Projet de Software Evolution
Année Académique 2014-2015

Tom Mens et Maëlick Claes

28 janvier 2015

Résumé

Ce document présente le projet devant être réalisé dans le cadre du cours de Software Evolution. Le travail à rendre, par groupe de deux étudiants, compte pour 40% de la note finale du cours. Si vous pensez certaines consignes incomplètes, ambiguës ou contradictoires, veuillez en faire part aux enseignants en charge du projet.

1 Objectifs

Le projet s'inscrit dans le cours "Software Evolution". Le but est de mettre en pratique les techniques et outils étudiés pendant le cours théorique et les travaux pratiques. Plus précisément, dans le cadre de ce projet, vous serez amené à

- **analyser la qualité** d'un 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.

2 Approche

2.1 Répartition du travail

Le travail sera réalisé par équipes de deux étudiants. Si le nombre d'étudiants participant au projet est impair, une équipe d' un à trois étudiants sera constituée. Vous êtes libres de constituer vos équipes. Si cela s'avère nécessaire, les enseignants créeront des équipes en y affectant aléatoirement les étudiants n'ayant pas

trouvé d'équipiers lors du déroulement de votre projet. Dès qu'une équipe est créée, il lui est attribué un numéro qui sera utilisé pour l'identifier.

Les équipes seront fixées pour toute la durée du projet. Vous êtes priés d'avertir les enseignants en cas de problème majeur avec votre/vos coéquipier(s).

L'évaluation du travail sera réalisée par équipe, à moins qu'un membre d'une équipe ait fourni un travail de qualité radicalement différente de celle de ses coéquipiers.

2.2 Étapes clés

Le projet est constitué de plusieurs étapes clés que vous devez réaliser successivement. Au terme du projet, chaque équipe devra rendre les différentes versions du code source (incluant les tests unitaires) et un rapport, suivant les consignes de la section 2.3.

Les étapes clés du projet sont les suivantes :

1. Analyse de la qualité de la première version du code (section 3.1)
2. Ajout de tests unitaires à la première version du code (section 3.2), et vérification de la couverture des tests
3. Refactoring du code pour en améliorer la qualité et la structure (section 3.3)
4. Analyse des améliorations de qualité et tests de régression (section 3.4)
5. Extension du logiciel et ajout des tests unitaires pour cette extension (section 3.5)
6. Analyse de la qualité de cette extension et tests de régression (section 3.6)
7. Étude de l'historique de la qualité logicielle entre toutes les versions du code (section 3.7)

2.3 Livrables

Le travail effectué sera évalué sur base d'un travail rendu pour le **2 avril 2015** au plus tard. **Aucun retard ne sera toléré.** Si vous ne respectez pas l'échéance vous aurez d'office 0/20 pour le projet, qui compte pour 40% de la note finale du cours.

Le travail à rendre sera une archive composée de plusieurs choses :
— Le code source et exécutable des différentes versions du logiciel (une pour chaque étape clé)
— Un rapport en format PDF. Les noms des étudiants doivent être mis sur la première page du rapport. Tout d'abord, le rapport doit expliquer la démarche suivie par les étudiants, et présenter et justifier les outils utilisés. Ensuite, l'historique du code doit être détaillé : quel est le but de chaque version du logiciel, et quelle est la différence de chaque version par rapport à la version précédente ? Une analyse de qualité de chaque version doit être fournie, avec une explication sur la manière dont cette analyse a été effectuée, ainsi qu'une justification de la manière dont certains aspects de la qualité ont été améliorés ou détériorés par rapport à la version précédente. D'autres observations, remarques ou avis importants peuvent être mis dans le rapport également.
— Tout autre document ou fichier supplémentaire peut être joint au travail si l'étudiant le juge nécessaire (par exemple, jeux de données de test).

Le nom de l'archive et du rapport doivent respecter le format suivant : "rapportEvol<*année*>-<*numéro équipe* >.<*file extension* >"

3 Étapes clés – en détail

Ci-dessous, nous détaillerons ce qui est attendu pour chaque étape clé. C'est à vous de choisir et justifier les outils qui vous permettront de réaliser le travail, tenant compte de ce qui a été expliqué lors du cours et des travaux pratiques, et tenant compte de la disponibilité des outils.

3.1 Première analyse de la qualité du logiciel

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 nous vous 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 dégoulottage du système.

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

3.2 Ajout de tests unitaires

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.

Étendez 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* ?

3.3 Refactoring en vue d'améliorer la qualité

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.

3.4 Analyse de la qualité du logiciel

Réalisez une étude similaire à celle décrite en Section 3.1. 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 ?

3.5 Extensions

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 différentes, décrites dans la section 4.

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.

