

HomePlans - Rapport INFO-F-307

Titouan Christophe
Pierre Gérard
Florentin Hennecker
Walter Moulart
Bruno Rocha Pereira
Julian Schembri

27 novembre 2014

Table des matières

1	Itération 1	2
1.1	Introduction	2
1.2	Librairies introduites	2
1.2.1	Enregistrement du projet	2
1.2.2	Librairie GUI	2
1.2.3	Librairie 3D	4
1.3	Rapport de fin d'itération	5
1.3.1	Suivi du planning	5
1.3.2	Architecture	5
1.3.3	Bonnes pratiques utilisées	7
1.3.4	Réflexion sur les librairies choisies	11
1.3.5	Conclusion - What's next	11

Chapitre 1

Itération 1

1.1 Introduction

Nous avons décidé de développer l'histoire 1 proposée par le client lors de cette phase. Ce choix a été fait dans le but de fournir le plus vite possible un socle pour les fonctionnalités à suivre, et se présentait logiquement comme le seul choix possible.

Vous pourrez lire dans la suite de ce chapitre la description des choix qui ont été faits quant aux librairies que l'équipe a décidé d'utiliser, suivie d'un rapport sur cette phase.

1.2 Librairies introduites

1.2.1 Enregistrement du projet

Utilisation de SQLite pour enregistrer les projets, à l'aide de ORMLite, et ses dépendances (ormlite-jdbc, ormlite-core, sqlite-jdbc)

- sqlite a les avantages d'une DB relationnelle, mais enregistre dans des fichiers : facilité de déplacement des projets, pas besoin de serveur
- sqlite permet d'avoir une DB en mémoire vive : pratique pour les tests
- sqlite est utilisable dans de nombreux autres langages : augmente la productivité en permettant d'éditer les fichiers de projets dans le langage préféré de chaque dev
- ORM facilite l'accès aux données (Design pattern DataMapper <http://martinfowler.com/eaaCatalog/dataMapper.html>)
- ORMLite permet d'utiliser plusieurs types de bases de données différentes à travers JDBC. Possibilité donc d'utiliser autre chose qu'SQLite si nécessaire.

1.2.2 Librairie GUI

Les requirements de l'interface graphique sont les suivants

- Afficher une fenêtre esthétique
- Mettre des menus / boutons
- Incorporer une vue 2D/3D
- Compatible avec la lib 3D choisie

Les différentes possibilités de GUI sont :

- AWT (Abstract Window Toolkit)

- Swing
- SWT (Standard Widget Toolkit)
- JavaFX
- Apache Pivot
- Qt Jambi

Swing

Nous avons retenu **Swing**.

- Customisable : Oui
- Léger : Non
- Cross-Platform : Oui
- Documentation : Complet
- Faiblement couplé
- Suit le design pattern MVC
- Compatible avec jMonkeyEngine

Toutes ces raisons font qu'on utilisera Swing comme librairie pour construire notre interface graphique. Voici un aperçu des autres libraires éligibles, et les raisons pour lesquelles elles ont été refusées.

AWT

- Customisable : Non
- Léger : Oui
- Cross-Platform : Oui
- Documentation : Complet
- AWT utilise les objets du système
- Swing hérite des objets de AWT

On n'utilisera pas AWT car il n'est pas assez customisable.

SWT

- Customisable : Non
 - Léger : Oui
 - Cross-Platform : Oui
 - Documentation : Très complet
 - Développé par l'équipe d'Eclipse
 - Se place un peu comme premier concurrent de Swing
- On n'utilisera pas SWT car il n'est pas assez customisable.

Java FX

On n'utilisera pas Java FX car ils se sont spécialisés dans les interfaces d'application web et ce n'est pas forcément utile ici.

Apache Pivot

On n'utilisera pas Apache Pivot pour les mêmes raisons que Java FX.

