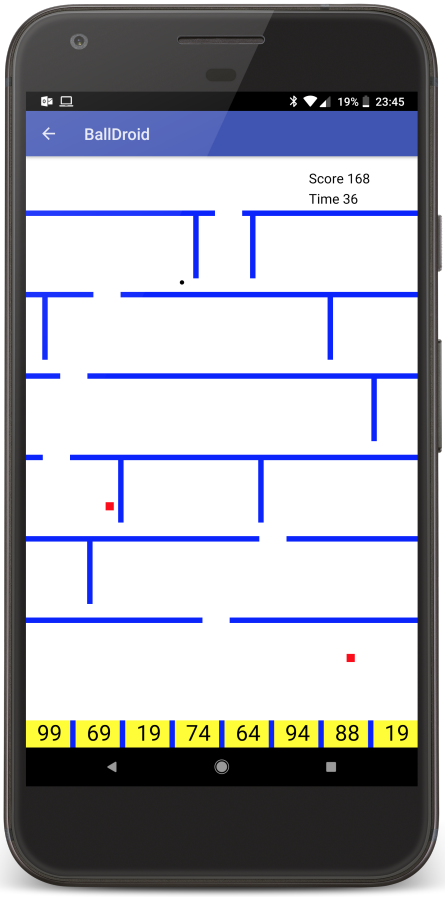
BallDroid - Android Project

Maxime Lovino

5 janvier 2018



1 Introduction

BallDroid est un jeu Android tirant parti de l'accéléromètre présent dans la plupart des appareils mobiles pour diriger une balle qui tombe à travers des obstacles en vue d'arriver dans des zones définies par des points et empocher le plus de points possibles dans un temps limité. Au cours du jeu, le joueur peut récupérer des bonus et des malus qui lui permettront de gagner ou perdre du temps respectivement.

2 Structure et spécifications de l'application

L'application a été développée et testée principalement sur Android 8.1 (API 27) sur un Google Pixel XL avec une résolution d'écran de 2560x1440. La version minimale de SDK supportée est la version 24 (Android 7.0). L'application a également été testée via l'émulateur sur les version de 24 à 27 de l'API et des résolutions de 1920x1080 à 2880x1440.

L'application est composée de 4 activités : un écran d'accueil, le jeu, une vue des meilleurs scores et une page "A propos".

3 Main Activity

L'activité principale est assez simple et va se contenter de servir de point d'entrée au jeu, un menu a été réalisé pour choisir la difficulté, qui est définie dans l'énumération `DifficultyLevel`. Au moment de lancer le jeu via le bouton sur l'écran d'accueil, la difficulté est passée dans l'Intent en tant qu'extra en passant l'ordinal de la valeur choisie dans l'enum.

3.1 Menu

Le menu est défini dans un fichier XML `activity_main_menu.xml` et comprend les 3 niveaux de difficultés sous la forme de Radio Buttons et une entrée pour accéder à la page "A propos". Par défaut le niveau de difficulté est le niveau le plus simple. Le niveau de difficulté choisi par l'utilisateur est stocké dans les `SharedPreferences` Android sous la forme de l'ordinal de la difficulté choisie dans l'enum pour pouvoir être persistant pour l'application installée..

4 High Score Activity

L'activité High Score affiche tous les scores enregistrés dans le jeu dans l'ordre décroissant avec leur niveau de difficulté dans une `ListView` en utilisant un `ArrayAdapter` à partir d'un `ArrayList` contenant les scores. Le layout `score_list_item.xml` définit une entrée de la liste.

