Développement pour mobiles - Space Dynamics

Jean-Emile PELLIER, M1 informatique

9 novembre 2018

Résumé

Space Dynamics est un jeu vidéo pour mobiles qui consiste à déplacer des balles à travers la galaxie afin qu'elles puissent atteindre leur destination le plus rapidement possible en évitant différents obstacles.

Ce jeu de stratégie au gameplay volontairement simpliste s'est finalement révélé particulièrement complexe au cours de sa conception. En particulier, le choix d'utiliser des cercles pour tous les objets visibles aurait dû simplifier la conception mais il a fallu que ceux-ci la compliquent en devenant des ellipses du point de vue du mécanisme de détection des collisions.

Dans ce document, nous allons étudier les fonctionnalités et l'implémentation de ce jeu.

Introduction

Space Dynamics est un jeu vidéo conçu spécifiquement dans le cadre de ce projet. Il a été pensé pour être aussi simple que possible afin d'éviter de s'éterniser sur des mécanismes de jeu excessivement complexes au profit d'une utilisation plus poussée du potentiel matériel de l'appareil mobile.

Le jeu consiste à déplacer des balles colorées via l'inclinaison de l'appareil mobile jusqu'à leur destination de même couleur en allant le plus vite possible. En effet, un temps est affiché au-dessus de la partie courante, il va correspondre au score associé à la partie dès lors qu'elle sera terminée.

Plusieurs stratégies permettent de terminer une partie mais certaines d'entre elles sont plus rapides à mettre en oeuvre que d'autres, elles seront à privilégier dans la perspective d'obtenir un bon score.

Le contexte global se déroule dans l'espace, l'objectif étant de se servir de la richesse des comportements des objets célestes pour bâtir un univers très dynamique d'où le nom du jeu.

L'ambiance générale se veut apaisante, elle est renforcée par une musique d'arrière-plan relaxante.

Dans cette version on ne va considérer que 4 entités distinctes :

- les balles jouables : contrôlables par le joueur via l'inclinaison de son appareil mobile
- les cercles de destination : permettant de terminer un niveau de jeu par la capture des balles
- les trous noirs : ramenant les balles à leur position initiale pour ralentir le joueur
- les trous blancs : repoussant les balles au loin pour les éloigner de leur objectif.

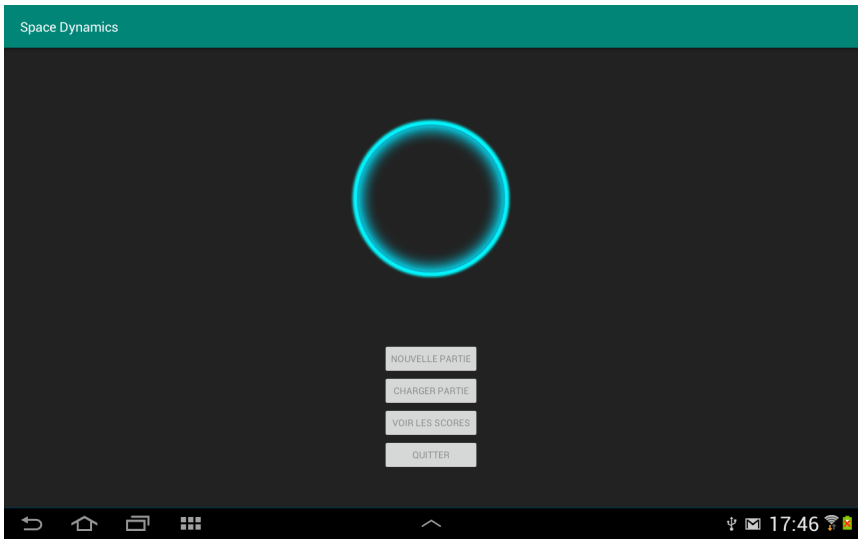
Les balles jouables ne peuvent pas sortir de l'écran.

Comment ce jeu vidéo mobile a-t-il été implémenté ?

