COOL, TL, QL

Rapport de synthèse

CHEVRIER Thibaud, CREVAN Romain, ZOUABI Youssef - 6 February 2017

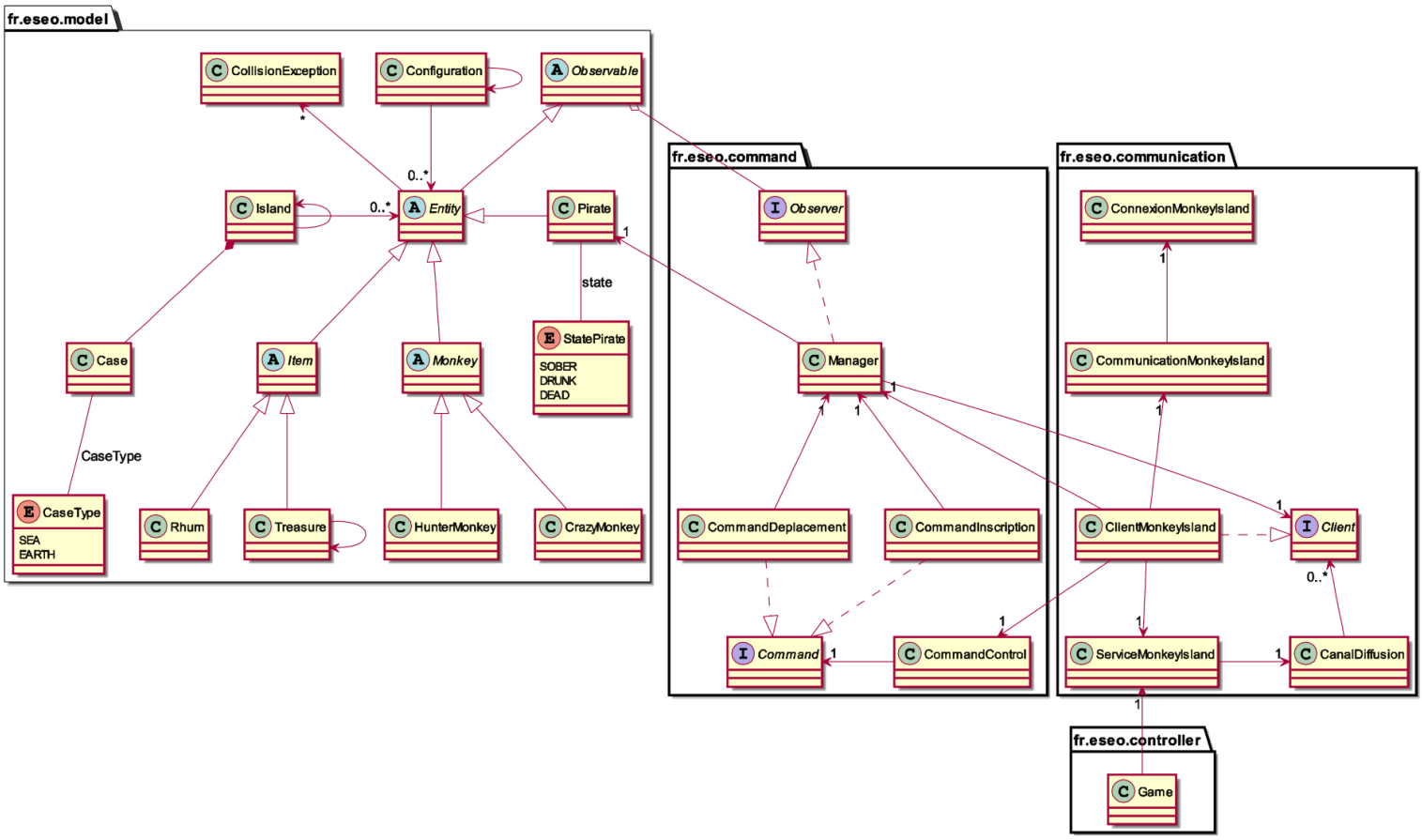
1. Introduction

Ce document présente la synthèse du projet MonkeyIsland produit pour des fins pédagogiques. Il présente les activités de conception, de développement et de test menées durant le projet ainsi que les exigences de qualité mises en place pour ce projet.