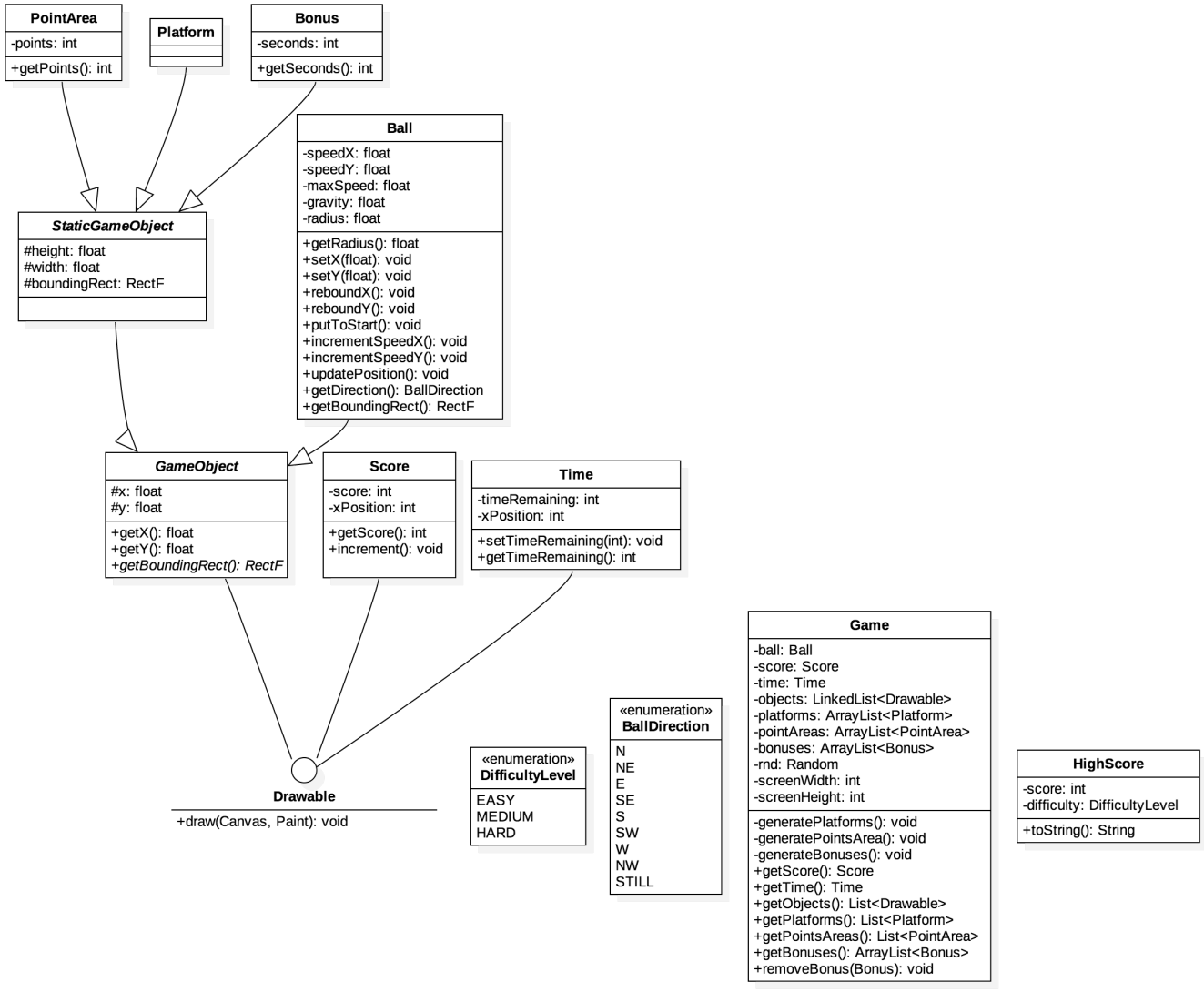
4.1 Stockage des high scores

Le stockage de ces scores est réalisé par une base de données SQLite dans l'application (pratique commune pour les applications Android).
La structure de la base de données est de sa table est définie dans le package `ch.hepia.lovino.balldroid.models.db`. Il s'agit d'une table contenant un index auto-généré, le score et le niveau de difficulté. Au moment de l'affichage de l'activité High Score, un SELECT est réalisé dans la base de données en triant par score descendant.

5 Game Activity - Le jeu

L'activité principale de l'application est l'activité contenant le jeu, celle-ci est entièrement vide de base en terme de layout et est entièrement générée au runtime. En terme de structure, j'ai décidé d'utiliser un modèle MVC bien que pas 100% strict par rapport à la définition MVC comme on pourra le voir. Etant donné que les modèles contiennent un petit peu de logique et de rendu pour simplifier les choses et éviter d'écrire 3 fois les classes pour rien, étant donné l'ampleur réduite du projet.

5.1 Structures de données - Model



En terme de modèles, j'ai réalisé une architecture tirant avantage de la POO et de l'héritage pour regrouper et catégoriser les différents éléments nécessaires pour le jeu.

