CARVAJAL Tom
GUIL Vanessa
DELIN Alexis

R 3.04
Qualité de développement

Projet - Application Zoo Fantastique

Spécification Fonctionnelle Technique

Semestre 3

2023 - 2024

Sommaire

CARVAJAL Tom
GUIL Vanessa
DELIN Alexis

I – Introduction

A - Présentation du document

Cette spécification technique décrit les exigences pour le développement de l'application du Zoo Fantastique. Il fournit des indications claires sur les objectifs et servira de référence sur la portée et la structure du projet. Il constitue une ressource complète pour l'équipe en détaillant les fonctionnalités attendues, les contraintes techniques, ainsi que les lignes directrices en matière de tests et de documentation.

B - Objectif de l'application

L'objectif principal de l'application est d'offrir une expérience dans un monde magique rempli de créatures mythiques (telles que les licornes, les mégalodons, les sirènes…). Les utilisateurs auront la possibilité d'observer, de gérer et d'interagir avec ces créatures dans des enclos spécialement conçus pour répondre à leurs caractéristiques uniques.

Au cœur de l'application se trouve le gardien de zoo, qui joue un rôle central, en supervisant à la fois les enclos et les créatures individuelles. Le gardien du zoo (l'utilisateur) a la possibilité d'effectuer diverses tâches, comme par exemple l'alimentation des créatures, le nettoyage des enclos, et la satisfaction des besoins de chaque créature. La gestion peut se faire de manière manuel ou automatique.

II - Description générale

A - Vue d'ensemble du système

L'application fonctionne selon une architecture Modèle-Vue-Contrôleur (MVC), offrant un système complet de gestion et de présentation du monde des créatures mythiques. Au cœur du système se trouvent les créatures, les enclos et le zoo lui-même. Chaque action est appelé par le contrôleur. Ce dernier envoie les informations à afficher à la vue pour l'affichage. Cela permet de prévenir l'utilisateur de tout ce que l'application est en train de faire. En mode gestion manuelle, l'utilisateur aura la possibilité de choisir les actions qu'il souhaite réaliser.

B - Architecture MVC (Modèle-Vue-Contrôleur)

1 – Modèle

Dans le modèle, l'application encapsule toutes les données et la logique liées aux créatures, aux enclos et au zoo. Le modèle sert de colonne vertébrale à l'application, en garantissant l'intégrité des données et en gérant les fonctionnalités de base qui dictent le comportement des créatures et de leurs habitats.

2 – Vue

Les composantes de vue se concentrent sur l'interface et l'expérience utilisateur, responsable de la présentation du monde du Zoo Fantastique aux utilisateurs. En communication constante avec le contrôleur, la vue va permettre de prévenir l'utilisateur sur l'ensemble des actions réalisés par l'application, mais aussi de récupérer directement les entrées qu'il pourra faire.

3 – Contrôleur

Le contrôleur sert de médiateur entre le modèle et la vue. Il orchestre les interactions des utilisateurs et les réponses du système. Il interprète les entrées des utilisateurs, met à jour le modèle en conséquence et s'assure que la vue reflète les changements les plus récents. Les contrôleurs permettent d'assurer une flexibilité et une modularité de l'application en améliorant les fonctionnalités globales.

C - Les Design Pattern

1 – Singleton

Le modèle de conception Singleton est utilisé pour les instances du zoo, du maitre du zoo, ou encore pour le gestionnaire de temps. En utilisant Singleton, l'application s'assure qu'il n'y a qu'une seule instance du zoo et une seule instance du maitre zoo tout au long de son cycle de vie. Ce choix de conception maintient un point de contrôle et de coordination unique, évitant ainsi de multiples instances conflictuelles et améliorant la cohérence et l'intégrité du zoo fantastique.

2 – Factory

Le design pattern Factory est implémenté pour la création de créatures au sein de l'application. Il impose de créer les créatures par l'intermédiaire de la fabrique désignée, fournissant ainsi une approche structurée et contrôlée de l'instanciation des créatures. Ce modèle permet de passer des types de créatures en tant que paramètres, ce qui assure une souplesse et une extensibilité au processus de création. En centralisant la création de créatures par le biais du modèle Factory, le zoo fantastique garantit une approche standardisée et personnalisable de l'introduction de nouvelles créatures.