Etudions en détails les fonctionnalités et l'implémentation de ce jeu.

1 Description générale de l'application

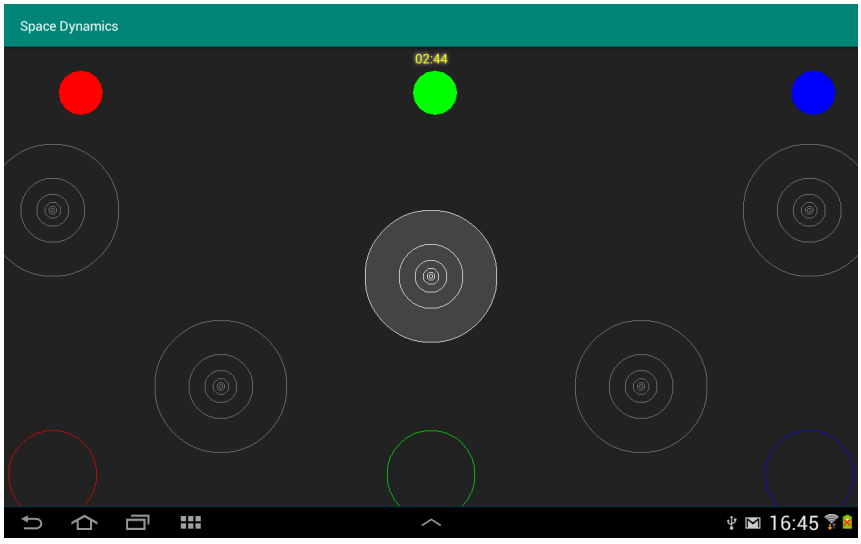
1.1 Menu principal



Le menu principal offre au joueur la possibilité de :

- démarrer une nouvelle partie
- charger une partie enregistrée
- consulter la liste des scores enregistrés
- quitter l'application

1.2 Interface de jeu



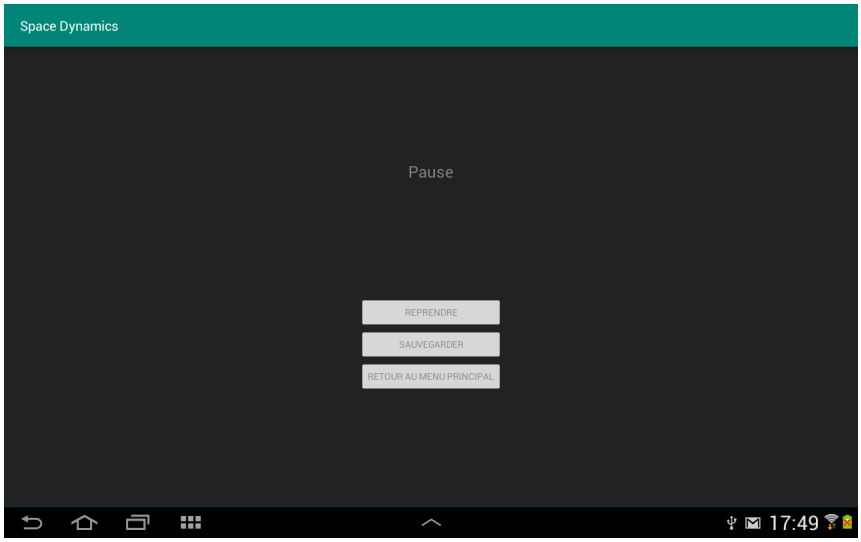
L'interface de jeu propose de :

- jouer à Space Dynamics
- déplacer les balles selon l'inclinaison de l'appareil mobile
- faire une pause via un double tap

Dans cette capture d'écran on peut voir 3 balles jouables l'une rouge, une autre verte et une dernière bleue en haut de l'écran, accompagnées de leur cercles de destination en bas de l'écran.

