

# Rapport du projet Jpacman

## Projet de Software Evolution

Activité d'Apprentissage S-INFO-029

Groupe numéro : **10**

Membres du groupe :

Collin Arnaud

Craeye Mathieu

Cominu Billal

Année Académique 2015-2016

Année d'études : Bloc complémentaire en MA1 info

Faculté des Sciences, Université de Mons

6 mai 2016

### Résumé

Ce *rapport du projet Jpacman* est rendu dans le cadre de l'AA S-INFO-029 "Projet de Software Evolution", dispensé par le Prof. *Tom Mens* en année académique 2015-2016 . Le but de ce rapport d'énumérer les différentes parties des membres du groupe ainsi que l'analyse du code.

# 1 Introduction

## 1.1 Présentation

Ce rapport est en lien avec le cours de Software evolution donné en Master 1 Informatique. Plusieurs points seront vu en détails dans ce document tels que : les extensions des différents membres du groupe avec les choix d'implémentation de chacun et la modification du code d'origine, l'historique du code de chaque membre (différents commit) et pour finir, une analyse du code avec divers outils comme PDM, Codacy, EclEmma, Jdepend et visualVm.

## 1.2 Répartition du travail

Chaque membre du groupe ont fait leur propres parties (les extensions et historique du code) ainsi que deux points pour l'analyse du code.

Nom du membre	Extension	Historique du code	Analyse du code
Collin Arnaud	Cases et fruits spéciaux	Cases et fruits spéciaux	PMD et code coverage
Cominu Billal	IA pour Pacman	IA pour Pacman	Analyse de dépendances et analyse de performance
Craeye Mathieu	Multi-Joueur sans Pacman	Multi-Joueur sans Pacman	Code duplication et code auditing

## 2 Les Extensions

Comme demandé dans les consignes du projet, chaque membre du groupe a réalisé une extension. Les points suivants expliqueront en détails les choix d'implémentation des différents membres.

### 2.1 Cases et fruits spéciaux - Collin Arnaud

#### 2.1.1 Choix d'implémentation

Pour cette extension, deux nouveaux packages ont été créés. Ces packages ont été conçus afin d'éviter les problèmes lors de la fusion des extensions des groupes.

##### 1. **nl.tudelft.jpacman.fruit**

Ce package contient les classes propres à l'implémentation des fruits et légumes spéciaux.

On a une classe principale qui est la classe `Fruit` qui définit en outre le timer pour les effets de ces fruits.

Ensuite on a les autres classes, chacune du nom du fruit à implémenter qui hérite de la classe `Fruit`. Ces classes surchargent des méthodes de la classe `Fruit` comme par exemple la fonction « **effect()** » donnant le nom de l'effet.

##### 2. **nl.tudelft.jpacman.specialcase**

Ce package contient les classes propres à l'implémentation des cases spéciales.

On a également une classe principale qui est `SpecialSquare` qui définit aussi un timer propre à ces cases. Ensuite on a les autres classes, chacune du nom d'une case spéciale à implémenter qui hérite de la classe `SpecialSquare`.

Vous trouverez en détail ci-dessous d'amples informations sur les deux packages cités précédemment.

#### 1. Les fruits et légumes spéciaux

##### (a) *Effets positifs*

**Grenade** : Lorsque que Pac-man rentre en collision avec ce fruit, on va calculer sa position sur le tableau, ensuite on va calculer de la même façon la position des monstres. Ceux qui se trouvent à 4 cases (horizontalement, verticalement ou en oblique) sont définis comme morts et disparaissent du tableau.

**Poivron** : Lorsque Pac-man le mange, sa vitesse augmente. Un timer s'enclenche. N'ayant pas de variable vitesse prise en compte pour son déplacement, c'est la vitesse des monstres qui diminue, rendant ainsi Pac-man plus rapide jusqu'à ce que le timer arrive à la fin du temps.

**Tomate** : Lorsque Pac-man le mange, il devient invisible et un timer s'enclenche. C'est-à-dire que lorsque les fantômes vont rentrer en collision avec lui, ils vont vérifier si Pac-man n'est pas invisible (défini par un booléen). Lorsque le timer arrive à la fin, Pac-man redevient visible et donc vulnérable.

**Haricot rouge** : Pour des raisons de temps (délai individuel), ce légume n'a pas été implémenté. La gestion de l'affichage des tirs buggant, il n'a pas été comité.

(b) *Effets négatifs*

**Pomme de terre** : Lorsque que Pac-man le mange, la vitesse des fantômes augmentent et un timer s'enclenche. Cet effet est similaire à celui du poivron sauf que l'on augmente la vitesse de déplacement des monstres cette fois jusqu'à la fin du timer.

**Poisson** : Lorsque Pac-man le mange, un timer se lance et il ne peut plus se déplacer pendant un certain temps. A chaque fois que le joueur va essayer de faire un mouvement, Pac-man va rester sur sa position de départ et cela jusqu'à la fin du timer.

(c) *Remarques*

Le timer est implémenté dans la classe Fruit. Pour plus de simplicité, on ne différencie pas fruits/légumes/poisson car ils n'ont aucunes différences entre eux.

Lorsque Pac-man va manger un fruit avec un timer, on va le démarrer et puis tant que l'effet associé est présent, on va vérifier si le temps du timer est dépassé et dans ce cas on annule l'effet.

Les effets des fruits **poivron** et **pomme de terre** s'annulent entre eux.

Les effets peuvent être cumulatifs, Pac-man peut être invisible et avoir par exemple sa vitesse augmentée.

2. *Les cases spéciales*

**Pièges** : Lorsque qu'un fantôme tombe dans un piège, il ne peut plus se déplacer pendant un certain temps. Comme pour les fruits, un timer se lance et tant qu'il n'a pas atteint le temps définit, le fantôme ne sera pas se déplacer. Le principe est similaire au poisson. Pac-man ne sait pas tomber dans le piège, car sinon cela rendrait le poisson inutile.

**Téléportation** : Lorsque Pac-man tombe sur cette case, il est téléporté directement sur une autre case définit. Dans cette extension, c'est son point de départ qui a été choisi. La téléportation n'est pas aléatoire afin de ne pas mettre Pac-man dans une situation « **difficile** ». On va modifier la position du joueur sans délai.

**Pont** : Lorsque Pac-man ou les fantômes passent dessus, selon comment ils sont rentrés, ils seront en haut du pont ou bien en dessous. S'ils arrivent par le nord ou le sud de la case, ils seront au-dessus, sinon ils seront en-dessous.

Lorsqu'il marche sur cette case, on va enregistrer comment ils sont rentrés en fonction de leur direction. Pour sortir de cette case, ils ne peuvent qu'aller dans les directions correspondantes, à savoir retourner en arrière ou bien continuer tout droit.

Les fantômes ne peuvent pas tuer Pac-man s'ils ne sont pas à la même hauteur sur le pont.

A la page suivante, on identifie les différents fruits et cases spéciales ainsi que deux images de jeu.

