Rapport de projet : JPacMan
Partie 2 : Intégration et analyse de qualité

Software Evolution

Groupe 7 :

MILLEMACI Mikis

VULLO Amaury

Année Académique 2015-2016
Bloc complémentaire en Master Informatique
Faculté des Sciences, Université de Mons

Table des matières

1. Intégration des extensions

Pour intégrer les extensions liées à notre groupe, nous avons pris comme dépôt de base celui de Vullo Amaury et nous avons effectué un pull request du dépôt de Millemaci Mikis.
Une fois le pull request accepté de la part de Vullo Amaury, l'intégration s'est parfaitement faite car il n'y avait aucun conflit entre nos deux projets.

Pour clarifier, Amaury a réalisé l'extension « Score » et Mikis a développé l'intelligence artificielle pour les fantômes. A la base, nous étions un groupe de trois personnes mais le dernier membre du groupe a visiblement abandonné le projet et suite à l'accord de Mr. Mens, nous avons seulement mis en commun nos deux extensions.

Dépôt de Vullo Amaury : https://github.com/momomiami/jpacman-framework

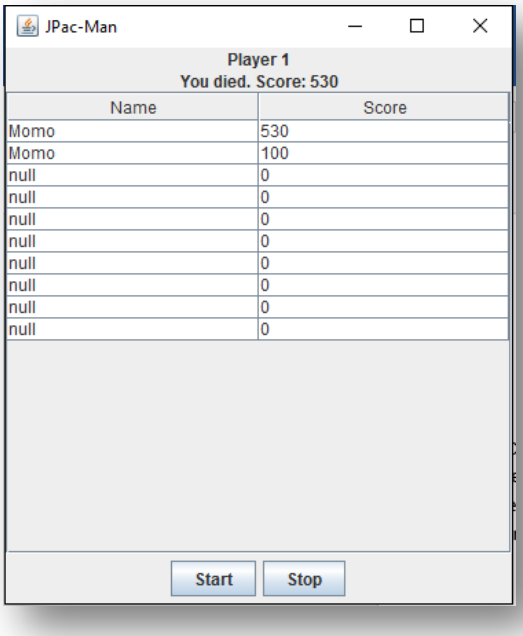
Dépôt de Millemaci Mikis : https://github.com/Soupline69/jpacman-framework

2. Explication des extensions

2.1. Score

2.1.1. Hall of fame

Dans un premier temps, le but de l'extension est de permettre la possibilité de garder en mémoire les précédents scores de sorte à les inscrire dans un tableau des scores qui s'affiche en fin de partie. Il était aussi demandé d'ajouter les fruits pour pouvoir augmenter le score. Chaque fruit apporte un montant de point différent.



Ce tableau des scores devait être sérialisé car il doit s'afficher en fin de partie avec les scores des précédents joueurs. Il fallait également permettre au joueur de pouvoir réinitialiser le tableau des scores à l'aide d'un bouton disponible dans le menu.

FileSerialization.java

La classe est principalement responsable de rendre des services comme lecture/écriture dans un fichier de données. Les classes qui sont sérialisées sont le profil et le tableau des scores. Le profil dans un fichier qui a pour nom le créateur du profil et le tableau des scores dans le fichier « HallOfFame.ser ».

HallOfFameTable.java

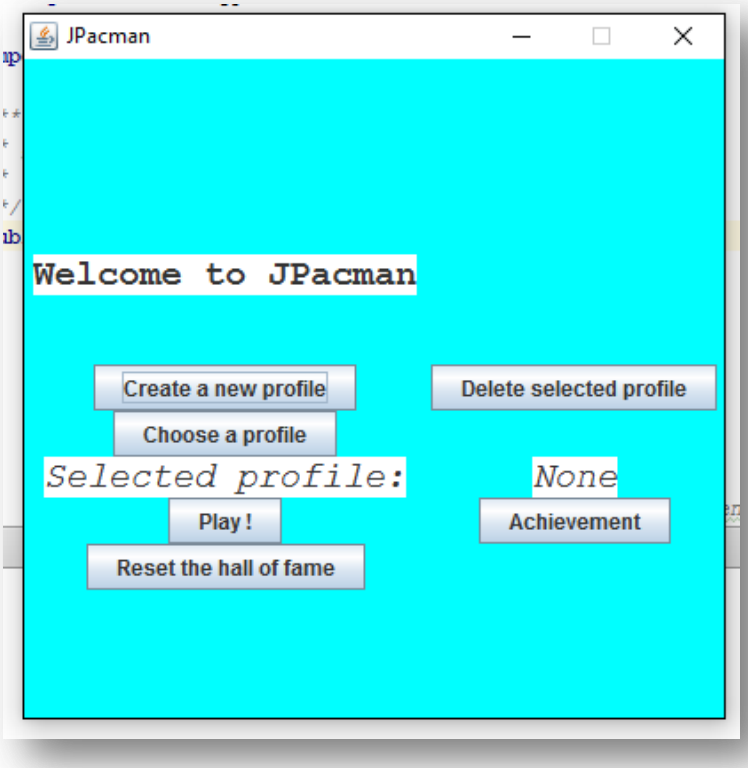
La classe est responsable de créer la JTable swing dans laquelle les informations relatives au tableau des scores sont introduites. La JTable est donnée au JPanel après la fin de la partie.

