Clôture du travail

Projet développé par DIAZ Gwenn, JACOB Alexandre et MEGNOUCHE Riad
2e Année GB-B

Au terme de ce projet, on a réussi à implémenter plusieurs objectifs techniques et conceptuels.

L'application développée respecte les exigences minimales, tout en proposant une architecture suffisamment flexible pour être étendue.

1. Synthèse globale

Le travail mené a permis de :

- Mettre en place une modélisation orientée objet cohérente.
- structurer une simulation dynamique intégrant des comportements comme le combat, hiérarchie, états internes, gestion du temps.
- Organiser les interactions entre personnages, lieux et systèmes avec la hiérarchie romaine/gauloise ou hiérarchie de meute dans le TD4.
- Intégrer des mécaniques d'alimentation, de santé, de potion magique ou encore de domination entre lycanthropes.
- Renforcer l'immersion grâce à un système de génération de noms culturels cohérents (NameRepository).
- Enrichir l'univers avec des créatures légendaires autonomes (Sphinx, Korrigans) offrant des interactions uniques.

L'ensemble a nécessité une bonne compréhension :

- Des principes de programmation Java.
- De la gestion d'événements simulés dans le temps.
- Des modèles de conception adaptés (singleton (bon il faudra juste vérifier s'il est vraiment)).

2. Bilan technique

Architecture logicielle

- Séparation claire des responsabilités entre classes :
  Personnages, Lieux, Chefs de clan, Théâtre d'envahissement, Aliments, Meutes, Hurlements...

- Mise en place d'héritages et d'interfaces permettant d'éviter la duplication.

- Gestion centralisée du déroulement temporel via une classe maîtrise (main loop).

- Implémentation de la généricité via une classe conteneur personnalisée (Box<T>) pour la gestion flexible des listes.

- Gestion de la concurrence (Threads) pour permettre le recrutement asynchrone des personnages sans bloquer la simulation.

- Utilisation de fonctionnalités avancées (Sealed Classes) pour sécuriser l'héritage des types Gaulois et Romains.

Gestion des entités

- Les personnages disposent d'états internes évolutifs comme la santé, faim, potion, belligérance…

- Les lycanthropes du TD4 possèdent des attributs en plus comme le rang, domination, impétuosité…

Mécanismes fonctionnels

- Combat et conséquences impact force/endurance.

- Nutrition et effets négatifs des aliments inadaptés.

- Potion magique.

- Hiérarchie des meutes :
  - domination, soumission, création du couple α, reproduction, vieillissement.

- Simulation :
  - rafraîchissement d'états, apparition d'aliments, gestion des actions du joueur, hurlements entre lycanthropes.

Qualité du code

- Respect global de la norme (nommage, indentation, séparation des fichiers).
- Utilisation réfléchie des encapsulations et getters/setters.
- Mise en place de tests fonctionnels élémentaires.

Problèmes rencontrées

- La complexité croissante liée aux interactions (ex : domination → changement de rang → propagation dans la meute).
- Gestion des contraintes spécifiques aux lieux.
- Équilibrage de la simulation pour éviter chaos immédiat ou monotonie.
- Mise en place d'une Intégration Continue (CI) via GitHub Actions pour automatiser la compilation et les tests.

Mesures d'amélioration

- Ajout de l'option de la création de la vie où les Gaulois ou les Romain puissent se reproduire avec la création d'enfant qui prendra le même clan que ses parents. (Avec un discussion que j'ai pu avoir avec Louis SIMON qui a pu le mettre en place lui de son côté)
- Interface utilisateur plus ergonomique.
- Mettre en place l'interaction des lycanthropes avec les chefs de clan
- Persistances des données entre simulations.

Écart avec les prévisions

Complexité de la gestion de meute :

- L'intégration de la logique "TD4" (Lycanthropes) dans le projet "TD3" a nécessité une refonte de l'architecture pour gérer les doubles hiérarchies (grades militaires vs domination animale).

Aspect technique vs Fonctionnel :

- Sous-estimation du temps nécessaire à la conformité technique stricte (Sealed Classes, Javadoc, Généricité). Une part importante du temps a été allouée au refactoring pour respecter ces contraintes académiques.

Richesse de la simulation :

- La simulation est finalement plus vivante que prévu grâce à l'ajout non planifié de PNJ autonomes et à l'automatisation complète des combats, rendant le théâtre d'opérations dynamique même sans intervention du joueur.

3. Conclusion

Ce projet a permis de manipuler intensivement les notions fondamentales de Java tout en affrontant un scénario riche. Malgré la taille du cahier des charges, l’application obtenue est fonctionnelle, stable et extensible.