Qt Jambi

- Customisable : Oui
 - Léger : Non
 - Cross-Platform : Oui mais limité à QT4.6
 - Documentation : Mauvaise
 - Intégré à Eclipse
 - Assez complet
 - Accès a une database sql incluse
- Inconvénients :
- Pas de doc sur le site QtJambi
 - Compliqué pour l'intégration de la 3D . Il existe Qt3D mais on s'est dirigé vers Jmonkey
 - Bloqué a la version 4.6

1.2.3 Librairie 3D

Nous choisissons d'utiliser jMonkeyEngine comme librairie 3D pour ces raisons :

- Elle est sous license BSD
- Une des interfaces les plus high level
- Community driven
- Développement encore actif
- Documentation extensive
- Engine complet
- Refonte complète de JME2 pour le mieux
- Intégration facile dans un GUI swing : http://hub.jmonkeyengine.org/wiki/doku.php/jme3:advanced:swing_canvas

Justification de l'écartement d'autres librairies :

JOGL :

- Utilise SWT comme système de fenêtrage

GL4Java :

- Vieux et obsolète

Java3D :

- Abandonné

Ardor3D :

- Prise en main difficile et risque de requérir beaucoup plus de temps

1.3 Rapport de fin d'itération

Le développement de l'itération 1 s'est bien déroulé. La charge de travail a été correctement répartie le long des 3 semaines de développement, tout le monde a pris part de manière équilibrée au travail et on ne déplore aucun conflit.

1.3.1 Suivi du planning

Toutes les sous-histoires de l'itération 1 ont été implémentées et nous estimons avoir réussi à le faire tout en gardant une architecture propre.

Le temps étant court et la fonctionnalité demandée étant plutôt grande, nous avons d'abord commencé en petits groupes pour la découverte des outils et librairies que nous allions utiliser. Nous avons ensuite essayé de tout mettre ensemble pour tester le produit complet à un stade pré-alpha.

C'est à ce moment que nous avons effectué un premier refactoring pour pouvoir tous continuer sur les différents aspects du projet dans une architecture propre.

Timesheet

Tâche	Bruno	Florentin	Julian	Pierre	Titou	Walter
Management	6,00	19,00	6,00	6,00	6,11	6,00
Découverte lib 3D	7,00	5,75				
Compilation projet Eclipse/Ant			1,00	1,00	5,75	
Découverte GUI (swing)			5,00	8,33		1,00
Model / Découverte ORMLite					4,00	
Refactoring/Debug		8,00		11	3,00	
1.1 Enregistrement d'un projet					18,00	
1.2 GUI	6,33		4,00	25,83	5,00	4,00
1.3 Affichage du monde 2D/3D	15,00	11,00	9,00		2,00	12,00
1.4 Navigation dans le monde 3D	12,00					2,00
1.5 Modif. de la géométrie des pièces		8,00			6,00	12,00
1.6 Création d'un demo project						
1.7 Enregistrement automatique				3	2,00	
1.8 Intégration GUI+3D	4,00	2,00	3,00	1,00	5,00	

1.3.2 Architecture

Architecture générale

On voit directement l'utilisation du MVC, couplé avec un design pattern Observable et un DAO. Cette architecture nous permet de :

- rajouter/supprimer une vue (+ contrôleur) très facilement
- garder toutes les vues à jour
- découpler un maximum toutes les parties de l'application
- garder l'état du projet entre deux sessions d'utilisation