HighScoreBoard.java

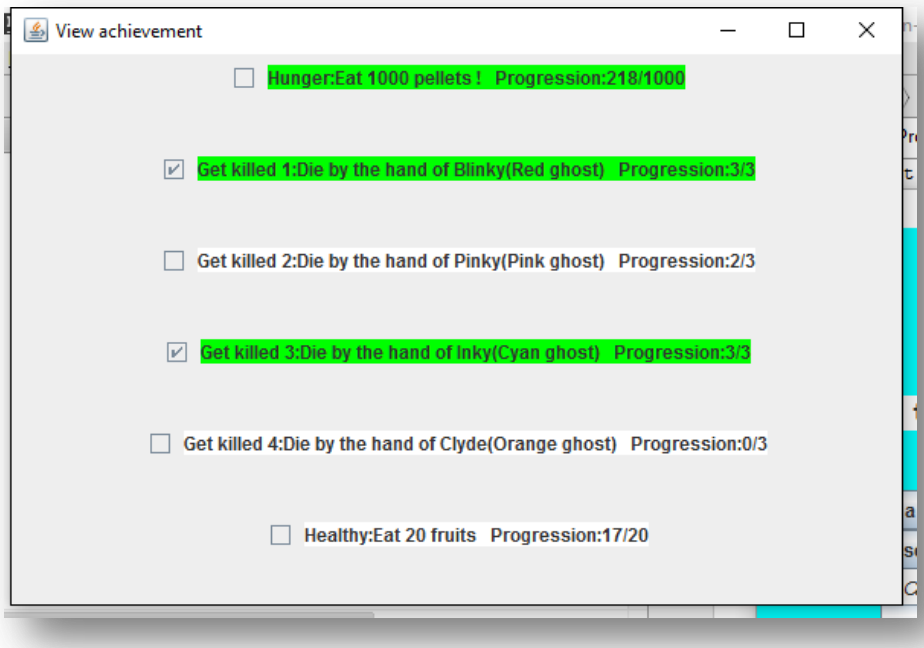
La classe fournit toute la logique derrière la gestion du tableau des scores (insertion, suppression et tri).

2.1.2. Hauts faits/ Défis

Pour cette extension, il est demandé de pouvoir créer un profil. Ce profil peut être utilisé par un joueur pour sauvegarder son état d'avancement pour des haut-faits/défis réalisés en cours de jeu. Il est possible de vérifier l'état de son profil à partir du menu principal. Le menu principal permet aussi les opérations CRUD basique sur les profils.



Trois haut-faits peuvent être actifs à la fois et ceux déjà réalisés sont cochés :



Achievement.java

Ceci est la super classe responsable de modéliser un haut-faits. Pour ajouter un haut-fait il suffit de créer une sous-classe de celle-ci en remplissant le constructeur de la super classe avec les valeurs adéquates. La classe permet de faire avancer la progression du haut-faits.

Profile.java

Classe responsable de modéliser le profil d'un joueur. Ce profil est caractérisé par un nom et par la liste de haut-faits.

2.2. IA pour fantômes

Il était demandé de réaliser une extension permettant aux monstres de se déplacer selon une certaine stratégie (qui pouvait changer au cours de la partie).

2.2.1. Mise en place de l'extension

Strategy.java

Pour implémenter cette extension, une classe abstraite « Strategy » a été implémentée. Elle devra être étendue par la stratégie « concrète » pour que celle-ci puisse être créée.
Elle possède un attribut « Board » représentant la carte du niveau. Cet attribut permettra de créer une stratégie selon les différents points importants de la carte (comme par exemple, savoir où se trouve la maison d'un fantôme). Cet objet « Board » sera instancié dès la création de la stratégie puisque pour être créer, une stratégie doit connaître la carte du niveau.
Elle possède également une méthode « move » qui renverra le prochain mouvement du fantôme. Elle devra être réécrite par la stratégie concrète qui définira le comportement d'un fantôme.

StrategyFactory.java

Cette classe instancie toutes les stratégies et les stocke dans une « Map », pour que le programme puisse ensuite les retrouver grâce à leurs noms.

ScatterStrategy.java

Stratégie définissant le comportement des fantômes lorsque ces derniers doivent retourner dans leurs maisons.

ChaseStrategy.java

Stratégie définissant le comportement des fantômes lorsque ces derniers chassent PacMan.

Ghost.java

Le fantôme possède un attribut supplémentaire qui est de type « Strategy » et qui définira la stratégie actuelle dudit fantôme.

Clyde.java / Blinky.java / Inky.java / Pinky.java

La méthode « nextMove() » a été réécrite et demande à présent, le prochain mouvement pour le fantôme sur base de la stratégie actuelle en donnant le fantôme concerné en paramètre.

