Rapport projet GeoGuess

I) Technologies utilisées

Pour ce projet Android, la technologie Java a été utilisée. Pour l'édition du code, il a été fait le choix de l'éditeur Android Studio. Une gestion des versions du code a été effectuée tout au long du projet à l'aide de l'outil git.

Pour l'affichage de la Street View et de Maps sur la même activité, je me suis basé sur un template fourni par Google et proposant déjà deux fragments disposés dans la même activité. Ce choix a été motivé par l'idée de ne pas réinventer la roue et de bénéficier d'une solution déjà existante pour se focaliser sur la qualité du code et de la solution fournie au niveau algorithmique.

II) Choix techniques

Pour cette application, j'ai fait en sorte de choisir les meilleures structures de données pour chaque partie de l'application. J'ai fait en sorte de découper au maximum mon code dans différentes classes en essayant de conserver un couplage le plus faible possible. Des commentaires (en anglais) sont aussi disponibles dans le code source

1) Architecture

L'application se compose de 9 classes :

- CheckGeneratedLocation : Classe chargée de faire une requête http en fond, utilisée pour vérifier qu'une position générée est disponible sur Street View
- GameLogic : Classe chargée du calcul du score
- Level : Enumération contenant les différents niveaux de difficulté du jeu
- MainActivity : La page d'accueil qui lancera le jeu
- Point : Un simple POJO définissant ce qu'est un point
- PossibleLocation : La classe chargée de générer des points aléatoirement
- PossibleLocationList : La classe contenant la liste des positions disponibles
- Score : La classe entité représentant le design d'un score
- SplitStreetView : L'activité contenant la street view en haut et la map en bas

2) Générer les positions

Pour générer les positions, j'ai fait le choix de définir au préalable des zones dans lesquelles je souhaiterai piocher des coordonnées aléatoirement. J'ai donc créé une classe PossibleLocationList qui contient une EnumMap avec toutes les zones possibles pour chaque niveau de difficulté. J'ai fait le choix d'une EnumMap car, pour mes niveaux de difficultés, j'ai créé une énumération. Cela me permet d'avoir une structure beaucoup plus solide pour gérer mes niveaux de difficultés et moins source d'erreur. Une fois que l'utilisateur a choisi le nombre de position qu'il souhaitait découvrir, je crée une pile qui contiendra mes zones. Je crée ma pile en créant tout d'abord un set qui contiendra les différentes zones pour lesquelles on va vouloir

générer des positions. J'ai fait le choix d'un set pour tirer au hasard dans mes zones possibles et ne pas avoir à gérer de doublons. Je transforme ensuite mon set en pile (ArrayDeque) car je ne vais que dépiler au fur et à mesure pour proposer une nouvelle position à l'utilisateur au cours du jeu. Une pile correspond donc parfaitement à cette exigence.

Une fois les zones pour lesquelles je souhaite générer des positions définies, je génère des points aléatoirement dans lesdites zones et teste avec l'API de Street View si le point généré est disponible. Si c'est le cas, je passe au suivant, si ce n'est pas le cas, je régénère des points jusqu'à ce qu'une position correcte soit trouvée.

3) Calculer et enregistrer les scores

Pour calculer les scores, j'ai décidé de faire un système où une unité d'écart est accordée pour chaque ville et chaque niveau. Par exemple, pour Paris, j'accorde un écart de seulement 3km car il s'agit d'une zone très facile. Pour Moscou par contre, l'écart accordé est de 30km car c'est une zone bien plus difficile. Par exemple, si l'utilisateur clique à 16km du point généré pour la zone Paris, il obtiendra le score suivant :

100 – (16 / 3) = 100 – 5 = 95

16 / 3 nous donne 5 car nous travaillons sur des int

Le joueur aura donc ici un score de 95.

Pour calculer la distance entre deux longitudes et deux latitudes, j'ai utilisé un code trouvé sur Stackoverflow.com, la source est mise en commentaires du code.