Florian JANISHON
Thomas MARGOTTIN

ITI, 3D

ENSEA
Ecole Nationale
Supérieure
de l'Electronique
et de ses Applications

Compte-rendu TP Android

Labyrinthe basculant

Professeur référent : M. Perez

Année : 2018-2019

I. Introduction

Le but de ce TP est de développer une application Android pour tablette Samsung Galaxy. L'application sera un jeu de « labyrinthe basculant ». Pour cela, nous mettrons en applications les connaissances acquises lors du cours. La notation se basera sur 2 points :

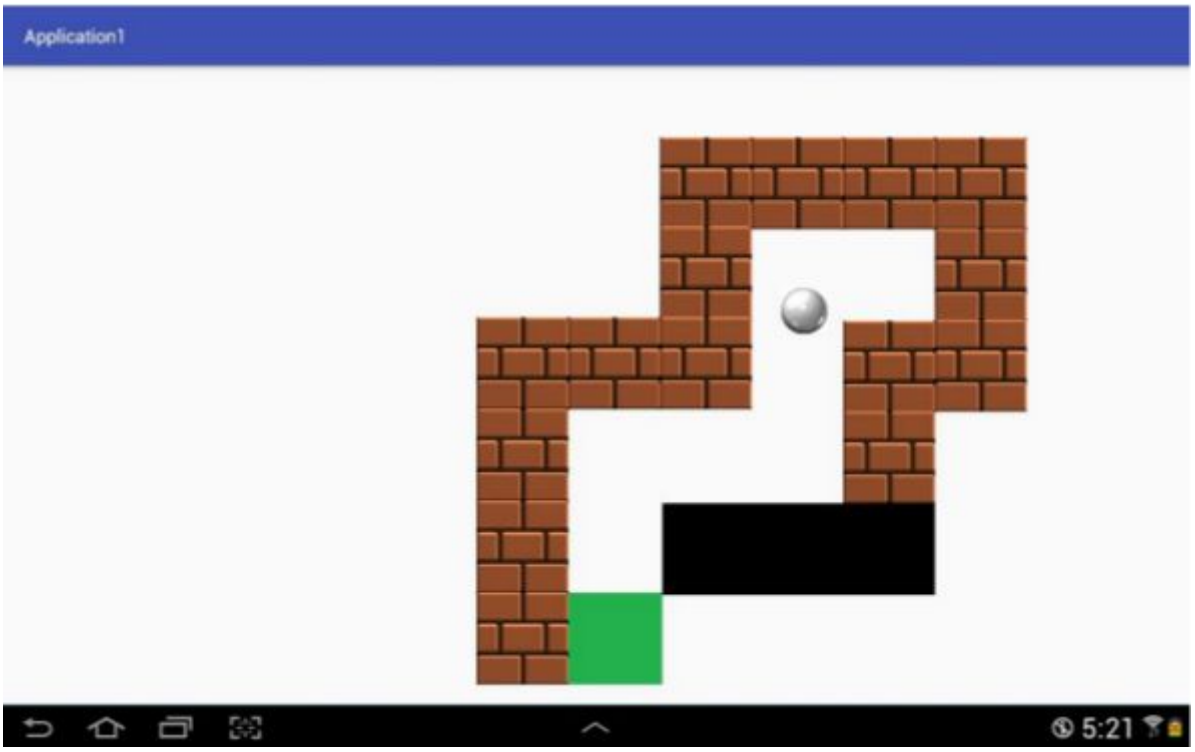
- l'avancement du TP. A la fin de chaque séance, le professeur notera l'avancement de chacun des groupes (maximum 2 personnes).
- le compte rendu, détaillé ci-dessous. Vous aurez du temps supplémentaire après la dernière séance pour le finir si vous le désirez.

Le compte rendu attendu doit contenir :

- Un fichier texte qui décrit les choix que nous avons effectué, leurs motivations et leur fonctionnement, aussi bien au niveau de l'architecture de notre application que du choix des classes, des attributs, etc.
- Les sources de l'application avec le code commenté. On y expliquera le rôle des attributs, variables locales, méthodes, classes, ressources, etc.

Note : si le contexte s'y porte, on pourra utiliser des dessins ou des *screenshots* pour mieux expliquer nos propos.