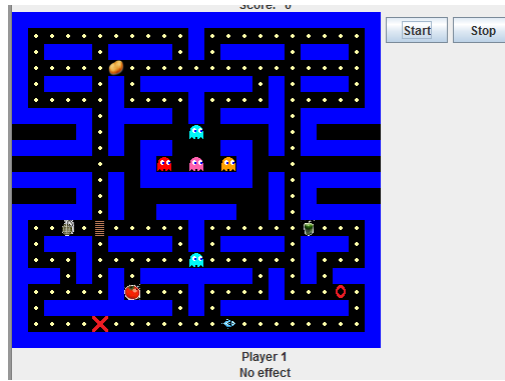


FIGURE 1 – Fruits et cases speciales

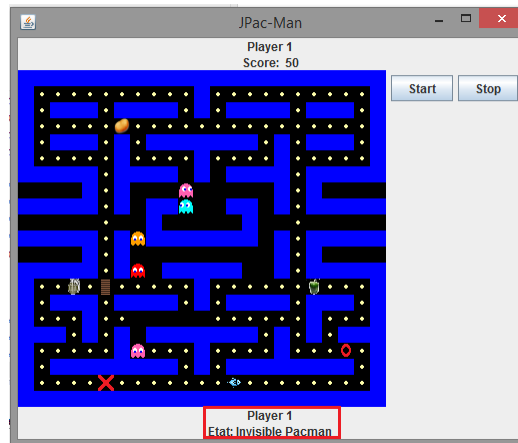










FIGURE 2 – Effets

-  **Grenade** : effet de bombe
-  **Poivron** : effet de boost de vitesse Pac-man
-  **Tomate** : effet d'invisibilité Pac-man
-  **Pomme de terre** : effet de boost vitesse des fantômes
-  **Poisson** : Paralyse pac-man
-  **Téléporteur** : téléporte Pac-man
-  **Piège** : Paralyse fantômes
-  **Pont** : donne une « hauteur » à l'unité dessus

### 2.1.2 Modification du code d'origine

Plusieurs classes ont dû être modifiées pour mettre en place cette extension, en particulier pour les plus importantes :

#### 1. La classe PlayerCollision

De nouveaux facteurs devaient être pris en compte lors des collisions. Est-ce que Pac-Man est invisible, est-ce que le fantôme est sur le même niveau du pont que Pac-Man ? De nouvelles conditions ont fait leur apparition pour vérifier cela.

```
public void playerVersusGhost(Player player, Ghost ghost)
{
    if(player.isInvisible()==false)
    {
        if((player.checkOnBridge(player.getSquare()) instanceof Bridge) && (ghost.checkOnBridge(ghost.getSquare()) instanceof Bridge))
        {
            Bridge playerBridge=(Bridge) player.checkOnBridge(player.getSquare());
            Bridge ghostBridge=(Bridge) ghost.checkOnBridge(ghost.getSquare());

            if(playerBridge.getEnterDirection(player).equals(ghostBridge.getEnterDirection(ghost)) )
            {
                player.setAlive(false);
                changePlayer(player);
            }
        }
        else
        {
            player.setAlive(false);
            changePlayer(player);
        }
    }
}
```

#### 2. La classe MapParser

De nouveaux éléments devaient être pris en compte lors de la création de la carte, afin de créer les cases spéciales et les fruits correspondants.

Le switch de la fonction addSquare() a donc été complété.

#### 3. La classe Level

- (a) La fonction move(Unit unit, Direction direction) a été complétée afin de pouvoir prendre en compte les effets et les cases sur lequel se trouve le personnage(unit).
- (b) La fonction fruitEffect(Unit fruit, Square unit) a été implémenté afin de définir ce qu'il se passe lorsque Pac-man mange un fruit. Si c'est un effet instantané alors on l'effectue, si cela requiert un timer alors on enclenche celui-ci.

## 2.2 IA pour Pacman - Cominu Billal

### 2.2.1 Choix d'implémentation

Pour cette extension, le package «**nl.tudelft.jpacman.controls**» a été ajouté dans la hiérarchie initiale. Ce package contient une classe abstraite `ControlStrategy` conçue selon le design pattern «**Strategy**».

La classe `PlayerStrategy` permet de modifier la façon de contrôler Pacman avec le clavier, le dernier mouvement reçu est répété tant que la case ciblée est accessible, jusqu'à ce qu'un autre mouvement soit reçu.

La classe abstraite permet aussi d'implémenter une intelligence artificielle pour Pacman avec diverses stratégies.

Deux stratégies sont implémentées pour le projet : une stratégie aléatoire (`RandomStrategy`) et une stratégie basée sur le score (`ScoreStrategy`).

La première stratégie change aléatoirement de direction à chaque intersection, alors que la deuxième sélectionne une direction en fonction des pellets et des fantômes qui s'y trouvent.

Le malus induit par la présence d'un fantôme est inversement proportionnel à la distance qui le sépare de Pacman. Le même raisonnement est utilisé pour le bonus induit par la présence de pellets.

L'IA basée sur le score vérifie aussi la présence de fantômes dans les cases adjacentes à la ciblée de manière à éviter un maximum les fantômes.

Un bouton a été ajouté à l'interface graphique afin de pouvoir activer et désactiver l'intelligence artificielle. Lors du démarrage du jeu, il suffit d'activer le bouton «**IA**» pour démarrer la partie et activer l'intelligence artificielle de Pacman.

Le bouton «**IA**» peut être utilisé en cours de partie pour activer ou désactiver l'IA à tout moment.

### 2.2.2 Modification du code d'origine

Certaines classes ont été modifiées afin de mettre de place cette extension :

#### 1. `nl.tudelft.jpacman.game.SinglePlayerGame`

(a) Ajout de la méthode `public ControlStrategy getPacmanControls`

```
public ControlStrategy getPacmanControls(){  
    return this.pacmanControls;  
}
```

Retourne une référence vers la classe qui gère le contrôle de Pacman(`ControlStrategy`) quelle que soit la stratégie utilisée.

(b) Ajout de la méthode `public void toggleAI`

```
public void toggleAI(){
    if (pacmanControls instanceof PlayerStrategy){
        pacmanControls.stop();
        pacmanControls = new ScoreStrategy(this,player);
    }
    else{
        pacmanControls.stop();
        pacmanControls = new PlayerStrategy(this,player);
    }
}
```

Permet de passer du mode de contrôle manuel (clavier) au mode automatique (intelligence artificielle) et inversement.

## 2. `nl.tudelft.jpacman.Launcher`

(a) Modification de la méthode `protected void addSinglePlayerKeys`

```
ControlStrategy controls = game.getPacmanControls();
if (controls instanceof PlayerStrategy) {
    switch (d) {
        case EAST:
            game.getPacmanControls().setDirection(Direction.EAST);
            break;
        case WEST:
            game.getPacmanControls().setDirection(Direction.WEST);
            break;
        case NORTH:
            game.getPacmanControls().setDirection(Direction.NORTH);
            break;
        case SOUTH:
            game.getPacmanControls().setDirection(Direction.SOUTH);
            break;
        default:
            break;
    }
}
```

L'action associée à chaque direction a été modifiée.

Initialement, les directions reçues (clavier) étaient directement traitées par la classe `Game`.

Après la modification, les directions reçues sont traitées uniquement si la stratégie de contrôle est de type `PlayerStrategy`.



### 3. nl.tudelft.jpacman.ui.PacManUiBuilder

(a) Ajout de la méthode private void addAIButton

```
private void addAIButton(final Game game){
    assert game != null;
    buttons.put(AI_CAPTION, new Action() {
        @Override
        public void doAction() {
            game.toggleAI();
            game.start();
        }
    });
}
```

Ajout d'un bouton «IA» faisant appel à game.start() et game.toggleAI()

(b) Modification de la méthode public PacManUI build

```
public PacManUI build(final Game game) {
    assert game != null;

    if (defaultButtons) {
        addStartButton(game);
        addStopButton(game);
        addAIButton(game);
    }
    return new PacManUI(game, buttons, keyMappings, scoreFormatter);
}
```

Ajout d'un appel à addAIButton dans la liste des boutons par défaut.

## 2.3 Multi-Joueur sans Pacman - Craeye Mathieu

### 2.3.1 Choix d'implémentation

Pour cet extension, j'ai décidé de travailler avec des nouvelles classes totalement indépendantes du projet initial afin d'éviter des problèmes dans la seconde phase (rassemblement des différentes extensions du groupe).

Ces classes se trouvent dans le package « **nl.tudelft.jpacman.multiplayers** » pour la version commune ou « **Craeye-Mathieu** » pour la version individuelle.

De plus, dans l'énoncé il était demandé de travailler avec plusieurs joueurs en même temps sur la carte mais je trouvais cela bizarre que 4 joueurs seraient en même temps sur le même clavier. j'ai donc décidé de séparer chacun des joueurs.

