Itération 4

Release notes

Planification du projet

Les objectifs de l'itération 3 n'ayant pas été complètement atteints à cause de difficultés techniques rencontrées, concernant principalement la mise en place d'une architecture client/serveur et la configuration des ressources dans les environnements de développement/release, une réestimation des points pour l'itération 4 a été nécessaire.

Voici la liste des objectifs pour l'itération 4 :

- Complétion des objectifs de l'itération 3, révision de l'architecture pour éviter de nouvelles erreurs d'estimation.
- Correction des erreurs survenues lors de la démonstration (apparition des popup sur l'écran secondaire en mode projecteur, problèmes de connexion avec un compte nouvellement créé).
- Partage des informations d'un marqueur sur Twitter, avec lien vers les coordonnées du marqueur sur Google Map.
- Recherche et filtrage des marqueurs sur la carte (basées sur les caractéristiques de nom et de type des pokémons concernés, conditions combinées avec opérateurs ET, OU, NON).
- Sauvegarde des filtres par les utilisateurs ayant un compte, utilisation de filtres sauvegardés par n'importe quel utilisateur.

Le premier point a été résolu sans problème, une fois les solutions aux problèmes techniques rencontrés l'architecture du code s'est révélée correcte, nous avons tout de même veillé à effectuer certaines restructurations pour assurer plus de modularité dans le code.

Une fois cette difficulté surmontée, les nouveaux objectifs ont tous été rencontrés de façon satisfaisante.

Structure du code et choix techniques

La séparation du projet en *client*/*serveur* (ainsi que *common*, la partie du code partagée) implique également l'apparition de nouvelles classes. En effet les classes situées dans *common* et dont les instances sont échangées à travers le réseau doivent à présent implémenter une *interface* correspondante, et un système d'héritage est parfois requis pour séparer des fonctionnalités qui elles ne sont pas communes. Les classes situées dans *common* sont qualifiées de *sendable model* et leurs interfaces sont dites des *query model*. La plupart des contrôleurs font le lien entre un modèle et une vue et se trouvent du côté client, mais des fonctionnalités plus complexes que de simples requêtes existent également côté serveur, comme l'envoi d'e-mails.

La séparation entre "Modèle", "Vue" et "Contrôleur" existe toujours et les choix effectués à propos de leurs utilités et portées restent vrais. Chaque type de classe se trouvant désormais dans un *package* propre à ce type, il est redondant de reprendre la liste extensive dans un tableau. Depuis le début de l'itération deux, des efforts ont été faits pour que les classes contenant des données se trouvent bel et bien dans la catégorie "Modèle".
(parler des sous "catégorie" dans les packages)

Difficultés techniques rencontrées

Refactorisation du code :

Une vérification du code a été faite avec les outils présentés en cours. Dans un premier temps, l'ensemble du code a été revu à l'aide de PMD et des modifications et améliorations ont été apportées lorsque celles-ci ont été considérées judicieuses. Ensuite, de manière à estimer la qualité de l'architecture ainsi que les redondances dans le code, CodeCity a été utilisé. Peu d'incohérences ou de situations alarmantes ont été trouvées. Cependant, il nous est apparu clairement que les contrôleurs sont fortement liés (ce constat a déjà mis en avant lors de la troisième code review via Moose). Au vu de la deadline de la quatrième itération et de la date de la troisième code review, il a été décidé de ne pas se lancer dans un refactor complet de l'architecture car celle-ci aurait été trop chronophage.

(Wontfix) - bug d’affichage des marqueurs sur la map

Depuis la première itération, le système de clustering utilisé a causé des problèmes d’affichage des marqueurs : après un declustering, et parfois après un changement du niveau de zoom, les images représentant les marqueurs ne correspondent pas toujours au pokemon indiqué. Ce *glitch* est uniquement graphique et un simple zoom/dézoom suffit à retrouver l’apparence correcte, mais nous avons évidemment déployé des moyens pour y remédier.

- Une issue a été ouverte sur le *repo* officiel de google maps - marker clusterer où se trouve le code JavaScript employé. Il s’est avéré que ce repo n’était plus maintenu de façon active, mais que son auteur avait créé un fork pour le maintenir lui même. Après s’être entretenus avec cet employé de Google, nous avons pu déterminer que le glitch était entièrement lié à l’environnement WebView dans JavaFX, et donc que le code utilisé n’était pas en tort.
- Pour l’instant, le seul moyen trouvé pour arranger l’affichage est de recréer toutes les instances de marqueurs à chaque changement de zoom, ce qui n’a finalement pas été merge sur master par soucis de propreté et d’efficacité du code.

Bug affichage-chargement du css

Lors de la dernière itération, un “bug” concernant le thème css est apparu. Par déduction, nous avons déterminé qu’il provient d’un conflit lors du chargement du css de l’application javafx et de la google map (il est difficile d’apporter une preuve formelle). Nous nous sommes rendu compte que ce “bug” ne survenait pas lorsqu’on active le bouton du panel avant le chargement de la map. Nous avons donc créé une méthode activant par défaut le bouton. Ainsi, le “bug” ne survient plus.