La figure suivante présente un aperçu de l'activité principale de l'application finale attendue :



II. Créer notre première application

Rappel la documentation officielle d'Android est disponible sur http://developer.android.com/

Afin de créer notre première application, nous lançons Android Studio pour créer notre premier projet.
Il faut d'abord renseigner le nom de l'application et le nom de la société (ici ENSEA).
On sélectionne le type de plateforme cible (*Phone and Tablets*). La version de SDK sera l'**API 15**. On créé une activité vide (Empty Activity).
Lors du premier lancement d'Android Studio, il va installer plusieurs types de fichiers (manifest, activité, ressources, scripts de compilation, etc.).

Désormais notre première application est prête à être coder.
Pour cela, nous allons instancier tous les *call-back* de l'activité. Pour les visualiser, nous allons utiliser les *logs* (journal d'activité) grâce à la classe **Log** (android.util.Log). On importe le code fourni dans le sujet de TP.
Nous lançons notre *activity* en mode **Debug**. L'application se lance automatiquement sur la tablette, on peut alors observer nos *logs* dans ***Android Monitor***, onglet ***logcat***.

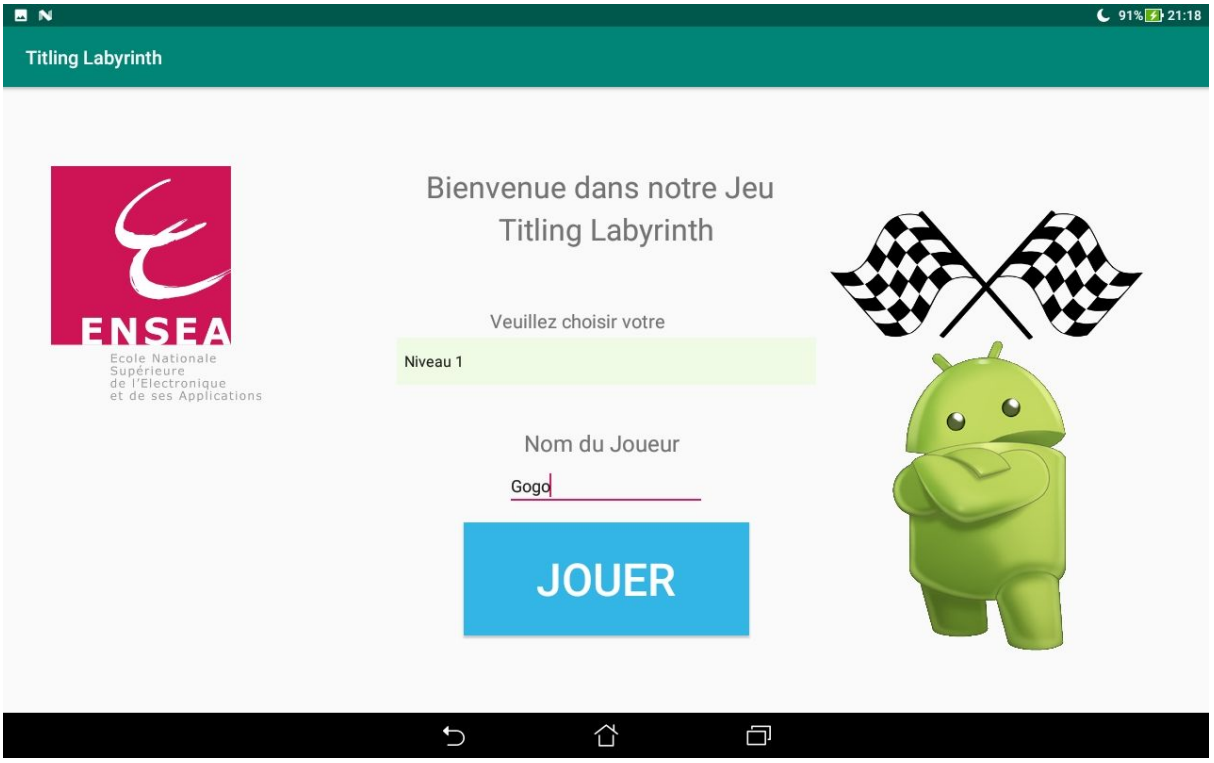
III. L’activité menu

L’objectif de l'activité ***MainActivity*** est de lancer le jeu et de renseigner les informations sur le joueur, typiquement son pseudo. Pour cela, on va commencer par ajouter 3 widgets :

- le nom du champ de saisi
- le champ de saisie pour le pseudo
- le bouton « Jouer »

On ouvre le fichier XML de l’activité. On commence par configurer le rendu pour l’appareil cible. On y sélectionne l’appareil cible, l’orientation (figée en paysage), le thème (light ou dark), le langage associé, la version d’Android (API 15).

