



# Projet Android

---

## Trainer

Rémi BOMMELAERE  
Laurent GENTY  
Luis PALLUEL

UQAC - 2018



# Approche initiale

---

Plusieurs idées :

- Utilisation de l'accéléromètre
- Mini-jeux
  - WarioWare, Pou, ...
- Personnage qui évolue avec le temps
- Podomètre
- ...



# Sommaire

## 1. Rendu partiel

- a. Gestion du personnage
- b. Mini-jeux
- c. Interface
- d. Modèle 3D

## 2. Problèmes

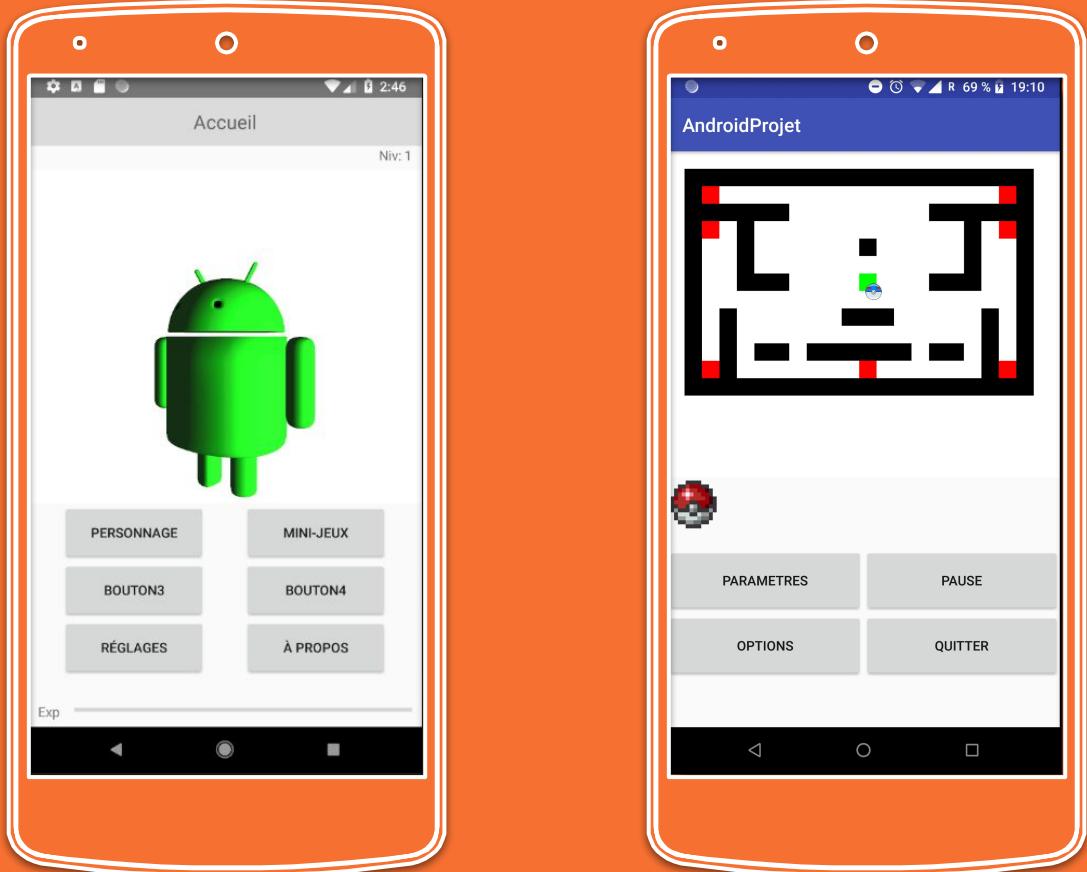
- a. Senseurs et mini-jeux
- b. Modèle 3D

## 3. Améliorations envisagées

# 1.

## Rendu partiel

*Mini-jeux  
Expérience  
Interface  
Modèle 3D*



# Personnage

---

## Expérience



Evolution du personnage

# Personnage

---

## Expérience

- ▶ Plusieurs sources d'expérience

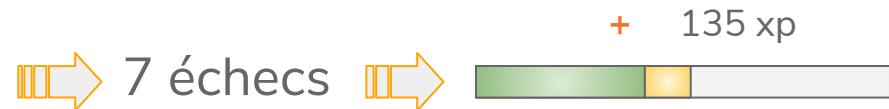
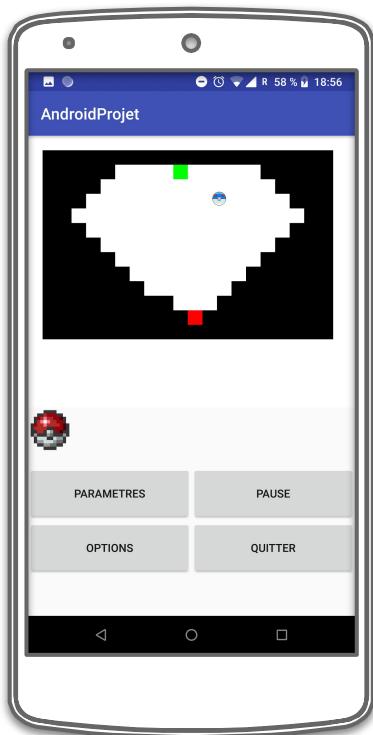


