Preuves

Je sais utiliser les Intent pour faire communiquer deux activités.

Le fichier GameOverDataBundle implémente Parcelable pour sauvegarder des données GameMaster.java instincie ce GameOverDataBundle à la ligne 285

Puis la communication se fait à la ligne 186 du fichier GameActivity.java GameOverActivity.java récupère ces données à la ligne 29

Je sais développer en utilisant le SDK le plus bas possible.

Nous avons réussi à descendre à 14

Je sais distinguer mes ressources en utilisant les qualifier

Les images sont dans le dossiers drawable, les fichiers « raw », c'est à dire fichier texte et musiques/sons sont dans le dossier raw, les layout dans les layout etc.

Je sais modifier le manifeste de l'application en fonction de mes besoins

Nous avons du modifier le fichier manifest pour par exemple ajouter des vues et modifier l'orientation de l'écran.

```
1  < activity android:name=".Activity.GameOverActivity"
2  android:screenOrientation="landscape"/>
3
4  < activity android:name=".Activity.GameActivity"
5  android:screenOrientation="landscape"/>
6
7  < activity android:name=".Activity.LoadingActivity"
8  android:screenOrientation="landscape"/>
9
10 < activity android:name=".MainActivity"
11 android:screenOrientation="landscape">
12
13 < intent-filter>
14 < action android:name="android.intent.action.MAIN" />
15 < category android:name="android.intent.category.LAUNCHER" />
16 < /intent-filter>
17 < /activity>
18 < activity android:name=".Activity$SettingsActivity"
19 android:parentActivityName=".MainActivity" />
```

Mais aussi pour ajouter une icone et changer le nom de l'apk

```
android:icon="@mipmap/ic_launcher"
android:label="@string/app_name"
android:roundIcon="@mipmap/ic_launcher_round"
```

Je sais faire des vues xml en utilisant layouts et composants adéquats

Nous utilisons beaucoup le constraint layout, qui permet de mettre nos canvas et le timer dans activity_game par exemple, la même situation est présente dans layout_gameover pour mettre en place l'animation de chute avec un canvac et afficher le temps, score, et la distance parcouru

Je sais coder proprement mes activités, en m'assurant qu'elles ne font que relayer les évènements

Par exemple dans GameActivity, les evenements du touchers sont envoyer au GameMaster

```
@Override
public boolean onTouch(View v, MotionEvent event) {
    surfaceView.getRootView().performClick();

    int maskedAction = event.getActionMasked();
    int pointerId = event.getPointerId(event.getActionIndex());

    switch (maskedAction) {
        case MotionEvent.ACTION_DOWN:
        case MotionEvent.ACTION_POINTER_DOWN:
        case MotionEvent.ACTION_MOVE: {
            MotionEvent.PointerCoords pointerCoords = new
                MotionEvent.PointerCoords
                try {
                    event.getPointerCoords(pointerId, pointerCoords);

                    fingerPoints[pointerId].x = pointerCoords.x /
                        WindowDefinitions.RESOLUTION_FACTOR;
                    fingerPoints[pointerId].y = pointerCoords.y /
                        WindowDefinitions.RESOLUTION_FACTOR;
                } catch (Exception e) {}
                gameManager.setPointsFingerPressed(fingerPoints);
                break;
        }
        case MotionEvent.ACTION_UP:
        case MotionEvent.ACTION_POINTER_UP: {
        // permet d eviter une collision accidentel
```

```
            fingerPoints[pointerId].x = -1;
            fingerPoints[pointerId].y = -1;

            gameManager.setPointsFingerPressed(fingerPoints);
            break;
        }
    }
    return true;
}
});
```

Je sais coder une application en ayant un véritable métier

Aucune vue n'est présente dans le metier, et plusieurs métiers sont disponibles, le principal étant GameMaster. Mais il y a aussi GameOverMaster

Je sais parfaitement séparer vue et modèle

Les vues sont dans le package Activity, et les modeles dans le package Game. Aucune vue n'est présente à l'intérieur du modèle et l'inverse est vrai aussi. La vue ne fait qu'envoyer les événements au modéle lui correspondant et le modèle traite ces données là. Comme données, il y'a notemment les événements liés aux chagement de positions des doigts, à savoir si onStop, onPause, onResume sont appelés pour par exemple couper la musique, libérer des ressources etc