Dans ce package, on retrouve 3 classes :

1. La première intitulée « **ChoiceMonster** » : Cette classe définit le choix du nombre de joueurs ainsi que le monstre associé au joueur. Dans un premier temps, on retrouve 3 boutons (2 joueurs, 3 joueurs, 4 joueurs) ensuite en fonction du choix du nombre de joueurs vous aurez la possibilité de choisir pour chaque joueur le monstre que vous désirez (1 monstre différent par joueur) - *Figure 3*. Une fois les monstres définis aux joueurs, la partie peut commencer avec le dernier monstre choisi - *Figure 4* (avec extension fruit d'Arnaud Collin)
2. La seconde classe « Joueur » : crée un joueur (un monstre) en fonction de son nom (Clyde, Pinky, Inky, Blinky), son numéro, son score.  
De plus, dans cette classe on retrouve une liste de joueurs permettant d'ajouter les monstres en fonction du choix des joueurs.  
Cette liste est utilisée dans la classe « **launcher** » afin de charger le dernier joueur de celle-ci pour débiter la partie. Pour cela, j'ai dû modifier légèrement la classe « **launcher** » (voir section sur les modifications du code).
3. La dernière et troisième classe « **Classement** » : elle permet de voir le score de chaque joueur ayant participé - *Figure 5*.

Comment se déroule une partie ? Tout simplement en commençant à choisir pour chaque joueur le monstre désiré. Ensuite, la partie débute avec le dernier monstre choisi donc le dernier joueur. Une fois que le joueur a fini sa partie (mort ou gagné) le second joueur débute sa partie. Et ainsi de suite jusqu'au moment où tous les joueurs ont joué. Entre chaque partie, nous retrouvons les scores des différents joueurs dans une autre fenêtre.



FIGURE 3 – Choix des monstres/joueurs

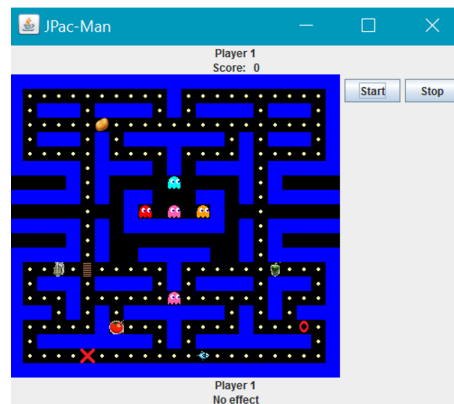


FIGURE 4 – Jeu



FIGURE 5 – Classement

### 2.3.2 Modification du code d'origine

Afin de pouvoir mettre en place mon extension, j'ai dû modifier quelques classes du projet d'origine que je vais détailler ci-dessous (les modifications apportées seront mises en italiques et en gras).

#### 1. La classe Launcher

- (a) Pour pouvoir commencer le jeu avec le choix des joueurs, j'ai créé une fonction « before » permettant d'appeler la classe « **choiceMonser** » afin d'afficher l'écran des joueurs *Figure 1*.

```
public void before()
{
    cM.ButtonPlayer();
}
```

- (b) J'ai modifié la fonction « **makeGame** » en ajoutant en paramètre un joueur afin de pouvoir créer un joueur en fonction du niveau et du joueur passé en paramètre de la fonction « **makeGame** » se trouvant dans la méthode « **launch** ».

```
public Game makeGame(Joueur joueur) {
    GameFactory gf = getGameFactory();
    Level level = makeLevel();
    return gf.createSinglePlayerGame(level, joueur);
}
```

- (c) Pour finir j'ai dû apporter quelques modifications à la fonction « **launch** ». D'abord, afficher le score de chaque joueur avant de commencer la partie. Ensuite, passer en paramètre le dernier joueur de la liste (en fonction du nombre de joueur). Pour finir, de diminuer le nombre de 1 afin de charger le joueur suivant.

```
public void launch() {
    new Classement().ClassementJoueur();
    game = makeGame(cM.listJ.get(cM.j.getNbrJoueur()-1));
    PacManUiBuilder builder = new PacManUiBuilder().withDefaultButtons();
    addSinglePlayerKeys(builder, game);
    pacManUI = builder.build(game);
    pacManUI.start();
    cM.j.SetNbrJoueur(cM.j.getNbrJoueur()-1);
}
```

#### 2. La classe GameFactory

- (a) J'ai dû réviser cette classe afin de pouvoir charger un ghost au lieu du pacman comme joueur. Pour cela, la méthode « **SinglePlayerGame** » de la classe « **PlayerFactory** » crée un monstre en fonction du joueur passer en paramètre.

```
public Game createSinglePlayerGame(Level level, Joueur name) {
    return new SinglePlayerGame(playerFact.createGhost(name), level);
}
```

---

### 3. La classe PlayerFactory

- (a) Dans cette classe, j'ai ajouté une fonction «**createGhost** » qui va retourner un player avec l'aspect du monstre choisi par le joueur grâce à l'objet « **name** » passé en paramètre. En fonction du nom du joueur, il retournera l'aspect du monstre adéquat.

```
public Player createGhost(Joueur name)
{
    if(name.getName()=="pinky")
        return new
Player(sprites.getGhostSprite(GhostColor.PINK),sprites.getPacManDeathAnimat
ion(),"pinky");
    if(name.getName()=="clyde")
        return new
Player(sprites.getGhostSprite(GhostColor.ORANGE),sprites.getPacManDeathAnim
ation(),"clyde");
    if(name.getName()=="inky")
        return new
Player(sprites.getGhostSprite(GhostColor.CYAN),sprites.getPacManDeathAnimat
ion(),"inky");
    if(name.getName()=="blinky")
        return new
Player(sprites.getGhostSprite(GhostColor.RED),sprites.getPacManDeathAnimati
on(),"blinky");
    return new
Player(sprites.getGhostSprite(GhostColor.RED),sprites.getPacManDeathAnimati
on(),"blinky");}
```

### 4. La classe PlayerCollision

- (a) J'ai créé une nouvelle fonction qui va faire appel à la méthode « **launch** » de la classe « **launcher** » afin de démarrer une nouvelle partie avec le second joueur. Avant de charger la nouvelle partie, on définit le score du joueur en fonction de son nom. Cette fonction est appelée lorsque le joueur meurt ou gagne (dans la méthode « **playerVersusGhost** » de cette classe quand il meurt et, dans la méthode « **updateObservers** » quand il gagne de la classe « **level** »).

```
public void changePlayer(Player player)
{
    Launcher l=new Launcher();

    switch(player.getName())
    {
        case "blinky":
            l.cM.jBlinky.setScore(player.getScore());
            break;
        case "inky":
            l.cM.jInky.setScore(player.getScore());
            break;
        case "clyde":
            l.cM.jClyde.setScore(player.getScore());
            break;
        case "pinky":
            l.cM.jPinky.setScore(player.getScore());
            break;
    }
    l.launch();
}
```

### 3 Historique du code

#### 3.1 Cases et fruits spéciaux - Collin Arnaud

Date	Nom du commit	Details	version	Différence(s) entre versions
11-04-16	Implementation grenade	Implémentation du fruit grenade.	1.A	Aucune
12-04-16	Implements potato and pepper	Implémentation des fruits poivrons et pomme de terre.	1.B	Ajout de fruits interactifs
14-04-16	Potato and fish	Refactoring du fruit pomme de terre et implémentation du poisson et tomate.	1.C	Ajout de fruits interactifs et indicateur d'effet
14-04-16	Refactoring fruits	Refactoring des fruits implémentés.	1.D	Amélioration des fonctions
16-04-16	Implements special cases	Implémentation des cases spéciales : téléporteur, piège et pont.	2.A	Ajout de cases spéciales
17-04-16	Jpacman Groupe10 CaseFruitSpeciaux	Version final de l'extension	3.A	Javadoc complète
21-04-16	Importation Mathieu	Ajout extension Mathieu Craeye.	3.B	Aucune