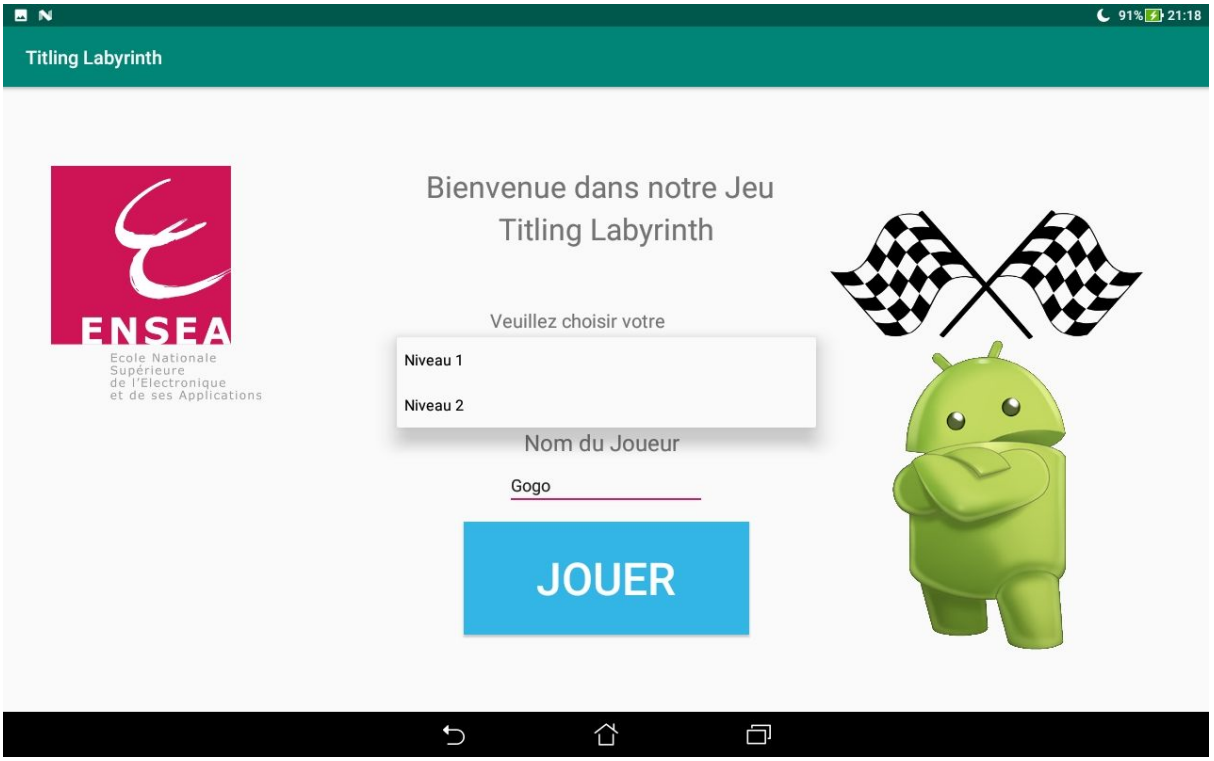
La figure suivante présente un aperçu de la page d'accueil de notre application :



Cette classe hérite de la classe *AppCompatActivity* et va permettre de gérer l’affichage du menu de démarrage via l’objet *INTENT*. C’est cette classe de base qui permet d’utiliser les bibliothèques des différentes fonctionnalités présentes dans Android Studio.

Dans notre activité “mère”, l’objet *INTENT* va permettre de lancer l’activité en question (soit *JeuActivity* ou *JeuActivity2* selon le choix de l’utilisateur).

Un aperçu de la sélection du niveau se trouve à la page suivante.



Comme nous pouvons le voir, le menu principale de l'application se découpe en plusieurs parties :

- Nous pouvons choisir de jouer deux niveaux différents que l'utilisateur peut choisir simplement à l'aide d'une déroulante.
- Il rentre ensuite son nom, bien que ce soit facultatif.
- Enfin, il suffit de cliquer sur le bouton *Jouer* pour faire apparaître la fenêtre de jeu du niveau choisi.