# Personnage

## Expérience

↳ Plusieurs sources d'expérience

► Mini-jeux

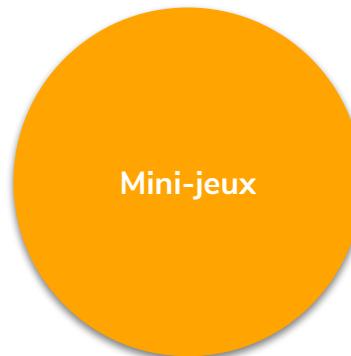


# Personnage

---

## Expérience

► Plusieurs sources d'expérience



# Personnage

---

## Expérience

### ↳ Plusieurs sources d'expérience

#### ► Temps inactif

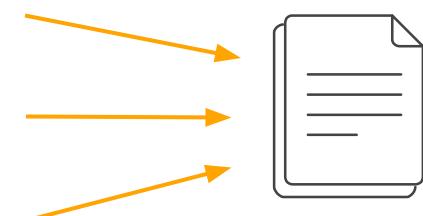
**onDestroy()**



Date()

Niveau

Experience



# Personnage

## Expérience

### ↳ Plusieurs sources d'expérience

#### ► Temps inactif

```
public void save_last_connexion(){
    @SuppressLint("SimpleDateFormat") DateFormat dateFormat = new SimpleDateFormat( pattern: "yyyy/MM/dd HH:mm:ss");
    Date date = new Date();
    try {
        OutputStreamWriter outputStreamWriter = new OutputStreamWriter(this.getApplicationContext().openFileOutput( s: "last_connexion.txt", Context.MODE_PRIVATE));
        outputStreamWriter.write(dateFormat.format(date));
        outputStreamWriter.close();
    }
    catch (IOException e) {
        Log.e( tag: "Exception", msg: "File write failed: " + e.toString());
    }
}
```

# Personnage

## Expérience

### ↳ Plusieurs sources d'expérience

#### ► Temps inactif



# Personnage

## Expérience

### ↳ Plusieurs sources d'expérience

#### ▶ Temps inactif

```
if(this.baseContext.getFileStreamPath("last_connexion.txt").exists()) {  
  
    try {  
        FileInputStream fis = this.applicationContext.openFileInput("last_connexion.txt");  
        InputStreamReader isr = new InputStreamReader(fis);  
        BufferedReader bufferedReader = new BufferedReader(isr);  
        StringBuilder sb = new StringBuilder();  
        String line;  
        while ((line = bufferedReader.readLine()) != null) {  
            sb.append(line);  
        }  
        if(sb.toString() != ""){  
            this.date_lc = sb.toString();  
        }  
    }  
}
```

