Rapport

Jules Hospital, Gabriel Rouge, Maxime Tamarin

Sommaire :

Synthèse
Bilan Technique
Problèmes rencontrés
Ecart des prévisions
Mesures d'amélioration
Etude de conception
Manuel d'utilisation

Synthèse

Ce rapport se veut au mieux résumer les points importants qu'il nous est demandé d'aborder et d'expliquer ; ce qu'on a appris, ce qu'on a pas pu faire, les solutions que l'on a apportées … Globalement, notre ressenti sur notre projet est positif ; nous avons pu accomplir une partie importante du travail en faisant particulièrement attention à la qualité du code et de son architecture, puisqu'il s'agit de ce sur quoi nous sommes évalués. Malgré ce contexte de projet noté, nous nous sommes amusés à le développer, et sommes contents de vous montrer le résultat de notre travail.

Bilan Technique

Ce projet a permis la mise en application de plusieurs points, et ainsi, a permis l'approfondissement de plusieurs connaissances telles que ; la manipulation des Threads, les Design Pattern et leurs implémentation, et l'orienté objet en général.

En effet, nous avons pu pousser jusqu'au bout notre compréhension de l'orienté objet autant pendant la conception que dans la production de l'application, car le sujet était propice à l'utilisation de tout ce que ce paradigme a à proposer : les différentes portées (privées, publiques et protégées), des types de propriétés (static, final), différents types de classes (classes, classes abstraites, interfaces, enum) avec l'utilisation poussée de ce qu'elles ont à proposer (méthodes dans un enum, méthodes par défaut d'interface, gestion de l'héritage des propriétés avec leurs portées) et puis les différentes manières d'hériter et manipuler des propriétés (étendre, surcharger, implémenter et forcer l'implémentation…).

Nous avons également pu améliorer encore une fois nos compétences dans l'utilisation de git et GitHub, en gestion de projet de groupes, et en UML.

Le produit final démontre de l'utilisation de tous ces concepts, et l'historique des commits sur GitHub peuvent témoigner de l'évolution importante du projet autant que de l'équipe pendant ce mois de création de Zoo.

Problèmes rencontrés

Le premier problème rencontré était de réaliser que la conception n'allait pas, et ce, pendant la réalisation de l'application. En effet, les sujets nécessitent l'utilisation de design pattern pour résoudre certains problèmes, et nos connaissances dans ce domaine au début du projet étaient presque nulles. Ce n'est que plus tard que nous avons pu comprendre que certaines situations étaient propices à l'utilisation de design pattern (l'utilisation d'une factory pour les créatures, d'un décorateur pour les lycanthropes, d'un Singleton pour la gestion des interactions…). Si la même chose était à refaire, maintenant que nous avons de meilleures connaissances en Design Pattern, ce problème aurait moins de chance d'apparaître. Évidemment, il nous faudrait une connaissance absolue dans ce domaine pour ne plus jamais y faire face ; et nous aspirons à cela.

Ensuite, le second problème, nous devons l'évoquer ; il s'agit du temps. Pendant cette période chargée, trouver du temps pour coder l'application et s'organiser entre nous était compliqué. C'est principalement pour cette raison que l'application n'est pas finie et que tous les points demandés ne sont pas remplis ; dans quelques années peut-être, nous serons biens plus performants et sauront produire le double, voire plus, en autant de temps.

Ecart des prévisions

Nous avons au mieux omis tous les points qui n'étaient pas directement demandés, puisque nous avions conscience du défi de conception auquel nous faisions face, et rajouter ne serait-ce qu'un seul élément pourrait beaucoup trop complexifier la chose.

