UMONS
Université de Mons

Faculté des Sciences

Rapport du projet Pac-Man

Projet réalisé dans le cadre
de la 1ère Master en Sciences Informatiques
pour le cours de « Software Evolution »

Réalisé par

CAMBIER	Robin	ROBIN.CAMBIERR@student.umons.ac.be
OPSOMMER	Sophie	SOPHIE.OPSOMMER@student.umons.ac.be

Faculté
des Sciences

Sous la direction de: *Titulaire* : T. MENS
Assistant : M. CLAES

Année Académique 2014-2015

ACADÉMIE
UNIVERSITAIRE
WALLONIE-
BRUXELLES

Table des matières

Résumé

Ce *rapport* est rendu dans le cadre du cursus de première année de « Master en Sciences Informatiques »pour le cours de *Software Evolution* (dont le titulaire est Mr. *T. Mens* et l'assistant est Mr. *M. Claes* en année académique 2014-2015) . Le but de ce rapport est de présenter les résultats de la réalisation du projet Pac-Man.

1 Introduction

1.1 Problème posé

A partir d'un code existant, ce projet consiste à :

- analyser la qualité du logiciel, en utilisant des techniques d'analyse statique du code (par exemple, la détection du code dupliqué et des bad smells, les diverses métriques de qualité) et les outils d'analyse dynamique du code (par exemple, le profilage, la couverture du code et des tests) ;
- améliorer la qualité et la structure du code (en utilisant des refactorings, en introduisant des design patterns, et en modularisant le code) ;
- étendre le logiciel avec de nouvelles fonctionnalités (évolution), et étudier l'effet de cela sur la qualité du code ;
- tester le logiciel avant et après chaque modification. Ceci implique que vous devez ajouter des tests unitaires (unit tests) pour au moins les fragments du code modifiés ou ajoutés, et d'appliquer des tests de régression à chaque modification.

1.2 Etapes clés

Les étapes clés du projet sont les suivantes : (chronologiquement)

1. Analyse de la qualité de la premiére version du code (section **??**)
2. Ajout de tests unitaires à la première version du code (section ...), et vérification de la couverture des tests
3. Refactoring du code pour en améliorer la qualité et la structure (section ...)
4. Analyse des améliorations de qualité et tests de régression (section ...)
5. Extension du logiciel et ajout des tests unitaires pour cette extension (section ...)
6. Analyse de la qualité de cette extension et tests de régression (section ...)
7. Etude de l'historique de la qualité logicielle entre toutes les versions du code (section ...)

2 Etape 1 : Première analyse de la qualité du logiciel

2.1 Enoncé

Le but de la première étape clé est d'analyser la qualité du logiciel pour avoir une première idée de ce qu'il faudra corriger si l'on désire améliorer la qualité. Cette première analyse est également l'occasion de comprendre l'architecture et la dynamique du système. Pour cette étape clé il est demandons de :

1. Calculer la valeur de plusieurs métriques logicielles permettant d'estimer la qualité du logiciel, d'interpréter les résultats des métriques, et de mettre en évidence les modules (packages, classes ou méthodes) qui devraient être traités prioritairement afin d'améliorer leur qualité ainsi que la raison pour laquelle ils sont prioritaires.
2. Déterminer les classes et les méthodes qui sont couvertes par des tests unitaires, et mettre en évidence les méthodes pour lesquelles de nouveaux tests unitaires devraient être créés prioritairement.
3. Répertorier les portions de code qui ne sont pas utilisées et qui pourraient donc être supprimées sans altérer le comportement du système.
4. Répertorier les portions de code qui sont redondantes (code dupliqué) et qui pourraient donc être éliminées par une restructuration du système sans altérer son comportement.
5. Détecter la présence de bad smells. En trouvez-vous une plus forte concentration dans certains modules ?
6. Analyser les performances du système en terme d'utilisation CPU et de consommation de mémoire. Repérez les parties du code créant un goulot d'étranglement et précisez les modules qui devraient être retravaillés afin de procéder à un déoulottage du système.

Décrivez la qualité globale du système. Quel serait, selon vous, le coût nécessaire à sa maintenance et à son évolution ?