3.6 Analyse de la qualité du logiciel

Pour chaque extension ajoutée, réalisez une analyse de qualité similaire à celle décrite en Section 3.1. Au vu de cette analyse, quels sont les points qui devraient à présent être améliorés ?

3.7 Analyse de l'évolution de la qualité logicielle

Analysez l'évolution de la qualité du logiciel entre les différentes versions, en utilisant les résultats d'analyse de qualité des sections 3.1, 3.4 et 3.6. Montrez cette évolution graphiquement et interprétez-la.

4 Énoncé : Pac-Man

Pac-Man est un jeu vidéo créé en 1980 (`http://fr.wikipedia.org/wiki/Pac-Man`). Le joueur dirige un personnage jaune en forme de camembert appelé **Pac-Man**. Ce jeu vidéo a connu un grand succès en salle d'arcade, et de nombreux clones et variantes du jeu ont été réalisés sur diverses plateformes, y compris sur PC. Le but du jeu est de terminer une série de niveaux. Chaque niveau est constitué d'un labyrinthe dans lequel Pac-Man se promène. Le labyrinthe est également peuplé de **fantômes** qui, la plupart du temps, tentent de toucher (et ainsi de tuer) Pac-Man. Celui-ci doit éviter les fantômes et manger toutes les **gommes** se trouvant sur son chemin. Un niveau est terminé dès que toutes les gommes du labyrinthe ont été mangées.

Dans chaque labyrinthe se trouvent quatre fantômes, chacun ayant une couleur et un nom uniques. Il s'agit de :
— Blinky, le fantôme rouge ;
— Pinky, le fantôme rose ;
— Inky, le fantôme bleu ;
— Clyde, le fantôme orange.

Chaque fantôme essaie d'attraper Pac-Man en ayant recours à une méthode qui lui est propre. Dans certaines circonstances, la proie devient chasseur : les fantômes s'enfuient pour éviter que Pac-Man ne puisse les toucher (et ainsi de les tuer). Dans le jeu, les fantômes sont dirigés par une intelligence artificielle (IA) presque entièrement déterministe tandis que Pac-Man est contrôlé par le joueur. Le comportement des fantômes est détaillé dans la section 4.6.

Chaque équipe devrait réaliser au moins deux extensions du jeu Pac-Man, selon la répartition suivante :

Équipe	Extension 1	Extension 2
1	IA pour les fantômes	Série de labyrinthes
2	IA pour les fantômes	Supergomme
3	IA pour les fantômes	Hall of Fame
4	IA pour Pac-Man	Série de labyrinthes
5	IA pour Pac-Man	Supergomme
6	IA pour Pac-Man	Hall of Fame
7	IA pour Pac-Man	Diriger les fantômes

4.1 Hall of Fame

Au début du jeu, le score du joueur est nul. Chaque fois que Pac-Man mange une gomme, un fantôme ou un fruit, cela augmente la valeur du compteur. Les gommes comptent pour 10 points et les fantômes augmentent le compteur de 10 points. Les fruits mangés par Pac-Man font également monter le compteur de la manière suivante :

— Les cerises de 100 points ;
— Les fraises de 300 points ;
— Les oranges de 500 points ;
— Les melons de 1000 points.

À la fin du jeu, si le score du joueur est supérieur au dixième score le plus élevé, il doit entrer son nom afin de figurer dans le *Hall of Fame* (ne retenant que les 10 meilleurs scores) qui s'affiche à la fin de chaque partie.

Les dix meilleurs scores doivent être enregistrés, de sorte que le *Hall of Fame* survive à un redémarrage du jeu. Le joueur doit aussi avoir la possibilité de remettre les scores du *Hall of Fame* à zéro. Pour éviter la remise à zéro par erreur, il doit confirmer son choix dans une fenêtre.

4.2 Déplacement continu

Actuellement, Pac-Man se déplace d'une case chaque fois que le joueur appuie sur une touche. Modifiez le jeu pour que Pac-Man continue à avancer devant lui tant qu'il ne fait pas face à un mur et qu'il ne reçoit pas d'autres instructions. Si l'instruction ne peut pas être réalisée (par exemple, si le joueur appuie sur la flèche du bas alors qu'un mur se trouve en dessous de Pac-Man), elle est simplement ignorée.

4.3 Série de labyrinthes

Actuellement, la partie s'achève dès qu'un fantôme a touché Pac-Man (et Pac-Man meurt) ou dès que Pac-Man a avalé toutes les gommes du labyrinthe.