LevelFactory.java

Création de la classe « StrategyFactory » dans la méthode « createLevel » afin de connaître toutes les stratégies lors du début du niveau.

Level.java

Un attribut « Strategy » a été ajouté sous forme d'une « Map » afin que le niveau connaisse toutes les stratégies pouvant être utilisées durant celui-ci.

NpcMoveTask (dans Level.java)

C'est cette classe qui est responsable de la mise en place des stratégies pour le niveau. Elle va définir dans quel ordre les stratégies vont se succéder ou au contraire définir la seule stratégie que le niveau utilisera.

2.2.2. Ajout de nouvelles fonctionnalités

Dans le but de rendre l'extension très facile à utiliser, voyons comment ajouter de nouvelles stratégies et comment les utiliser dans un niveau.

1. Créer une stratégie « concrète » définissant le comportement de chaque fantôme l'ayant comme stratégie actuelle.

2. Avertir la Factory que cette nouvelle stratégie « concrète » existe.

3. Définir dans le niveau à quel moment utilisé cette nouvelle stratégie « concrète ».

2.3. Contribution de chaque membre

Concernant l'implémentation des extensions, chaque membre du groupe a effectué sa propre extension, tout en se concertant avec son partenaire afin d'avoir un code compatible lors de la phase d'intégration.

Du point de vue de la qualité du code source, chaque membre a essayé d'améliorer le code du mieux qu'il le pouvait en utilisant les outils expliqués dans le point 3.

En ce qui concerne le rapport, chaque membre du groupe a écrit les parties du rapport qui concerne son extension.

3. Outils utilisés

Pour vérifier la qualité du code source, nous avons utilisé les outils suivants :

3.1. Les métriques

Pour calculer les métriques de notre projet, nous avons utilisé l'outil « Metrics » intégré à eclipse. Grâce à cet outil, nous avons calculé quelques informations assez basiques sur notre projet telles que :

- Le nombre de ligne de code ;
- Le nombre de classes ;
- Le nombre d'attributs ;
- Le nombre de méthodes ;

3.2. Style et structure de code

Nous avons utilisé Codacy pour analyser le style et la structure du code (voir le commit « first and second analyse with codacy »).

Pourquoi avoir utilisé Codacy ?

- Il est très simple d'utilisation ;
- Permet de lier un projet en utilisant simplement le lien du dépôt git ;
- Met à jour automatiquement l'interface en fonction des modifications effectuées dans l'environnement de développement ;
- Les remarques/erreurs détectées sont très faciles à comprendre et à corriger (un exemple de comment corriger l'erreur est disponible) ;

3.3. Bad Smells

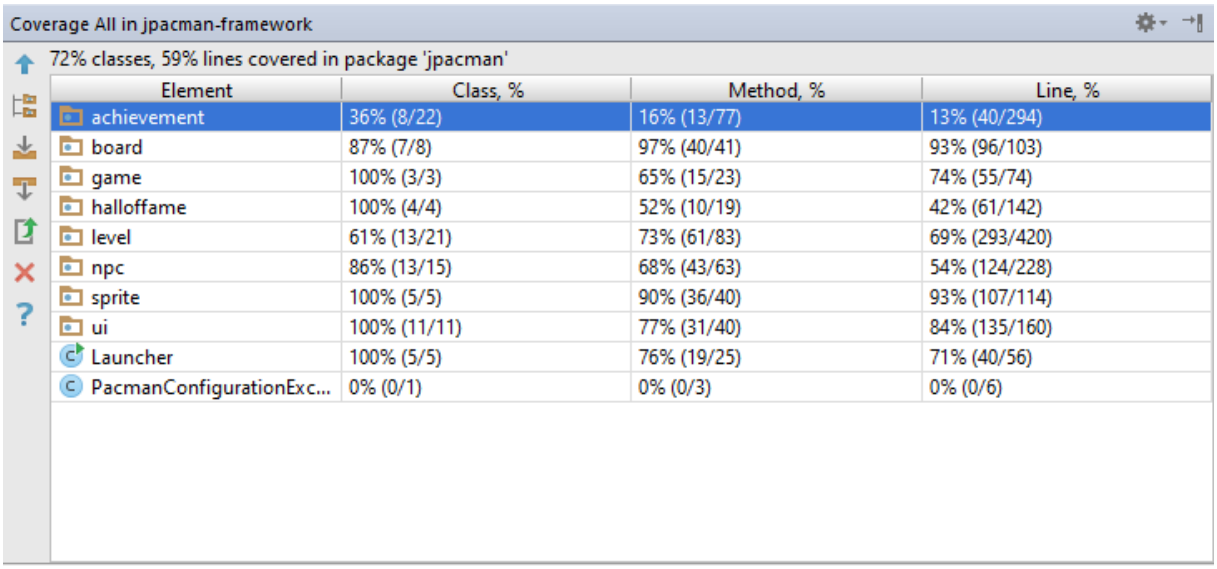
L'outil PMD (intégré avec Eclipse) nous a permis d'analyser les différents « bad smells » de notre projet et de les corriger :

- N'utilisez pas une variable globale, une variable locale suffit (voir la classe « checkAchievPanel »)
- Changez le type d'une arrayList ;

Pour plus de détails, voir le commit « apply PMD tools to the code » (la plupart des bad smells ayant été corrigés avec codacy, l'analyse PMD n'a fait que vérifier la vérification de codacy).