2.2 Résultat

Voici le tableau des résultats de l'analyse des différentes métriques :

Métrique	Outil utilisé + Résultat Interprétation des résultats Modules à traiter prioritairement + justification
Techniques d'analyse statique du code	
Code dupliqué	CodePro Tools -> Find Similar Code : 10 blocs de code (visible sur la figure A (page 10) et les suivantes) sont semblable. Ce résultat n'est pas alarmant mais mérite d'être refactoré. Les classes Tile.java, GameTest.java, ButtonPanel.java, MainUITest.java, MapParser.java, PacmanKeyListener.java,PacmanKeyListener.java, PacmanKeyListener.java, GameTest.java,Board sont à traiter puique c'est elle qui contiennent du code dupliqué.
Respect des conventions de codage	
Dépendences cycliques	
Performances	
Structure du code	
Style du code	
Design flaws	
Antipattern	
Test	
Dataflow	
Documentation	
Outils d'analyse dynamique du code	
Profilage	
Couverture du code	
Tests	

NB : - slide 4 : outils

Images : - $Mtrique_0.png$ -

3 Etape 2 : Ajout de tests unitaires

3.1 Enoncé

Votre première analyse a révélé la présence de certains problèmes de qualité du code de l'application. Avant d'envisager la correction de ces problèmes, il faut s'assurer que les modifications que vous apporterez au code source ne modifieront pas le comportement du logiciel. Etendez et complétez le jeu de tests unitaires fourni avec le code source afin de vous prémunir d'une telle modification. Effectuez également une analyse de couverture de tests. Quelle garantie avez-vous que vos futures modifications ne pourront pas casser le système ?

3.2 Résultat

4 Etape 3 : Refactoring en vue d'améliorer la qualité

4.1 Enoncé

Avec les tests unitaires ajoutés dans l'étape précédente, vous pouvez vérifier automatiquement (jusqu'à un certain point) la préservation du comportement du logiciel. Réalisez les modifications nécessaires à l'amélioration de la qualité et la structure du logiciel. Vos ressources et votre temps étant limités, commencez par établir les modifications devant être réalisées en priorité. Sur base de quels critères réalisez-vous cette priorisation ? Refactorisez progressivement votre code, en vous assurant systématiquement que tous les tests déjà présents s'exécutent avec succès. Souvenez-vous que vos modifications doivent améliorer la qualité du code, et non étendre ou modifier le comportement du logiciel.

4.2 Résultat

5 Etape 4 : Analyse de la qualité du logiciel

5.1 Enoncé

Réalisez une étude similaire à celle décrite en Section ... La qualité du logiciel s'est-elle améliorée ? Les problèmes les plus critiques ont-ils été résolus ? Au vu de cette seconde analyse, quels sont les points qui devraient à

présent être améliorés ?

5.2 Résultat

6 Etape 5 : Extensions

6.1 Enoncé

Il vous est demandé d'étendre le logiciel afin d'y ajouter certaines fonctionnalités ou d'en améliorer la qualité. Chaque équipe doit réaliser au moins deux extensions diférentes, décrites dans la section ... . Utilisez un processus de développement dirigé par les tests (test-driven development) : lors du développement des extensions, ajoutez de nouveaux tests unitaires pour tester le comportement prévu de l'extension. Effectuez également des tests de régression avec les tests unitaires déjà présents, afin de vous assurer que le comportement initial n'a pas été modifié.

6.2 Résultat

7 Etape 6 : Analyse de la qualité du logiciel

7.1 Enoncé

Pour chaque extension ajoutée, réalisez une analyse de qualité similaire à celle décrite en Section ... Au vu de cette analyse, quels sont les points qui devraient à présent être améliorés ?

7.2 Résultat

8 Etape 7 : Analyse de l'évolution de la qualité logicielle

8.1 Enoncé

Analysez l'évolution de la qualité du logiciel entre les diférentes versions, en utilisant les résultats d'analyse de qualité des sections ..., ... et ... . Montrez cette évolution graphiquement et interprètez-la.

8.2 Résultat

Quelque exemple d'utilisation. *"Un peu d'italique"* **Du Gras**. Pour séparer deux paragraphe il suffit de mettre deux enter (une ligne blanche en gros)

Et voila un nouveau paragraphe :)
On peut egalement simplement revenir en arriere comme ca

Une petite liste a puce ? numéroté ?

- V est un ensemble fini de noeuds
- E est un ensemble d'arcs reliant deux noeuds

1. Remplacer la valeur de la racine par celle du dernier noeud, celui qui sera le plus à droite de la dernière ligne (le "1" dans l'exemple de la figure **??**(a)).
2. Supprimer ce dernier noeud
3. Faire redescendre tant que nécessaire la nouvelle racine.

Une image ? Avec une reference dans un texte ? no probleme :p

Un graphe orienté est défini par un couple : $G = (V, E)$. La figure **??** illustre un graphe.