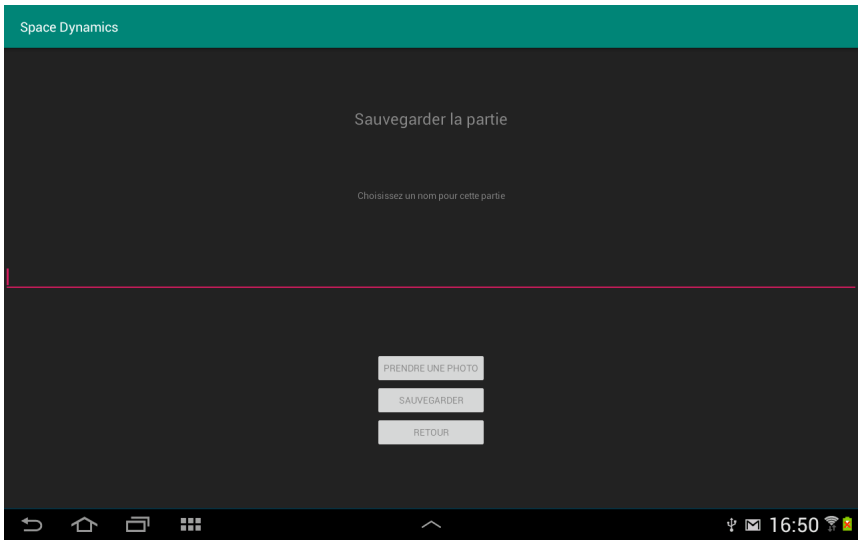
Quant aux cercles concentriques foncés et clairs, ils matérialisent respectivement les trous noirs et les trous blancs.

1.3 Menu pause



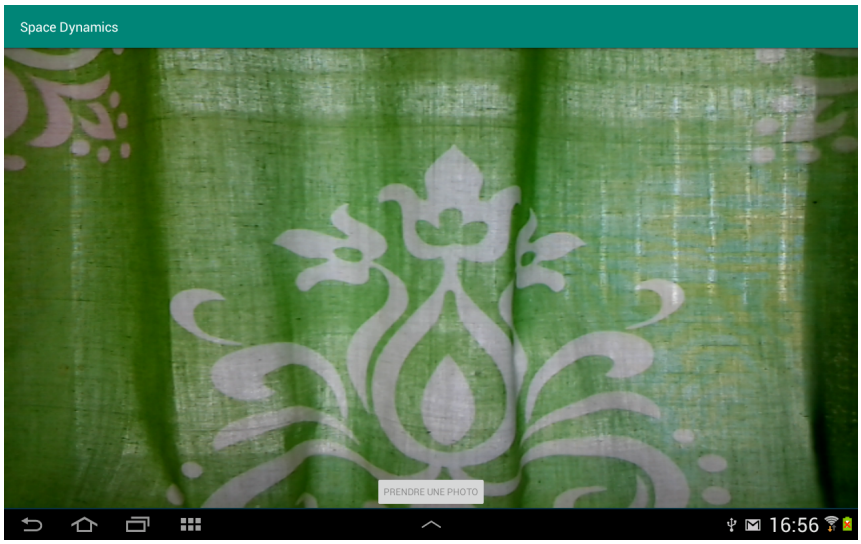
Le menu pause propose au joueur de :
- reprendre sa partie
- sauvegarder l'état actuel de sa partie
- retourner au menu principal