3.4. Code Coverage

Pour voir le pourcentage de code utilisé lorsque le programme est exécuté, nous avons utilisé le plugin propre à IntelliJ.


Coverage All in jpacman-framework

72% classes, 59% lines covered in package 'jpacman'

Element	Class, %	Method, %	Line, %
achievement	36% (8/22)	16% (13/77)	13% (40/294)
board	87% (7/8)	97% (40/41)	93% (96/103)
game	100% (3/3)	65% (15/23)	74% (55/74)
halloffame	100% (4/4)	52% (10/19)	42% (61/142)
level	61% (13/21)	73% (61/83)	69% (293/420)
npc	86% (13/15)	68% (43/63)	54% (124/228)
sprite	100% (5/5)	90% (36/40)	93% (107/114)
ui	100% (11/11)	77% (31/40)	84% (135/160)
Launcher	100% (5/5)	76% (19/25)	71% (40/56)
PacmanConfigurationExc...	0% (0/1)	0% (0/3)	0% (0/6)

3.5. Code Duplication

Pour voir les morceaux de codes identiques (ou presque), nous avons utilisé l'outil intégré à l'IDE IntelliJ. Grâce à cette analyse, nous avons pu regrouper des morceaux de codes identiques (à quelques valeurs près) en une seule fonction paramétrée.

L'extension IA pour fantôme a également pu être modifiée via cette analyse puisqu'elle a permis de déplacer la méthode « nextMove », présente dans les classes Blinky/Inky/Pinky/Clyde, dans la classe générique Ghost. La méthode « move » présente dans les stratégies concrètes ont également pu être déplacées puisqu'il était possible de l'implémenter directement dans la classe abstraite (vu que le traitement de cette méthode est la même pour toutes les stratégies). Voir le commit « method 'move' in abstract class now, not in real strategy ».

3.6. La Performance

YourKit Java Profiler a été utilisé pour mesurer les performances du programme. Il permet de se rendre compte des ressources utilisées par le programme (voir 5.3.)

<u>Pourquoi avoir utilisé YourKit Java Profiler ?</u>

- Une interface très simple à utiliser ;
- Programme très complet proposant beaucoup de fonctionnalités ;
- Permet de visualiser la liste triée des méthodes les plus « gourmandes » ;
- Permet de voir sous forme graphique la consommation du programme et des différentes mémoires ;

4. Historique du code source

4.1. Dépôt « momomiami »

4.1.1. Ajout de la classe Cherry (cerise), test ajout de la cerise grâce au map parser, creation cerise grâce au levelfactory, ajout colision pacman/cerisse

- Ajout du fruit cerise tout en respectant l'architecture originale de l'application

4.1.2. Ajout de la classe qui gère les high score et ajout d'un moyen de récupérer le nom du joueur en fin de partie

- Ajout du modèle qui gère le score en fin de partie

4.1.3. Ajout d'une JTable après un jeu pour afficher les scores

- Mise en place de la JTable swing pour permettre d'afficher les scores en fin de partie

4.1.4. Ajout des trois autres fruits

- Simple extension du précédent commit pour ajouter les trois autres fruits, tâche facile car les Assets se trouvaient déjà dans le projet

4.1.5. Ajout d'un menu principal et ajout de la possibilité de reset le ladder.

- Création du menu principal en prévision de la prochaine extension ;
- Ajout du bouton permettant de réinitialiser les scores

4.1.6. Ajout des fonctionnalités relatives à la création, suppression, chargement des profils et création des hauts-faits

- Création d'une classe qui définit le comportement d'un profil ;
- Création du modèle des haut-faits

4.1.7. Ajout de la possibilité de visualiser les haut-faits en fonction d'un profil

- La logique derrière la visualisation des haut-faits est ajoutée dans le menu (Activation, lecture des hauts-faits).

4.1.8. Connexion d'un profil au contrôleur game

- Le contrôleur game a dû être lié à l'interface graphique et vice-versa pour permettre une communication bidirectionnelle du GUI vers le contrôleur.

4.2. Dépôt « Soupline69 »

4.2.1. Add the ghost's strategy using strategy design pattern

- Mise en place du strategy design pattern ;
- Création des différentes stratégies ;
- Adaptation des tests unitaires (ajout de paramètres pour quelques constructeurs)

4.2.2. Apply some tools to improve code

- Vérification du code à l'aide de l'outil PMD

4.2.3. Adapt unit tests with my extension

Lorsque la suite de tests unitaires était lancée, des erreurs se produisaient (principalement des null pointer exception), il a donc fallu résoudre ce problème en modifiant le code :

- La StrategyFactory n'est plus créée dans le launcher mais plutôt dans la LevelFactory ;
- Une stratégie a été utilisée dans le test unitaire LevelTest afin de ne pas avoir un null pointer exception (par la suite, cette stratégie sera supprimée et une liste de stratégie vide sera donnée au niveau)

4.2.4. Change constructor of strategy

- Au lieu d' « attacher » l'objet Board à chaque stratégie après la création de ces stratégies, il a été remarqué que cet objet peut être « attaché » dès la création des stratégies.