III - Exigences fonctionnelles

A – Les fonctionnalités de l'application principale

1 – Les créatures

Chaque créature (dragon, lycanthrope, licorne, nymphe…) doit être dotée de caractéristiques telles que le nom, le sexe, le poids, la taille, l'âge, un indicateur de faim, un indicateur de sommeil, et un indicateur de santé. Les créatures peuvent accomplir des actions basiques telles que manger, émettre un son, être soignées, s'endormir, se réveiller, et vieillir.

De plus, chaque créature aura une famille (vivipare ou ovipare) avec des actions spécifiques à leur genre comme pondre un œuf ou mettre bas. Les créatures pourront aussi hériter de méthodes spécifiques à leur genre (aquatique, terrestre, immortel…) comme courir, nager, voler ou renaitre.

2 – Les enclos

Le Zoo fantastique comprend trois types d'enclos distincts : Enclos classique, Volière (avec un toit), et Aquarium (avec des considérations de profondeur et de salinité de l'eau). Il faut assurer la compatibilité des types de créatures avec leurs enclos respectifs afin de permettre une cohabitation harmonieuse. Les enclos contiennent des fonctions de gestion essentielles telles que le nettoyage de l'enclos et l'alimentation des créatures.

3 – Le zoo fantastique

L'entité globale, le Fantastic Zoo, agit comme un Singleton, encapsulant l'ensemble de l'écosystème du zoo. Ce composant de gestion central facilite une navigation efficace, permettant de retrouver les enclos par leur nom et fournissant une vue d'ensemble du nombre total de créatures dans le zoo. Le Zoo Fantastique sert de point central pour la gestion globale du zoo.

4 – Le maitre zoo

Le maître zoo est lui aussi un Singleton qui représentera l'utilisateur. C'est le gestionnaire qui aura la responsabilité d'examiner des enclos, de les nettoyer, de nourrir les créatures, et de transférer de créatures entre les enclos. L'aspect temporel de la simulation est pris en compte, avec un compteur décrémenté selon le nombre d'actions réalisées.

L'aspect temporel de la simulation doit être pris en compte, introduisant des éléments aléatoires tels que des changements d'état pour certaines créatures (maladie, sommeil, etc.) et des altérations dans les enclos (propreté, salinité, etc.). Nous avons choisi pour cela d'implémenter dans un premier temps le temps sous forme de compteur qui sera décrémenté selon le nombre d'action réalisé. Une fois le compteur à 0, un changement d'année sera effectué avec toutes les actions que cela engendre (les créatures vieillissent, les enclos se dégradent, des enfants naissent…).

5 – Gestion automatique du zoo

L'application intègre des méthodes de contrôle automatisées qui gèrent dynamiquement le zoo, assurant le bien-être des créatures et la cohérence globale de l'écosystème. Cette fonctionnalité permet au zoo d'être géré automatiquement grâce à des méthodes de contrôle des différentes caractéristiques.

6 – Gestion manuelle du zoo

La gestion manuelle du zoo par l'utilisateur permet une prise de décision directe. Les utilisateurs peuvent choisir d'intervenir dans l'entretien et l'organisation du zoo, ce qui favorise une interaction personnalisée et engageante avec les créatures mythiques et leurs habitats. Leur but est d'avancer dans leur aventure et de créer un zoo de plus en plus grand et sain.

7 – La gestion du temps

La gestion du temps dans ce système est facilitée par la classe GestionnaireTemps, qui agit comme un contrôleur temporel. Elle permet aux utilisateurs de manipuler et de suivre le temps dans l'environnement simulé. La classe utilise une approche basée sur le calendrier, offrant des méthodes pour avancer, ajouter et récupérer des dates. En outre, grâce à une énumération, les utilisateurs peuvent commodément associer des actions spécifiques à des durées prédéfinies, ce qui simplifie la simulation des aspects temporels dans le programme. Cette approche modulaire permet de contrôler et de synchroniser avec précision diverses actions au sein de la simulation, offrant ainsi un moyen souple et efficace de gérer le temps dans le système. De plus, cette gestion du temps permet de lancer des évènements à intervalles régulières (et notamment l'évènement qui modifie chaque année l'état des enclos et des créatures).

