Projet R3.04 : Qualité de développement

Lien Github : VERNAGUT-Titouan-2326112-A/Projet-R3.04

Etude de conception

TD 3: Hôpital fantastique

Dans cette première partie de projet, nous avions à implémenter de nombreux éléments,afin de réaliser une simulation d'hôpital fantastique, que nous allons vous présenter ici tout en détaillant nos choix d'implémentations.

I. Créature

Toutes les créatures ont les mêmes caractéristiques suivantes: un nom, un sexe, un poids, une taille, un âge, un indicateur moral et une liste de maladies.

Chaque créature doit pouvoir: attendre, hurler, s'emporter, tomber malade, être soignée,trépasser.

En trépassant, certaines créatures peuvent démoraliser (elfes, vampires), contaminer (orques,hommes-bêtes,lycanthropes, vampires) et régénérer (zombies, vampires).

De plus, certaines créatures attendent plus patiemment (orques, homme-bêtes, lycanthropes et vampires) et d'autres non (elfes, nains, vampires, reptiliens).

Enfin, les créatures ont de grandes chances de contaminer les autres créatures.

Pour implémenter tout cela, nous nous sommes basés sur les fondements du modèle de conception Décorateur à l'exception qu'ici nous créons une classe abstraite Créature à la place d'une interface dans le but de simplifier les tests. Chaque sous classe fonctionne donc comme des objets décorés car elles sont construites sur le principe de la classe Créatures avec leurs comportements spécifiques en plus.

Afin de forcer les créatures ayant besoin d'implémenter les fonctions contaminer ou régénérer nous avons créé deux interfaces qui implémentent chaque les comportements liés à ces caractéristiques (une pour contaminer et l'autre pour régénérable).

I. Maladie

En plus d'avoir des caractéristiques personnelles, la créature peut aussi avoir une ou plusieurs maladies. Il en existe plusieurs avec bien évidemment un nom pour chacune d'entre elles. Chaque maladie qui est présente dans ce jeu à la même probabilité d'arriver à une créature.
Comme maladie nous avons :
Maladie débilitante chronique (MDC)
Syndrome fear of missing out (FOMO)
Dépendance aux réseaux sociaux (DRS)
Porphyrie érythropoïétique congénitale (PEC)
Zoopathie paraphrénique lycanthropique (ZPL)

Pour les implémenter, nous avons décidé de leur dédier leur propre classe maladie comportant toutes les méthodes nécessaires pour le respect du sujet et le fonctionnement de la simulation.

II. Service Médicaux

Les services médicaux doivent pouvoir contenir une ou plusieurs créatures et chaque service médical doit avoir des créatures de même type.

un service médicaux doit avoir :
un nom, une superficie, le nombre maximum de créatures qu'il peut contenir, le nombre de créatures présentes, les créatures présentes, un budget (inexistant, médiocre, insuffisant ou faible) ainsi qu'un type de créature qu'il peut contenir.

il doit permettre aussi:
d'afficher ses caractéristiques ainsi que les caractéristiques des créatures qu'il contient, d'ajouter et d'enlever des créatures, de soigner les créatures qu'il contient, de réviser le budget.

il existe aussi plusieurs services médicau: les centres de quarantaine et les cryptes:
Les centres quarantaines ne contiennent que des créatures contagieuses et possèdent des espèces supplémentaires: l'isolation.
Les cryptes ne peuvent contenir que des créatures régénérantes et possèdent deux caractéristiques supplémentaires: le niveau de ventilation et la température.

Pour implémenter cela, nous avons créé une classe mère service médical qui représente tous les services médicaux génériques (pas abstraite car ici on peut avoir de simples services médicaux sans spécificité). CentreDeQuarantaine et Crypte sont des classes filles héritant par le principe d'héritage simple des comportement des services médicaux génériques avec en plus l'implémentation des comportements qui leur sont dédiés.

III. Médecin

Les médecins possède des caractéristiques comme : le nom, le sexe ainsi que l'âge et le type de créature qu'il peut prendre en charge.

Ils doivent pouvoir : examiner un service médical,soigner les créatures d'un service médical, réviser le budget d'un service médical puis transférer une créature d'un service médical à un autre.

Nous avons pour ceux-ci implémenté un classe simple contenant tous les comportements dédiés.

IV. L'hôpital fantastique

Enfin, pour la simulation nous pouvons gérer les créatures, gérer les services médicaux,gérer les médecins, gérer les maladies, afficher l'ensemble des créatures présentes dans l'hôpital,afficher l'ensemble des services médicaux et leurs créatures, incarner un médecin et enfin quitter le jeu.
Pour implémenter cela, nous avons d'abord décrit les comportements que doit posséder l'hôpital. Ensuite, nous avons implémenté une simulation par système de menus. Nous avons le menu principal qui permet de choisir au joueur qu'est-ce qu'il veut gérer ou s' il veut incarner le médecin comme le sujet le demande. Ensuite, il est redirigé vers un nouveau menu spécialisé dans un domaine précis (par exemple : la gestion des médecins). Tout subdiviser nous a permis de réduire la charge des méthodes mais également d'isoler les erreurs quand celles-ci se présentaient afin d'optimiser au maximum les performances et l'expérience utilisateur. Enfin, en plus de cela, nous avons implémenté un Thread qui tourne en arrière plan et qui permet de gérer automatiquement toute la partie événementielle aléatoire dans la simulation à l'aide de probabilités d'obtenir des évènements régulées au maximum afin de rendre la simulation fluide.

TD 4: colonie de lycanthropes

Dans cette partie du projet, nous avions à implémenter également de nombreux éléments, afin de pouvoir créer une simulation de colonie de lycanthrope, que nous allons vous détailler ici tout en vous présentant nos choix techniques.