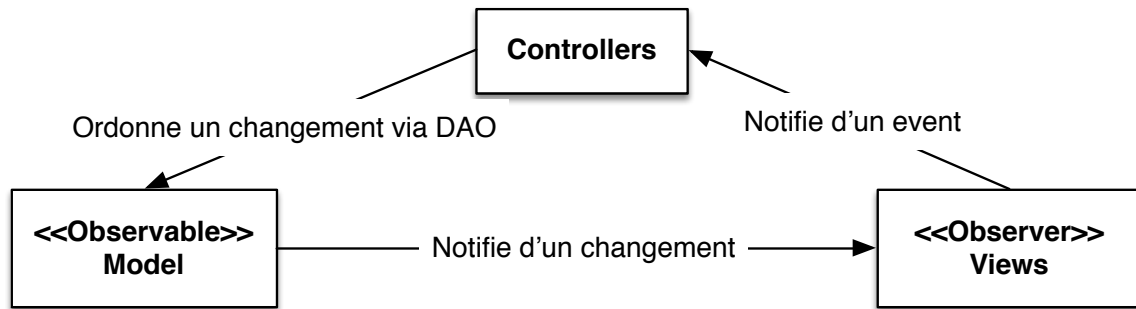


FIGURE 1.1 – Diagramme de composants général

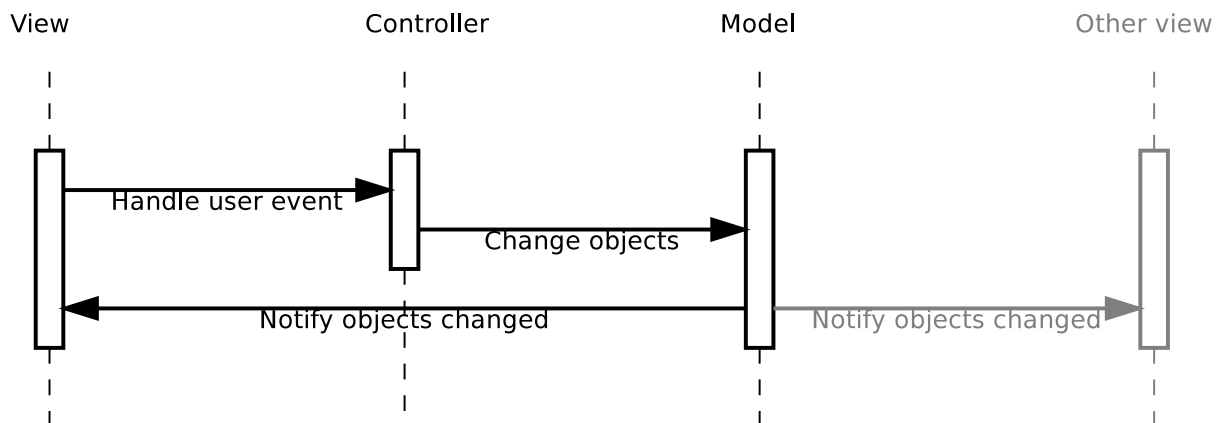


FIGURE 1.2 – Diagramme de séquence MVC

Prenons un exemple : le passage d'une vue 3D à une vue 2D et inversement. Ce passage a des implications dans plusieurs vues. La première est bien évidemment l'éditeur général, qui doit afficher le monde différemment en fonction du mode ; la deuxième est la toolbar, qui permet de changer de mode.

Quand on clique sur le bouton 2D/3D, la `ToolBarView` notifie son contrôleur d'un event reçu. Le contrôleur utilise alors le DAO pour enregistrer que le mode a été changé. Le modèle a en effet une valeur de configuration qui stocke le mode. Ce dernier notifie alors toutes les vues concernées du changement (via le pattern Observer), dont l'éditeur principal qui va alors changer son mode d'affichage.

Architecture du modèle

Le modèle est architecturé selon le design pattern Data Access Object. La classe `Project` a la responsabilité de l'accès au fichier du projet, et des valeurs de configurations générales. Il donne en outre accès au `GeometryDAO`, qui est un singleton pour le projet, et qui permet d'effectuer les opérations CRUD¹, ainsi que des recherches d'objets selon

1. Create Refresh Update Delete

différents critères.

Les objets enregistrables par le **GeometryDAO** implémentent tous l'interface **Geometric**. On distingue 4 catégories d'objet dans le modèle (Figure 1.3) :

- Les **Point** représentent une position dans l'espace
- Les **Shape** définissent des formes bidimensionnelles composées de **Point**
- Les **Grouped** définissent des constructions sur les **Shape**
- Les **Floor** permettent de grouper ces éléments par étage.

La séparation de la forme et des constructions nous permet d'associer facilement différents éléments de la pièce (murs, sols, plafonds) qui sont bâtis à partir des mêmes points. En outre, plusieurs pièces peuvent contenir le même point, facilitant ainsi la détection de pièces adjacentes, le déplacement d'un coin de mur entre plusieurs pièces, ...

Architecture de la GUI

L'interface graphique est composée de plusieurs vues qui ont chacune leur contrôleur propre. La classe GUI est celle qui gère la fenêtre et les composantes de premier niveau :

- la barre de menus
- les popup de sélection de fichiers
- la barre d'outils
- l'éditeur principal (qui gère la vue 3D/2D et la TreeView)

La **WorldView** est une sous-classe d'une **SimpleApplication** **jMonkey**. C'est le canevas 3D à proprement parler.

1.3.3 Bonnes pratiques utilisées

Plusieurs pratiques ont été mises en place pour faciliter le développement en groupe, comme par exemple le pair programming ou les sprints d'une journée.