1.4 Menu sauvegarder



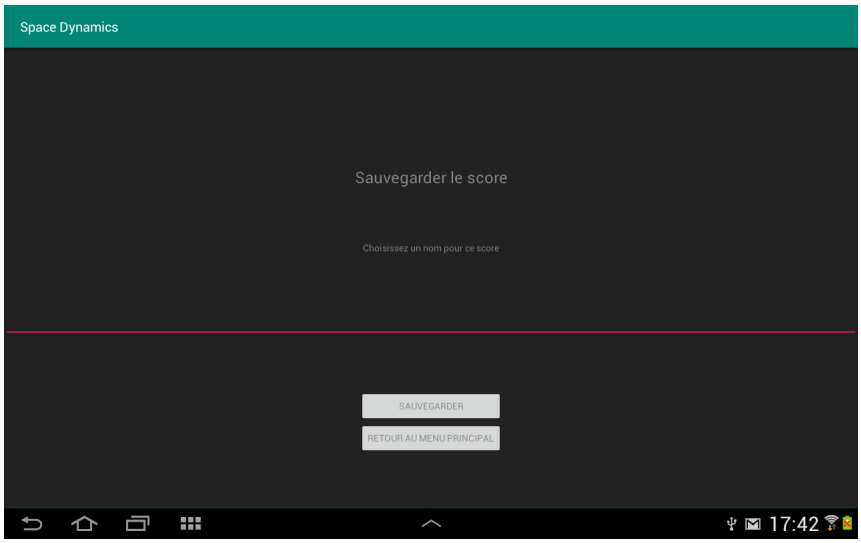
Le menu de sauvegarde propose de :
- nommer la partie en cours
- prendre une photo pour l'associer à la partie
- sauvegarder la partie en cours
- retourner au menu pause

1.5 Menu caméra



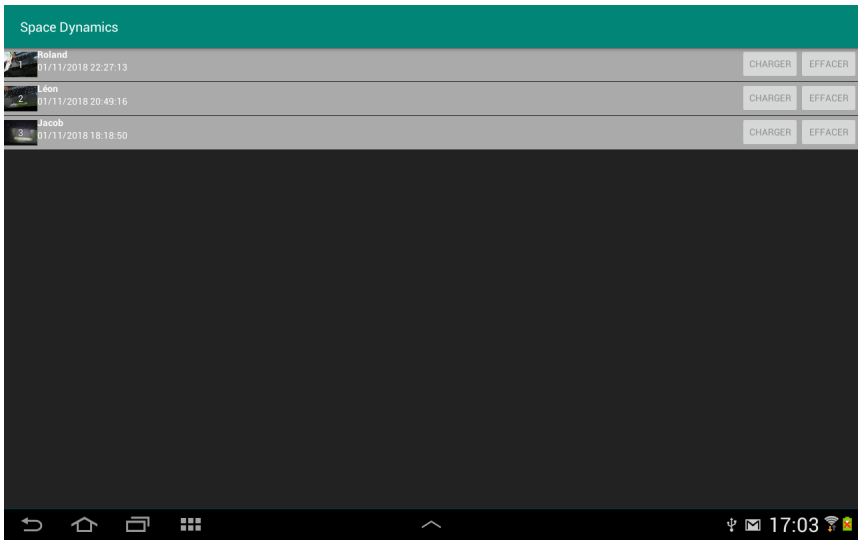
Le menu caméra propose de :
- prendre une photo via un bouton approprié
- prévisualiser l'environnement pour obtenir une photo de qualité optimale

1.6 Menu fin de partie



Le menu de fin de partie propose de :
- nommer le score
- sauvegarder le score du joueur
- retourner au menu principal

1.7 Menu chargement des sauvegardes



Le menu de chargement des sauvegardes propose de :

- consulter les parties sauvegardées ainsi que les photos associées à chaque sauvegarde
- charger une partie sauvegardée via le bouton charger
- effacer une partie sauvegardée via le bouton effacer

Les sauvegardes sont présentées dans une liste classée par ordre chronologique décroissant.