#### 3.2 IA pour Pacman - Cominu Billal

Date	Nom du commit	Details
02-03-2016	PlayerMovesHandler.java	Gestion des commandes de mouvement du joueur
02-03-2016	ControlKeysTest.java	Tests pour commandes mouvement du joueur
02-03-2016	AiStrategy.java	IA pour pacman
02-03-2016	AiTest.java	Tests de l'IA pour pacman
02-03-2016	modification du launcher	Utilisation de la classe PlayerMovesHandler
17-04-2016	refactoring	gestion clavier avec pattern strategy
17-04-2016	suppr PlayerMovesHandler	suppression classe inutile après refactoring
17-04-2016	rename	renommer classes et package
17-04-2016	ajout bouton AI	ajout bouton AI
17-04-2016	refactor tests	adapter les tests au refactoring des classes
17-04-2016	final	version finale

### 3.3 Multi-Joueur sans Pacman - Craeye Mathieu

Dans le tableau ci-dessous, on retrouve les différents commit que j'ai effectué pour mettre en place mon extension, un petit résumé sur le commit, le numéro de version ainsi que la différence entre deux versions.

Date	Nom du commit	Détails	Version	Différence(s) entre versions
11-04-16	Mise en place des boutons pour l'interface « <b>choix nombre de joueurs</b> »	On retrouve deux groupes de boutons, le premier reprend le nombre de joueurs (2, 3 ou 4 joueurs) le second sont les différents monstres disponibles (Pinky, Inky, Clyde, Blinky). <b>Création de la classe « ChoiceMonster »</b>	1.A	Aucune
11-04-16	Ajout ennemi player	Remplacement du sprite pacman par un monstre en fonction du choix du joueur. <b>Modification de la classe « GameFactory » et « PlayerFactory »</b>	1.B	Par apport à la version 1.A, ajout d'une fonction pour afficher le sprite du monstre choisi.
17-04-16	Classement	Mise en place du système des scores des joueurs. <b>Création de la classe « Classement » et « Joueurs »</b>	1.C	Par apport à la version 1.B, ajout de l'interface des scores.
18-04-16	Suppression du problème classement	Suppression du problème du classement (connaître le vainqueur). <b>Modification de la classe « Classement »</b>	2.C	Par apport à la version 1.C, suppression du système du classement
18-04-16	UnitTest	Mise en place des tests unitaires pour la création d'un joueur. <b>Création de la classe « joueurTest »</b>	1.D	Aucune
18-04-16	Correction beug chargement joueur	Suppression de la ligne de code « ajouter un monstre à la liste » (on ajoutait deux fois le monstre alors qu'il devait le faire qu'une seule fois). <b>Modification de la classe « ChoiseMonster »</b>	2.A	Par apport à 1.A, suppression d'une ligne de code.
20-04-16	JavaDoc	Mise en place de la javaDoc.	1.E	Aucune
21-04-16	Sync Arnaud	Synchronisation du mon code avec celui d'Arnaud Collin. Ajout des classes ainsi que la correction d'un conflit dans la classe« <b>PlayerCollision</b> ».	1.F	Aucune

	Sync Arnaud 4 days ago by 160673	50 ▶
	javaDoc 5 days ago by 160673	2 ▶
	correction beug chargement joueur 7 days ago by 160673	2 ▶
	UnitTest 7 days ago by 160673	3 ▶
	suppression problème classement 7 days ago by 160673	3 ▶
	classement 8 days ago by 160673	10 ▶
	ajout ennemi player 14 days ago by 160673	4 ▶
	Mise en place des boutons choix nombre joueurs 14 days ago by 160673	2 ▶



## 4 Analyse du code

Nous avons utilisé différents outils ou techniques pour vérifier la qualité du code. Ces outils et techniques ne nous sont pas inconnus car nous les avons vus en partie durant le cours.

Différents points du code vont être analysés :

1. Code duplication
2. Code auditing (style et structure du code)
3. Analyse de dépendances
4. Code coverage
5. Performance
6. Bad smells

### 4.1 Code duplication

Pour vérifier la présence de duplication de ligne de code dans le projet, nous avons utilisé l'outil intégré à IntelliJ vu au cours. Nous avons pu alors détecter la présence de code identique et ainsi les corriger afin d'améliorer le projet. Dans cette partie d'analyse, il n'y a vraiment pas de comparaison à définir car le code détecté se trouvait dans le code d'origine et final.

On retrouvait également de la duplication de code dans les classes de tests unitaires que nous n'avons pas jugé nécessaire de factoriser.

Voici différentes fonctions améliorées suite à l'analyse :

1. Une partie de code de la fonction « **nextmove()** » se retrouvant dans les classes inky, pinky et blinky, a pu être rassemblée en une seule fonction se trouvant dans la classe « **ghost** » appelée « **path (Square destination)** ».
2. La fonction « **addSinglePlayerKeys** » de la classe « **launcher** » a pu être factorisée afin d'éviter la duplication de certaines lignes. Pour cela, une nouvelle fonction « **direction (Direction d)** » a été créée afin de rassembler les différentes copies de code en une seule.

Pour de plus amples détails voir commit «**Optimisation code duplicate** ».

## 4.2 Code auditing

### 4.2.1 Codacy

Codacy est un outil qui analyse le code en ligne via une association avec votre dépôt github. Il nous a permis d'analyser le style, la structure ainsi que des parties de code sujettes à erreurs. Cet outil est gratuit pendant 14 jours ensuite il faut une licence pour pouvoir continuer à l'employer.

Comment fonctionne-t-il ? Une fois sur le site codacy, il suffit de cliquer sur « **Try it for free** », ensuite se connecter via votre github et choisir le projet à analyser.

Lorsque l'analyse est terminée, un tableau de bord apparaît avec l'état de votre code. On y retrouve plusieurs sections : la complexité, le style, la compatibilité, les erreurs, la performance, la sécurité, le code inutilisé, etc ? . L'outil vérifie également la qualité de chaque commit via la section « commits » dans le volet de gauche.

Vous trouverez plus bas l'analyse du code d'origine et du code final (après la réunification des extensions).

#### 1. Code d'origine

Sur l'image ci-dessous, on voit que le style et la structure du code sont excellents. Juste un petit point faible pour la section « **Error Prone** » où il y a 3 points qui peuvent être améliorés (seconde image).

The screenshot displays the Codacy dashboard for a project named 'master'. The dashboard is divided into several sections:

- Project Certification:** A grid of metrics showing various code quality scores: Code Complexity (100%), Code Style (100%), Compatibility (100%), Documentation (No Patterns), Error Prone (86%), Performance (100%), Security (100%), and Unused Code (100%).
- Issues Breakdown:** A donut chart showing 3 total issues, categorized as Error Prone (3), Unused Code (0), and Documentation (0).
- Coverage:** A section with a cartoon character and a 'Setup Coverage' button, indicating that coverage results and notifications are not yet set up.
- Goals:** A section for setting and tracking project goals.
- Issues:** A table listing current issues, filtered by 'Error Prone'.