Pour cette raison, nous avons globalement produit que le nécessaire, et voici ce que nous n'avons pas pu produire :

- Concernant le Zoo Fantastique, nous sommes heureux de pouvoir dire que nous avons rempli toutes les consignes, à l'exception de la dernière "Équilibrez la simulation pour la rendre ludique". Effectivement, nous avons préféré accorder du temps au reste plutôt que de s'improviser Game Designer et fabriquer un jeu de simulation. En revanche, avec du temps supplémentaire, nous aurions trouvé ça amusant et enrichissant de remplir ce point. Actuellement, il est possible de faire tout ce qui est demandé dans l'énoncé, mais le jeu ne comporte aucun enjeu et l'interface n'est pas particulièrement travaillée.
- L'implémentation de tests unitaires et d'une implémentation continue ; il s'agit d'un point auquel nous sommes que très peu familier, et n'avons pas accordé assez de temps pour en faire usage, bien que le TDD aurait pu nous faire économiser du temps à certains moments.
- L'utilisation de collection, généricité, et exceptions ; nous avons effectivement utilisé brièvement les exceptions à quelques endroits, mais globalement, nous n'avons pas trouvé l'utilité d'employer ces outils pour notre application.
- Concernant la meute de lycanthropes, c'est plus délicat. Evidemment, il n'était pas prévu qu'un groupe puisse finir ce sujet entièrement puisque la charge totale de travail est très importante par rapport à notre niveau. Pour cette raison, nous ne sommes pas déçus de ne pas avoir pu finir l'entièreté des points. Ceux que nous avons pas pu remplir sont : le corps de certaines méthodes, le comportement des loups entre eux, la transformation en humain, le calcul du facteur de domination …

Mesures d’amélioration

Finalement, nous aurions pu changer notre manière d’aborder les problèmes pour les résoudre plus rapidement et finaliser le produit pour le rendu. Comme dit précédemment, les principales ressources qui nous manquent sont le temps mais surtout des connaissances, que grâce à ce travail, nous avons pu acquérir.

Certaines décisions étaient de bonnes décisions, comme la plupart des choix de conceptions que nous avons fait (voir prochaine partie). En revanche, il aurait été préférable de ne pas négliger d’autres points comme l’utilisation du TDD et de l’implémentation continue, qui nous aurait pris des ressources à initialiser, mais nous en aurait fait économiser beaucoup plus sur la fin (en plus de majorer la note).

Sur le plan de la gestion de projet et du travail d’équipe, nous n’avons pas eu de souci particulier. On en conclut que nous avons assez pris l’habitude de travailler ensemble pour y arriver sans créer aucun ennui.

Le dernier point touche à la jouabilité du jeu. Nous aurions en effet aimé avoir le temps de rendre le jeu plus ludique ; intégrer un système d’argent pour financer les enclos et les créatures, régler les comportements des créatures qui salissent plus ou moins ou mangent plus ou moins selon leur espèce, équilibrer le nombre d’action possible par intervalles …

Etude de conception

- **Les répertoires :**

La racine de l’application donne sur deux répertoires : ***Player*** qui contient les classes représentant le joueur, son terrain de jeu, et les classes permettant la communication entre le joueur et le jeu, puis ***Game*** avec toutes les autres classes (créatures, enclos, gestion de la logique du jeu …).

Pour faire ce choix d’architecture, nous avons analysé les différentes couches d’intéractions entre le joueur et l’application, ou entre l’application et l’application : Le joueur n'interagit directement qu’avec quelques classes (Player) ; le reste du jeu s’organise dans une couche suivante, où les classes instanciées par les classes ***Player*** s'auto gèrent.

- **Le Singleton :**

Le point central de notre application est une classe Singleton nommée “Controler”. Elle contient l’instance du Zoo, et organise les entrées de données dans l’application. Lorsqu’un objet doit avoir une influence sur le Zoo (par exemple, une créature meurt car on ne s’est pas occupé d’elle), ledit objet le fait savoir au Contrôleur qui se charge de le communiquer à l’application.

- **Les classes joueur :**

Ces classes sont le point d’entrée de l’application et celles qui permettent les interactions. On peut considérer une division semblable à une architecture MVC où la classe Interface est la vue car elle seule affiche et reçoit l’entrée utilisateur, les classes Contrôler, Asker et Creator sont celles qui orchestrent et traitent les données, puis le reste de l’application est le Model.