L'objectif de ce projet à consister à développer l'application serveur de ce jeu, le client étant fourni par l'équipe pédagogique. Ce développement s'est axé autour d'une problématique de qualité logicielle importante. Des outils au service de la qualité et du test logiciel ont donc été imposés à utilisation.

2. Conception

2.1 Conception générale



Le diagramme ci-dessus représente les instances entre toutes les classes et packages implémentés. Trois patrons de conception s'y distinguent :

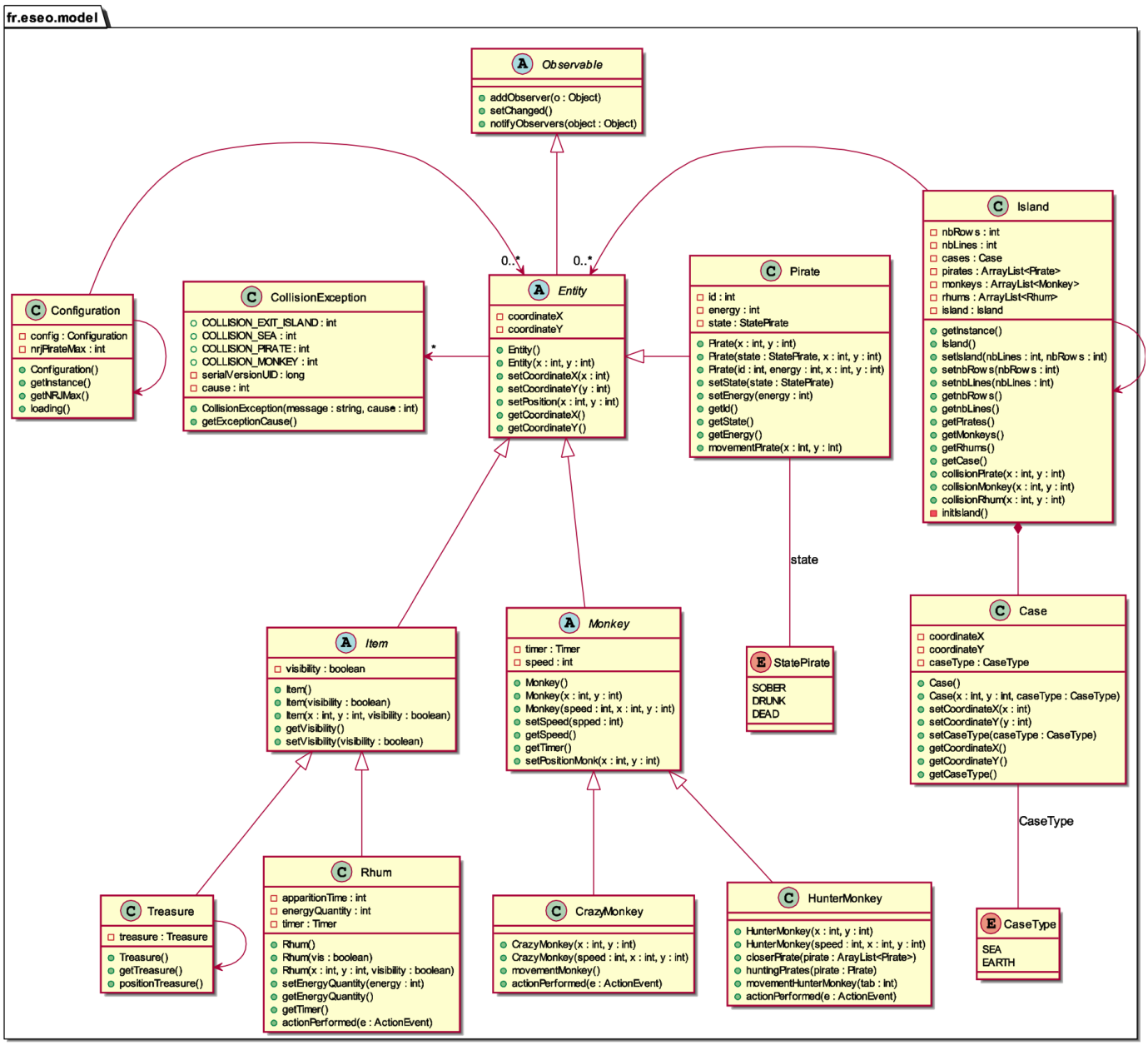
- Singleton : les trois classes Treasure, Island et Configuration sont des singletons et ne sont instanciées qu'une seule fois chacune.
- Observateur : l'observateur est représenté par la classe Manager et l'observable par la classe Entity
- Commande : le patron de conception Commande est représenté par le package fr.eseo.command

