eXtreme Programming

Planning game

✅ J'ai commencé par lister les fonctionnalités et les saisir dans des issues sur le dépôt github. je les ai ensuite répertoriées dans l'outil kanban qui y est intégré.

C'est comme ça que j'ai commencé par développer les fonctionnalités liées à l'inventaire du matériel de l'archer: les flèches étaient nécessaires pour les deux autres fonctionnalités de base. Pour prioriser mes développements, j'ai surtout établi les dépendances des fonctionnalités entre elles.

Petites releases

✅ Au début du projet, je ne faisais pas assez attention à ce point. J'ai donc amélioré mes pratiques pour aller vers le fonctionnement suivant: pour chaque issue, une branche de travail contenant autant de commits qu'il le faut pour la résoudre, et lorsque c'est fait, elle est fusionnée dans la branche principale en réécrivant l'historique pour ne garder que les textes de commits importants.

pour d'éventuelles évolutions futures, le déploiement se fera à chaque "milestone", c'est-à-dire pour chaque "parcours utilisateur" complet développé, testé et validé.

Utilisation de métaphores

✅ Ayant travaillé sur un projet dans un domaine avec beaucoup d'objets concrets et relativement peu d'abstraction, je ne pense pas qu'il y avait besoin de plus utiliser de métaphores que simplement utiliser le vocabulaire métier, lui-même assez courant.

Conception simple

je ne pense pas avoir au final une conception trop complexe. en revanche, pendant le développement j'aurais pu travailler avec des modèles plus simplifiés (ils contenaient trop de champs dès le début), quitte à les compléter plus tard.

Tests unitaires/TDD

✅ présence de tests globalement surtout pour vérifier la non-régression générale de l'application, mais aussi comme une aide au développement sur la fonctionnalité "statistiques" (validation de l'algorithme utilisé)

Refactoring

✅ notamment pour la fonctionnalité de calcul du meilleur carquois

Pair programming

❌ pour avoir été le seul à vraiment travailler sur la partie code du projet, cette manière de travailler à plusieurs n'a pas pu avoir lieu.

Appropriation collective du code

❌ pour avoir été le seul à vraiment travailler sur la partie code du projet, cette appropriation collective n'a pas pu avoir lieu.

Intégration continue

✅ mise en place d'une exécution automatique des tests avec github actions qui s'exécute à chaque fois que du code est poussé peu importe la branche, ou qu'il y a une opération de fusion dans la branche principale.

la prochaine étape est d'automatiser aussi le déploiement, de manière à n'avoir plus qu'une seule commande à lancer, ou un seul bouton

Client sur site

✅ étant le principal "client" j'étais forcément "sur site"

Standard de code

- ✅ utilisation du gestionnaire de dépendances et d'environnement virtuel poetry
- ✅ outil d'inspection du code: Pylint, dont la configuration est dans un fichier pylintRC
- ✅ black, avec les options par défaut, lancé automatiquement par l'IDE à chaque sauvegarde du code:wq