Le complément des explications se trouve directement dans les commentaires de la classe *MainActivity.java.*

IV. L’activité Jeu

L’objectif de l’activité ***JeuActivity*** est d’afficher le jeu.

A. Création de l’activité

On commence par ajouter un attribut *String* à l'activité qui contient le pseudo du joueur renseigné sur la page d'accueil.

B. Communications entre activités

On déclare un attribut constant « NOM » qui servira de clé pour les filtres d’intention :
public final static String NOM = "fr.ensea.monnom.application1.intent.NOM";

Puis on ajoute un bouton :
Button bJouer = (Button) findViewById(R.id.buttonJouer);

Enfin on ajoute le *listener* sur le bouton :

```
bJouer.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick (View v) {
                Intent intent = new Intent(MainActivity.this, JeuActivity.class);
                EditText playerName = (EditText) findViewById(R.id.editTextPlayerName);
                Log.d(TAG, "Player name = " + playerName.getText());
                intent.putExtra(NOM, "" + playerName.getText().toString());
                startActivity(intent);
        }
});
```

On déclare un attribut *String* contenant le nom du joueur.

Dans le *call-back* approprié, on récupère l’*INTENT* et à partir de ce dernier, le nom du joueur.

```
Intent intent = getIntent();
nomJoueur = intent.getStringExtra(MainActivity.NOM);
Log.d(TAG, "Player name = " + nomJoueur);
```

Pour afficher le nom du joueur, on peut utiliser un *toast*.
Toast.makeText(this, "Player name = " + nomJoueur, Toast.LENGTH_LONG).show();

On teste notre application, un *toast* apparaît avec le nom du joueur.

C. Récupérer l'orientation de la tablette

1. L'Accéléromètre

L'accéléromètre permet de récupérer les données d'accélération sur les 3 axes de la tablette. Nous ne nous servirons que de deux axes dans ce TP.
Les valeurs d'accélération sur chaque axe sont exprimé sont forme de Float en m/s². Donc si nous tenons la tablette droite, l'accélération est de 9.81 sur Y. Ou encore, si la tablette est à plat, l'accélération est nulle sur X et sur Y.

2. Récupérer les données de l'accéléromètre

L'accéléromètre permet de récupérer les données d'accélération sur les 3 axes de la tablette. Nous ne nous servirons que de deux axes dans ce TP.

Pour récupérer les données de l'accéléromètre, nous allons construire une classe Accelerometre qui possède plusieurs attributs. :

- float X, l'accélération sur l'axe X
- float Y, l'accélération sur l'axe Y
- JeuActivity mActivity, l'activité Jeu
- SensorManager mManager, qui va gérer l'accéléromètre
- Sensor mAccelerometre, l'accéléromètre

Pour initialiser ces deux derniers attributs on utilise :

- mManager = (SensorManager)mActivity.getBaseContext().
  getSystemService(Service.SENSOR_SERVICE);
- mAccelerometre = mManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);

Pour récupérer les valeurs de l'accéléromètre, attention à bien redéfinir les 2 méthodes :

```
SensorEventListener mSensorEventListener = new SensorEventListener() {
        @Override
        public void onSensorChanged(SensorEvent pEvent) {
                X = pEvent.values[0];
                Y = pEvent.values[1];
                Log.d(JeuActivity.TAG, "new sensor values: X=" + X + " Y=" + Y);
        }
        @Override
        public void onAccuracyChanged(Sensor pSensor, int pAccuracy) { }
```

Il faut aussi prévoir les méthodes pour s'abonner et se désabonner au capteur à l'aide des fonctions suivantes :

- mManager.unregisterListener(mSensorEventListener, mAccelerometre);
- mManager.registerListener(mSensorEventListener, mAccelerometre,
  SensorManager.SENSOR_DELAY_GAME);

3. Afficher la bille

Pour afficher la bille nous allons commencer par définir une classe Bille avec les attributs :

- int sizeX; // taille de la bille (px)
- int sizeY; // taille de la bille (px)
- float posX; // position de la bille
- float posY; // // position de la bille
- int maxX; // coordonnée max en X
- int maxY; // coordonnée max en Y
- ImageView imageView; // ImageView

Pour accéder à la position de l'ImageView :

- imageView.getX();
- imageView.getY();

De même pour la taille de l'ImageView :

- imageView.getWidth();
- imageView.getHeight();

Vous pouvez connaitre les dimensions maximales en X et en Y avec (dans l'activité) :