The 'Issues' table shows the following details:

Language	Category	Level	Pattern	Author
src/main/java/nl/tudelft/jpacman/board/Square.java	Error Prone	Low	Use explicit scoping instead of the default package private level	
src/main/java/nl/tudelft/jpacman/level/LevelFactory.java	Error Prone	Low	Use explicit scoping instead of the default package private level	

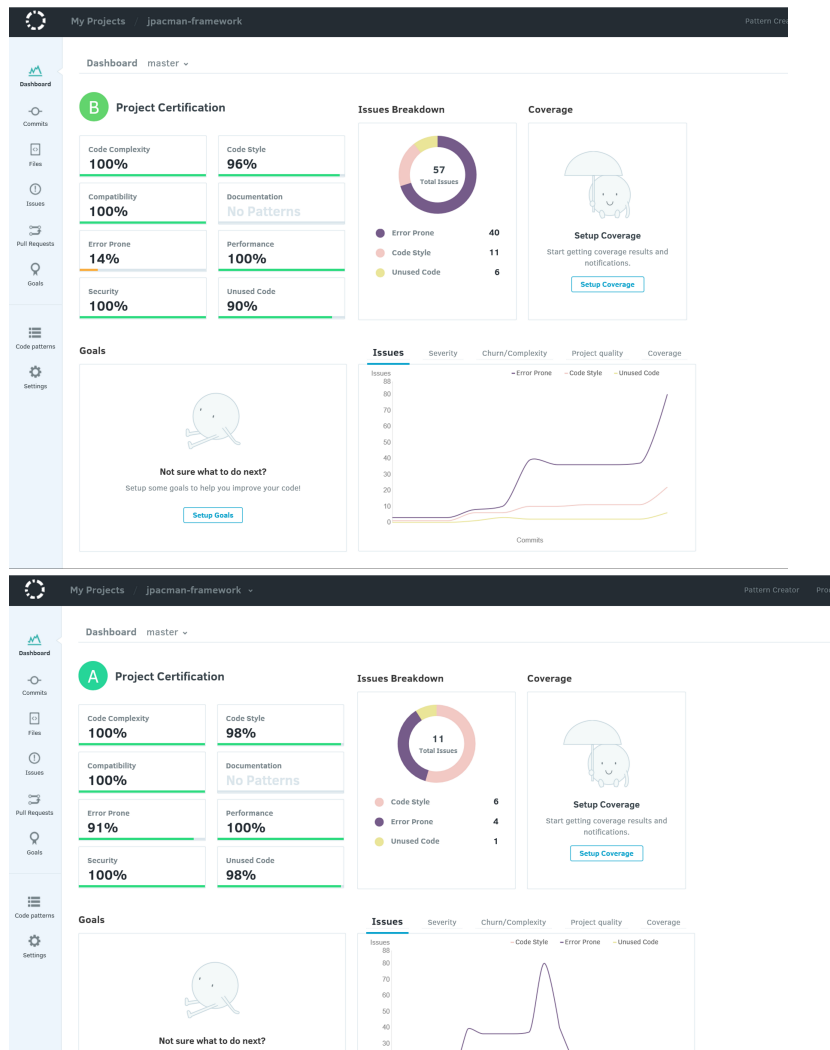
Navigation links at the bottom include '< Previous' and 'Next >'.

## 2. Code final

Sur la première image, on retrouve l'analyse du code final, on peut s'apercevoir que certains points ont diminué en qualité mais admissibles tandis que la section « Error Prone » est devenue très critique. Du coup, grâce à cet outil, nous avons pu identifier les parties de code qui posaient problème à ce sujet en cliquant simplement sur la section adéquate.

Une fois ces différents problèmes résolus dans le projet, nous remarquons sur la second image que le pourcentage est devenu presque parfait même meilleur que le code d'origine suite à la résolution de quelques problèmes.

Pour suivre l'évolution des différents commits sur l'optimisation, je vous invite à consulter les commits du 29/04 et du 30/04 sur l'optimisation du code avec Codacy.



## 4.3 PMD

Pour renforcer notre analyse du code, en plus de Codacy, nous avons utilisé PMD. Nous avons installé PMD sur notre programme de développement « eclipse » grâce à un plugin téléchargeable via le marketplace d'eclipse.

Une fois l'installation terminée, il suffit d'analyser votre projet avec PMD. Pour cela, un simple clic droit sur le projet, choisir PMD, vérifier le code avec PMD. L'analyse faite, PMD va identifier les violations de code.

Pour notre part, nous avons décidé de passer outre certaines de ces identifications. Exemple, le nom du variable trop court ou trop long, le manque de déclarations du type « final » pour les variables passées en paramètre dans une méthode, l'utilisation du « **tiré bas** » dans les variables autres que les finales.

Mais grâce à l'analyse précédente avec Codacy, PMD a identifié quelques violations « **Erreur haute** ». La plupart d'entre elles ont été corrigées excepté un cas qui revient deux fois dans différentes classes lorsqu'on appelle une fonction dans le constructeur de la classe.

Afin de renforcer notre analyse avec ce logiciel, nous avons testé la présence de copier/coller dans le code et un retour positif nous est revenu.

## 4.4 Code coverage

### 4.4.1 EclEmma

EclEmma est un plugin permettant d'analyser le pourcentage de code employé lors de l'utilisation du Jpacman. Grâce à cette vérification, on pourra plus facilement supprimer certaines lignes de codes inutilisés par le projet.












Avant tout, il faut installer le plugin via le marketplace d'eclipse comme pour PMD. Ensuite, on exécute la classe

« **launcher.java** » via le bouton  dans la barre de tâche de votre IDE (ici utilisé avec eclipse), puis on joue au Jpacman jusqu'à la fin de la partie.

Une fois celle-ci terminée et l'application fermée, on retrouve une fenêtre indiquant le pourcentage de code utilisé lors de l'exécution de l'application.

### 1. Code d'origine

On peut s'apercevoir sur la figure ci-dessous que le pourcentage d'utilisation du code est de **76%** avec le code d'origine. Ce pourcentage est assez variable, cela dépend de l'utilisation de certaines classes.

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
▼ jpacman-framework	 58,1 %	3.314	2.386	5.700
> src/test/java	 0,0 %	0	1.342	1.342
▼ src/main/java	 76,0 %	3.314	1.044	4.358
> nl.tudelft.jpacman.level	 66,8 %	863	429	1.292
> nl.tudelft.jpacman.board	 65,4 %	369	195	564
> nl.tudelft.jpacman.ui	 80,4 %	619	151	770
> nl.tudelft.jpacman.sprite	 78,7 %	485	131	616
> nl.tudelft.jpacman.npc.ghost	 91,6 %	608	56	664
> nl.tudelft.jpacman	 81,9 %	245	54	299
> nl.tudelft.jpacman.game	 81,3 %	122	28	150
> nl.tudelft.jpacman.npc	 100,0 %	3	0	3
















## 2. Final

Concernant le code final, on retrouve un pourcentage plus élevé de l'utilisation du code (image ci-dessous). Cela peut se traduire par l'ajout des extensions qui sont employées en même temps. On peut remarquer que par exemple pour le package « **fruit** », on a une utilisation presque parfaite mais cela peut varier en fonction du choix du joueur d'utiliser les fruits présents sur la carte.

Autre exemple, on voit que pour le package « **multiplayers** » on a une utilisation de **81,9%**. On peut l'expliquer en partie à cause de la classe « **player** » où certains getters et setters définis par défaut ne sont pas utilisés ainsi que par la classe « **classement** » qui, en fonction du nombre de joueur utilise plus de ligne de code. Afin d'augmenter l'utilisation du code de cette classe, nous pourrions supprimer les setters et getters initialisés.

Nous pourrions faire de même pour les différents packages du projet afin d'essayer d'utiliser au maximum toutes les lignes de code du projet.