Pair programming

À plusieurs moments, nous avons travaillé par deux sur un même ordinateur pour les problèmes plus épineux. Un développeur écrivait du code, et l'autre essayait de le corriger et de penser aux edge cases en même temps. Cela nous a permis de partager beaucoup plus facilement des idées et de compléter des features complexes de manière rapide et robuste.

Sprints d'une journée

En déployant un outil de statistiques de notre repository **Git**, on voit que le samedi est le jour de la semaine où l'équipe commite le plus. La raison de ce pic est simple : à deux reprises lors de cette itération, nous nous sommes retrouvés ensemble physiquement, autour d'une grande table, pour travailler sur le projet.

Au début de la matinée, nous faisons une réunion pour établir les objectifs de la journée puis nous nous lançons tous sur la production de features. C'est lors de ces sprints que l'avancement était le plus marqué et que nous pouvions prendre du recul facilement sur ce qui avait déjà été fait, et ce qu'il restait à faire.

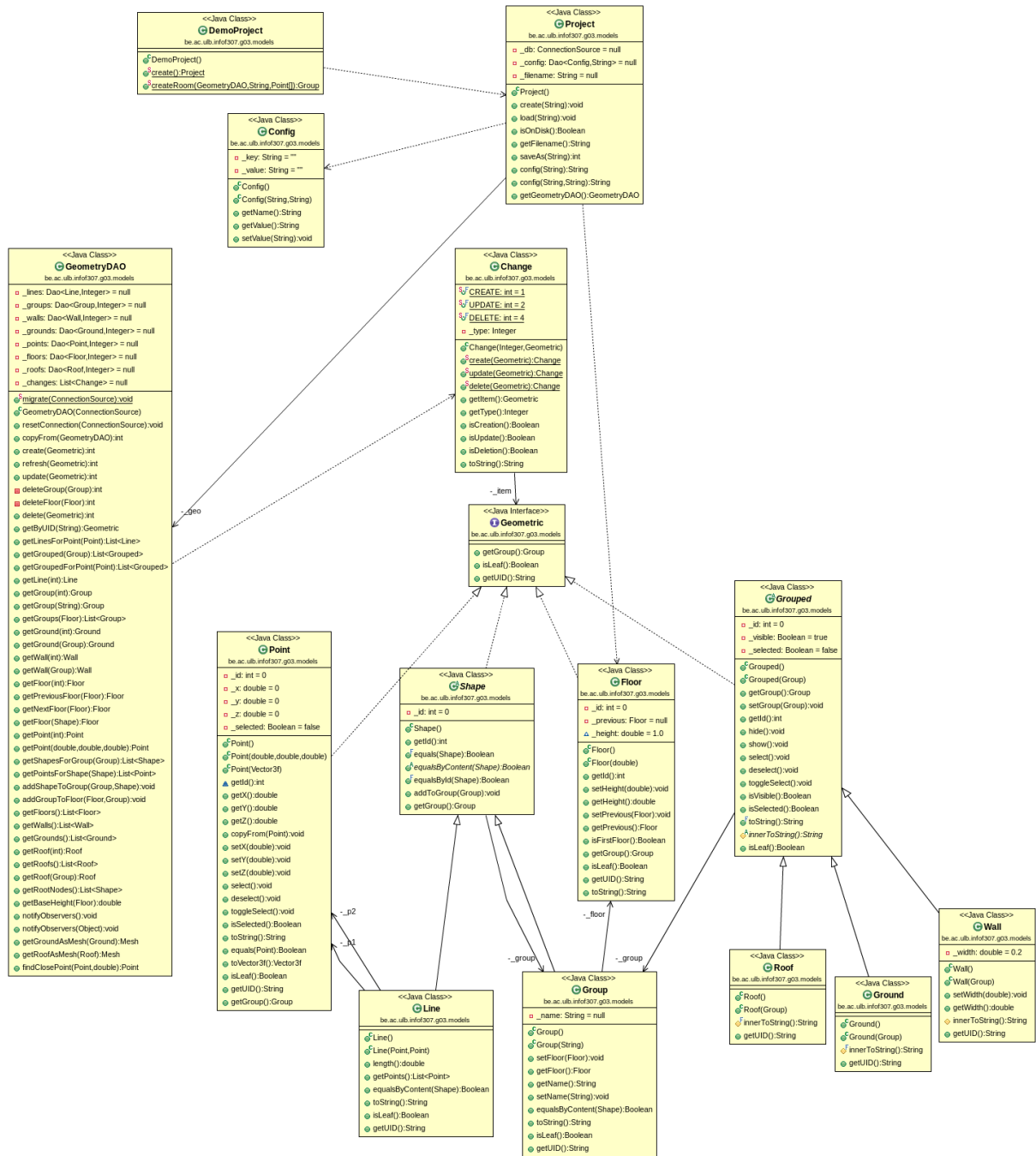


FIGURE 1.3 – Diagramme de classes du modèle

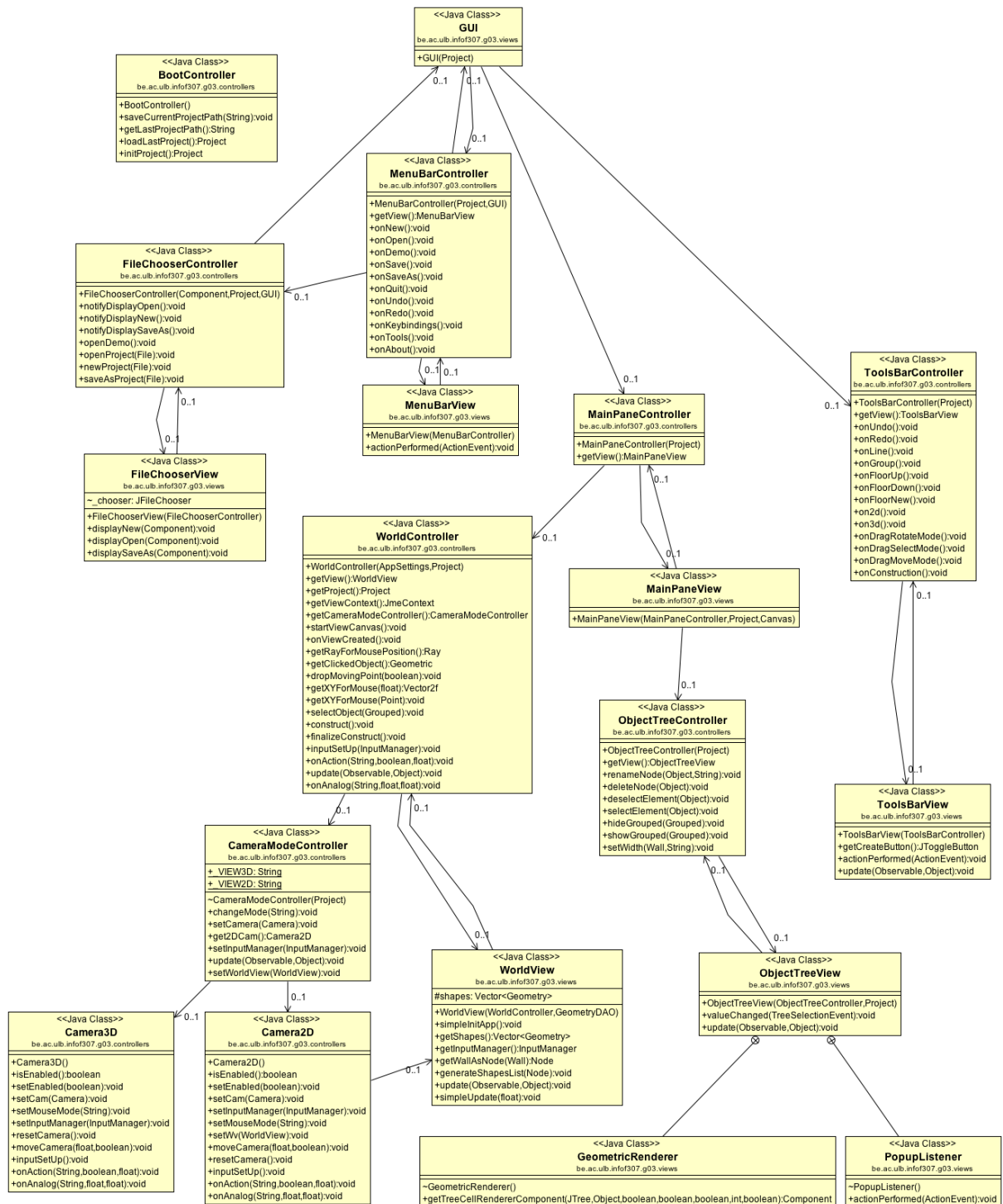


FIGURE 1.4 – Diagramme de classes de la GUI

Développement itératif/fractal

Cette pratique a été adoptée dès le début, mais nous nous sommes rendus compte vers la moitié de l'itération que nous l'utilisions mal. Le principe de cette technique est de considérer chaque feature comme une unité qu'on peut développer à plusieurs niveaux de *perfection*.