4.2.5. Ghost can move even if they haven't strategy

Comme le nom du commit l'indique, une modification a été effectuée afin de permettre à un monstre sans stratégie de se déplacer aléatoirement. Cela permet également de donner une « Map » de stratégie vide lors du test unitaire LevelTest.

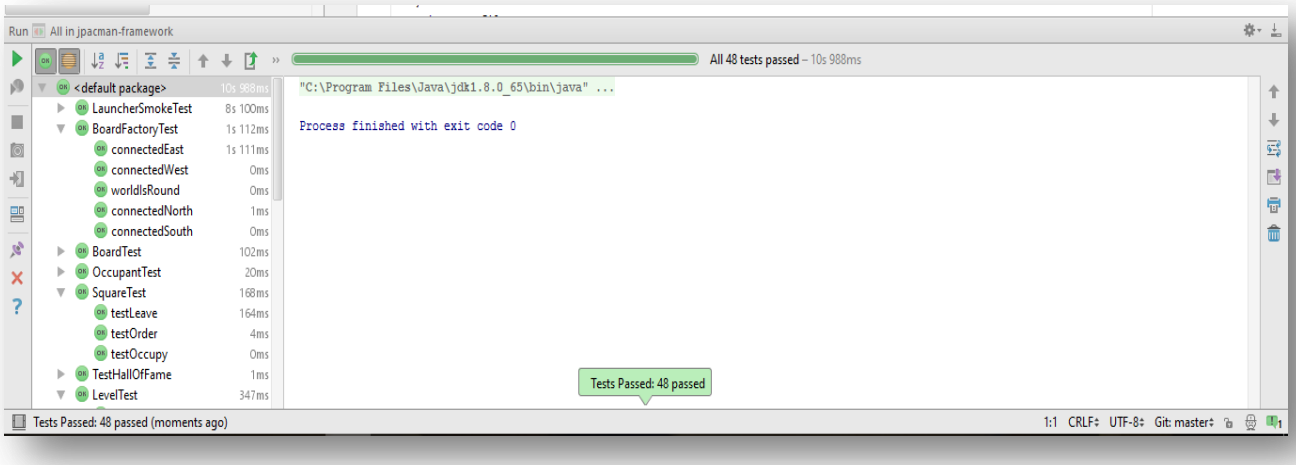
4.2.6. Autres commits

- Changements provenant des différentes analyses réalisées sur le code dans le but de l'améliorer.

5. Comparaison entre le projet de base et le projet des étudiants

5.1. Au niveau des tests unitaires

Aucune régression n'a été remarquée au niveau des tests unitaires malgré les quelques tests unitaires que nous avons ajoutés. Lorsque la suite de tests est lancée, tous les tests réussissent. Signe que le comportement du code de base n'a pas été modifié.



5.2. Au niveau du code source

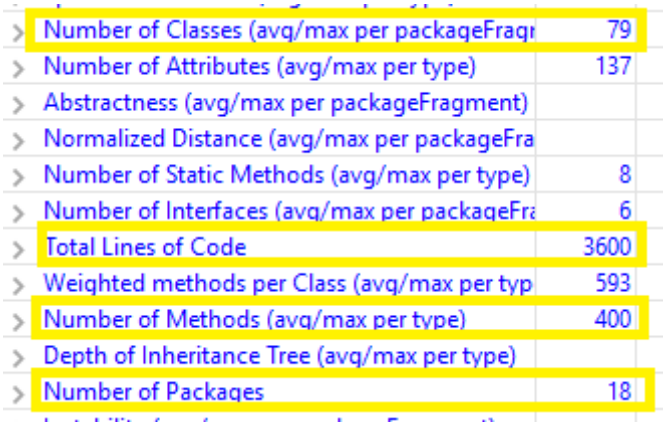
Il est important de remarquer que le comportement des classes de base n'a pas été changé et que la logique du programme est également restée la même. Pour faire nos extensions, nous avons essayé d'utiliser au maximum les méthodes/classes déjà présentes dans le projet.