5.1.1 L'interface Drawable

Ici est contenu en fait la plus grande digression par rapport au modèle MVC, l'interface `Drawable` qui définit une fonction `draw(Canvas,Paint)` et qui est implémentée par tous les modèles qui seront affichés à l'écran. Cela permet

comme on le verra plus tard de passer à la vue une liste de `Drawable` et on pourra simplement appeler `draw` sur tous ces éléments. Si nous voulions coller au plus proche du MVC, il aurait fallu réaliser une classe de rendu pour chaque modèle et implémenter cette interface sur ceux-ci, mais cela était plus pratique de cette façon dans ce cas.

5.1.2 Score et Time

Score et Time sont deux classes qui définissent les deux textes en haut de l'écran affichant le score et le temps restant, ils implémentent l'interface `Drawable`

5.1.3 GameObject

A partir de là, `GameObject` est une classe abstraite qui définit une position (x,y) pour un élément de jeu. On définit également une méthode abstraite pour récupérer la bounding box rectangle autour de l'élément qui sera définie par les sous classes.

5.1.4 Ball

La balle hérite de `GameObject` et y ajoute un rayon, ainsi qu'une vitesse sur chaque axe, sa force de gravité, sa vitesse maximale (ces deux dernières dépendent de la difficulté) et des constantes pour les forces de frottement et les positions de départ. Ensuite, on définit des méthodes permettant de déplacer la balle, augmenter sa vitesse, la faire rebondir (inverser sa vitesse en la diminuant) et récupérer sa direction sous la forme d'une énumération BallDirection qui est définie comme les 8 directions cardinales ou la balle à l'arrêt. Cela va nous servir pour la gestion des collisions. On implémente également la méthode pour créer un rectangle autour de la balle et le retourner pour gérer ses collisions.

5.1.5 StaticGameObject

La balle étant le seul élément du jeu qui bouge, j'ai créé une classe abstraite `StaticGameObject` qui hérite de `GameObject` pour définir un élément fixe, tous ces éléments sont définis comme des rectangles, donc cette classe ajoute des champs de largeur et hauteur à la classe de base, ainsi qu'une implémentation de la méthode pour récupérer la bounding box, qui sera la même pour toutes les sous-classes étant donné que tout sera un rectangle désormais.

5.1.6 Platform

Je n'ai pas distingué la plateforme et le mur dans mon jeu, les deux sont définis comme des plateformes, leur taille et leur position sert à les distinguer, et au niveau de leur physique ils sont gérés de la même façon de toute façon.

5.1.7 Bonus

Un bonus contient un certain nombre de secondes (positif ou négatif) à ajouter au temps restant du jeu, la seule différence entre un bonus est sa couleur verte ou rouge en fonction des secondes positives ou négatives.

5.1.8 PointArea

PointArea définit une zone d'arrivée, elle est définie par un nombre de points qu'elle rapporte au joueur.

5.1.9 Game

Game est en fait le modèle principal qui est appelé par le moteur de jeu. Le plateau de jeu, ses plateformes, ses bonus et ses zones d'arrivée sont générés aléatoirement dans cette classe lors de la création d'une instance de Game. Le plateau s'adapte à la résolution par exemple en définissant plus "d'étages" de plateformes si la résolution verticale est plus élevée.

Tous les éléments du plateau sont placés dans deux listes à chaque fois, une liste contenant tous les éléments `Drawable` du jeu (récupérée par la vue) et une liste contenant tous les éléments de leur type (une pour les bonus, une pour les plateformes, une pour les zones d'arrivées) qui seront récupérées par le moteur de jeu pour vérifier les collisions et la victoire de points.

5.2 Le rendu - View

Je ne vais pas m'attarder plus en profondeur sur la partie vue, car elle suit ce qui était expliqué dans le PDF fourni, on a donc une `GameSurfaceView` avec un thread d'affichage `DrawingThread` qui à chaque frame prend un lock sur le canvas et appelle `onDraw`. Dans cette méthode, on appelle la méthode pour que le moteur effectue les mouvements et ensuite on récupère les éléments `Drawable` à afficher et on appelle `Draw()` sur ceux-ci.

5.3 Le moteur de jeu - Controller

5.3.1 La fonction update

5.3.2 Les collisions en fonction de la direction de la balle

5.3.3 Prise de bonus

5.3.4 La gestion des pauses

5.3.5 Timer thread et fin de jeu

6 Conclusion