Les classes Asker et Creator contiennent des méthodes statiques qui auraient pu appartenir au Controler ; nous avons préféré les diviser pour fluidifier le code, où le Asker permet de demander des entrées spécifiques à l’utilisateur (un age, un sexe …) et le Creator permet d'interagir avec l’utilisateur pour créer certains objets (Des enclos, des créatures …).

Ensuite, le Zoo stocke l’entièreté du jeu, et le ZooMaster permet toutes les interactions entre l’utilisateur et le Zoo.

- **Les classes jeu:**

Ces classes sont, globalement, le jeu. Les répertoires principaux sont ; Logic (contenant des classes utilisées par toute l'application pour certaines tâches, comme la gestion de l'aléatoire et des temps d'attentes), Corral (Contenant les enclos et les classes qui les agrègent ou les composent), Creature (Contenant les créatures et les classes qui les agrègent ou les composent), et Lycantropus (concernant le deuxième sujet sur la meute de loup).

- **Les caractéristiques**

Les créatures, enclos, et même le maitre de zoo et les Lycantropes ont des caractéristiques particulières (age, sexe, taille, poids, etc). Celles-ci seraient naturellement exprimées par des entiers puisqu'il s'agit de valeurs quantifiables et mesurables ; nous aurions pu indiquer la taille en mètres, l'âge en année, le poids en kilogrammes, etc. Or, nous avons préféré éviter l'emploi de métriques puisqu'elles semblent incohérentes ici ; ainsi nous nous sommes inspirés du jeu de rôle ***Forbidden Lands*** qui exprime brillamment les quantités de ressources en dés, soit de manière abstraite, plutôt qu'en mesure précise. De cette manière, les caractéristiques sont des classes Enum, et leur sens varie selon la créature qu'elle compose ; un mégalodon"Petit" est sans doute plus grand qu'un Lycanthrope "Grand", … En outre, nous avons pris en compte la **relativité**.

- **Les comportements des créatures**

Chaque créatures sont des classes finales qui implémentent l'interface Runnable. De ce fait, elles peuvent exister dans leur propre thread, et expérimenter à leur échelle le passage du temps. Cette manière de procéder est en effet la plus logique puisqu'elle mimétise la réalité où chaque individu expérimente différemment le passage du temps ; par exemple, une personne active et en bonne santé n'aura pas la même fatigue qu'un ouvrier pauvre. Ainsi, ces deux individus peuvent avoir le même âge en termes d'année, mais l'un semblera beaucoup plus jeune que l'autre car moins usé : encore une fois, nous prenons en compte la **relativité** des choses dans l'expérience que chaque créature fait de sa vie virtuelle et du temps.

Concernant leur "libre arbitre", la décision de ce qu'elles font est donnée en jet de dé ; selon son résultat, elle fait une action spécifique, ou n'en fait aucune.

Manuel d'utilisation

Pour lancer le jeu, il suffit d'exécuter l'application depuis le point d'entrée qu'est la méthode statique main de la classe Player.Zoo, exactement comme il était demandé dans la consigne.

Le jeu est simpliste ; L'interface est un simple terminal, qui attend constamment les entrées de l'utilisateur, en lui expliquant préalablement ce qu'on attend de lui.

Au lancement du jeu, il sera demandé de donner un nom et un sexe au maitre de zoo, puis un nom au zoo. Vous pourrez ensuite commencer à construire des enclos en tapant ***Add*** et en suivant le reste des instructions, ensuite vous rapprocher d'un enclos pour interagir avec en tapant ***Zoom***, etc.

Au premier coup d'œil, il est possible de ne pas savoir exactement comment jouer ; mais le jeu indique clairement toutes les commandes que l'on peut entrer, ou l'entrée précise qu'il attend. Si la saisie de l'utilisateur n'est pas comprise par l'application, aucun souci, il pourra la ressaisir.

De cette manière, il est possible de créer des enclos, d'y créer des créatures, puis de les laisser vivre en regardant régulièrement leurs états, s'assurant que leurs besoins sont remplis.