I. Lycanthrope

Un lycanthrope est une créature ayant un nom, un sexe, une force, un rang,un niveau ainsi qu'un facteur d'impétuosité.
En plus d'implémenter l'ensemble des comportements attendus par le sujet dans une classe Lycanthrope, nous avons décidé d'utiliser le modèle de conception Factory pour implémenter ceux-ci. Ce modèle permet d'implémenter de façon automatique des lycanthropes dès qu'on en a besoin dans le code sans avoir à tout spécifier à chaque fois.
De plus, il nous était demandé qu'un système de hurlement avec d'autres lycanthropes pouvant réagir à ceux-ci. Pour cela, nous avons utilisé le modèle de conception Observer qui permet de créer et gérer des listes de listeners via une interface qu'on peut implémenter dans les classes nécessaires (notamment dans Meute que nous verrons après). Cela nous a permis de gérer efficacement tous les comportements liés à ceux-ci.
Pour ce qui est des hurlements, nous avons ajouté de nouvelles réactions en cas de conflit afin de rendre la simulation plus ludique : un autre lycanthrope de la meute peut réagir au cri de domination ou de soumission d'un lycanthrope en essayant de dominer celui-ci à son tour.

II. Meute

Une meute est composée de lycanthrope, elle possède un couple alpha, des membres et une hiérarchie. En plus des comportements attendus, nous avons ici implémenté notre Observable via la méthode reagirAuHurlement () qui permet de notifier tous les listeners afin qu'ils puissent réagir de façon appropriée au hurlements.

III. Colonie

Une colonie est composée de meutes, de lycanthropes solitaires et de groupes de lycanthropes solitaires. Nous avons complété ce modèle avec les groupes de

lycanthropes solitaires afin de rendre la simulation plus complète et ainsi améliorer l'expérience utilisateur. En plus des comportements attendus d'une colonie, nous avons ajouté le cœur de la simulation dans cette classe. Comme pour l'hôpital fantastique, celui-ci se repose sur une menu général permettant d'accéder à des éléments de gestion précis (par exemple : la gestion des meutes), toujours dans un souci d'optimisation et afin d'isoler tout problème qui aurait fait face. Nous avons également mis un Thread permettant de gérer les évènements aléatoires de la simulation toujours sur le même principe avec en plus de cela la gestion de l'avancée des saisons et les évènements liés à celle-ci (que nous détaillerons dans une partie spécifiée).

IV. Les saisons

Afin de rendre la simulation plus ludique, nous avons décidé d'implémenter un système de saisons. Celui-ci fonctionne sur le principe d'un timer qui toutes les durées données passe à un état suivant correspondant à une nouvelle saison. En plus de cela, à chaque saison est lié une catégorie d'événements que nous avons choisi :

- pour le printemps : période de reproduction du couple alpha
- pour l'été : période où les lycanthropes deviennent plus agressifs et combattent entre eux
- pour l'automne : période où les lycanthropes ont plus de chances de se transformer en humain
- pour l'hiver : période de changement de catégorie d'âge d'un lycanthrope leur permettant d'avancer dans leur cycle de vie jusqu'à mourir.

V. Les évènements

Pour rendre la simulation complète, nous avons ajouté comme demandés deux évènements à celle-ci :

- les reproductions du couples alpha
- les conflits entre deux lycanthropes

A la base nous gérions ces évènements dans une classe de gestion, car nous nous sommes basés sur la construction d'un gestionnaire de commande comme modèle de conception. Nous avons ensuite adapté ce modèle à notre simulation pour que les commandes soient directement gérées par les classes concernées plutôt que par un gestionnaire de commandes afin de rendre le code plus optimal. Bien sûr, chacun des deux commandes implémente les comportements d'une classe commande afin d'éviter tous les cas particuliers d'erreurs et régler celles-ci plus simplement encore une fois.

Implémentation des tests

Afin de développer de manière efficace, nous avons implémenté de nombreux tests avec JUnit 5 afin d'éviter au maximum les cas d'erreurs que peuvent contenir notre code et rendre celui-ci utilisable.
Dans le cas de classes avec héritages, nous avons appliqué des tests sur la classe mère car celle-ci comportait pratiquement tous les comportements et attributs à tester car il s'agit de classe abstraite.

Bilan de clôture du projet

Dans l'ensemble, nous avons été capables de réaliser l'ensemble du projet sans rencontrer de réels problèmes excepté quelques uns :

- un problème de budget aléatoire qu'on n'a pas réussi à totalement fixer qui peut très rarement nuire à une petite partie de l'expérience utilisateur.
- afficher les maladies, qui pendant un certain temps à été compliqué à implémenter ce qui nous a freiné sur l'avancée du projet un certain temps.
- soigner les créatures, qui a été la méthode la plus compliquée à implémenter correctement afin qu'elle soit effective dans la simulation.
- conversion de Intellij à Eclipse, qui a été compliqué de part la variété de contenu et les méthodes parfois trop récentes pour Eclipse utilisées dans notre projet qui ont dû être remplacées.
- difficulté de gestion des thread, principalement sur l'affichage qui se surcharge beaucoup trop que nous n'avons pas pu régler car nous n'utilisons pas d'interface graphique.

Afin de rendre ce projet plus complet, nous avons pensé à certaines améliorations possibles :

- optimiser l'affichage en JAVAFX pour le rendre plus beau et plus complet dans la représentation des événements.
- ajouter plus de modes de jeu pour rendre le jeu plus ludique comme pouvoir incarner son propre lycanthrope dans le TD4.