Faites en sorte que Pac-Man dispose en début de partie de 3 vies. Lorsqu'il est touché par un fantôme et s'il lui reste une vie, Pac-Man perd une vie et est *téléporté* sur une case aléatoire se trouvant à au moins 4 cases du fantôme le plus proche. Lorsque Pac-Man perd sa dernière vie, la partie est terminée. Chaque fois que Pac-Man gagne 10.000 points, cela lui procure une vie supplémentaire.

Faites également en sorte que, lorsque Pac-Man termine un niveau avec succès, un autre niveau soit mis en place. Pac-Man commence ce nouveau niveau avec le nombre de vies qu'il avait à la fin du niveau précédent. Mettez en place un système permettant de facilement indiquer l'ordre dans lequel les niveaux doivent être présentés. Proposez une suite de quelques niveaux.

4.4 IA pour Pac-Man

Faites en sorte que Pac-Man puisse être dirigé par une intelligence artificielle et non par un humain. Celle-ci doit tenter d'avoir un score final aussi élevé que possible. Voyez par exemple le site de konket.net [1] pour des stratégies de déplacement. Avant de commencer une partie, le joueur doit pouvoir choisir entre contrôler Pac-Man ou être un spectateur passif de la partie.

Utilisez un *Strategy Design Pattern* pour implémenter le comportement de Pac-Man. Le contrôle de Pac-Man par un humain sera obtenu au moyen d'une stratégie particulière.

Le système doit être tel que l'ajout de nouvelles stratégies soit possible et facile : les stratégies doivent avoir toutes les informations nécessaires pour agir en état de cause. En aucune manière, les stratégies ne doivent pouvoir tricher en manipulant les données du jeu ou en donnant des instructions contraires aux règles du jeu.

1. `http://archive.kontek.net/sngp.classicgaming.gamespy.com/games/pac/pac_faq.htm`

4.5 Diriger les fantômes

Avec la fonctionnalité d'IA pour Pac-Man déjà implémentée (voir section 4.4), prévoyez la possibilité d'inverser le jeu : au lieu de diriger Pac-Man dans le but de manger toutes les gommes sans être mangé par les fantômes, le joueur peut choisir de diriger un des fantômes. Après sélection du fantôme, le but du jeu devient de manger Pac-Man. Pour rendre le jeu plus difficile, l'IA de Pac-Man peut tenir compte du fait que l'un des fantômes est dirigé par le joueur.

4.6 IA pour les fantômes

Actuellement, le comportement des fantômes est assez erratique, car la direction empruntée par chacun d'eux est déterminée aléatoirement. Commencez par attribuer une couleur à chacun des quatre fantômes du jeu, afin que le joueur puisse les différencier. Créez des IA pour les fantômes afin qu'ils prennent des décisions plus *intéressantes* pour le joueur. Une décision de direction n'est prise que lorsqu'un fantôme arrive à un embranchement. La décision est basée sur le calcul de la plus courte distance entre la position de case de l'embranchement et la position que le fantôme souhaite atteindre. Si plusieurs directions sont pareillement préférables, un fantôme préférera toujours aller en haut, puis à gauche, puis en bas. Un fantôme ne peut donc aller à droite que si cette direction est la seule représentant la plus petite distance jusqu'à la destination.

Le comportement des fantômes alterne entre un mode de **poursuite** et un mode de **dispersion**.

En mode poursuite, chaque fantôme a un comportement et une vitesse spécifiques et déterministes [2] :

— **Blinky** a une philosophie très simple : « Droit au but ! ». En toute circonstance, il tente de se rendre sur la case où se trouve Pac-Man par le chemin le plus court. Il a une vitesse de déplacement de 100%.
— **Pinky** aime tendre des embuscades. Il devine où sera Pac-Man 4 mouvements plus tard et se rend à cette position. Il a une vitesse de déplacement de 80%.
— **Inky** est difficilement prévisible pour un humain, car son comportement dépend des positions et des directions de Pac-Man et de Blinky. Imaginez un vecteur joignant la position de Blinky à celle où se trouvera Pac-Man dans 2 mouvements. Doublez la longueur de ce vecteur. La position que Inky cherche à atteindre est à l'extrémité de ce vecteur. Il a une vitesse de déplacement de 100%.
— **Clyde** ne semble pas se préoccuper des autres. Il a en fait deux modes de fonctionnement, et il passe de l'un à l'autre en fonction de la distance existant entre Pac-Man et lui. S'il se trouve à plus de 8 cases de Pac-Man, Clyde a le même comportement que Blinky. Dans le cas contraire, il se rend sur la case située dans le coin inférieur gauche du labyrinthe. Il a une vitesse de déplacement de 100%.

En mode dispersion, chaque fantôme se dirige vers sa position "maison". Ces positions sont situées aux quatre coins du labyrinthe :

— **Blinky** se rend dans le coin supérieur droit.
— **Pinky** se rend dans le coin supérieur gauche.
— **Inky** se rend dans le coin inférieur droit.
— **Clyde** se rend dans le coin inférieur gauche.