# Personnage

---

## Expérience

► Plusieurs sources d'expérience



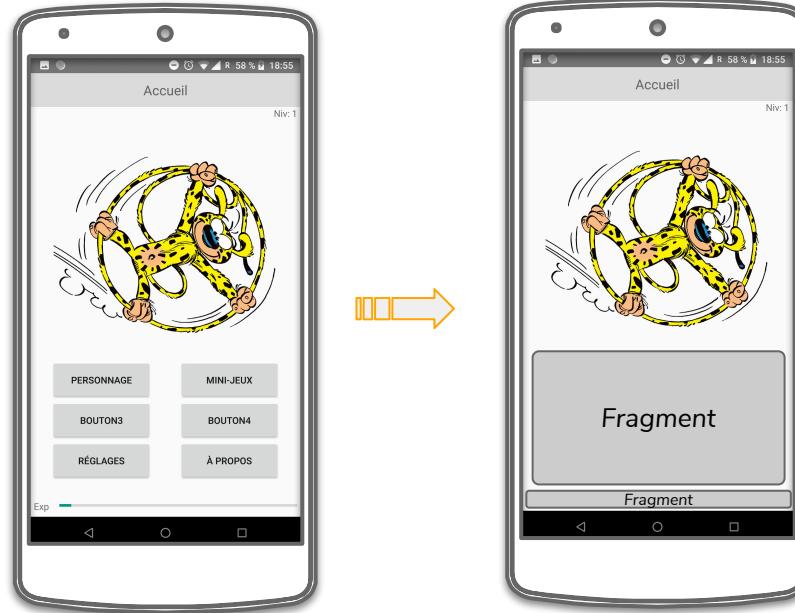
# Interface

---

# Interface

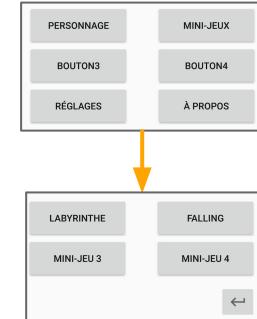
---

## Fragments



onClick() →

```
Fragment1.hide();  
Fragment2.show();
```



# Interface

## Fragments

```
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="0dp"
    android:layout_weight="35"
    android:layout_gravity="center">

    <fragment
        android:id="@+id/frag_main_menu"
        android:name="ugac.dim.androidprojet.menu_fragments.Main_menu"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <fragment
        android:id="@+id/frag_stats"
        android:name="ugac.dim.androidprojet.menu_fragments.Statistiques"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <fragment
        android:id="@+id/frag_menu_mini_jeu"
        android:name="ugac.dim.androidprojet.menu_fragments.Menu_mini_jeu"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <fragment
        android:id="@+id/frag_menu_labyrinthe"
        android:name="ugac.dim.androidprojet.menu_fragments.Menu_labyrinthe"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

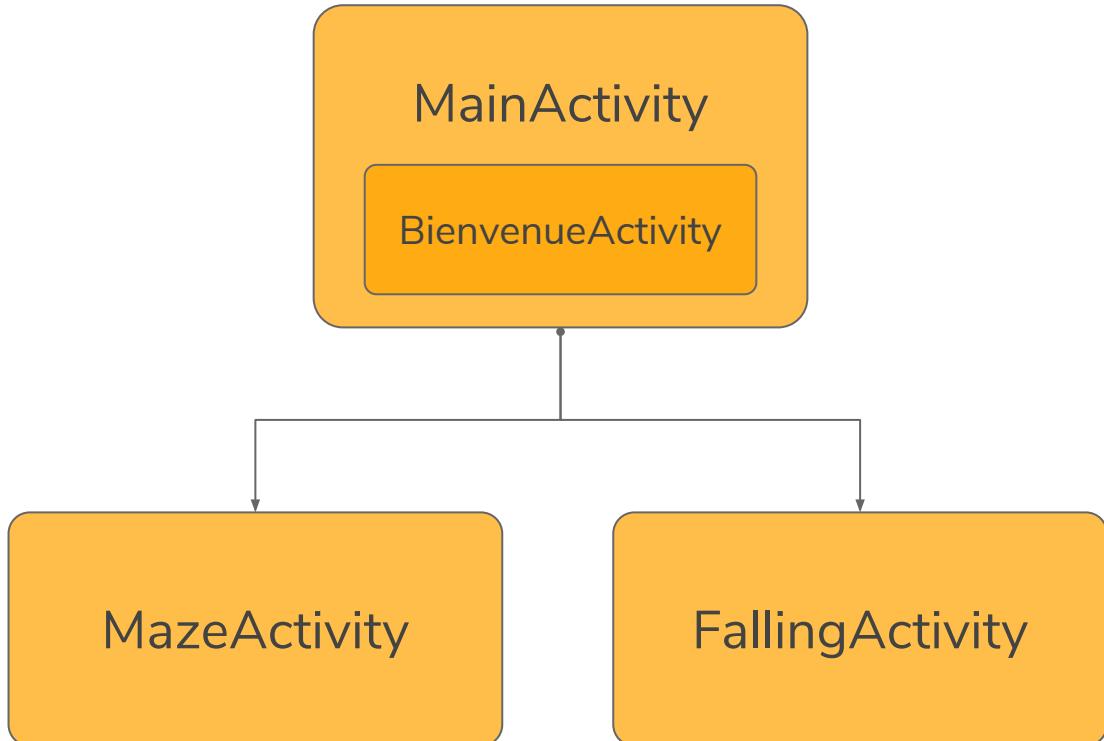
```
public void show_mini_jeu(View view) {
    FragmentManager fm = getFragmentManager();
    fm.beginTransaction()
        .hide(this.main_menu)
        .commit();

    fm.beginTransaction()
        .setCustomAnimations(android.R.animator.fade_in, android.R.animator.fade_out)
        .show(this.mini_jeu)
        .commit();
}
```

# Interface

---

## Activités



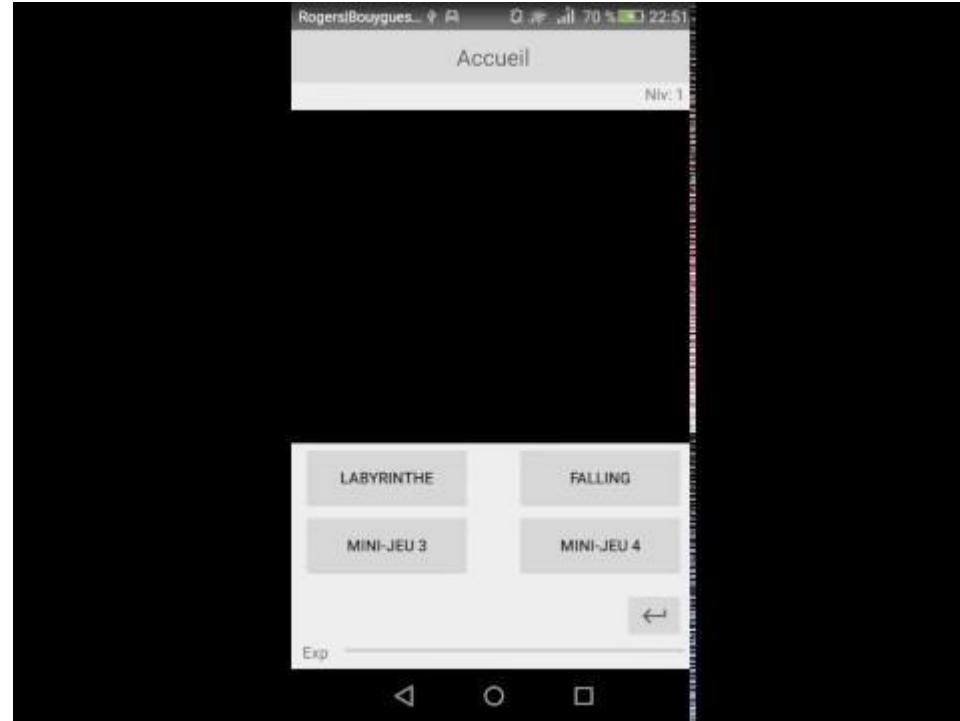
# Mini-jeux

---

# Mini-jeux

---

## Maze



<https://goo.gl/BXFh4s>

# Mini-jeux

---

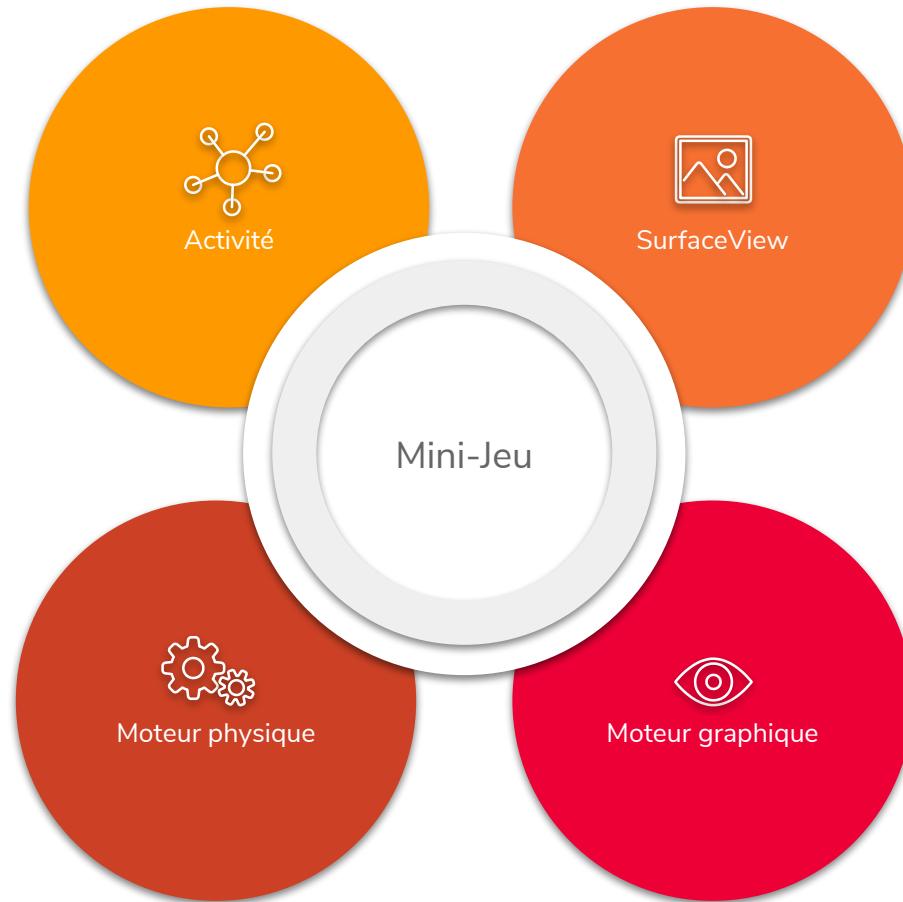
## Falling



<https://goo.gl/Zr9dzn>

# Mini-jeux

## Architecture générale



# Mécanismes générales

## Activité



Activité contient :

- Moteur physique
- Moteur graphique
- **Tous les éléments du jeu**
  - Ring, Catcher, Walls, ...
- Widgets
  - Buttons, TextView, ImageView

Faire le liens entre les différents moteurs

- Instancier et gérer

Gérer les niveaux

Gérer les constantes

...