Intéressons-nous aux métriques :

- Dans le projet de base, soulignons ces quelques métriques

Number of Classes (avg/max per packageFragr	55
Number of Attributes (avg/max per type)	89
Abstractness (avg/max per packageFragment)	
Normalized Distance (avg/max per packageFra	
Number of Static Methods (avg/max per type)	9
Number of Interfaces (avg/max per packageFra	6
Total Lines of Code	2458
Weighted methods per Class (avg/max per typ	376
Number of Methods (avg/max per type)	264
Depth of Inheritance Tree (avg/max per type)	
Number of Packages	15

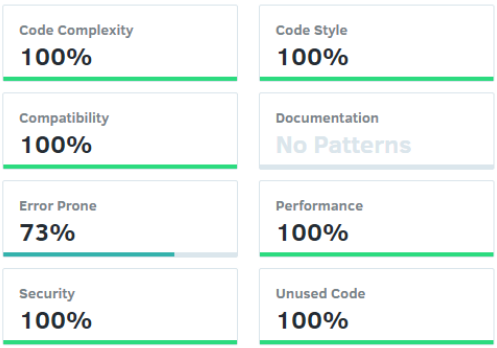
- Dans notre projet



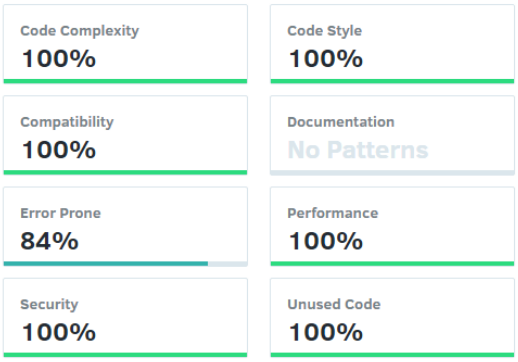
⇨ Plus de ligne de code, plus de méthodes, plus de classes.

Intéressons-nous à l'analyse du code :

- Dans le projet de base



- Dans notre projet

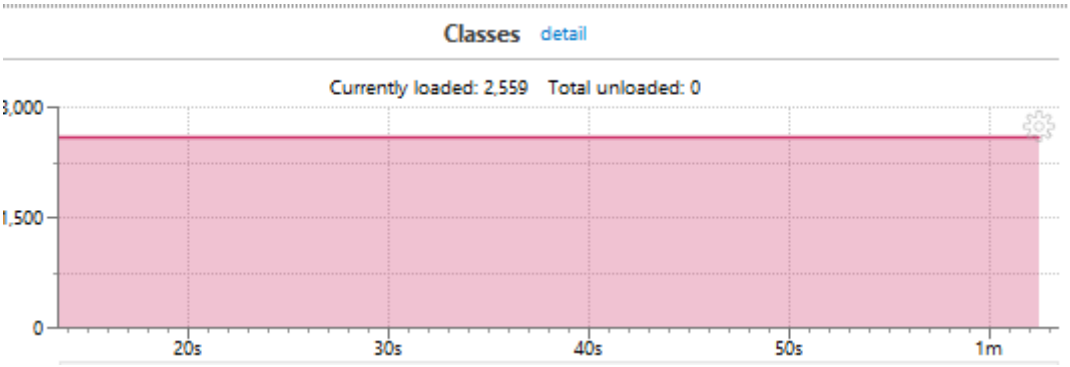


⇨ Malgré l'ajout de code, remarquons qu'il n'y a pas plus d'erreurs qu'avant ! Le code est donc resté très propre.

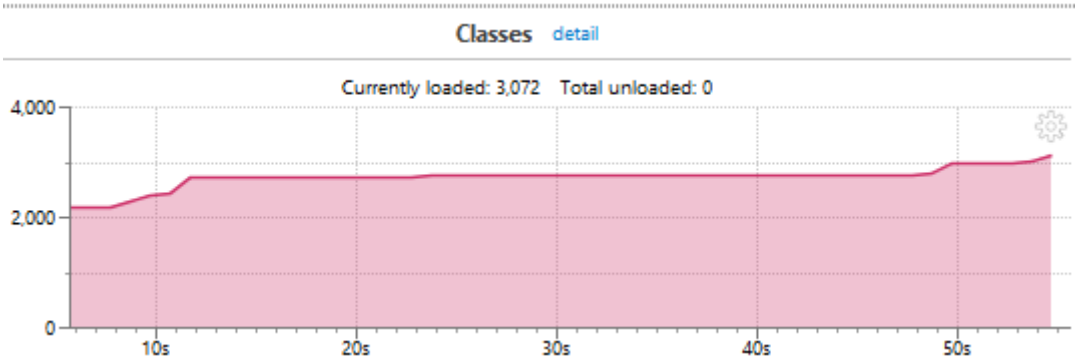
5.3. Au niveau de la performance (profiling)

5.3.1. Les classes

Le projet de base (comme le montre l'image ci-dessous), charge un peu plus de 2000 classes du début jusqu'à la fin du programme.

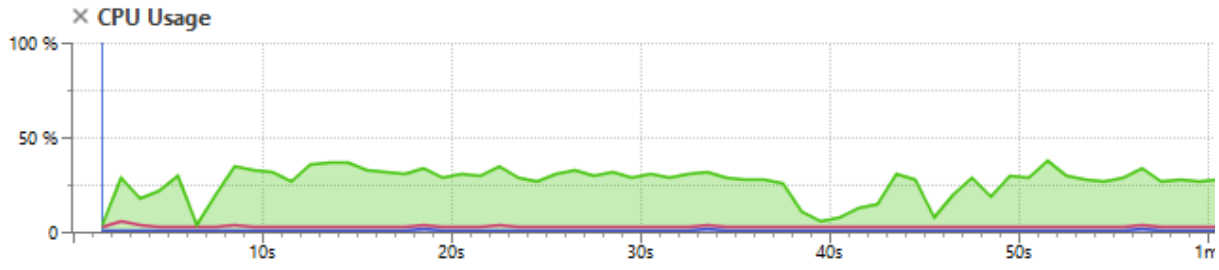


Contrairement au projet de base, notre projet (avec les extensions) charge le même nombre de classes au lancement et pendant le jeu mais atteint les 3000 classes chargées à la fin, car à la fin du jeu, le joueur doit entrer son pseudo afin d'être enregistré (ou pas) dans les 10 meilleurs scores.