- Display display = getWindowManager().getDefaultDisplay();
- Point size = new Point();
- display.getSize(size);
- int width = size.x;
- int height = size.y;
- Log.d(TAG, "dim = " + width + " * " + height);

Pour modifier l'ImageView :

- imageView.setX(Float); // pour régler la position
- imageView.setY(Float);
- imageView.getX(Float); // pour obtenir la position
- imageView.getY(Float);
- imageView.setScaleX((Float); // pour effectuer un agrandissement
- imageView.setScaleY((Float);

Pour récupérer les attributs de ImageView dans notre activité dans le onCreate() :

- @Override public void onWindowFocusChanged(boolean hasFocus){ }

V. Le Jeu

Rentrons maintenant dans le coeur du jeu. Pour bien démarrer, nous allons définir ce qui sera nécessaire pour un jeu minimal. Nous aurons besoin de plusieurs éléments :

- La bille, l'élément central du jeu.
- Le décor, constitué :
    - De murs, qui bloquent la bille

- De trous, qui capturent la bille et fait perdre la partie
- D'un point d'arrivée, qu'il faut atteindre avec la bille pour gagner

Pour gérer ces éléments, nous allons séparer notre code en deux parties :

- Un moteur graphique qui va gérer l'affichage des éléments.
- Un moteur physique qui va gérer le comportement des éléments du jeu (position, déplacement, interaction avec d'autres éléments).

A. Le modèle physique

1. La gravité

Pour avoir un rendu réaliste sur le mouvement de la bille nous devrons utiliser un modèle basé sur l'accélération. Sur chacun des axes X et Y définissez un attribut de Bille, accélération, vitesse et position.
La vitesse est la dérivée de la position, c'est-à-dire la différence entre deux positions successives.
De même, l'accélération est la dérivée de la vitesse, c'est-à-dire la différence entre deux vitesses successives.

Pour calculer la position de la bille à partir de l'accélération (exemple) :
(T=0) conditions initiales aX = 0; vX = 0; pX = 0;
(T=1) L'accélération passe à 1 donc on a : aX = 1; vX = 1; pX = 1;
(T=2) L'accélération reste constante : aX = 1; vX = 2; pX = 3;
(T=3) L'accélération reste constante : aX = 1; vX = 3; pX = 6;

2. Les frottements

Pour ne pas donner une impression encore plus réaliste la balle va subir une force de frottement. Physiquement cela veut dire que sa vitesse va suivre la formule suivante :
$vX \mathrel{+}= - \mu * vX$.
Avec mu le coefficient de frottement ($0 < \mu < 1$).

3. Le rebond

Lorsque la bille rencontre un obstacle « mur » elle rebondit. Physiquement cela veut dire que sa vitesse va suivre la formule suivante : $vX = - \alpha * vX$.
Avec alpha le coefficient de rebond ($0 < \alpha < 1$).

4. Les collisions

Pour simplifier les collisions du modèle, nous considérons que tous les objets sont des « rectangles » pour les collisions. Les blocs de décor sont déjà rectangulaire, donc pas de problème. Pour la bille, on peut considérer qu'il s'agit du carré circonscrit au cercle de la bille.

B. Le modèle graphique

Le modèle graphique sera notre View principale de notre activité *JeuActivity*.

1. Le temps

Pour simuler temps nous avons besoin de choisir un pas qui va déterminer la fréquence d'actualisation de l'affichage. Pour cela nous avons besoin d'un nouveau thread différent du thread UI.

```
private class monThread extends Thread {
	@Override public void run() {
		while (isRunning) {
			/* mise à jour de la vue */
			 try {
				Thread.sleep(rate);
			}
			 catch (InterruptedException e) {}
		}
	}
}
```