Le jeu s'exécute par le biais du package fr.eseo.controller par la classe Game qui instancie la classe ServiceMonkeyIsland située dans le package fr.eseo.communication.

Celui-ci n'est volontairement pas détaillé puisqu'il nous est fourni, seul les classes instanciants ou instanciés par les classes du package fr.eseo.command y sont représentées. La classe ProtocoleMonkeyIsland n'est pas représenté.

ServiceMonkeyIsland instancie la classe CanalDiffusion qui instancie plusieurs clients. Chaque client est associé à un manager qui lui est propre. Le Manager crée un pirate pour son client et se charge de gérer les commandes de ce dernier.

2.2 Package model



Ce diagramme ci-dessus représente le package fr.eseo.model . Il définit les relations entre l'île, les pirates, les singes, les bouteilles de rhum et le trésor. Etant donné qu'il n'y a qu'une île et un trésor pour une partie, les classes Island et Treasure sont des singletons. L'île est composée de deux types de case (classe Case) définie par l'énumération CaseType : case « SEA » et case « EARTH».

La classe Entity est une classe abstraite représentant une entité qui regroupe à la fois les singes, les pirates, les bouteilles de rhums et le trésor. La classe Entity est un héritier de la classe Observable, chaque changement de certains de ses attributs est notifié à l'aide des deux méthodes setChanged() et notifyObservers(Object o). Ces méthodes sont invoquées dans les classes suivantes :

- Classe Entity par setCoodinateX(int x) et setCoordinate(int y)
- Classe Item par setVisibility(boolean visibility)
- Classe Pirate par setState(StatePirate state)

Chaque modification de ses attributs invoque la fonction update(Observable o, Object arg1) de l'interface Observer. Autrement dit chaque fois qu'une entité se déplace, qu'un objet change de visibilité ou qu'un pirate change d'état, ce changement est notifié dans la fonction update.

