**Preuves :**

*Je sais utiliser l’accéléromètre :*

* Fonctionnalité principale de notre jeu
* public class GameActivity extends Activity implements SensorEventListener

*Je sais utiliser les Intent pour faire communiquer deux activités :*

* Intent intent = new Intent(this, MainActivity.class);  
  intent.putExtra(*NAME\_PLAYER*, name.getText().toString());  
  intent.putExtra(*NUMBER*,number\_score);  
  startActivity(intent);
* Cf : Class LoseActivity

*Je sais développer en utilisant le SDK le plus bas possible :*

* minSdkVersion 15
* Cf : build.gradle (module ‘app’)

*Je sais modifier le manifeste de l’application en fonction de mes besoins :*

* <activity android:name="com.android2019.spaceinvaderchinesebootleg.Activity.ScoreActivity"  
   android:parentActivityName="com.android2019.spaceinvaderchinesebootleg.Activity.MainActivity"/>
* Ajoute une nouvelle activité (ScoreActivity) et définit MainActivity comme son activité parent, afin de pouvoir y retourner via la flèche de la Toolbar

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

* Le xml activity\_highscore.xml est la fenêtre permettant de rentrer son High Score. Elle utilise une Constraint Layout et 3 composants différents qui sont tous bien liés au Constraint Layout et rend la fenêtre adaptative. Ces 3 composants sont : TextView, Button, EditText
* Le xml activity\_lose.xml quant à lui utilise un Linear Layout et 2 composants différents (Button, TextView)

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

* Les 4 activités ne relayent que des évènements, ou instancie un Manager dans leurs onCreate(), elles sont donc proprement codées et transmette les informations nécessaires au Manager
* Cf : Class MainActivity, Class GameActivity, Class LoseActivity, Class ScoreActivity

*Je sais coder une application en ayant un véritable métier :*

* Les classes métier sont instanciés par le Manager nécessaire. Par exemple la Class Score permet de sauvegarder un score (Entier + String nom du joueur). Une liste de Score est donc créée pour être affiché.
* Cf : Class Player, Class Enemy, Class Shot, Class Score

*Je maîtrise le cycle de vie de mon application :*

*&&*

*Je sais gérer la persistance légère de mon application :*

* protected void onCreate(Bundle savedInstanceState) {  
   if(savedInstanceState != null){  
   is\_highscore = savedInstanceState.getBoolean(*IS\_HIGHSCORE*);  
   name\_player = savedInstanceState.getString(*NAME\_PLAYER*);  
   }  
   setGoodContent();  
  }  
    
  @Override  
  public void onSaveInstanceState(Bundle savedInstanceState) {  
   savedInstanceState.putBoolean(*IS\_HIGHSCORE*, is\_highscore);  
   if(name != null) savedInstanceState.putString(*NAME\_PLAYER*, name.getText().toString());  
   super.onSaveInstanceState(savedInstanceState);  
  }  
    
  @Override  
  public void onRestoreInstanceState(Bundle savedInstanceState) {  
   super.onRestoreInstanceState(savedInstanceState);  
   is\_highscore = savedInstanceState.getBoolean(*IS\_HIGHSCORE*);  
   name\_player = savedInstanceState.getString(*NAME\_PLAYER*);  
  }
* La fenêtre correspondante est celle qui s’affiche lorsque l’on a perdu, elle permet notamment de rentrer son nom pour enregistrer son score, ou alors de recommencer une partie.
* Les constantes sont des clés, nécessaire au bundle (IS\_HIGHSCORE et NAME\_PLAYER)
* Ainsi, si on retourne le téléphone après avoir tapé notre nom, il n’est pas effacé.
* Intent intent = new Intent(this, MainActivity.class);  
  intent.putExtra(*NAME\_PLAYER*, name.getText().toString());  
  intent.putExtra(*NUMBER*,number\_score);
* De plus une fois que le joueur a écrit son nom et cliqué sur ‘Submit’, j’envoie ce score à l’activité principale, pour qu’elle puisse ensuite le transmettre à l’activité gérant le score pour pouvoir l’enregistrer/
* Cf : Class LoseActivity

*Je sais parfaitement séparer vue et modèle :*

* Les activités relayent des événements et instancie des managers (expliqué précédemment).
* La vue personnalisé AnimatedView ne fait qu’afficher les objets que l’on lui demande d’afficher (coordonné par la Class GameLoop qui notifie le manager puis la vue afin qu’ils se mettent à jour).
* Cf : Class AnimatedView

*Je sais gérer la persistance profonde de mon application :*

* Le ScoreManager (appelé par l’activité ScoreActivity) va créer une Class qui va se charger d’enregistrer les scores. ScoreManager n’est pas dépendant du type de sauvegarde et se contente d’appeler les bonnes méthodes au bon moment
* Le Sauveur ‘SaveIntoFile’ est une class héritant d’une class abstraite ‘Save’. Cela me permet de référencer le sauveur en tant que ‘Save’ et donc pouvoir créer plusieurs types de sauveurs différents sans problèmes (respect pattern)
* SaveIntoFile va donc sauvegarder la liste des scores dans un fichier, ou alors lire ce même fichier pour en sortir une liste de scores. Le parser est unique, créée pour l’occasion.
* Cf : Class ScoreManager, Class SaveIntoFile

*Je maîtrise l’utilisation de Git :*

* Je prends pour preuve notre dépôt sur la forge
* Les commandes basiques (add, rm, clone, checkout, ignore, commit et push) sont comprises et maitrisés.

*Je sais utiliser le findViewById à bon escient :*

* Les événements liés aux Boutons sont gérés via une référence dans le xml vers la méthode (listener) dans l’activité.
* J’utilise plutôt le findViewById pour récupérer des composants comme EditText, ou ListView
* name = (EditText) findViewById(R.id.*name*);
* Cf : Class LoseActivity
* ListView lv = findViewById(R.id.*list\_scores*);
* Cf : Class ScoreActivity

*Je sais afficher une collection de données :*

*&&*

*Je sais coder mon propre adaptateur :*

* La fenêtre xml menu\_scores.xml comporte une ListView qui va être chargé d’afficher la liste des scores
* Dans l’activité du score, je définis un ListAdapter et récupère la ListView afin de lui donner nos données à afficher (en l’occurrence ici, la liste des scores)
* ListAdapter la;
* la = new ListAdapter(this,R.layout.*layout\_listview*,scoreManager.getScores());
* ListView lv = findViewById(R.id.*list\_scores*);
* lv.setAdapter(la);
* La Class ListAdapter est un adapter personnalisé, notre adapter, utilisant le layout layout\_listview.xml pour sa mise en page
* Cf : Class ScoreActivity, Class ListAdapter