C. L'activité Jeu

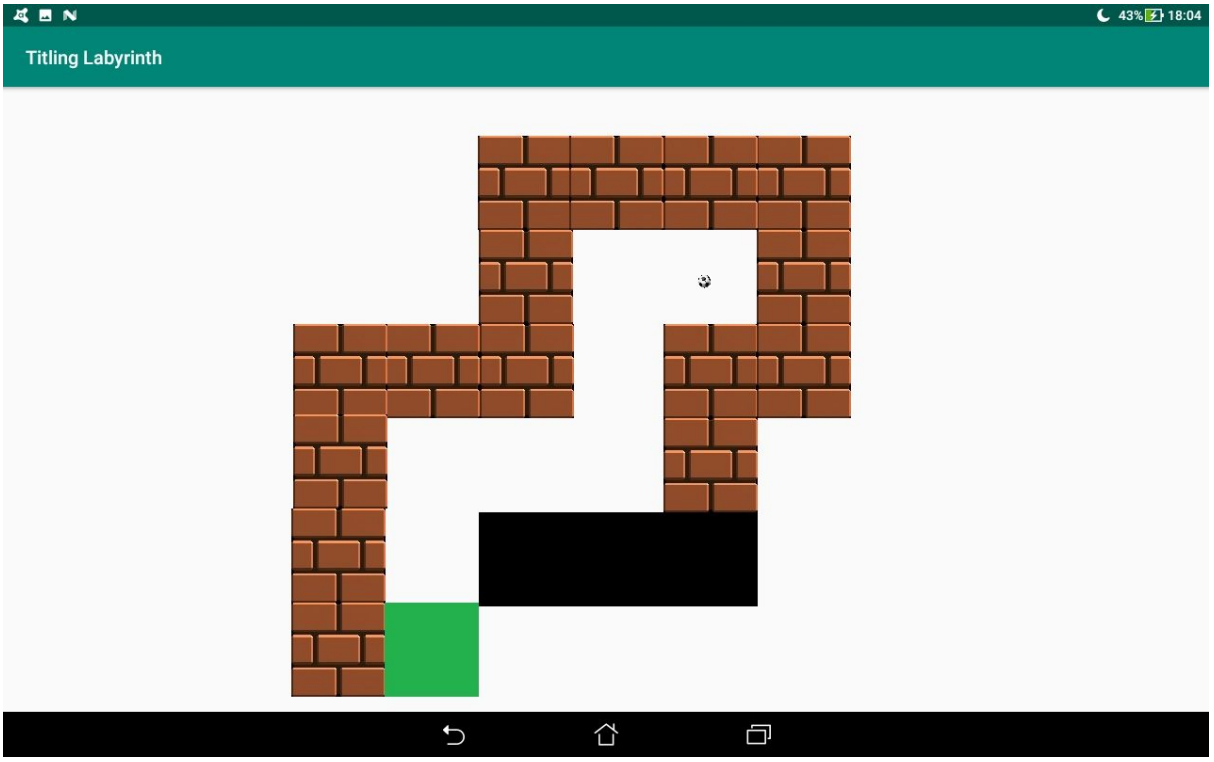
La classe ***JeuActivity*** va permettre de gérer l'affichage de l'application. Ce sera donc la classe principale qui contiendra la partie graphique du jeu et qui appelera les classes gérant l'accéléromètre ou encore le mouvement de la bile. Elle contient plusieurs méthodes :