Je maîtrise le cycle de vie de mon application

Dans le fichier GameActivty, il y a onPause qui permet de mettre en "pause" le jeu, pour ne pas mettre à les éléments à jour et mettre en pause la musique, onResume qui permet de réprendre la mise à jour des éléments du jeu et remettre la musique. Finalement quand onStop est appelée, on gére la libération de la mémorie et la désintaciations des éléments qui ne servent plus.

```
@Override
public void onStop() {
    super.onStop();
    gameManager.stopUpdate();
    gameManager.kill();

    try {
        gameManager.interrupt();
        gameManager.join();
    } catch (InterruptedException ignore) {
    }

```

```
13      launchLoseActivity(GameMaster.registerDataBundle);
14  }
15
16  @Override
17  public void onPause() {
18      super.onPause();
19      instanceOnPause = true;
20      gameManager.pauseUpdate();
21  }
22
23  @Override
24  public void onResume() {
25      super.onResume();
26      instanceOnPause = false;
27      gameManager.resumeUpdate();
28  }
29
30  @Override
31  public void onConfigurationChanged(Configuration newConfig) {
32      super.onConfigurationChanged(newConfig);
33
34      WindowDefinitions.HEIGHT =
            WindowUtil.convertDpToPixel(getApplicationContext(),
            newConfig.screenHeightDp) /
            WindowDefinitions.RESOLUTION_FACTOR;
35      WindowDefinitions.WIDTH =
            WindowUtil.convertDpToPixel(getApplicationContext(),
            newConfig.screenWidthDp) /
            WindowDefinitions.RESOLUTION_FACTOR;
36
37      gameManager.updatePoolPoints();
38  }
```

Je sais utiliser le findViewById à bon escient

Nous utilisons les findViewById notemment dans GameOverMaster pour mettre à jour les textview permettant d'afficher le temps du joueur, son score et sa distance parcourue

```
1  TextView timer = findViewById(R.id.textview_gameover_time);
2  TextView distance =
       findViewById(R.id.textview_gameover_distance_player);
3  TextView score = findViewById(R.id.textview_gameover_score);
4
5  .....
6
```

```
timer.setText(dataBundle.getTimer());
distance.setText(String.format("%s%%",
    Util.roundFloatNDigits(percentage, 2)));
score.setText(String.valueOf(dataBundle.getScore()));
```

Je sais gérer les permissions dynamiques de mon application

Comme notre application est un jeu qui ne demande en rien d'acceder à des fonctions spécifiques du téléphone, nous n'en avons pas utiliser. Néamoins, pour en ajouter il suffit de mettre dans le fichier manifest, de préférence au début:

```
< uses-permission android:name="android.permission.NOMPERMISSION"/>
```

Pour internet, ce serait:

```
< uses-permission android:name="android.permission.INTERNET"/>
```

Pour vérifier que l'application a bien le droit d'acceder à internet:

```
if (ContextCompat.checkSelfPermission(thisActivity,
    Manifest.permission.INTERNET) !=
    PackageManager.PERMISSION_GRANTED) {
    // l application n'a pas accés à internet
}
```

Je sais gérer la persistance légère de mon application

Nous avons utiliser les shared preferences afin de garder en mémoire le nom du joueur. Le nom du joueur se rentrant avant le jeu, il est donc stocké dans les shared preferences et récupéré dans la gameOverActivity afin d'afficher à la fin de la partie le nom du joueur. Pour stocker dans les shared preferences :

```
SharedPreferences.Editor editor =
    getSharedPreferences("shared_pref_user", 0).edit();
editor.putString("username",u.getPseudo());
```

Pour récupérer la valeur stockée :

```
SharedPreferences prefs = getSharedPreferences("shared_pref_user",
    0);
String userName = prefs.getString("username", null);
```

Et pour finir, pour l'afficher sur l'écran lors du Game Over :

```
TextView showUsername = findViewById(R.id.textview_username);
showUsername.setText(userName);
```