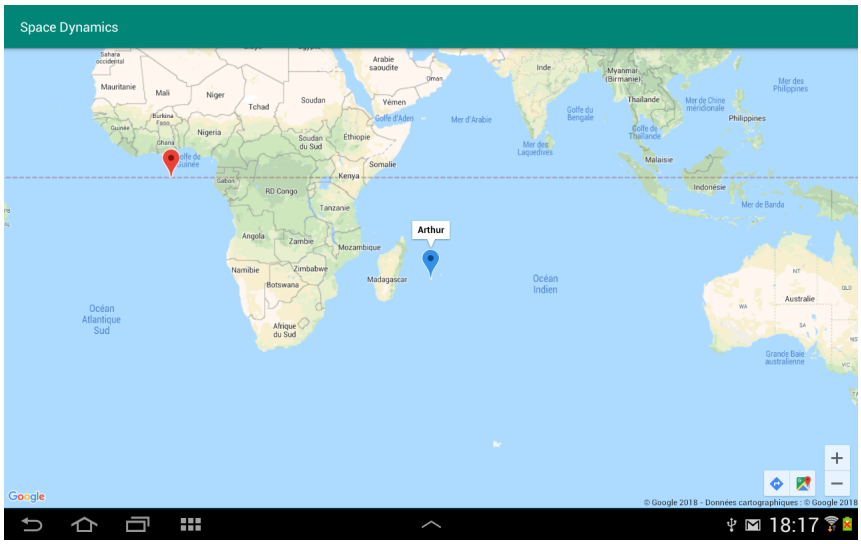
1.8 Menu affichage des scores

Le menu d'affichage des scores propose de :

- afficher les scores enregistrés
- localiser un score sur une carte via le bouton localiser
- effacer un score via le bouton effacer

Les scores sont présentés dans une liste classée d'abord par ordre croissant des scores puis par ordre alphabétique sur les noms associés aux scores.

1.9 Menu localisation

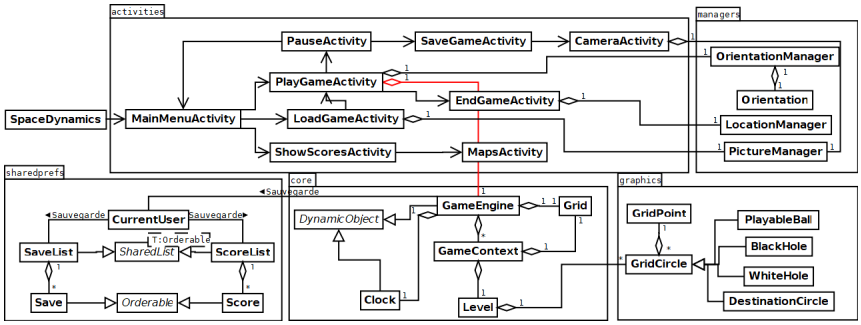


Le menu localisation propose de :
- situer en bleu le score d'un joueur sélectionné depuis le menu d'affichage des scores
- situer en rouge tous les autres scores

2 Architecture du code

2.1 Android

2.1.1 Diagramme de classes



Le diagramme de classe met en évidence 5 packages que nous allons détailler. Il ne permet pas de représenter tous les fichiers du projet mais donne une vision d'ensemble des interactions entre les classes, nous détaillerons également ces fichiers non représentables.

SpaceDynamics Une classe représentant l'application servant essentiellement à partager le contexte d'application à toutes les autres classes via un attribut statique.

Tools Le fichier Tools.kt contient un ensemble d'outils génériques destinés à toutes les classes.

2.1.2 Les activités

MainMenuActivity L'activité qui gère le menu principal est constituée de boutons permettant de :
- démarrer une nouvelle partie (via PlayGameActivity)

- charger une partie existante (via LoadGameActivity)
- consulter la liste des scores (via ShowScoresActivity)
- mettre fin à l'application

PlayGameActivity L'activité qui gère l'interface de jeu via la vue GameBoardView traite la totalité des mécanismes de jeu en utilisant :
- GestureDetectorCompat pour le double tap déclenchant le menu pause (via PauseActivity)
- OrientationManager pour les changements d'inclinaison
- MediaPlayer pour la musique d'arrière-plan
- GameEngine pour la supervision des évènements de jeu

PauseActivity L'activité qui gère le menu pause est constituée de boutons permettant de :
- reprendre la partie suspendue (via l'arrêt de l'activité courante)
- sauvegarder la partie suspendue (via SaveGameActivity)
- retourner au menu principal (via MainMenuActivity et l'arrêt des activités précédentes)