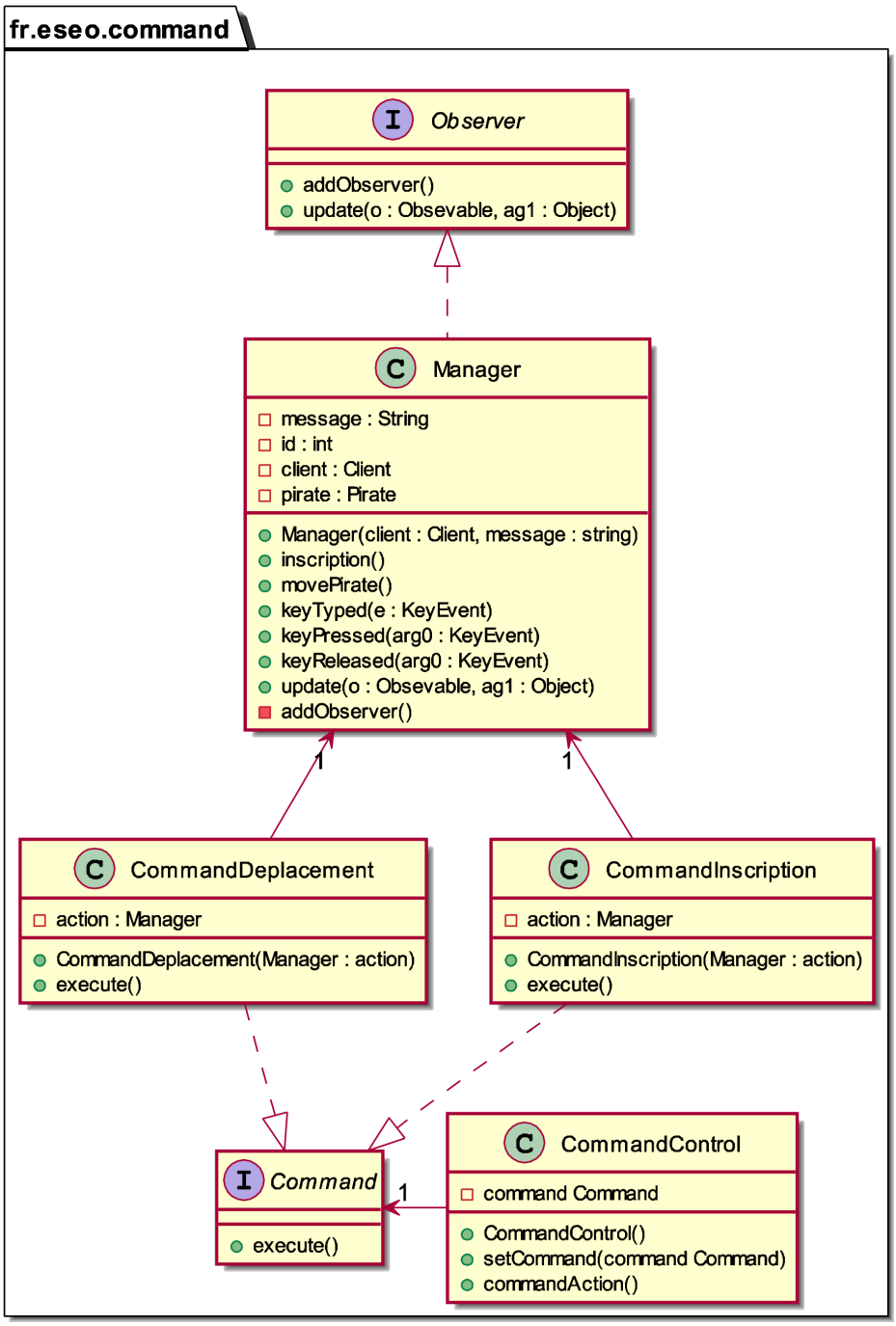
2.2 Package command

La classe Manager est un Observateur, elle implémente l'interface Observer. C'est dans cette classe que la méthode update est appelée à chaque modification d'attribut de la classe Entity. A chaque invocation, la méthode update envoie les commandes spécifiques de changement d'entités au client qui lui est propre. La classe Manager à également pour fonction de gérer les commandes du client.
On distingue deux types de commande :

- Demande d'inscription (classe CommandInscription appelant la méthode inscription() de la classe Manager) pour l'inscription d'un client. La méthode inscription() créer aléatoirement un nouveau pirate pour son client et envoie toutes les commandes d'initialisation de l'île.
- Demande de déplacement (classe CommandDeplacement appelant la méthode movePirate de la classe Manager) pour le déplacement d'un pirate. Cette commande est invoquée à chaque appui sur les touches directionnelles du clavier du client par la méthode KeyTyped(KeyEvent e) de la classe Manager.

Les deux classes CommandDeplacement et CommandInscription implémentent l'interface Command instancié par la classe CommandControl. Ce

patron de conception Commande permet d'encapsuler l'invocation des commandes du client.