La complexité c'est pas un simple O(lg n) mais $\mathcal{O}(\log_2 n)$.

un petit titre qui n'apparait pas dans la table des matiere

blablabla

encore un plus petit titre

blablabla

Un algorithme ? ouille ouille ouille :p mais on si habitue. Le caption sera le titre visible et le label sera ce que tu devra utiliser pour le referencer comme ca Alogo **??**

Des theorme, lemme, propriete ? sa marche aussi :). De nouveaux tu peux reference le label :) Lemme 1

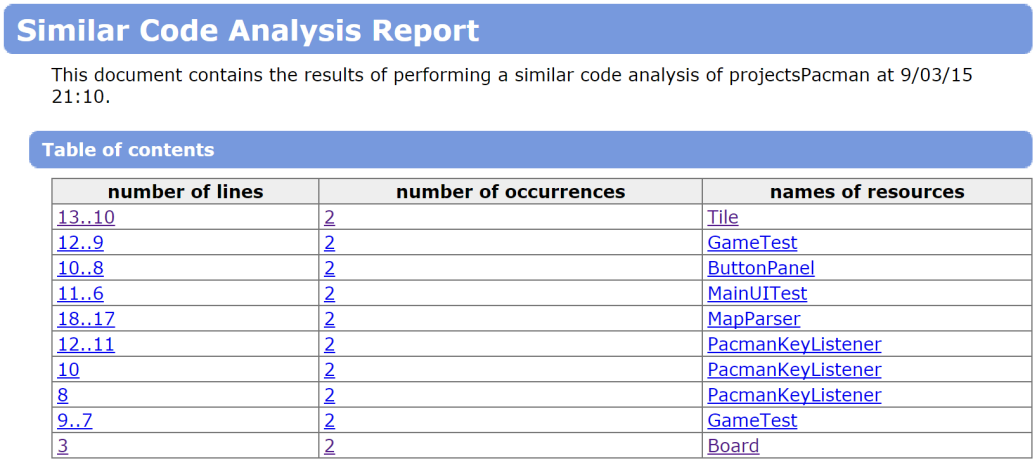
Lemme 1 (Propriété de borne supérieur)**.** *Nous avons à tout moment* $v.d \geq \delta(s, v) \forall v \in V$*, et une fois que* $v.d$ *atteins* $\delta(s, v)$ *il ne changera plus.*

Corollaire 1 (Propriété d'absence de chemin)**.** *Si il n'existe pas de chemin allant de* s *à* v *alors nous avons à tout moment* $v.d = \delta(s, v) = \infty$*.*

Propriété 1 (Propriété de convergence)**.** *Si $s \rightsquigarrow u \rightarrow v$ est un plus court chemin dans G pour $u, v \in V$ donné et si $u.d = \delta(s, u)$ avant que l'arc (u, v) ne soit relaxé alors $v.d = \delta(s, v)$ après la relaxation.*

Théorème 1 (Thm de convergence)**.** *Si $s \rightsquigarrow u \rightarrow v$ est un plus court chemin dans G pour $u, v \in V$ donné et si $u.d = \delta(s, u)$ avant que l'arc (u, v) ne soit relaxé alors $v.d = \delta(s, v)$ après la relaxation.*

des symbole grec, mathematique $\delta(a), \sigma(a), \neq, \leq, \geq, ...$
Ca doit etre entre dollard. Pareil pour toutes les variables, formule, ...
on n'ecrit pas l'index i mais l'index i



Similar Code Analysis Report

This document contains the results of performing a similar code analysis of projectsPacman at 9/03/15 21:10.

Table of contents

number of lines	number of occurrences	names of resources
13..10	2	Tile
12..9	2	GameTest
10..8	2	ButtonPanel
11..6	2	MainUITest
18..17	2	MapParser
12..11	2	PacmanKeyListener
10	2	PacmanKeyListener
8	2	PacmanKeyListener
9..7	2	GameTest
3	2	Board

FIGURE 1 – Résultat de l'analyse de code redondant par CodePro

9 Annexes

10 Annexe : 1

Dans cette section se trouve les différentes annexes qui permettent d'identifier les blocs de dode dupliqués détecté par CodePro. La première image répertorie les 10 blocs. De la seconde à l'avant dernière, les images illustres les blocs de code. La dernière image illustres les 6 derniers blocs de code et est issue du rapport généré par CodePro parce que Eclipse les masque.