Une fois sa position "maison" atteinte, le fantôme va se déplacer dans un circuit qui consiste à toujours emprunter le chemin de gauche pour Pinky et Clyde, et celui de droite pour Blinky et Inky (cf. Figure 1)

L'alternance entre ces modes est définie comme suit :

2. `http://gameinternals.com/post/2072558330/understanding-pac-man-ghost-behavior`

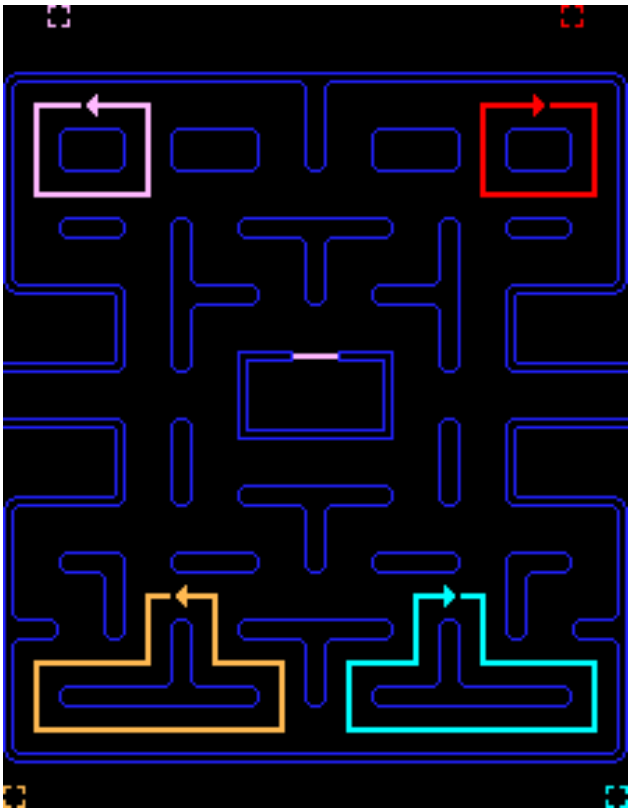

FIGURE 1 – Circuits empruntés par les fantômes en mode dispersion.

— Dispersion pendant 7 secondes, puis poursuite pendant 20 secondes.
— Dispersion pendant 7 secondes, puis poursuite pendant 20 secondes.
— Dispersion pendant 5 secondes, puis poursuite pendant 20 secondes.
— Dispersion pendant 5 secondes, puis poursuite indéfiniment.

Utilisez un *Strategy Design Pattern* pour implémenter les comportements des fantômes. L'alternance entre la dispersion et la poursuite sera ainsi réalisée grâce à un changement de stratégie. De même, les aspects communs des comportements des fantômes seront implémentés sous la forme de stratégies partagées par les antagonistes concernés.

Le système doit être tel que l'ajout de nouvelles stratégies soit possible et facile : les stratégies doivent avoir toutes les informations nécessaires pour agir en état de cause. En aucune manière, les stratégies ne doivent pouvoir tricher en manipulant les données du jeu ou en donnant des instructions contraires aux règles du jeu.

4.7 Supergomme

Modifiez le jeu afin qu'il prenne en compte la présence de *supergommes*. Lorsque Pac-Man mange une supergomme, elle lui rapporte 50 points.

Pendant un court moment, les règles du jeu changent (le jeu se met en mode *fuite*) et la proie devient chasseur. Lorsque Pac-Man mange une supergomme, les fantômes sont effrayés. Le chronomètre du mode dans lequel ils se trouvent s'arrête et les fantômes entrent en mode fuite : ils deviennent bleus et leur vitesse de déplacement est réduite à 50% (y compris pour Pinky). En mode fuite, chaque fantôme décide aléatoirement de la direction à prendre à chaque embranchement. Si Pac-Man touche un fantôme en mode fuite, celui-ci disparaît du jeu et réapparaît au milieu du labyrinthe après 5 secondes.

Si Pac-Man touche un fantôme en mode fuite, il le mange, ce qui lui rapporte :

— 200 points pour le premier fantôme mangé.
— 400 points pour le second fantôme mangé.
— 800 points pour le troisième fantôme mangé.
— 1600 points pour le quatrième fantôme mangé.

Il y a quatre supergommes par niveau. Les deux premières à être avalées effraient les fantômes pendant 7 secondes, tandis que les deux dernières les effraient pendant 5 secondes. Après ce délai, les fantômes retournent dans le mode dans lequel ils étaient avant de fuir. Le chronomètre associé à ce mode est repris là où il s'était arrêté.