3. Réalisation

Cette section reprend les différents points et exigences du cahier des charges pour présenter l'état d'avancement du projet.

La présentation des réalisations s'effectue de la manière suivante :

Fonctionnalité	Testé	Validé	Outil(s) utilisé(s)
Fonctionnalité X	Oui/Non	Oui/Non	JUnit, JMeter, Test Validation(TV)

3.1 Connexion client-serveur

Pour lancer une partie, un joueur doit se connecter au serveur MonkeyIsland à l'aide du client Guybrush.

Fonctionnalité	Testé	Validé	Outil(s) utilisé(s)
Connexion d'un client au serveur	Oui	Oui	JMeter
Ajout d'un pirate avec ID lors de la connexion du client	Oui	Oui	JMeter
Envoi de la carte du jeu avec sa dimension et le type de cases (terre ou mer)	Oui	Oui	JMeter
Envoi de la position initiale du pirate associé au client	Oui	Oui	JMeter
Envoi de la position des singes erratiques sur la carte	Oui	Oui	JMeter
Envoi de la position des singes chasseurs sur la carte	Oui	Oui	JMeter
Envoi d'un déplacement refusé	Oui	Oui	JMeter
Envoi d'un déplacement accepté	Oui	Oui	JMeter
Envoi de la position du rhum sur la carte et de la quantité d'énergie	Oui	Oui	JMeter

3.2 Configuration d'une partie

Le jeu contient un niveau décrit dans un fichier de configuration. L'analyse de ce fichier permet de récupérer toutes les informations nécessaires au lancement d'une nouvelle partie.

Fonctionnalité	Testé	Validé	Outil(s) utilisé(s)
Les informations du fichier de configuration sont utilisées pour le lancement d'une partie	Oui	Oui	Unit, TV
Création des singes erratiques avec les caractéristiques du fichier de configuration	Oui	Oui	TV
Création des singes chasseurs avec les caractéristiques du fichier de configuration	Oui	Oui	TV
Création des rhums avec les caractéristiques du fichier de configuration	Oui	Oui	TV, JMeter
Création du trésor avec les caractéristiques du fichier de configuration	Oui	Oui	TV

3.3 Initialisation d'une partie

Fonctionnalité	Testé	Validé	Outil(s) utilisé(s)
Création d'une carte avec des cases terre et mer	Oui	Oui	JUnit, TV
Création impossible d'un pirate sur une case mer	Oui	Oui	JUnit, TV
Création impossible d'un singe erratique sur une case mer	Oui	Oui	JUnit, TV

Fonctionnalité	Testé	Validé	Outil(s) utilisé(s)
Création impossible d'un singe chasseur sur une case mer	Oui	Oui	JUnit, TV
Un seul personnage à la fois sur une case du jeu	Oui	Oui	TV
Le trésor est placé sur la carte	Oui	Oui	JUnit

3.4 Déroulement d'une partie

3.4.1 Pirate

Fonctionnalité	Testé	Validé	Outil(s) utilisé(s)
Déplacement du pirate (haut, bas, gauche, droite)	Oui	Oui	JUnit, TV, JMeter
Déplacement du pirate coûte 1 point d'énergie	Oui	Oui	JUnit, TV, JMeter
Déplacement du pirate refusé si un autre pirate est déjà présent	Oui	Oui	JUnit
Le pirate est tué s'il se déplace sur un singe	Oui	Oui	JUnit, TV
Le pirate est tué s'il n'a plus d'énergie	Oui	Oui	JUnit, TV
Déplacement impossible d'un pirate sur une case mer	Oui	Oui	JUnit, TV
Indication de l'emplacement du trésor en cas de découverte	Oui	Oui	TV

3.4.2 Singe

Fonctionnalité	Testé	Validé	Outil(s) utilisé(s)
Déplacement aléatoire du singe erratique(haut, bas, gauche, droite)	Oui	Oui	JUnit, TV
Les singes chasseurs et erratiques tuent le pirate quand ils se déplacent dessus	Oui	Oui	JUnit, TV
Mouvement impossible d'un singe erratique sur une case mer	Oui	Oui	JUnit, TV
Mouvement impossible d'un singe chasseur sur une case mer	Oui	Oui	JUnit, TV
Mouvement impossible d'un singe sur un autre singe	Oui	Oui	JUnit, TV