SaveGameActivity L'activité qui gère la sauvegarde de la partie est constituée d'un champ de texte permettant de modifier le nom de la partie courante et de boutons permettant de :
- prendre une photo (via CameraActivity)
- sauvegarder la partie
- retourner au menu pause (via l'arrêt de l'activité courante)

CameraActivity L'activité qui gère la prise de photo est constituée d'une surface de prévisualisation et d'un bouton permettant de prendre une photo. Une fois la photographie prise, celle-ci est enregistrée et référencée par PictureManager puis l'activité se termine.

EndGameActivity L'activité qui gère la sauvegarde des scores une fois la partie terminée est constituée d'un champ de texte permettant de nommer le score et de boutons permettant de :
- sauvegarder le score
- retourner au menu principal (via MainMenuActivity et l'arrêt des activités précédentes)

Pour la sauvegarde du score, l'activité récupère la position de l'utilisateur via GeolocationManager.

LoadGameActivity L'activité qui gère le chargement des sauvegardes va les présenter dans une liste, classée par ordre chronologique décroissant puis alphabétique sur les noms, qu'elle récupère à partir du singleton SaveList. Chacune des entrées est constituée d'un rang (à gauche), surplombant éventuellement une photographie, et de deux boutons à droite, l'un pour charger la partie (via PlayGameActivity), l'autre pour la supprimer.

ShowScoresActivity L'activité qui gère l'affichage des scores va les présenter dans une liste, classée par ordre croissant selon les scores puis par ordre alphabétique sur les noms, qu'elle récupère à partir du singleton ScoreList. Chacune des entrées est constituée d'un rang (à gauche) et de deux boutons à droite, l'un pour localiser le score (via MapsActivity), l'autre pour le supprimer.

MapsActivity L'activité qui gère la représentation de la position géographique des scores sur une carte va représenter en bleu le score choisi depuis le menu d'affichage des scores et tous les autres scores en rouge. Les marqueurs sont nommés d'après les noms associés aux scores et la caméra est centrée sur le score sélectionné préalablement.

2.1.3 Les éléments graphiques

GridCircle Un cercle de grille est défini par une position dans une grille, une couleur et un rayon. Il définit une entité abstraite pour un objet circulaire qui peut être dessiné dans une grille.

PlayableBall Une balle jouable est un cercle de grille conçu pour être déplacé, il a une position initiale, un état de verrouillage et un état d'achèvement. Ces paramètres permettent respectivement de réinitialiser sa position, de la verrouiller pour effectuer des animations, et valider l'atteinte de sa destination.

DestinationCircle Un cercle de destination est un cercle de grille correspondant à la destination d'une balle jouable, il va servir à capturer les balles de la même couleur que lui.

BlackHole Un trou noir est un cercle de grille qui réinitialise la position des balles jouables.

WhiteHole Un trou blanc est un cercle de grille qui repousse les balles jouables.

2.1.4 Le coeur

DynamicObject Un objet dynamique est une entité abstraite assurant l'autonomie des objets instanciés par ses classes filles via un thread dédié et un mécanisme du contrôle du cycle de vie.

Clock Une horloge est un objet dynamique dessinable décrivant l'écoulement du temps.

GameEngine Un moteur de jeu est un objet dynamique supervisant l'ensemble d'une partie en définissant la grille, l'horloge et en déclenchant le passage des niveaux.

GameContext Un contexte de jeu est une sorte de moteur physique dans la mesure où il traite les mouvements et différencie 3 types de collisions (toucher, chute et capture). Le contexte ne dure que le temps d'un niveau, il change à chaque passage au niveau suivant.

Grid Une grille est une abstraction mathématique pour les dessins sur l'écran. Les dimensions théoriques de celle-ci sont constantes tandis que ses dimensions réelles s'ajustent automatiquement par rapport à la taille de l'écran. Cette grille peut être matérialisée pour la conception de niveaux de jeu mais cette possibilité est inactive au cours d'une partie.