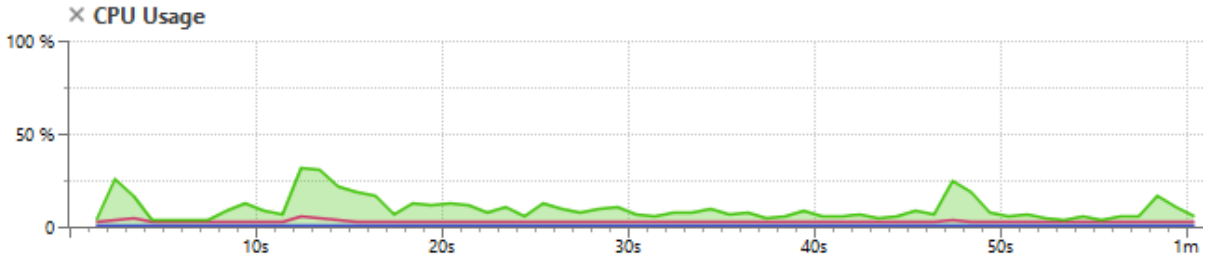


5.3.2. Le CPU

Au niveau de la consommation du CPU, une très petite partie est utilisée pour le projet de base, ce qui est assez positif (un programme qui utilise toute la mémoire CPU n'est pas un très bon programme car, il se peut qu'il ait une fuite mémoire)



Pour notre projet, nous avons remarqué que la consommation du CPU est plus faible qu'à la base ! Ce qui ne peut être que bon signe et une preuve que nos extensions ont bien été intégrées au bon endroit dans le projet de base.



5.3.3. Les méthodes les plus gourmandes

La différence de consommation du CPU entre les deux projets peut s'expliquer par le fait que les méthodes de notre projet sont moins « gourmandes ».

Notre projet :

java.awt.**EventDispatchThread.run**()	1,993	43 %	457
nl.tudelft.jpacman.**Launcher.main**(String[]) Launcher.java	1,515	33 %	0
java.lang.**Thread.run**()	953	21 %	78
nl.tudelft.jpacman.level.**Level.updateObservers**() Level.java	796	17 %	0
nl.tudelft.jpacman.achievement.**MainPanel$3.actionPerformed**(ActionEvent) MainPanel.java	785	17 %	0
nl.tudelft.jpacman.level.**Level.move**(Unit, Direction) Level.java	781	17 %	0
nl.tudelft.jpacman.level.**Level$NpcMoveTask.getMove**() Level.java	781	17 %	0
nl.tudelft.jpacman.level.**Level$NpcMoveTask.run**() Level.java	781	17 %	0
nl.tudelft.jpacman.**Launcher.launch**() Launcher.java	770	17 %	0
nl.tudelft.jpacman.game.**Game.endGame**() Game.java	765	17 %	0
nl.tudelft.jpacman.game.**Game.levelLost**() Game.java	765	17 %	0
java.lang.**ClassLoader.loadClass**(String)	703	15 %	859
nl.tudelft.jpacman.**Launcher.makeGame**() Launcher.java	598	13 %	0
java.awt.**Window.setVisible**(boolean)	593	13 %	593
nl.tudelft.jpacman.**Launcher.makeLevel**() Launcher.java	567	12 %	0
nl.tudelft.jpacman.level.**MapParser.parseMap**(char[][]) MapParser.java	535	12 %	0
nl.tudelft.jpacman.level.**MapParser.parseMap**(List) MapParser.java	535	12 %	0
nl.tudelft.jpacman.level.**MapParser.parseMap**(InputStream) MapParser.java	535	12 %	0
nl.tudelft.jpacman.level.**MapParser.addSquare**(Square[][], List, List, int, int, char) MapParser.java	520	11 %	0
nl.tudelft.jpacman.level.**MapParser.makeGrid**(char[][], int, int, Square[][], List, List) MapParser.jav	520	11 %	0
nl.tudelft.jpacman.sprite.**PacManSprites.loadSprite**(String) PacManSprites.java	504	11 %	0
nl.tudelft.jpacman.sprite.**SpriteStore.loadSprite**(String) SpriteStore.java	504	11 %	0
nl.tudelft.jpacman.sprite.**SpriteStore.loadSpriteFromResource**(String) SpriteStore.java	504	11 %	0
nl.tudelft.jpacman.achievement.**MainMenu.<init>**() MainMenu.java	500	11 %	0
nl.tudelft.jpacman.game.**Game.addPlayerToHallOfFame**() Game.java	500	11 %	0
javax.swing.**JOptionPane.showInputDialog**(Component, Object, String, int)	437	9 %	437
javax.imageio.**ImageIO.read**(InputStream)	426	9 %	426
javax.swing.**JFrame.<init>**(String)	421	9 %	421