```
public class MazeGame extends AppCompatActivity {

    //ma vue : éléments sur l'écran
    private MazeView view;

    //données
    private MazeData mData;

    //éléments du labyrinthe
    private Ball ball;
    private ArrayList<Wall> walls = null;

    //gap
    private static int GAP;

    //widgets graphiques
    private LinearLayout all;

    private LinearLayout game;

    private LinearLayout buttonsLayout;
    private LinearLayout otherButtons;
    private LinearLayout gameButtons;
    private Button optionsButton;
    private Button parametersButton;
    private Button quitButton;
    private Button pauseButton;

    private ImageView imageLives;
    private TextView textLives;

    //vies du joueur
    private int lives = 5;
```

# Mécanismes générales

## SurfaceView



SurfaceView **implémentée** (et non pas un Layout)

Récupérer les éléments de l'activité

→ les **afficher seulement**

Possède un **thread** afin de gérer l'affichage

- ie. Moteur graphique
- Tâches longues (donc pas une AsyncTask)

```
public class MazeView extends SurfaceView implements SurfaceHolder.Callback {

    //ma boule
    private Ball ball;

    //ma liste de murs
    private ArrayList<Wall> walls = null;

    //dessin
    private SurfaceHolder mHolder = null;
    private Paint brush;

    private DisplayThread thread;

    //taille écran
    private static int WIDTH;
    private static int HEIGHT;

    public MazeView(Context context) {
        super(context);
        init(context);
    }

}
```

# Mécanismes générales

## Moteur graphique



Thread en classe **interne** (permettre d'avoir accès aux variables privées)

Instancié dès le **début** du mini-jeu

- Lors de l'instanciation de la SurfaceView

Utilisation d'Android Canvas pour afficher les éléments

- Canvas ↔ toile
- Paint ↔ pinceau **ET** palette
- Bitmap ↔ ressource

Gérer les redimensionnements

Ne s'occupe en **aucun cas**

de la gestion physique des objets

```
private class DisplayThread extends Thread {  
  
    private boolean drawing = true;  
  
    @Override  
    public void run() {  
        while (drawing) {  
            Canvas canvas = null;  
            try {  
                //recupere et bloque pour dessiner sur le canvas  
                canvas = mHolder.lockCanvas();  
  
                synchronized (mHolder) {  
                    // Et on dessine  
                    draw(canvas);  
                }  
            } finally {  
                if (canvas != null) {  
                    //libere le canvas pour l'afficher  
                    mHolder.unlockCanvasAndPost(canvas);  
                }  
            }  
            try {  
                //on met en pause le thread  
                Thread.sleep( millis: 35 );  
            } catch (InterruptedException e) {  
            }  
        }  
    }  
  
    public void setDrawing(boolean drawing) { this.drawing = drawing; }  
}
```

# Affichage des éléments

```
@Override
public void draw(Canvas canvas) {
    super.draw(canvas);

    Bitmap resizedBitmapBall = createBitmapResized( type: "BALL" );

    canvas.drawColor(Color.WHITE);
    //dessiner les blocs
    for (Wall w : walls) {
        brush.setColor(w.getColor());
        canvas.drawRect(w.getRectangle(), brush);

        if (w.getType() == WALL_TYPE.ENDING) {
            Bitmap resizedBitmapWall = createBitmapResized( type: "ENDING" );
            canvas.drawBitmap(resizedBitmapWall, left: w.getX() - Wall.WALL_SIZE, top: w.getY() - Wall.WALL_SIZE, brush);