Levels Le fichier Levels.kt contient la structure utilisée pour définir un niveau de jeu ainsi que la liste complète de ceux-ci. Cette liste est un tableau accessible exclusivement depuis le moteur de jeu via les extensions de propriétés que permet Kotlin.

DataClasses Le fichier DataClasses.kt contient un ensemble de classes distinctes mais similaires par le fait qu'elles ne définissent aucune méthode explicitement. Ainsi, il est préférable de les regrouper dans le même fichier plutôt que d'en avoir plusieurs dispersés un peu partout. Dans ce fichier, on trouve :

- Orientation : une structure regroupant les propriétés azimut, pitch et roll
- GridPoint : une structure regroupant les coordonnées bidimensionelles d'un point d'une grille
- Orderable : une classe facilitant le classement des instances de ses classes filles
- Save : une structure définissant les éléments à préserver dans une sauvegarde
- Score : une structure définissant les éléments à préserver dans un score

2.1.5 Les gestionnaires

IntentManager Le fichier IntentManager.kt facile la navigation et le passage de paramètres entre activités en définissant pour chacune d'elles des extensions de fonctions Kotlin au format :

```
fun X.toY(arg1, arg2, ...)
```

Exemple :

```
this@ShowScoresActivity.toMapsActivity(selectedScore)
```

GeolocationManager Le gestionnaire de géolocalisation va encapsuler les mécanismes de géolocalisation de l'API pour renvoyer, via un attribut statique, la localisation actuelle de l'utilisateur.

OrientationManager Le gestionnaire d'orientation va encapsuler les mécanismes d'orientation de l'API pour renvoyer, via un attribut statique, l'orientation courante de l'appareil mobile.

PictureManager Le gestionnaire d'images est un singleton qui va gérer l'association entre les sauvegardes et leur fichier image. Lors de son instanciation, il va supprimer tout fichier du dossier d'images non référencé par une sauvegarde ou ayant un identificateur non conforme. Il va permettre de récupérer un fichier image à partir de son identificateur depuis le dossier qu'il traite et il s'assurera de renvoyer l'identifiant le plus pertinent pour l'ajout de futures images au dossier.

2.1.6 Les préférences partagées

CurrentUser L'utilisateur courant est un singleton détenant toutes les informations concernant la partie en cours. Il garantit notamment l'accès et la persistance de ces données entre les activités via des attributs encapsulant les préférences partagées.

SharedList Une liste partagée est une liste abstraite d'éléments ordonnables encapsulant les préférences partagées. Bien qu'elle ne définisse que peu de méthodes publiques, elle peut être manipulée aussi simplement que les collections Java et Kotlin. Toutefois, afin de préserver la validité d'une instance, il convient de respecter un certain nombre de précautions :

- les préférences partagées encapsulées par l'instance ne devront plus être utilisées ailleurs
- les listes filles devront être des singletons
- les listes filles devront implémenter correctement les méthodes de mises en forme des données

SaveList La liste des sauvegardes est une liste partagée spécialisée pour les sauvegardes.

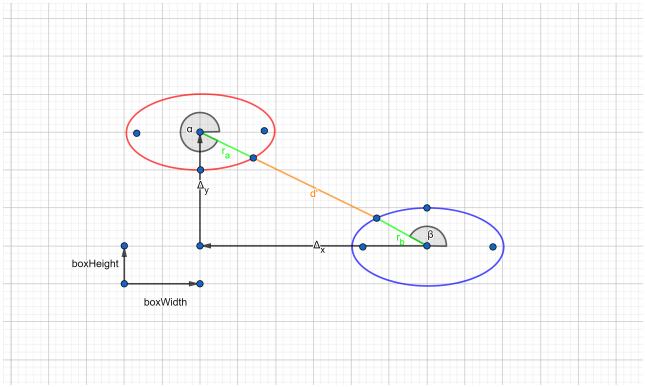
ScoreList La liste des scores est une liste partagée spécialisée pour les scores.