Le projet de base :

java.lang.**Thread.run**()	46,218	95 %	93
nl.tudelft.jpacman.level.**Level$NpcMoveTask.run**() Level.java	46,062	95 %	0
nl.tudelft.jpacman.npc.ghost.**Inky.nextMove**() Inky.java	46,062	95 %	0
nl.tudelft.jpacman.npc.ghost.**Navigation.shortestPath**(Square, Square, Unit) Navigation.java	46,046	95 %	2,296
java.util.**HashSet.add**(Object)	43,531	90 %	43,531
nl.tudelft.jpacman.**Launcher.launch**() Launcher.java	1,816	4 %	0
nl.tudelft.jpacman.**Launcher.main**(String[]) Launcher.java	1,816	4 %	0
nl.tudelft.jpacman.**Launcher.makeGame**() Launcher.java	1,129	2 %	0
nl.tudelft.jpacman.**Launcher.makeLevel**() Launcher.java	1,113	2 %	0
nl.tudelft.jpacman.level.**MapParser.parseMap**(char[][]) MapParser.java	1,020	2 %	0
nl.tudelft.jpacman.level.**MapParser.parseMap**(List) MapParser.java	1,020	2 %	0
nl.tudelft.jpacman.level.**MapParser.parseMap**(InputStream) MapParser.java	1,020	2 %	0
nl.tudelft.jpacman.level.**MapParser.addSquare**(Square[][], List, List, int, int, char) MapParser.java	1,004	2 %	0
nl.tudelft.jpacman.level.**MapParser.makeGrid**(char[][], int, int, Square[][], List, List) MapParser.java	1,004	2 %	0
nl.tudelft.jpacman.sprite.**PacManSprites.loadSprite**(String) PacManSprites.java	707	1 %	0
nl.tudelft.jpacman.sprite.**SpriteStore.loadSprite**(String) SpriteStore.java	707	1 %	0
nl.tudelft.jpacman.sprite.**SpriteStore.loadSpriteFromResource**(String) SpriteStore.java	707	1 %	0
nl.tudelft.jpacman.board.**BoardFactory.createWall**() BoardFactory.java	687	1 %	0
nl.tudelft.jpacman.sprite.**PacManSprites.getWallSprite**() PacManSprites.java	671	1 %	0
nl.tudelft.jpacman.ui.**PacManUiBuilder.build**(Game) PacManUiBuilder.java	578	1 %	0
nl.tudelft.jpacman.ui.**PacManUI.<init>**(Game, Map, Map, ScorePanel$ScoreFormatter) PacManUI.java	500	1 %	0
java.lang.**ClassLoader.loadClass**(String)	421	1 %	484
javax.imageio.**ImageIO.read**(InputStream)	410	1 %	410
java.awt.**EventDispatchThread.run**()	359	1 %	187
nl.tudelft.jpacman.level.**LevelFactory.createGhost**() LevelFactory.java	301	1 %	0
nl.tudelft.jpacman.level.**MapParser.makeGhostSquare**(List) MapParser.java	301	1 %	0
nl.tudelft.jpacman.sprite.**PacManSprites.directionSprite**(String, int) PacManSprites.java	301	1 %	0
nl.tudelft.jpacman.sprite.**PacManSprites.getGhostSprite**(GhostColor) PacManSprites.java	301	1 %	0
javax.imageio.**ImageIO.<clinit>**()	296	1 %	296
nl.tudelft.jpacman.npc.ghost.**GhostFactory.createBlinky**() GhostFactory.java	265	1 %	0
nl.tudelft.jpacman.sprite.**ImageSprite.split**(int, int, int, int) ImageSprite.java	250	1 %	0
java.awt.**Window.pack**()	218	0 %	218
nl.tudelft.jpacman.npc.ghost.**Navigation.addNewTargets**(Unit, List, Set, Navigation$Node, Square) Navigatior	218	0 %	62
javax.swing.**JFrame.<init>**(String)	203	0 %	203
nl.tudelft.jpacman.sprite.**ImageSprite.newImage**(int, int) ImageSprite.java	187	0 %	0
java.awt.**GraphicsEnvironment.getLocalGraphicsEnvironment**()	156	0 %	156
nl.tudelft.jpacman.sprite.**ImageSprite.draw**(Graphics, int, int, int, int) ImageSprite.java	156	0 %	0
nl.tudelft.jpacman.ui.**BoardPanel.paint**(Graphics) BoardPanel.java	156	0 %	0
nl.tudelft.jpacman.ui.**BoardPanel.render**(Square, Graphics, int, int, int, int) BoardPanel.java	156	0 %	0
nl.tudelft.jpacman.ui.**BoardPanel.render**(Board, Graphics, Dimension) BoardPanel.java	156	0 %	0