3.4.3 Rhum

Fonctionnalité	Testé	Validé	Outil(s) utilisé(s)
Le rhum disparait lorsqu'un pirate le boit	Oui	Oui	TV
Le pirate gagne l'équivalent de l'énergie du rhum lorsqu'il se déplace dessus	Oui	Oui	JUnit, TV
Le rhum réapparaît après un certain temps (timer)	Oui	Oui	TV

3.5 Ce qu'il reste à faire

Dans ce qui a été développé, il reste l'incrément 3 à réaliser. Les singes chasseurs ont commencé à être développé mais leur déplacement vers le pirate le plus proche n'a pas passé les test de validation, même si des tests avec JUnit ont été écrits.

4. Qualité

4.1 Subversion

Subversion est un outil qualité permettant la gestion de version par le partage de ressources.

Nous connaissions déjà cet outil par le biais d'autres projets réalisés les années précédentes. Ici, il a été indispensable afin de partager le code réalisé pour un partage clair du travail.

Utilisation élevée pour le partage du code.

4.2 Checkstyle

Checkstyle est un outil de contrôle de code permettant de vérifier le style d'un code source selon certaines règles établies.

Connu mais utilisé pleinement pour la première fois lors de ce projet, il a fallu un peu de temps pour établir des règles qui soient à la fois source de qualité et qui conviennent à chacun des membres. Il nous a semblé très utile car il nous oblige à respecter ces règles, amenant de l'efficacité et de la clarté au code.

Utilisation élevée dans le projet. Les règles Checkstyle sont cependant appliquées seulement dans le fichier source et non dans le fichier test.

4.3 Findbugs

Findbugs est un logiciel permettant de vérifier le code et d'en ressortir les erreurs pouvant amener des bugs de l'application.

Nous avons découvert cet outil lors de ce projet et malgré des exigences parfois difficiles à satisfaire, il nous a permis de respecter des exigences de qualité de développement.

Utilisation continue pendant le projet, installé lors du premier incrément et exécuter à chaque build du projet.

4.4 Emma

Emma est un outil qui mesure et rapporte la couverture du code de test.

Utilisé continuellement lors du projet pour vérifier que les tests effectués couvrent bien les fonctionnalités voulues. A la fin du projet, notre couverture sur notre model est de 66,2 %.

5. Bilan

Nous avons beaucoup appris par ce projet. En effet, nous avons pu découvrir une variété d'outils au service de la qualité et du test logiciel pour optimiser notre façon de façonner un projet. Le choix du jeu Monkey Island était judicieux et malgré le fait de ne pas avoir eu le temps de terminer, ce jeu nous a permis de pouvoir utiliser les outils d'une manière intéressante et constructive. La prise en main de ces derniers nous a d'abord semblé laborieuse et trop exigeante mais par la suite nous avons bien compris que, sans eux, il nous serait très difficile de produire en groupe un code de qualité et efficace. Nous avons d'abord eu l'impression d'être freiné dans notre développement à cause de cette nécessité de qualité et de test, mais cela nous a permis de pouvoir avancer rapidement lors de la fin du projet pour fournir quelque chose de fiable.

En plus de la qualité et du test logiciel, ce projet nous a apporté d'un point de vue conceptuel. Nous avons pu découvrir plusieurs patrons de conception et nous nous rendons compte de l'augmentation de l'efficacité par leur utilisation. Leur mise en place fut plus difficile que nous l'imaginions et nous a obligé à revenir sur certains point de notre conception, mais cela pour le bien du projet.

En analysant la participation de chacun des membres de l'équipe, nous sommes satisfaits de la qualité du code produit et de la couverture du code par les différents types de tests.