Launcher (1) (03-mai-2016 14:35:10)

Element	Coverage	Covered Ins...	Missed Instr...	Total Instruc...
▼ jpacman-framework	 67,4 %	5.569	2.689	8.258
▼ src/main/java	 80,0 %	5.569	1.390	6.959
> nl.tudelft.jpacman.npc	 100,0 %	3	0	3
> nl.tudelft.jpacman.fruit	 96,6 %	112	4	116
> nl.tudelft.jpacman.controls	 90,4 %	377	40	417
> nl.tudelft.jpacman.npc.ghost	 89,1 %	665	81	746
> nl.tudelft.jpacman	 84,4 %	260	48	308
> nl.tudelft.jpacman.sprite	 81,9 %	508	112	620
> nl.tudelft.jpacman.multiplayers	 81,9 %	564	125	689
> nl.tudelft.jpacman.ui	 81,4 %	808	185	993
> nl.tudelft.jpacman.specialcase	 74,8 %	89	30	119
> nl.tudelft.jpacman.board	 74,6 %	460	157	617
> nl.tudelft.jpacman.level	 74,4 %	1.576	542	2.118
> nl.tudelft.jpacman.game	 69,0 %	147	66	213
> src/test/java	 0,0 %	0	1.299	1.299

## 4.5 Analyse de dépendances

### 4.5.1 Analyse métriques - Jdepend V2.9

#### Version originale :

Package	CC	AC	Ca	Ce	A	I	D
jpacman	6	0	2	10	0	0.83	0.17
board	5	2	7	4	0.29	0.36	0.35
game	2	1	2	5	0.33	0.71	0.05
level	0	1	2	1	0.33	0.69	0.13
npc	6	0	2	10	0	0.33	0.33
ghost	8	1	1	6	0.11	0.67	0.22
sprite	5	1	5	9	0.17	0.64	0.19
ui	11	2	1	10	0.15	0.91	0.06

#### Version avec extensions :

Package	CC	AC	Ca	Ce	A	I	D
jpacman	7	0	3	12	0	0.8	0.2
board	6	2	9	5	0.25	0.36	0.39
controls	5	1	2	6	0.17	0.75	0.08
fruit	6	0	2	4	0	0.67	0.33
game	2	1	3	7	0.33	0.7	0.03
level	16	3	6	12	0.16	0.67	0.18
multiplayers	3	0	3	6	0	0.67	0.33
npc	0	1	2	2	1	0.5	0.5
ghost	6	1	4	7	0.1	0.5	0.5
specialcase	5	0	4	4	0	0.5	0.5
sprite	5	1	7	9	0.17	0.56	0.27
ui	14	3	1	11	0.18	0.92	0.09

CC : Nombre de classes concrètes.

CA : Nombre de classes abstraites.

Ca : Couplage afférent : nombre de packages qui dépendent des classes de ce package. Indicateur du niveau de responsabilité du package.

Ce : Couplage efférent : nombre de packages utilisés par les classes de ce package. Indicateur du niveau d'indépendance du package.

A : Abstraction : Ratio du nombre de classes abstraites par rapport au nombre total de classes du package.

I : Instabilité : Ratio du nombre de couplages efférents par rapport au nombre total de couplages (Ce + Ca).

D : Distance avec l'équilibre : Indicateur de l'équilibre entre abstraction et stabilité.

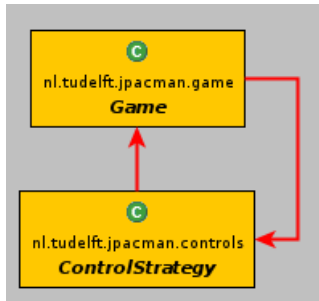
Une distance de zéro implique que le package est soit totalement abstrait et stable soit totalement concret et instable.

Certains packages sont très concrets et instables, il faut éviter les couplages afférents (en rouge) avec ces packages afin d'éviter de transmettre ces instabilités. Idéalement la distance avec l'équilibre entre abstraction et stabilité doit s'approcher de zéro mais une valeur inférieure à 0,5 est acceptable. Les packages ajoutés pour l'extension ont tous une valeur inférieure ou égale à 0,5. Pour les packages originaux, la distance augmente très légèrement. Sans compter les nouveaux packages, on peut observer une augmentation globale de la distance de 5% ce qui n'est pas considéré comme significatif.

#### 4.5.2 Dépendances cycliques

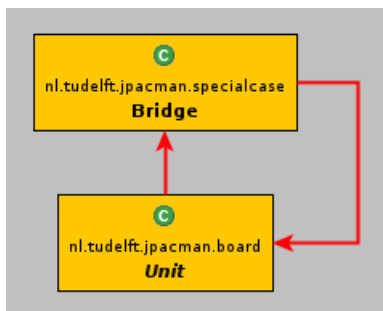
Les dépendances cycliques sont détectées grâce au logiciel Class Dependency Analyzer (V1.16.0). Ces dépendances sont indésirables parce qu'elles compliquent la séparation et la réutilisation du code source. Les cycles entre les packages qui appartiennent à la même branche de la hiérarchie sont ignorés, deux dépendances cycliques sont ajoutées par l'extension.

##### 1. Entre les packages game et controls :



Cette dépendance est due aux méthodes ajoutées à la classe **Game** qui permettent de récupérer l'objet de type **ControlStrategy** en cours d'utilisation et de changer la stratégie courante. La dépendance cyclique peut être évitée en supprimant les méthodes ajoutées à la classe **Game**. La récupération de l'objet de type **ControlStrategy** peut être évitée en conservant une référence vers cet objet et le changement de stratégie peut être géré en ajoutant une méthode qui le permet directement dans la classe **ControlStrategy**.

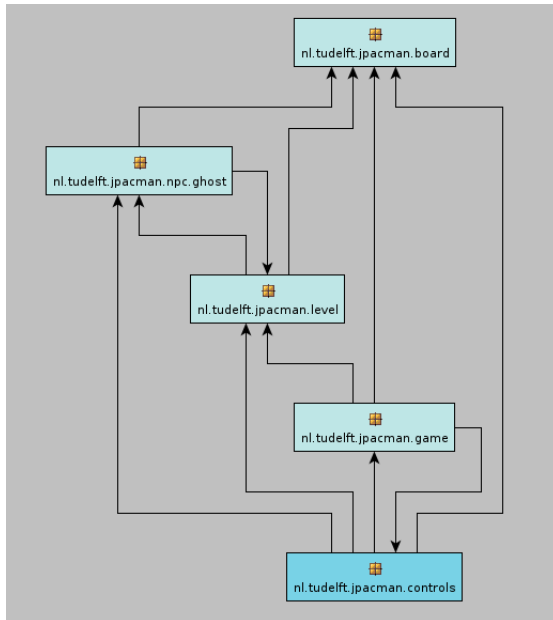
##### 2. Entre les packages bridge et unit



Cette dépendance est due aux méthodes ajoutées dans la classe **Bridge** permettant de récupérer un objet **Unit** en cours d'utilisation et de définir où il se trouve sur le pont. La dépendance cyclique peut être évitée en supprimant les méthodes ajoutées à la classe **Bridge**.

## Dépendances des packages ajoutés pour les extensions :

### 1. Dépendances du packages controls :



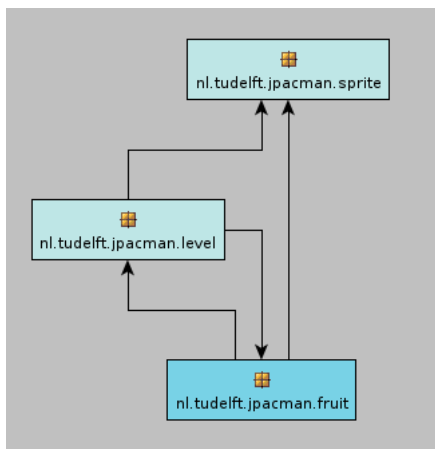
`nl.tudelft.jpacman.game` : nécessaire pour le changement de position du joueur.

`nl.tudelft.jpacman.level` : nécessaire pour la vérification du contenu des cases de la grille de jeu.

`nl.tudelft.jpacman.npc.ghost` : nécessaire pour la détection des fantômes.

`nl.tudelft.jpacman.board` : nécessaire pour la récupération des cases de la grille de jeu.