Nous nous sommes forcés de produire très vite des proofs of concept, ou Minimum Viable Products pour les unités à développer. Cette unité était alors souvent à un stade de fonctionnalité très pauvre, dont l'architecture n'était pas forcément bien pensée. Il fallait alors réécrire, rajouter ou déplacer du code pour améliorer le point de vue utilité autant que le point de vue de beauté interne du code, afin d'arriver à une unité fonctionnelle complète et qui s'intégrait bien dans l'architecture de l'application.

Toutefois, lors de la première moitié de la première itération, nous sommes souvent tombés dans le piège du "*perfectionnement d'abord*", en passant beaucoup de temps sur le perfectionnement utilitaire de l'unité. Nous aurions dû passer plus vite - une fois que l'architecture d'une unité était bien intégrée dans l'application - au développement des autres unités.

Test-Driven Development

Cette technique, qu'il ne faut plus expliquer, a été utilisée, surtout pour le développement du modèle et s'est effectivement avérée positive lorsqu'il a fallu se baser sur un modèle robuste.

Staging area

Nous avons créé une branche **stage** sur laquelle nous testions la version de développement la plus récente de l'application. De manière régulière, nous passons tous les changements de **stage** en production sur **master**. La condition de mise en production était simple : tout le code doit être documenté, et tout ce qui peut être testé doit être testé.

Couverture de la qualité du code

Des outils d'évaluation de la qualité du code ont été utilisés. Parmi eux, on peut en noter trois :

EclEmma Cet outil nous a permis d'évaluer facilement la couverture des tests dans l'application et nous permet d'estimer visuellement très rapidement cette dernière.

Missing Javadoc (Eclipse) On peut configurer Eclipse pour afficher des warnings là où le code n'est pas documenté, ce qui est très pratique.

PMD Plusieurs d'entre nous ont installé PMD en fin d'itération pour évaluer le respect des conventions de code. Nous n'avons cependant pas pris le temps de "réparer" les quelques erreurs mises en évidence.

Continuous integration

Nous avons choisi d'utiliser Travis CI² pour tester tous les commits pushés sur le dépôt, et ainsi assurer l'intégrité de la compilation et des tests, et d'être notifiés des erreurs éventuelles. Ant³ nous permet de compiler le projet hors d'Eclipse.

Réunions hebdomadaires

Nous avons réussi à tenir notre objectif d'une réunion hebdomadaire. Le bilan de ces réunions est positif; elles nous aidaient à prendre du recul sur notre progression et à remettre tout au clair.

1.3.4 Réflexion sur les librairies choisies

Généralement, nous sommes satisfaits de nos choix.

jMonkeyEngine

Même si elle est plutôt facile à utiliser, nous nous sommes rendus compte que la documentation et les ressources disponibles pouvaient parfois être rares. Une grande partie de ce que nous avons trouvé venait directement du site officiel de jMonkeyEngine qui, en soi, est très complet, mais qui est parfois lacunaire sur certains sujets.

En prenant un peu de recul, nous nous rendons compte que nous utilisons des fonctionnalités d'assez bas niveau et que nous aurions pu nous satisfaire de moins.

Swing

Swing reste un excellent choix, nous n'avons pas à nous plaindre.

1.3.5 Conclusion - What's next

La fonctionnalité atteinte lors de cette itération est satisfaisante. Il en est de même pour la stabilité de la release. Il reste toutefois quelques légers problèmes d'expérience utilisateur, et quelques bugs mineurs persistent.

Nous avons déjà prévu de commencer l'itération par une revue du code et par un refactoring en tout cas du modèle. Nous avons par exemple fait le choix de pouvoir imbriquer des groupes dans des groupes (structure récursive), ce qui implique une trop grande complexité dans le **GeometryDAO**, n'est pas en adéquation avec les principes d'une base de données relationnelle, et n'est pas utilisé.

2. <http://travis-ci.org>

3. <http://ant.apache.org/>