Je sais gérer la persistance profonde de mon application

Un niveau qui est inclus de base dans le jeu, dans res/raw/level.txt qui contient toutes les informations du niveau. Il est chargé grace au LevelLoader qui se trouve dans Game/Level/Loader

Je sais afficher une collection de données

Nous n'avons pas eu besoin d'afficher une collection de données, néamoins pour un faire un il faut:

```
ArrayAdapter<String> itemsAdapter = new ArrayAdapter<String>(this,
    android.R.layout.simple_list_item_1, items);
```

Puis pour mettre l'itemsAdapter dans une liste:

```
ListView listView = (ListView) findViewById(R.id.listview);
listView.setAdapter(itemsAdapter);
```

Je sais coder mon propre adaptateur

Nous n'avons pas eu besoin de faire notre propre adapteur Mais pour un faire un, il faut tout d'abord faire un modèle, qui va répertorier les différents attributs que l'on veut afficher. Pour ne pas surcharger l'exemple, il n'y aura pas de getter et setter, mais bien évidemment, il en faut si on veut faire un code propre. Par exemple:

```
class Model {
    public int variable1;
  public String variable2;

    public Model(int a, String b) {
        variable1 = a;
        variable2 = b;
    }
}
```

Puis viens le "view template", c'est à dire ce qu'on veut afficher et comment on va les afficher, que l'on va mettre dans le dossier layout de resources et qu'on va appeler adapter_view

```
< LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent" >

< TextView
```

```
    android:id="@+id/var1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Variable 1" />

< TextView
    android:id="@+id/var2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Variable 2" />

</LinearLayout>
```

Puis finalement créer la classe qui va hériter de ArrayAdapter

```
public class ModelAdapter extends ArrayAdapter<Model> {

    public ModelAdapter(Context context, ArrayList<Model> models) {
        super(context, 0, users);
    }

    @Override
        public View getView(int position, View convertView,
            ViewGroup parent) {
            Model model = getItem(position);

            if (convertView == null) {
                convertView =
                    LayoutInflater.from(getContext()).inflate(R.layout.adapter_view,
                    parent, false);
            }

            TextView var1 = (TextView)
                convertView.findViewById(R.id.var1);
            TextView var2 = (TextView)
                convertView.findViewById(R.id.vae2);


            var1.setText(String.valueOf(model.variable1);
            var2.setText(model.variable2);

            return convertView;
        }
}
```

Maintenant il faut attacher l'adaptateur à une listeview qu'on a définit dans une activity (celle où on veut afficher la liste appelé ici listeview_items)

```
ArrayList<Model> models = new ArrayList<Model>();

ModelAdapter adapter = new ModelAdapter(this, models);

ListView listView = (ListView) findViewById(R.id.listview_items);
listView.setAdapter(adapter);
```

Je maîtrise l'usage des fragments

Il n'y a pas eu besoin de l'utilisation de fragments dans le projet, néamoins voici comment en faire: Il faut d'abord créer un layout, celui que l'on veut, puis creer une classe qui va etendre de fragment

```
public static class UnFragment extends Fragment {

    @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup
            container,
                Bundle savedInstanceState) {
            return inflater.inflate(R.layout.fragment_layout,
                container, false); // penser au petit interrupteur
                rouge
        }
}
```

Puis, dans le layout où l'on veut utiliser le fragment:

```
< fragment android:name="iut.ipi.runnergame.UnFragment"
android:id="@+id/unfragment"
android:layout_width="match_parent"
android:layout_height="match_parent" />
```

Je maîtrise l'utilisation de GIT

Nous avons essayés de faire le plus de branches possibles, tout en évitant de push sur le master (sauf de toutes petites modifications testées et retéstées ou pour les fichiers de Documentation) Nous avons aussi commit à chaque fois dans la bonne branche les modifications apportés, et ceux réguliérement pour avoir un bon suivi de projet et pouvoir revenir quand on le souhaitait à une version antérieur.

Application

Je sais utiliser l'accéléromètre

Le fichier s'occupant de l'accéléromètre se trouve dans Engine/Sensor/Accelerometer.java