### 2. Dépendance du packages fruit :

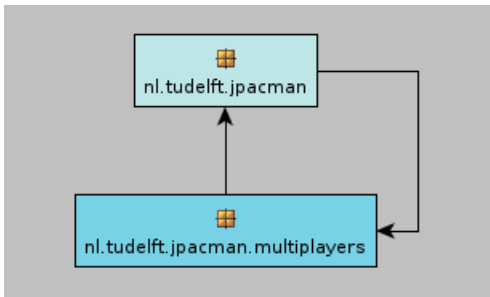


`nl.tudelft.jpacman.level` : Nécessaire pour associer les fruits à un niveau.

`nl.tudelft.jpacman.sprite` : Nécessaire pour les sprites des fruits.

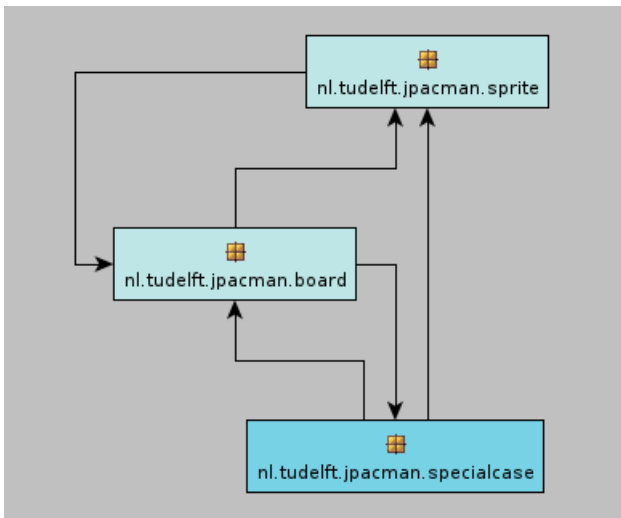


### 3. Dépendances du package multiplayer :



nl.tudelft.jpacman : indispensable pour la création du joueurs (Ghost)

### 4. Dépendances du package specialcase :



nl.tudelft.jpacman.board : nécessaire pour définir une case du board comme case spéciale.

nl.tudelft.jpacman.sprite : Nécessaire pour les sprites des cases spéciales.

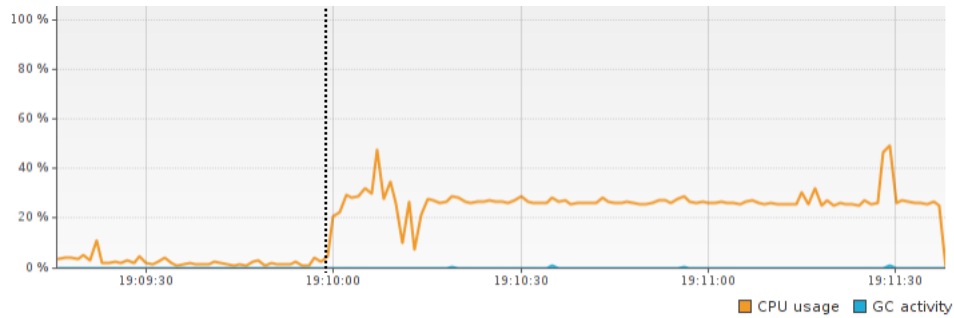
Les dépendances des packages ajoutés sont bien justifiées puisqu'elles sont nécessaires à la mise en ?uvre des fonctionnalités apportées par l'extension. Seules les deux dépendances cycliques détectées au point précédant doivent être corrigées.

## 4.6 Analyse de performance

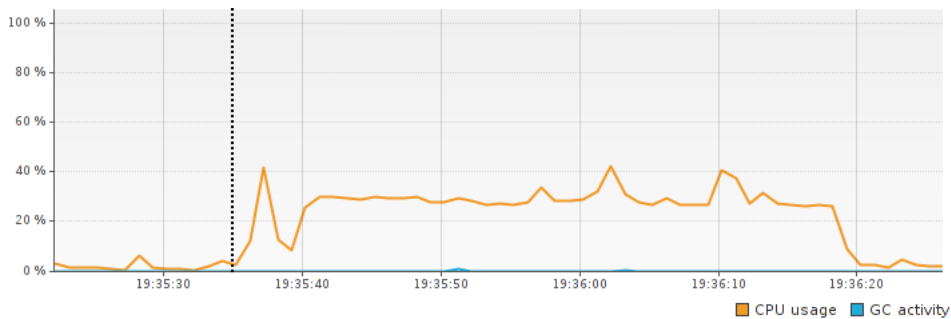
### 4.6.1 VisualVM V1.3.8

#### 1. Monitoring CPU

(a) Version originale



(b) Version finale



On observe que les performances entre les deux versions du logiciel sont assez similaires. Pendant l’affichage du menu (avant les pointillés), l’utilisation du processeur est assez faible (moins de 10%). En cours de partie l’utilisation du processeur augmente significativement mais reste toujours inférieure à 50%. On observe aussi que les ressources sont bien libérées en fin de partie, l’utilisation du processeur devient rapidement inférieure à 5 % quand la partie se termine.

## 2. Répartition du temps CPU

### (a) Verison originale

Hot Spots - Method	Self Time (%) ▼	Self Time	Total Time	Invocations
java.util.concurrent.ThreadPoolExecutor\$Worker.run ()	134.544 ... (80.3 %)	134.544 ms	156.927 ms	4
nl.tudelft.jpacman.npc.ghost.Navigation.shortestPath (nl.tudelft.jpacman.board.Square, nl.tudelft.jpacman.board.Square, nl.tudelft.jpacman.board.Unit)	16.317 ... (9.7 %)	16.317 ms	18.614 ms	602
nl.tudelft.jpacman.sprite.ImageSprite.draw (java.awt.Graphics, int, int, int, int)	2.264 ms (1.4 %)	2.264 ms	2.264 ms	860.644
nl.tudelft.jpacman.npc.ghost.Navigation.addNewTargets (nl.tudelft.jpacman.board.Unit, java.util.List, java.util.Set, nl.tudelft.jpacman.npc.ghost.Navigation.Node, nl.tudelft.jpacman.board.Square)	1.428 ms (0.9 %)	1.428 ms	2.167 ms	724.732
nl.tudelft.jpacman.ui.BoardPanel.render (nl.tudelft.jpacman.board.Square, java.awt.Graphics, int, int, int, int)	962 ms (0.6 %)	962 ms	7.331 ms	711.459
com.google.common.collect.Iterators.forArray (Object[], int, int, int)	771 ms (0.5 %)	771 ms	1.683 ms	992.022
nl.tudelft.jpacman.ui.BoardPanel.render (nl.tudelft.jpacman.board.Square, java.awt.Graphics, java.awt.Dimension)	769 ms (0.5 %)	769 ms	8.342 ms	1.473
javaw.swing.RepaintManager\$ProcessingRunnable.run ()	578 ms (0.3 %)	578 ms	8.926 ms	1.566
nl.tudelft.jpacman.level.Level.move (nl.tudelft.jpacman.board.Unit, nl.tudelft.jpacman.board.Direction)	565 ms (0.3 %)	565 ms	3.225 ms	838
nl.tudelft.jpacman.level.Level.remainingPellets ()	540 ms (0.3 %)	540 ms	2.633 ms	838
com.google.common.collect.ImmutableList.construct (Object[])	536 ms (0.3 %)	536 ms	2.366 ms	1.244.895
nl.tudelft.jpacman.board.Square.getSquareAt (nl.tudelft.jpacman.board.Direction)	530 ms (0.3 %)	530 ms	530 ms	9.410.823
nl.tudelft.jpacman.npc.ghost.Navigation.findNearest (Class, nl.tudelft.jpacman.board.Square)	470 ms (0.3 %)	470 ms	1.456 ms	602
com.google.common.collect.ImmutableList.copyOf (java.util.Collection)	437 ms (0.3 %)	437 ms	2.804 ms	1.244.895
nl.tudelft.jpacman.board.Square.getOccupants ()	420 ms (0.3 %)	420 ms	3.224 ms	1.244.895
com.google.common.collect.ObjectArrays.checkElementsNotNull (Object[])	387 ms (0.2 %)	387 ms	654 ms	1.244.895
com.google.common.collect.ImmutableList.asImmutableList (Object[], int)	380 ms (0.2 %)	380 ms	799 ms	1.244.895
com.google.common.collect.ImmutableList.asImmutableList (Object[])	376 ms (0.2 %)	376 ms	1.175 ms	1.244.895
com.google.common.collect.ImmutableList.listIterator ()	332 ms (0.2 %)	332 ms	2.304 ms	992.022
com.google.common.collect.ImmutableList.iterator ()	299 ms (0.2 %)	299 ms	2.895 ms	992.022
com.google.common.base.Preconditions.checkPositionIndex (int, int)	297 ms (0.2 %)	297 ms	441 ms	994.773
com.google.common.collect.ImmutableList.iterator ()	291 ms (0.2 %)	291 ms	2.595 ms	992.022
com.google.common.collect.RegularImmutableList.listIterator (int)	289 ms (0.2 %)	289 ms	1.972 ms	992.022
com.google.common.collect.ObjectArrays.checkElementsNotNull (Object[], int)	229 ms (0.1 %)	229 ms	267 ms	1.244.895
nl.tudelft.jpacman.ui.ScorePanel\$1.format (nl.tudelft.jpacman.level.Player)	209 ms (0.1 %)	209 ms	209 ms	1.510
nl.tudelft.jpacman.ui.PacManUI.nextFrame ()	189 ms (0.1 %)	189 ms	446 ms	1.510
nl.tudelft.jpacman.board.Board.squareAt (int, int)	181 ms (0.1 %)	181 ms	1.116.213	
nl.tudelft.jpacman.board.Direction.values ()	166 ms (0.1 %)	166 ms	852.131	
com.google.common.base.Preconditions.checkArgument (boolean)	157 ms (0.1 %)	157 ms	157 ms	992.022
com.google.common.collect.Iterators.emptyListIterator ()	157 ms (0.1 %)	157 ms	157 ms	989.271