B – Les fonctionnalités concernant les lycanthropes

1 – Caractéristiques

Un lycanthrope est une créature qui va dans un premier temps hériter de l'ensemble des caractéristiques de cette dernière. Il aura aussi des attributs comme sa force, un facteur de domination, son rang, son niveau, son facteur d'impétuosité, et la meute à laquelle il appartient (s'il en a une). Les lycanthropes peuvent aussi se transformer en humain.

2 – Hiérarchie de la meute et interactions

Les lycanthropes vivent en meute ou sont solitaires. Une meute est dirigée par un couple alpha, et les interactions entre les lycanthropes dépendent de leur rang qui permettra de connaitre leur facteur de domination. Pour communiquer, les lycanthropes hurlent afin d'exprimer leur appartenance, leur domination, leur soumission, ou leur agressivité.

3 – Gestion dans le contexte du zoo

Au sein du zoo, un enclos ne pourra avoir qu'une seule meute avec son couple alpha pour éviter tout conflit. Les lycanthropes seront intégrés au contrôle automatique du zoo, mais aussi au contrôle manuel par l'utilisateur.

IV - Contraintes techniques

A – Les structures de données

1 – Les ensembles (Sets)

L'application utilise les ensembles comme structure de données principale (en particulier pour la gestion des enclos dans le zoo). Les ensembles constituent un moyen efficace de gérer des collections uniques d'enclos, en veillant à ce que chaque enclos soit distinct et en évitant les doublons. Ce choix favorise l'organisation et la catégorisation des enclos, ce qui permet de rationaliser les opérations dans l'environnement du zoo. L'ajout et la suppression d'éléments est rapide et simplifié par rapport à un tableau.

2 – Les cartes (Map)

Les cartes sont utilisées pour organiser les créatures dans les enclos, chaque créature se voyant attribuer un identifiant unique (ID) comme clé. L'utilisation de cartes facilite l'accès rapide et direct à des créatures spécifiques sur la base de leur identifiant, optimisant ainsi la récupération et la manipulation d'êtres mythiques individuels. L'association d'une clé à chaque créature (qui est mise à jour à chaque ajout et suppression) permet aussi à l'utilisateur d'effectuer une sélection de créature de manière simplifié et rapide.

B – Les algorithmes

1 – Recherche linéaire

L'application utilise l'algorithme de recherche linéaire pour des tâches telles que la recherche d'un enclos sur la base de son nom. Le principe de la recherche linéaire consiste à parcourir une liste de manière séquentielle jusqu'à ce que l'élément recherché soit trouvé. Dans le contexte de la localisation des enclos, cet algorithme parcourt la liste des enclos en comparant les noms jusqu'à ce qu'une correspondance soit identifiée. La mise en œuvre implique une itération directe à travers la liste, ce qui la rend adaptée aux scénarios dans lesquels la taille de la liste est relativement petite ou n'est pas triée.

2 – Algorithme de tri

Afin de trier les créatures dans un enclos, nous avons choisi d'utiliser le tri par arbre binaire de recherche (ABR) afin que les créatures soient triées selon leur âge dans leur enclos. Chaque créature est représentée par un nœud de l'arbre. Les créatures plus jeunes sont situées du côté gauche, et les plus âgées sont placées du côté droit. Pour récupérer la liste des créatures de l'arbre, un parcours préfixe est effectué.

Le principal avantage de cette méthode est son efficacité, mais aussi que c'est une structure auto-équilibrée. C'est une méthode similaire aux arbres AVL. Cet algorithme de tri va permettre un maintien automatique de l'équilibre lors de l'insertion et de la suppression. Nous avons donc choisi cette méthode qui est déjà implémenté au sein de la structure TreeMap car elle correspond à nos besoins.

