuInflater().inflate(R.menu.activity_main, menu);
		return true;
	}

}
```

6. L'accéléromètre

Votre application est un jeu qui exploite la détection de mouvement. Si certains devices n'ont pas d'accéléromètre, peut-être est-il souhaitable, par exemple, de leur interdire le téléchargement sur le play store de celui-ci. Il vous faut tout simplement ajouter une ligne de type `<uses-feature>` dans le `Manifest` :

```
<uses-feature android:name="android.hardware.sensor.accelerometer" android:required="true">
```

La classe qui permet d'accéder aux capteurs est la classe `SensorManager`. Pour en obtenir une instance il suffit de faire :

```
SensorManager sensorManager =(SensorManager)getSystemService(Context.SENSOR_SERVICE);
```

Les capteurs sont représentés par la classe `Sensor`. Pour connaître la liste de tous les capteurs de l'appareil, vous devez utiliser la méthode `List<Sensor> getSensorList(int type)`, avec `type` égal à `Sensor.TYPE_ALL`. Pour la liste de tous les magnétomètres, par exemple, `type` sera égal à `Sensor.TYPE_MAGNETIC_FIELD`).

Pour obtenir une instance d'un capteur, utilisez la méthode `Sensor getDefaultSensor(type)`. Cette méthode ne donnera que l'appareil par défaut dans le cas de plusieurs capteurs avec le même objectif dans l'appareil. Vous devez toujours tester si le capteur est présent.

Exemple avec l'accéléromètre :

```
Sensor accelero = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
If (accelero != null)
	// il y a au moins un accéléromètre
else
	// y en a pas !
```

Ensuite, l'interface `SensorEventListener` permet de détecter :

a. Un changement de précision de celui-ci, avec la méthode de callback `void onAccuracyChanged(Sensor sensor, int accuracy)` avec `sensor` le capteur dont la précision a changée et `accuracy` la nouvelle précision (avec `SENSOR_STATUS_ACCURACY_LOW`, `SENSOR_STATUS_ACCURACY_MEDIUM`, `SENSOR_STATUS_ACCURACY_HIGH`, `SENSOR_STATUS_ACCURACY_UNRELIABLE` comme valeurs possibles).

b. Le capteur a calculé une nouvelle valeur, et dans ce cas, la méthode `void onSensorChanged(SensorEvent event)` est invoquée.

Un `SensorEvent` indique quatre informations : `accuracy` pour la précision de la mesure, `sensor` une référence sur ce capteur, `timestamp` l'instant en nanosecondes ou la valeur a été prise, et les valeurs de la mesure dans un tableau `values`.

Puis, il faut déclarer au capteur que nous sommes à son écoute, avec la méthode `boolean registerListener (SensorEventListener listener, Sensor sensor, int rate)` de `SensorManager`. Le paramètre `rate` peut prendre les valeurs suivantes : `SensorManager.SENSOR_DELAY_NORMAL,SensorManager.SENSOR_DELAY_UI, SensorManager.SENSOR_DELAY_GAME, SensorManager.SENSOR_DELAY_FASTEST`.

Enfin, il faut désactiver les capteurs pendant que l'activité n'est pas au premier plan (donc les désactiver pendant `onPause()` et les réactiver pendant `onResume()`) avec la méthode `void unregisterListener(SensorEventListener Listener, Sensor sensor)`.

Voici un exemple de code de base pour l'accéléromètre (sans test de présence...) :

```
private SensorManager mSensorManager = null;
private Sensor mAccelerometer = null;

final SensorEventListener mSensorEventListener = new SensorEvenListener() {
      public void onAccuracyChanged(Sensor sensor, int accuracy) {
            // que faire si changement de précision
      }
      public void onSensorChanged(SensorEvent sensorEvent) {
            // que faire en cas d'événement sur le capteur
            // dans notre cas, récupérer les valeurs x et y sufit…
            float x = sensorEvent.values[0];
            float y = sensorEvent.values[1];
      }
};

@override
public final void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
      mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
}

@override
protected void onResume() {
      super.onResume();
      mSensorManager.registerListener(mSensorEventListener,          mAccelerometer,
SensorManager.SENSOR_DELAY_NORMAL);
}

@override
protected void onPause() {
      mSensorManager.unregisterListener(mSensorEventListener, mAccelerometer);
}
```

7. Remarques

- Bloquer l'appareil en mode vertical dans le manifest. Interdire le jeu aux utilisateurs qui n'ont pas d'accéléromètre.

- Prévoir le chargement de différents plateaux de jeu. Vous réfléchirez à un modèle adapté pour celui-ci.

- Définir vous même la taille de votre surface de jeu, et ainsi le nombre de lignes / colonnes pour ce type de fichier en entrée.

7.1 Définir le chronomètre

Dans la vue fournie, une place est prévue pour afficher un chronomètre. Le principe est d'utiliser la méthode System.currentTimeMillis() et de comparer la valeur obtenue avec la valeur mémorisée au lancement du chronomètre. Ce traitement doit se faire en parallèle du jeu et doit agir sur la vue (pour mettre à jour régulièrement le temps écoulé). Il faut donc utiliser le Handler (http://developer.android.com/reference/android/os/Handler.html) d'Android. On le définira dans une classe spécifique.

Pendant le déroulement de la partie, le joueur souhaite faire une pause. Il faut donc :

1. Interrompre l'animation. Proposer un bouton de redémarrage.
2. interrompre le chronomètre. Le chronomètre redémarrera quand l'utilisateur cliquera de nouveau sur le bouton pause et que la grille sera donc à nouveau visible.

7.2 Autres événements

- Ajouter des Bonus / malus sur le plateau comme vous pouvez le voir sur la figure ; Une figure bonus ajoute du temps quand la boule passe dessus, l'autre figure malus en retire.

- Que se passe-t-il si on lance une nouvelle application (par exemple l'application téléphone) et que l'on revient sur le labyrinthe ?

- Effectuer les modifications nécessaires pour une bonne sauvegarde et restauration de l'état de l'application.

8. Conclusion

Vous devez rendre le code et un rapport expliquant clairement vos choix sous forme papier et par mail (stephane.malandain@hesge.com). La date de réddition vous sera communiquée en séance.