### (b) Version finale

Hot Spots - Method	Self Time (%) ▼	Self Time	Total Time	Invocations
java.util.concurrent.ThreadPoolExecutor\$Worker.run ()	70.532 ... (65.9 %)	70.532 ms	78.605 ms	9
nl.tudelft.jpacman.controls.ControlStrategy\$1.run ()	16.476 ... (15.4 %)	16.476 ms	17.233 ms	3
nl.tudelft.jpacman.npc.ghost.Navigation.shortestPath (nl.tudelft.jpacman.board.Square, nl.tudelft.jpacman.board.Square, nl.tudelft.jpacman.board.Unit)	4.268 ms (4 %)	4.268 ms	5.338 ms	245
nl.tudelft.jpacman.sprite.ImageSprite.draw (java.awt.Graphics, int, int, int, int)	2.826 ms (2.6 %)	2.826 ms	2.826 ms	1.117.603
nl.tudelft.jpacman.level.Level.move (nl.tudelft.jpacman.board.Unit, nl.tudelft.jpacman.board.Direction)	1.323 ms (1.2 %)	1.323 ms	2.588 ms	275
nl.tudelft.jpacman.ui.BoardPanel.render (nl.tudelft.jpacman.board.Square, java.awt.Graphics, int, int, int, int)	1.188 ms (1.1 %)	1.188 ms	8.757 ms	843.507
com.google.common.collect.Iterators.forArray (Object[], int, int, int)	906 ms (0.9 %)	906 ms	1.546 ms	682.783
nl.tudelft.jpacman.ui.BoardPanel.render (nl.tudelft.jpacman.board.Square, java.awt.Graphics, java.awt.Dimension)	882 ms (0.9 %)	882 ms	9.933 ms	1.747
javaw.swing.RepaintManager\$ProcessingRunnable.run ()	639 ms (0.6 %)	639 ms	10.580 ms	1.789
nl.tudelft.jpacman.npc.ghost.Navigation.addNewTargets (nl.tudelft.jpacman.board.Unit, java.util.List, java.util.Set, nl.tudelft.jpacman.npc.ghost.Navigation.Node, nl.tudelft.jpacman.board.Square)	541 ms (0.5 %)	541 ms	1.015 ms	222.032
com.google.common.collect.ImmutableList.construct (Object[])	458 ms (0.4 %)	458 ms	2.104 ms	1.006.605
com.google.common.collect.ImmutableList.copyOf (java.util.Collection)	398 ms (0.4 %)	398 ms	2.502 ms	1.006.605
nl.tudelft.jpacman.board.Square.getOccupants ()	347 ms (0.3 %)	347 ms	2.849 ms	1.006.605
com.google.common.collect.ImmutableList.asImmutableList (Object[], int)	330 ms (0.3 %)	330 ms	797 ms	1.006.604
com.google.common.collect.ImmutableList.asImmutableList (Object[])	308 ms (0.3 %)	308 ms	1.105 ms	1.006.604
com.google.common.collect.ObjectArrays.checkElementsNotNull (Object[])	289 ms (0.3 %)	289 ms	540 ms	1.006.605
nl.tudelft.jpacman.level.Level.remainingPellets ()	276 ms (0.3 %)	276 ms	1.117 ms	277
nl.tudelft.jpacman.ui.ScorePanel\$1.format (nl.tudelft.jpacman.level.Player)	264 ms (0.2 %)	264 ms	264 ms	1.770
nl.tudelft.jpacman.ui.PacManUI.nextFrame ()	240 ms (0.2 %)	240 ms	641 ms	1.770
com.google.common.collect.ImmutableList.iterator ()	236 ms (0.2 %)	236 ms	2.398 ms	682.783
com.google.common.base.Preconditions.checkPositionIndex (int, int)	211 ms (0.2 %)	211 ms	320 ms	685.044
com.google.common.collect.RegularImmutableList.listIterator (int)	211 ms (0.2 %)	211 ms	1.759 ms	682.783
com.google.common.collect.ImmutableList.listIterator ()	206 ms (0.2 %)	206 ms	1.966 ms	682.783
com.google.common.collect.ObjectArrays.checkElementsNotNull (Object[], int)	203 ms (0.2 %)	203 ms	250 ms	1.006.605
com.google.common.collect.ImmutableList.iterator ()	195 ms (0.2 %)	195 ms	2.162 ms	682.783
nl.tudelft.jpacman.board.Board.squareAt (int, int)	173 ms (0.2 %)	173 ms	977.298	
nl.tudelft.jpacman.npc.ghost.Navigation.Node.<init> (nl.tudelft.jpacman.board.Direction, nl.tudelft.jpacman.board.Square, nl.tudelft.jpacman.npc.ghost.Navigation.Node, nl.tudelft.jpacman.board.Square)	156 ms (0.1 %)	156 ms	259 ms	416.840
com.google.common.collect.SingletonImmutableList.<init> (Object)	149 ms (0.1 %)	149 ms	357 ms	323.823
nl.tudelft.jpacman.npc.ghost.Navigation.findNearest (Class, nl.tudelft.jpacman.board.Square)	140 ms (0.1 %)	140 ms	635 ms	245
nl.tudelft.jpacman.board.Square.getSquareAt (nl.tudelft.jpacman.board.Direction)	139 ms (0.1 %)	139 ms	139 ms	1.001.310

On remarque que des objets de type ControlStrategy utilisent environ 15% du temps CPU consacré au logiciel. Cette proportion semble acceptable puisque les objets de type ControlStrategy sont en charge des mouvements du joueur et de l'intelligence artificielle (activée dans ce cas) qui permet de déplacer le joueur automatiquement. Ces objets sont basés sur une boucle qui ne s'arrête qu'en fin de partie. Cette boucle est exécutée dans un nouveau fil d'exécution afin d'éviter de ralentir les autres fonctionnalités du logiciel.