C – Les classes abstraites

Les classes abstraites jouent un rôle crucial dans la conception de l'application, notamment en définissant des caractéristiques communes pour les créatures. Les classes abstraites, telles que celles des créatures, des ovipares et des vivipares, encapsulent des attributs et des comportements partagés tout en permettant des implémentations spécifiques dans leurs sous-classes concrètes. Cette abstraction fournit une hiérarchie structurée, favorisant la réutilisation du code et assurant la cohérence dans l'implémentation des différents types de créatures.

D – Les interfaces

Les interfaces sont utilisées pour classer les créatures en fonction de certains traits, comme le fait d'être aquatique, aérien ou immortel. Ce choix de conception facilite à la fois la vérification des caractéristiques d'une créature et la standardisation des méthodes propres à chaque type de créature. En mettant en œuvre des interfaces, le zoo fantastique globalise les méthodes associées à des types de créatures spécifiques, ce qui favorise une approche modulaire et extensible.

E– Convention de nommage

1 – Le nom des classes et des interfaces

Les noms de classes et d'interfaces respectent la convention PascalCase, commençant par une majuscule, et sont choisis pour être significatifs et indiquer leur but dans la base de code.

2 – Le nom des méthodes et des variables

Les noms de méthodes et de variables suivent la convention camelCase, commençant par une lettre minuscule. Cette convention garantit une approche cohérente et claire de l'attribution des noms, contribuant ainsi à la lisibilité et à la maintenabilité du code.

3 – Les constantes et membres statiques

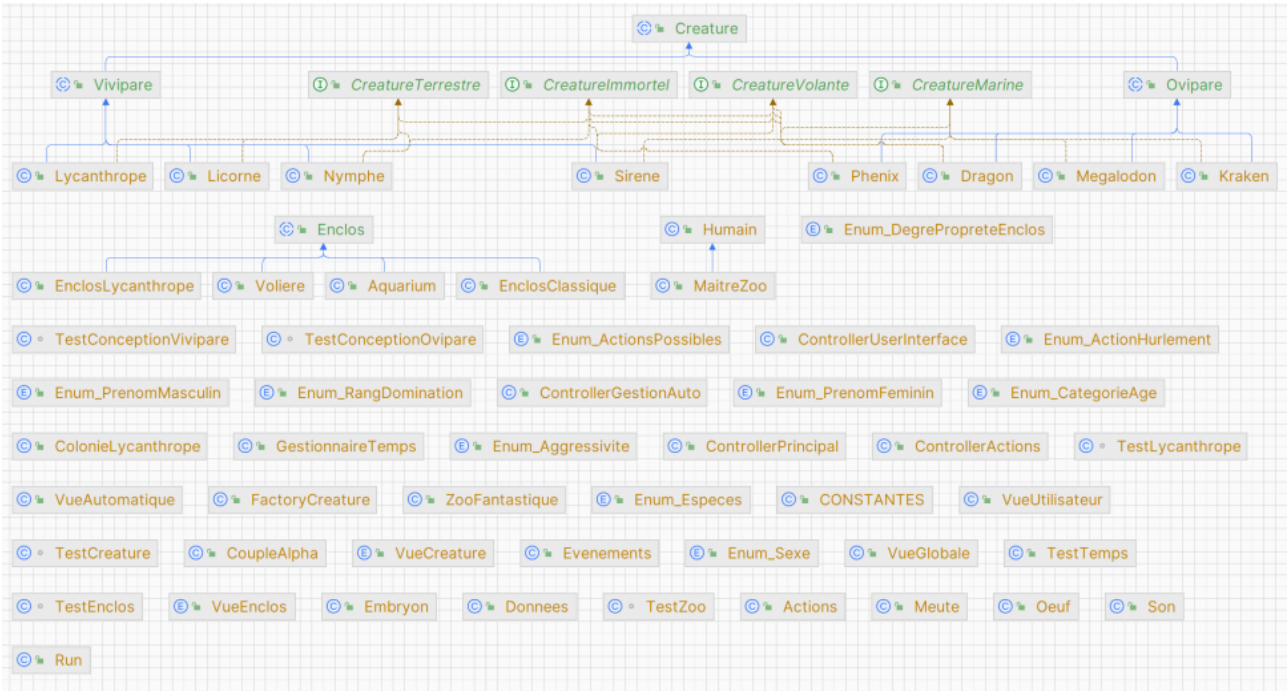
Les constantes et les membres statiques sont nommés en majuscules, les mots étant séparés par des traits de soulignement. Cette convention les distingue des variables ordinaires et met l'accent sur leur immuabilité ou leur nature statique dans la base de code.