        } else if (w.getType() == WALL_TYPE.STARTING) {
            Bitmap resizedBitmapWall = createBitmapResized( type: "STARTING" );
            canvas.drawBitmap(resizedBitmapWall, left: w.getX() - Wall.WALL_SIZE, top: w.getY() - Wall.WALL_SIZE, brush);
        } else if (w.getType() == WALL_TYPE.VOID) {

            //EDIT : lorsqu'on veut afficher toutes les images du jeu ca ram (trop demandant sans doute)
            /*Bitmap resizedBitmapWall = createBitmapResized("WALL");
            canvas.drawBitmap(resizedBitmapWall, w.getX() - Wall.WALL_SIZE, w.getY() - Wall.WALL_SIZE, brush);*/
            //canvas.drawRect(w.getRectangle(), brush);
        }
    }

    if (ball != null) {
        canvas.drawBitmap(resizedBitmapBall, left: ball.getX() - Ball.RADIUS, top: ball.getY() - Ball.RADIUS, brush);
    }
}
```

# Mécanismes générales

## Moteur physique

Récupérer les éléments de l'activité

→ gestion **physique**

Gère les interactions entre objets et les données

Utilisation des **sensors** pour jouer

- Stocker chaque mouvement et le traiter

```
//event listener
SensorEventListener mSEL = new SensorEventListener() {
    @Override
    public void onSensorChanged(SensorEvent sensorEvent) {
        float newY = 0;

        changementDirectionSprite(newY);

        //nouvelles valeurs dans values
        newY = sensorEvent.values[2];
        if (catcher != null) {
            //la hitbox correspond aux coordonnées de la boule actuellement

            RectF catcherHitbox = catcher.setNewCoordinates(newY);

            if (fallingRings != null) {
                for (Ring r : fallingRings) {
                    //si un wall touche la balle
                    RectF temp = new RectF(r.getLeft(), r.getTop(), r.getRight(), r.getBottom());

                    verifierIntersection(temp, catcherHitbox, r);
                }
            }
        }
        _lastY = newY;
    }
}
```

# Gestion des objets

---

Peut posséder un **thread** pour effectuer les actions en arrière plan (exemple mini-jeu Falling)

- Déplacement d'objets
- Vérifier les collisions
- Vérifier les sorties d'écran
- L'accélération
- ...

```
    @Override
    public void run() {
        while (falling) {
            sortieEcran();
            testLives();
            initRandomRing();
            descenteRings();
            accelerationRings();

            try {
                // on met en pause le thread
                Thread.sleep( millis: 50 );
            } catch (InterruptedException e) {
            }
        }
    }
}
```

# Processus de création d'un mini-jeu

---

3 étapes obligatoires



# Méthodologie

## 1. Crédit de l'activité, moteurs et vue custom

Créer toutes les classes que contiendra le mini-jeu

- Objets qui constituent votre mini-jeu

Créer les moteurs correspondants

- Les moteurs possèdent les références aux objets 

Initialiser tous les objets

- L'activité est le début de chaque mini-jeu !

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

    FW = new FallingView(this);
    setContentView(FW);

    //et les données physiques des objets
    FE = new FallingEngine(this, FW);

    //init la ball
    catcher = new Catcher();

    //je la met dans les deux moteurs
    FW.setCatcher(catcher);
    FE.setCatcher(catcher);

    initCatcherInitial();

    initRingsCatcher();

    fallingRings = initRings(fallingRings);

    FE.setFallingRings(fallingRings);
    FW.setFallingRings(fallingRings);
```

# Méthodologie

## 1. Crédit de l'activité, moteurs et vue custom

Les moteurs possèdent les même attributs que l'activité (... ou presque !)

Faire référence aux **mêmes objets** (très important)

- Les objets seront 'partagés'

```
FallingEngine init()
{
    public FallingEngine(Falling fA, FallingView fW) {
        init(fA);
        this.fW = fW;
    }

    private void init(Falling fA) {
        //l'activité c'est celle du paramètre
        setFallingActivity(fA);

        //classe permettant d'accéder aux capteurs : instance
        setSM((SensorManager) fA.getBaseContext().getSystemService(Service.SENSOR_SERVICE));

        //on link les sensors etc
        setmAcelerometre(sm.getDefaultSensor(Sensor.TYPE_ACCELEROMETER));

        setScore(0);
        setLives(5);
        fallingActivity.setLives(5);

        thread = new FallingThread();
        thread.start();
    }
}
```

```
FallingView
{
    public FallingView(Falling fA) {
        super(fA);
        init(fA);
    }

    private void init(Falling fA) {
        setFallingActivity(fA);

        mHolder = getHolder();
        mHolder.addCallback(this);
        thread = new DisplayThread();

        brush = new Paint();
        brush.setStyle(Paint.Style.FILL);
        brush.setTextSize(50);

        fallingRings = new ArrayList<Ring>();
        catcher = new Catcher();

        //faire en sorte que toutes les pieces font 1/5 de l'écran en largeur
        WindowManager wm = (WindowManager) fA.getSystemService(Context.WINDOW_SERVICE);
        Display display = wm.getDefaultDisplay();

        setWIDTH(display.getWidth());
        setHEIGHT(display.getHeight());
    }
}
```

# Méthodologie

## 2. Senseurs

## Utiliser le gestionnaire de capteurs : SensorManager

- Instancier le capteur que l'on souhaite

## Créer une implémentation de l'interface SensorEventListener : gérer les capteurs

- Changement de valeur d'un capteur
  - Changement de précision (rare)