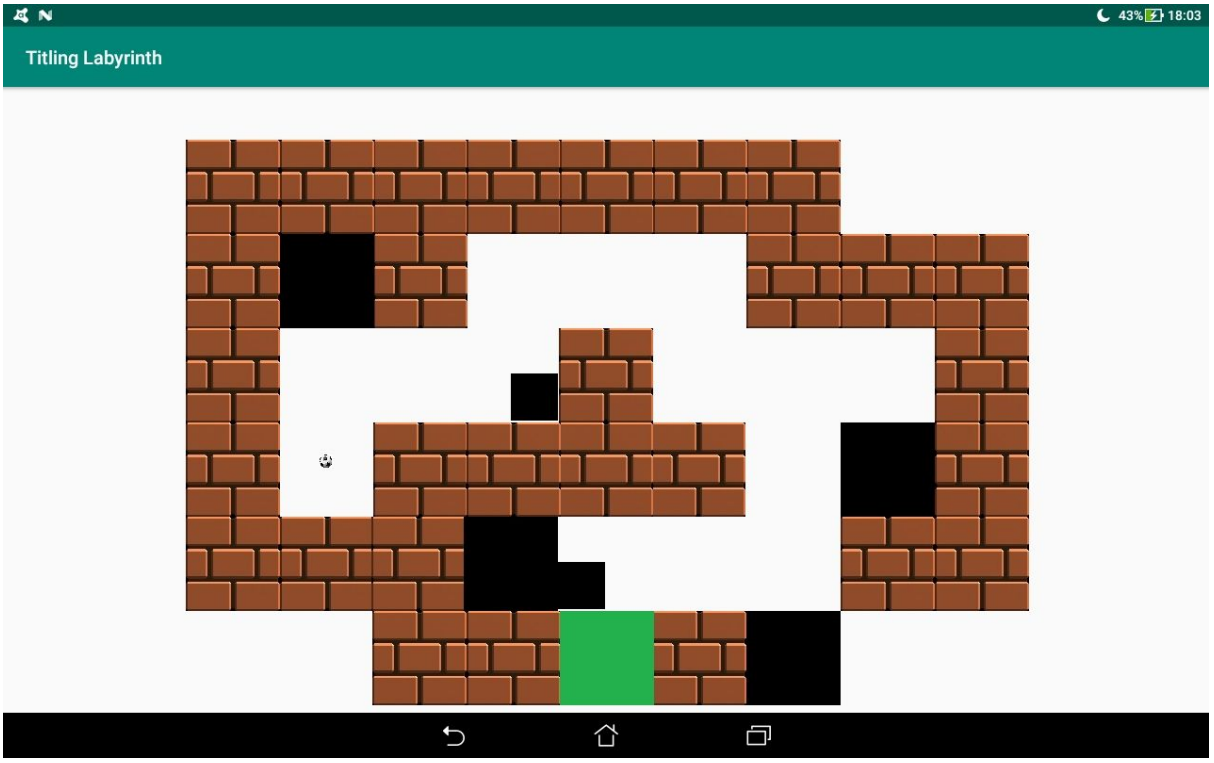
- ***onCreate()*** qui va permettre de générer une nouvelle vue graphique.
- ***onResume()*** qui va permettre de démarrer le *thread* et l'abonnement à l'accéléromètre.
- ***onPause()*** qui va permettre d'interrompre le *thread* et le désabonnement à l'accéléromètre.

La méthode *onWindowFocusChanged()* va permettre de récupérer toutes les informations nécessaires une fois que la vue graphique sera créée.

Comme nous décidons de créer deux niveaux, notre projet contiendra deux activités de jeu. Les deux activités fonctionnent exactement pareil en dehors des instanciations des blocs qui composent le jeu.

Voici un aperçu des deux niveaux créés (respectivement le niveau 1 et le niveau 2) :





Le complément des explications se trouve directement dans les commentaires de la classe *JeuActivity.java.*

1. La bille

L'objet Bille contient les méthodes régissant le comportement physique de la bille.

La classe ***Bille*** va permettre de gérer la récupération des informations sur le modèle graphique mais aussi de gérer toute la partie physique. Dans cette activité, nous allons gérer le mouvement de la bille et ses intéractions avec les éléments du décor.

Les paramètres graphiques de la bille sont passés en argument du constructeur de la classe Bille, ce qui nous permet d'avoir une visibilité sur la partie graphique. Cette classe possède une méthode ***updateBillePosition()*** qui va permettre de gérer le modèle physique de la bille. Cette méthode est appelé dans le thread externe pour actualiser les informations à fréquence fixe.

Les coefficients indiqués pour l'accélération, la vitesse ou encore la position ont été choisi arbitrairement. Nous avons choisi ces paramètres en fonction du rendu final sur la tablette.

Le complément des explications se trouve directement dans les commentaires de la classe *Bille.java.*

2. L'accéléromètre

La classe ***Accelerometre*** va permettre de gérer toutes les informations provenant de l'accéléromètre de la tablette. Elle contient des *getters* pour récupérer la valeur en X et Y. Des services existent déjà pour pointer sur l'accéléromètre de la tablette et pouvoir ainsi l'utiliser.

Elle contient également deux méthodes ***abonnementCapteur()*** et ***desabonnementCapteur()*** qui vont permettre, respectivement, de s'abonner et se désabonner de l'accéléromètre. Ces deux méthodes évitent d'utiliser inutilement certaines ressources de la tablette.

Le complément des explications se trouve directement dans les commentaires de la classe *Accelerometre.java.*

3. Le Thread

La classe du moteur graphique gère l'actualisation à intervalle régulier des View (bille, murs, tours, arrivée) en fonction de l'état du jeu calculé par le moteur physique.

Le complément des explications se trouve directement dans les commentaires de la classe *myThread.java.*

Remarque : Des bugs graphiques peuvent apparaître lors d'une collision entre la bille et un mur. Ceci provient des conditions de collisions implémentées très restrictives et également de la vitesse de rafraîchissement du Thread général.