2.2 iOS

Implémentation abandonnée

3 Quelques points délicats/intéressants

3.1 Les collisions



En apparence, la collision entre deux *ellipses* [1] n'est pas très différente de celle entre deux cercles. Cependant, en pratique, il faut connaitre le rayon des ellipses pour détecter leur collision et, puisque celui-ci dépend de l'angle selon lequel on le calcule, la formule est beaucoup plus lourde.

3.2 La géolocalisation

La récupération de la position géographique s'effectue via une fonction de rappel. Ainsi, il est possible qu'au moment où en ait besoin cette position ne se soit pas encore valide car la fonction n'aura pas encore été appelée. Cette difficulté peut être contournée par un stockage temporaire de la valeur que l'on va récupérer tardivement mais ça reste un procédé hasardeux.

3.3 Kotlin

Le langage *Kotlin* [3] a de très nombreux avantages par rapport à Java. Il est plus permissif, moins redondant, plus fiable et il dispose de nombreuses fonctionnalités supplémentaires. Détaillons quelques-unes de ses spécificités par rapport à Java :

Les extensions Kotlin propose l'*extension* [2] de propriétés et de fonctions afin d'enrichir les classes existantes. Il faut toutefois faire attention aux conflits de noms car la documentation indique que, dans ce cas, une extension de propriété serait ignorée alors qu'une extension de fonction serait prise en compte. L'exemple d'extension de fonction suivant provient du fichier Tools.kt :

```
fun Any.Logd(msg: String) = Log.d(this::class.java.simpleName, msg)
```

Cette extension visant la classe Any (racine de la hiérarchie de classe de Kotlin) va ajouter à toutes les classes une méthode d'instance qui s'appelera Logd et qui affichera le nom de la classe et un message. Il est intéressant de remarquer que si on tente d'appeler cette méthode comme s'il s'agissait d'une méthode statique, le nom affiché va correspondre à celui de l'objet compagnon puisque Kotlin traite les méthodes statiques comme des méthodes d'instances de cet objet compagnon.

Les singletons Kotlin définit la notion de *singleton* [4] via le mot clé object, il est tentant de vouloir l'utiliser autant que possible mais mal utilisé il peut causer des dysfonctionnements difficiles à identifier. De plus, Kotlin interdit le passage de paramètres aux constructeurs de ces singletons donc, s'ils sont indispensables, il faut soit se débrouiller pour ne pas avoir à passer de paramètres, soit utiliser un singleton pattern plus classique avec une classe ordinaire, un constructeur privé et une méthode statique getInstance.

Conclusion

Ce projet a été particulièrement enrichissant. En particulier, la découverte de Kotlin au cours de l'implémentation a été assez plaisante puisqu'elle a permis une progression rapide. Bien que le projet ait été pensé pour être très simple, il est finalement beaucoup plus élaboré que prévu et ce n'est pas plus mal puisqu'il a nécessité un approfondissement plus poussé de notions avancées.

Le projet n'est qu'une version de démonstration et n'a pas la prétention d'être publiable mais s'il devait l'être il est certain qu'il irait vers une représentation de l'espace plus réaliste avec un véritable moteur physique, des spirales, des astéroïdes à la place des balles, des structures complexes telles que des systèmes solaires ou même d'autres objets célestes plus imprévisibles.

Il semble que la totalité des dysfonctionnements critiques constatés aient été corrigés mais aucun test vraiment poussé n'a réellement été effectué donc il est concevable qu'une situation non considérée finisse par survenir un jour ou l'autre.

Références

[1] Ellipse. https://fr.wikipedia.org/wiki/Ellipse_(math%C3%A9matiques).

[2] Extension. https://kotlinlang.org/docs/reference/extensions.html.

[3] Kotlin. https://kotlinlang.org/.

[4] Singleton. https://fr.wikipedia.org/wiki/Singleton_(patron_de_conception).