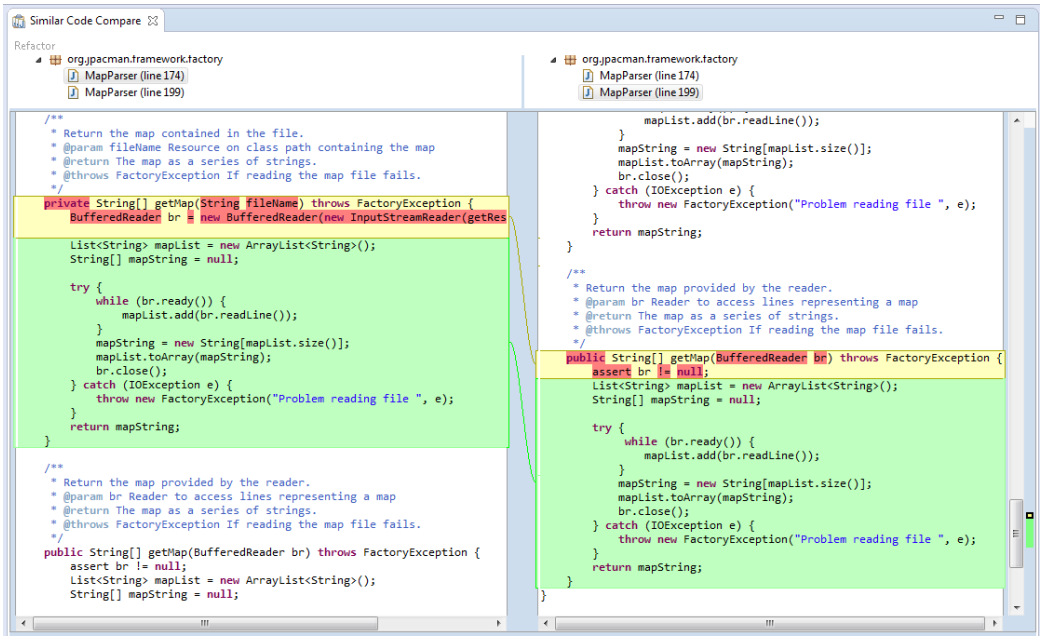


FIGURE 2 – Détail de l'analyse de code redondant par CodePro

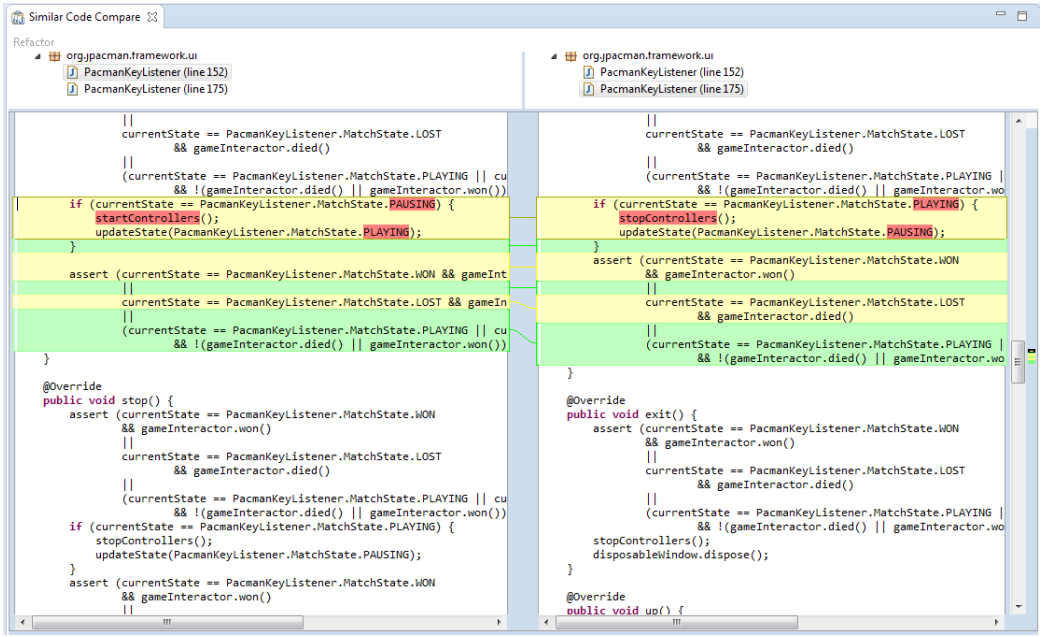


FIGURE 3 – Détail de l'analyse de code redondant par CodePro

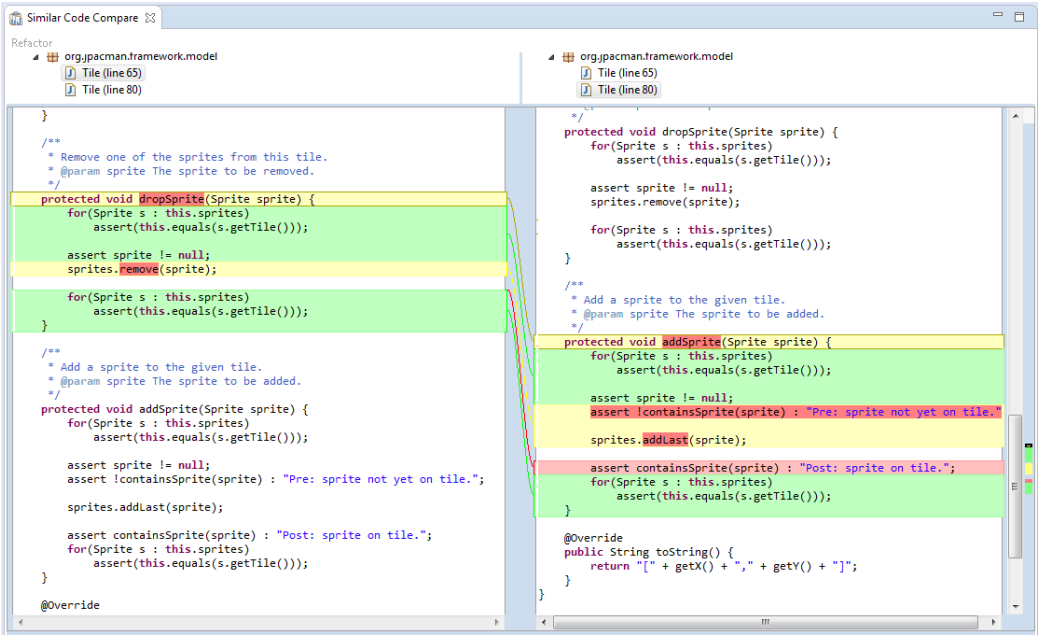


FIGURE 4 – Détail de l'analyse de code redondant par CodePro