A chaque mouvement du téléphone → interpréter ce mouvement

```

    @Override
    public void onSensorChanged(SensorEvent sensorEvent) {
        Log.i("MySensor", "Sensor changed");
        float newx = 0, newy = 0;

        //rest
        //MM.getOrientation(P,sensorEvent.values);
        //MM.getCoordinateSystem(S,MM.AXIS_X);
        //int screenRotation = get.WindowManager().getDefaultDisplay().getRotation();

        //nouvelles valeurs dans values
        newx = sensorEvent.values[1];
        Log.i("MySensor", "New x: " + Float.toString(newx));
        newy = sensorEvent.values[2];
        Log.i("MySensor", "New y: " + Float.toString(newy));
        Log.i("MySensor", "New z: " + Float.toString(newz));

        Log.i("MySensor", "New x: " + Float.toString(newx));
        Log.i("MySensor", "New y: " + Float.toString(newy));
        Log.i("MySensor", "New z: " + Float.toString(newz));

        if (ball != null) {
            Log.i("MySensor", "New hitbox");
            //la hitbox correspond aux coordonnées de la boule actuellement
            ball.setHitbox = ball.setNewCoordinates(newx, newy);
        }

        for (Wall w : walls) {
            //si la boule touche le mur
            RectF temp = new RectF(w.getRectangle());
            if (temp.intersects(ball.getHitbox)) {
                switch (w.getType()) {
                    case WALL_TYPE_LIVES:
                        lives--;
                        maseactivity.setLives(lives);
                        toaster(WALL_TYPE_VOID);
                        break;
                    case WALL_TYPE:
                        toaster(WALL_TYPE_ENDING);
                        break;
                }
            }
        }
    }
}

```

# Méthodologie

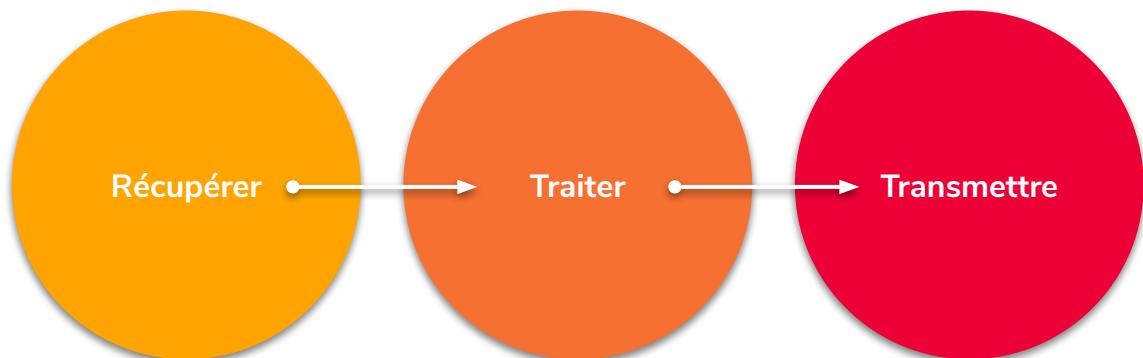
---

## 3. Calibrage

Partie la plus difficile

- Équilibrer les mouvements
- Utiliser les bons repères
- Calculer les mouvements pour déplacer les objets

...



## Modèle 3D

---

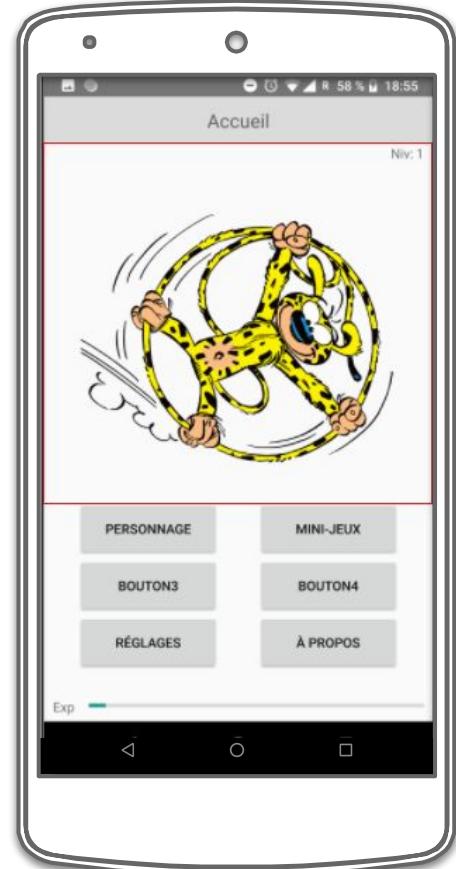
Approche  
initiale



# Modèle 3D

## Approche initiale

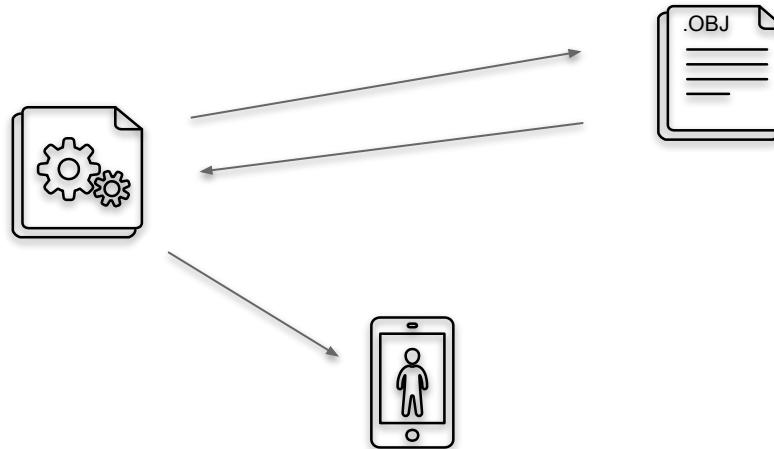
Remplacer l'image par une vue de notre personnage, en 3 dimensions



