Chapitre 1

Comparaison avec le projet AntSTO

1.1 Lisibilité

Contrairement au notre, le code est peu commenté (en partie en allemand...) mais aussi bien plus explicite. L'organisation des fichiers est similaire dans les deux projets, les différents rôles des modules ont été identifies de la même manière.

1.2 Correction

Le programme ne semble pas envoyer de requête (bien que ce soit implémenté) au serveur et n'a donc pas le comportement attendu. Le notre envoie effectivement les requêtes mais elle ne fonctionnement pas à cause des cookies. Accessoirement la boucle de jeu a aussi été implémentée dans notre programme.

Une large partie de l'API a été implémentée sans tenir compte d'un élément important : la plupart des actions de l'API requiert qu'un utilisateur soit connecté et cela n'a pas été traité dans leur module de communication. Toutefois, le code étant bien organisé, il ne serait pas difficile d'intégrer le cookie envoyé par le serveur dans les requêtes effectués.

1.3 Robustesse

Il est compliqué d'évaluer la capacité des traitements de données effectuées par leur projet car cela n'a pas été implémenté. On peut voir qu'il y a un début de traitement avec un module des codes d'erreurs du serveur et la récupération du statut de réponse du serveur suite à un appel à l'API. Toutefois, cette partie n'a pas pu être agréger au reste du projet donc elle n'est pas testable.

La manière par laquelle nous avons traité les réponses du serveur est de s'assurer que le statut renvoyé correspond à un succès puis effectuer le traitement correspondant à une action de l'API. Malgré le fait que nous ne pouvions pas tester notre traitement des données à cause des cookies, nous avons utilisé la documentation mise à notre disposition en début de projet afin de nous assurer que le traitement effectué pour chaque appel à l'API est correcte. Nous récupérions la réponse sous le format JSON pour le copier dans un fichier et tester nos fonctions dessus. Ce travail n'est pas visible que par l'implémentation que nous proposons des données puisque les tests ont été effectués en dehors de ce projet.

Pour conclure cette partie, on voit qu'un module a été mise en place pour traiter les erreurs du serveur mais que le traitement des données envoyés par le serveur n'a été effectué. Leur manière de traiter les erreurs est beaucoup plus extensible et précise que la nôtre mais nous avons effectué le traitement des données envoyés par le serveur contraitement à leur projet.

1.4 Efficacité

Il est difficile de juger (ni même d'observer) de l'efficacité des programmes, d'une part parce qu'ils ne sont pas corrects et d'autre part parce c'est le serveur qui donne le «rythme».

En terme de consommation mémoire, dans les deux cas c'est principalement l'interprétation des données reçues qui influe sur la consommation, le reste étant du calcul fait à partir de ces données. Ces données étant les mêmes pour les deux programmes, la consommation est, à peu de chose près, identique.

1.5 Généralisation

Notre code permet la création de comportement des fourmis (et des zombies) indépendamment du reste du programme (il suffit de donner la bonne valeur dans le fichier qui organise la boucle de jeu). On peut également modifier facilement les informations que contiennent les fourmis grâce a la représentation donné. L'autre projet n'a pas été assez loin dans l'implémentation de ces aspects du programme pour avoir cette flexibilité.

Par contre leur module de communication est beaucoup plus facilement modifiable pour suivre la potentiel évolution de l'API.