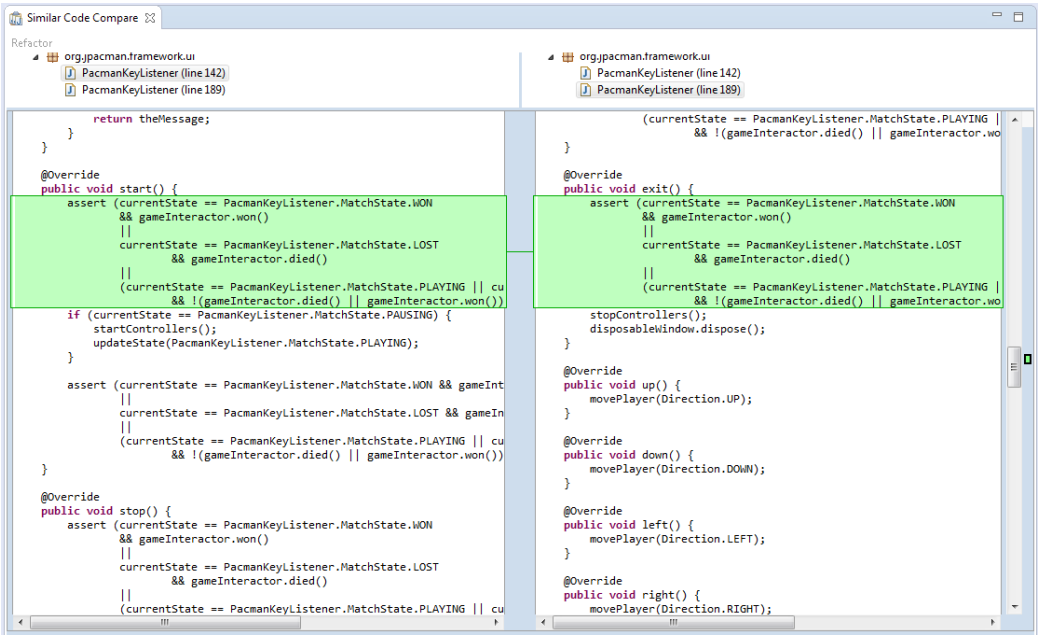


FIGURE 5 – Détail de l'analyse de code redondant par CodePro

FIGURE 6 – Détail de l'analyse de code redondant par CodePro (élément dont la visualisation n'est pas possible dans Eclipse)

11 Annexe : 2

Références