## Modèle 3D

---

### Approche initiale



### Problèmes :

- Gestion manuelle des textures
- Contrainte de temps, dûe à la complexité de la librairie



## Modèle 3D

---

## Changement de stratégie

## Utilisation de Rajawali



*“Rajawali is a 3D engine for Android based on OpenGL ES 2.0/3.0”*

# Modèle 3D

## Utilisation de Rajawali

1. Construction du renderer
2. Initialisation de la scène
3. Actions à l'affichage

```
public class myRenderer extends org.rajawali3d.renderer.Renderer {  
  
    private Object3D mModel;  
    private DirectionalLight mDirectionalLight;  
    private Context context;  
  
    public myRenderer(Context context) {  
        super(context);  
        this.context = context;  
        setFrameRate(60);  
    }  
}
```

# Modèle 3D

## Utilisation de Rajawali

1. Construction du renderer
2. Initialisation de la scène
3. Actions à l'affichage

```
30
31 ①
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
```

```
@Override
protected void initScene() {
    mDirectionalLight = new DirectionalLight( xDir: 1.0f, yDir: .2f, zDir: -1.0f );
    mDirectionalLight.setColor( r: 1.0f, g: 1.0f, b: 1.0f );
    mDirectionalLight.setPower(2);
    getCurrentScene().addLight(mDirectionalLight);

    LoaderOBJ objParser = new LoaderOBJ(mContext.getResources(), mTextureManager, R.raw.android_obj);

    try {
        objParser.parse();
    } catch (ParsingException e) {
        e.printStackTrace();
    }
    mModel = objParser.getParsedObject();
    //mModel.setMaterial(material);

    getCurrentScene().addChild(mModel);
    getCurrentScene().setBackgroundColor(0xFFFFFFFF);

    getCurrentCamera().setY(2f);
    getCurrentCamera().setZ(10f);
}
```

# Modèle 3D

## Utilisation de Rajawali

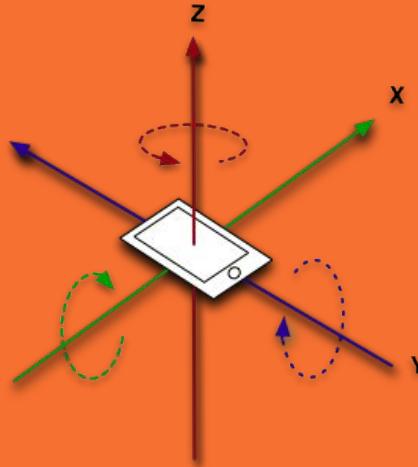
1. Construction du renderer
2. Initialisation de la scène
3. Actions à l'affichage

```
55
56 @Override
57     public void onRender(final long elapsedTime, final double deltaTime) {
58         super.onRender(elapsedTime, deltaTime);
59         mModel.rotate(Vector3.Axis.Y, angle: 1.0);
60     }
```

## 2.

### Problèmes rencontrés

*Liste des problèmes majeurs rencontrés*



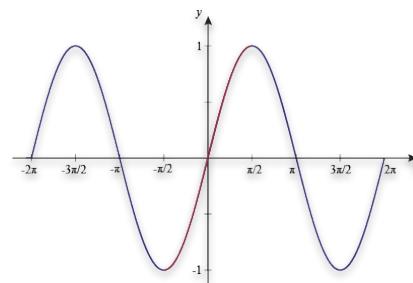
# Problèmes

---

## Repère mal calibré

Mauvais repère de base :

- Transformer la matrice de base en un repère que l'on souhaite



# Problèmes

---

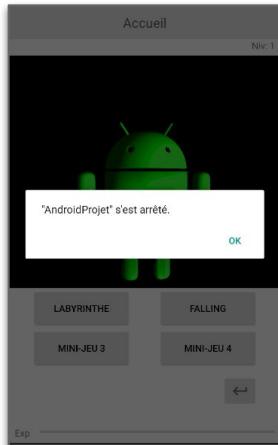
## Arrêts random Falling

Pas si random...

Rafraîchissement des deux threads en parallèles cause des problèmes :

- Lorsqu'un thread accède à un ArrayList, il le parcourt
- Engine peut supprimer des items dans ce conteneur de manière asynchrone

**PROBLEME :** concurrence des thread



**Solution :** Mutex, sleep les thread

# Modèle 3D

## Gestion des textures

```
v -2.350358 0.202702 0.000000
v -2.330119 0.203854 0.120854
v -2.339371 0.331888 0.124902
v -2.360188 0.331888 0.000000
vn 0.5958 -0.5385 -0.5958
vn 0.4421 -0.7804 -0.4421
vn 0.2617 -0.8274 -0.4968
vn -0.3119 -0.5319 -0.7872
vn 0.9159 -0.2091 -0.3427

f 6455//1 6457//2 6458//3
f 6458//3 5300//4 5298//5
f 6456//6 5269//7 6457//2
f 6457//2 5272//8 5300//4
```