4 – Les packages

Les noms de packages sont écrits en minuscules. Cette convention s'aligne sur les pratiques standard de Java, contribuant à une structure unifiée et organisée au sein de la base de code.

V - Diagramme de classes

A - Organisation entre les classes



B – Diagramme complet

Voir Annexe I.

VI – Tests

A – Importance des tests

Le processus de test mené pour l'application joue un rôle essentiel pour garantir la fiabilité et le fonctionnement de l'application. Les tests systématiques permettent d'identifier les problèmes potentiels et de valider rigoureusement les fonctionnalités de l'application. Un test est réalisé pour chaque méthode, en prenant en compte le cas où une erreur peut être renvoyé.

B – Tests effectués pour les contrôleurs

Le répertoire "testController" encapsule les tests spécifiquement axés sur les composants du contrôleur de l'application. Cet ensemble de tests vise à valider les fonctionnalités associées aux interactions avec l'utilisateur et aux réponses du système. Des tests sont notamment effectués pour la classe "Temps", qui garantissent la gestion précise des actions liées au temps, telles que le vieillissement des créatures et le passage des années. Les tests du contrôleur contribuent garantissent des interactions transparentes avec la logique de l'application.

C – Tests effectués pour le modèle

Le répertoire "testModele" comprend des tests adaptés aux différentes classes de modèles de l'application. Cette section de tests couvre un spectre de fonctionnalités, assurant la correction et la fiabilité de chaque composant du modèle. Des tests spécifiques aux enclos, à la conception d'une nouvelle créature, aux créatures elles-mêmes… valident l'implémentation de leurs fonctionnalités respectives. Les tests de modèle garantissent l'intégrité de la logique de base de l'application.

D – Tests de performance et de charge

En plus des tests fonctionnels, l'application a subit des tests de performance et de charge afin d'évaluer sa stabilité et son efficacité dans diverses conditions. Ces tests, exécutés avec un nombre important de créatures et d'enclos sur des périodes de simulation prolongées, évaluent la

capacité de l'application à gérer des charges accrues sans compromettre les performances. Grâce à une classe constantes qui comprend l'ensemble des paramètres qui peuvent changer (nombre de créature par enclos, âge maximale d'une créature…, nous avons pu tester l'application avec différents paramètres et évaluer ses performances.

Au cours de scénarios de test impliquant un grand nombre d'enclos et de créatures sur des périodes de simulation prolongées, nos observations ont révélé des informations sur le comportement de l'application. Au fur et à mesure que la complexité de l'environnement augmentait, nous avons constaté une augmentation proportionnelle de l'occurrence des erreurs. L'application a également connu une légère baisse de performance.

VII - Gestion des erreurs

A – Importance de la gestion des erreurs

Une gestion efficace des erreurs est un aspect crucial de la conception et de la mise en œuvre de l'application . Cela permet d'assurer la fiabilité globale, d'améliorer l'expérience de l'utilisateur, et d'assurer la maintenabilité de l'application. Une bonne gestion des erreurs permet d'identifier et de résoudre rapidement les problèmes potentiels, ce qui garantit un système plus résistant. Des tests sont effectués dans les méthodes du modèle. En cas de problème, une exception est envoyée. Le contrôleur gèrera ensuite ces exceptions.

B – Lancement d'exceptions à partir du modèle

Dans les composants de modèle de l'application, le lancement systématique d'exceptions est utilisé pour signaler et traiter efficacement les erreurs. Chaque classe de modèle est conçue pour détecter et répondre à des scénarios exceptionnels, tels que des entrées non valides, des opérations non autorisées ou des états inattendus. En utilisant des exceptions, le modèle garantit un moyen clair et normalisé de communiquer les erreurs aux composants appelants, ce qui permet une identification et une résolution précises des problèmes. Chaque exception renvoyée est formulée de manière assez claire pour cibler le problème de manière efficace.

C – Gestion des exceptions dans le contrôleur

Les composants du contrôleur de l'application possèdent des mécanismes de gestion des exceptions. Lorsqu'il interagit avec le modèle, le contrôleur attrape et gère les exceptions lancées par les classes du modèle. Cela permet de s'assurer que les scénarios inattendus ou les erreurs, qu'elles proviennent de l'entrée de l'utilisateur ou de processus internes, sont traités de manière efficace. La stratégie de gestion des exceptions du contrôleur consiste à fournir des messages d'erreur informatifs afin de maintenir la stabilité générale de l'application.

VIII – Clôture du travail

A – Synthèse et bilan

Nous avons trouvé ce projet très intéressant, notamment parce qu'il nous a permis d'explorer Java et ses différentes facettes. Les consignes du projet étaient assez générales, ce qui nous a permis d'avoir une certaine liberté dans nos choix. Cependant, le délai limité a eu un impact sur notre capacité à réaliser pleinement nos idées. Tout au long du processus de développement, nous avons acquis une expérience pratique, notamment en approfondissant différents aspects de la programmation orientée objet. Par exemple, nous avons réussi à mettre en œuvre différents design pattern comme une factory ou plusieurs singletons. Un travail d'équipe efficace a été essentiel, ce qui nous a confirmé l'importance de l'écoute active et du compromis pour pouvoir avancer de manière efficiente. Cette collaboration nous a également permis d'acquérir une connaissance approfondie du cycle de vie du développement d'une application, couvrant notamment les phases de spécification, de codage et de test.

B – Les problèmes rencontrés

Au cours du projet, nous avons rencontré plusieurs problèmes. Tout d'abord, il était parfois complexe ou long d'appréhender le code réalisé par un autre membre du groupe, que ce soit en entrant dans un projet déjà en cours ou lorsque des modifications ont été apportées par différents membres de l'équipe. Il a donc fallu s'assurer de la collaboration de chacun pour garantir une base de code cohérente (documentation, journal des modifications…). De plus, nous étions limités par le temps, ce qui nous a contraint de faire des choix techniques. Nous avons dû trouver

un équilibre entre le désir de mettre en œuvre toutes les fonctionnalités envisagées et la réalité des ressources.

Une autre difficulté a été la répartition des tâches, en particulier compte tenu des différents niveaux de compétences au sein de l'équipe. Il était donc difficile de déléguer une tâche à un autre membre de l'équipe qui aurait eu besoin de temps pour se familiariser avec le travail en cours. Malgré ces difficultés, notre esprit de collaboration nous a permis de les surmonter, contribuant ainsi à l'avancée de notre projet.

C – Les écarts avec les prévisions

Notre objectif premier était de respecter toutes les contraintes et lignes directrices du projet, ce que nous avons fait. Au fur et à mesure que les idées évoluaient, la portée du projet s'est naturellement élargie. Malheureusement, en raison des contraintes, toutes les fonctionnalités envisagées n'ont pas pu être mises en œuvre.

D – Les mesures d'améliorations

Pour l'avenir, nous suggérons d'améliorer le projet en mettant en œuvre un système de gestion automatique des zoos plus sophistiqué. Cela impliquerait de prendre en compte l'état individuel de chaque objet. De plus, l'étude de l'intégration de threads pourrait simuler des scénarios de gestion réels, rendant notre application plus réaliste et réactive.

D'autres actions pourraient aussi être proposée à l'utilisateur. Voici une liste non exhaustive :

- Proposer des aliments différents pour nourrir les créatures
- Implémenter des maladies lorsque les créatures sont en mauvaise santé
- Permettre la communication entre les créatures
- Gestion de la généalogie des créatures

CARVAJAL Tom
GUIL Vanessa
DELIN Alexis

Annexe I – Diagramme UML complet