android.obj

```
newmtl Material.001
Ns 94.117647
Ka 0.000000 0.000000 0.000000
Kd 0.000000 0.000000 0.000000
Ks 0.500000 0.500000 0.500000
Ke 0.000000 0.000000 0.000000
Ni 1.000000
d 1.000000
illum 2

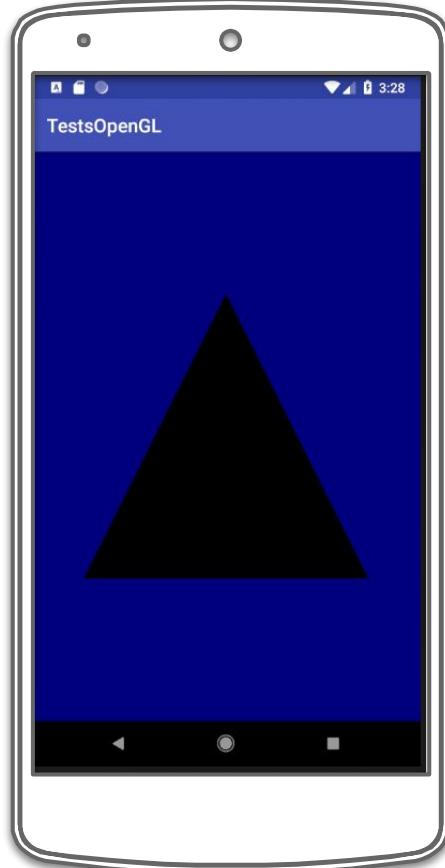
newmtl Material.002
Ns 37.254902
Ka 0.000000 0.000000 0.000000
Kd 0.070000 0.700000 0.070000
Ks 0.050000 0.250000 0.050000
Ke 0.000000 0.000000 0.000000
Ni 1.000000
d 1.000000
illum 2
```

android.mtl

# Modèle 3D

## Complexité de la librairie

- Difficulté à manipuler des objets simples
- Afficher un objet complexe demande de savoir utiliser correctement la librairie
- La contrainte de temps m'aurait empêché de terminer toutes les fonctionnalités



# Problèmes

---

## Autres problèmes annexes

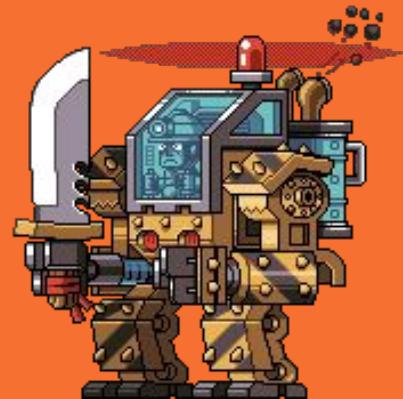
- Implémentation des boutons et des Layout
  - Dû au fait que SurfaceView est une classe **implémentée**
  - Layout des mini-jeux sont ... vides !
- MediaPlayer
- Gestion des hitbox à travers les moteurs
- Gestion des capteurs et traitement des données
  - Sensor Accéléromètre
  - Sensor Magnétomètre
- Problèmes liés aux images
  - Trop d'images ralentit le jeu
  - Images se s'affichant pas
- Chargement lent de la vue affichant le personnage



# 3.

## Améliorations envisagées

*Fonctionnalités  
supplémentaires*



# Améliorations

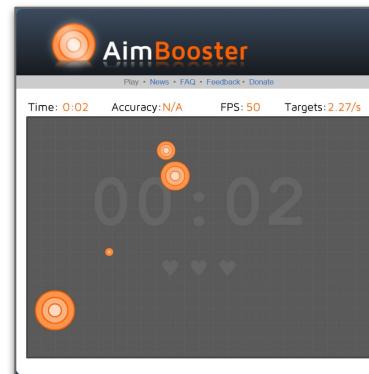
## Senseurs et mini-jeux

### Rééquilibrage des senseurs

- Utilisation du bon repère

### Implémentation supplémentaire :

- Nouveau mini-jeu
  - Accéléromètre ou non !



- Service de Step-Detector

Correction des bugs de d'actualisation de conteneurs

# Améliorations

---

## Choix du personnage



- ▶ Le joueur pourra choisir son personnage
- ▶ Le personnage correspondant s'affichera correctement

## Améliorations

---

### Observation du personnage



- ▶ Le personnage ne bouge plus automatiquement
- ▶ Le joueur peut faire tourner le personnage sur tous les angles

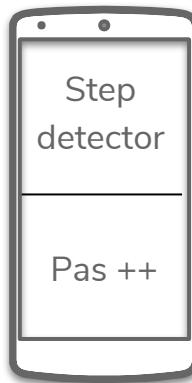
# Améliorations

---

## Expérience

### ↳ Plusieurs sources d'expérience

- ▶ Compteur de pas



1025 steps



+ 356 xp



Service

Compteur de pas

# Améliorations

## Objectif du joueur



Clicker Heroes

- ▶ Combat automatique
- ▶ Amélioration d'une statistique
- ▶ Combinaison des statistiques
- ▶ + en + difficile

# Conclusion

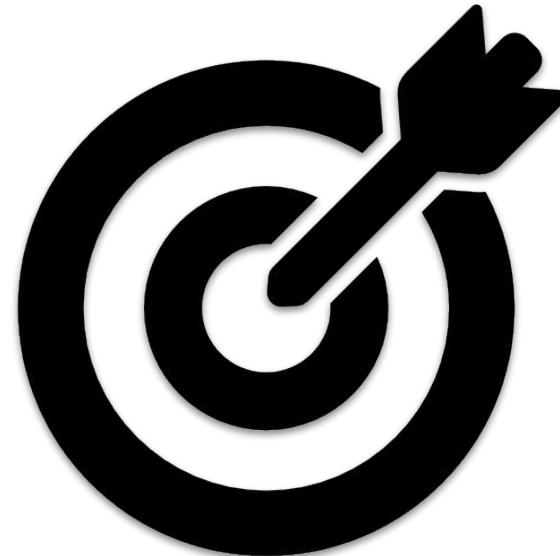
---

## Objectifs et résultats

Ensemble satisfaisant :

- Objectifs donnés → réussis

Malgré des bugs, application fonctionnelle



Merci de  
votre écoute

Avez-vous des questions ?

