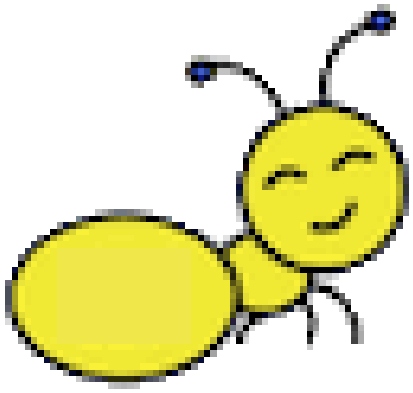
GHANEM Hakim
GARCIA Guillaume

Rapport de projet POO:

Ants.Vs.Bees



Introduction et Résumé du projet

Le but de ce projet était de concevoir un jeu vidéo du genre stratégie en temps réel, de type « tower defense » tel le jeu Plants vs. Zombies, que l'on nommera Fourmis contre Abeilles (Ants vs. Bees en anglais). Une image jeu est disponible en Annexe 2.

Le but de ce jeu est de protéger la reine des fourmis et d'empêcher que les abeilles atteignent le fond de la fourmilière sinon la partie est perdue. Une fois toutes les abeilles éradiquées, la partie est gagnée.

Afin d'éviter la duplication de code nous avons utilisé la programmation orientée objet (et le langage Java) et avons exploité les classes de bases fournies par le sujet,

Nous avons plus ou moins suivis l'ordre des consignes du sujet pour l'élaboration du projet. Une fois le travail imposé presque terminé nous avons commencé à améliorer le jeux avec de nouvelles fonctionnalités tout en cherchant à résoudre les quelques bugs et problèmes restants.

Difficultés rencontrées

La difficulté majeure rencontrée est le fait que nous ayons dû nous adapter à un code déjà construit et très dense (notamment AntGame.java). Il a fallu, en amont, lire et comprendre chacune des classes du squelette fourni et nous adapter à celui-ci.

Pour l'implémentation de l'eau (Water) nous avons du nous adapter au système de moatfrequency mais aussi au code permettant le bon affichage graphique de l'eau grace à g2d. Après quelques tests nous avons réussi à mettre en place un code efficace et fonctionnel.

L'une des plus grandes difficultés était de recenser et d'appliquer toutes les possibilités de fourmi Containing ou non et de veiller à ce que la fourmi contenante soit affichée en premier mais aussi que la fourmi contenante effectue bien son action. De nombreuses heures ont été passées à débuger le programme vis-à-vis des BodyGuardAnt. Nous avons fait le choix d'implémenter une classe abstraite implémentant l'interface Containing en intermédiaire, afin d'avoir un codage plus propre de ce type de fourmi (il nous a été dit que cela était conseillé).

Nous avions intégralement codé la NinjaAnt avant de nous rendre compte que son comportement ne collait pas à celui demandé. En effet elle ne frappait qu'une abeille au hasard sur sa place au lieu de toute les toucher. Une modification et un allègement du code a permis de corriger cela.

Une exception (de type NullPointer) nous a également ennuyée juqu'à la fin du projet, moment auquel nous avons enfin trouvé la solution, au niveau de la FireAnt. Nous avions 2 codes : l'un reproduisait exactement le comportement demandé mais déclenchait une exception dès qu'il y avait plus d'une abeille sur la place. L'autre fonctionnait mais ne

reproduisait pas le comportement “kamikaze” de la fourmi. Finalement un simple test sur place dans la méthode isBlockingBee() de la classe core.Bee nous a permis de résoudre le problème.

La HungryAnt nous a pris également un peu de temps, nous avions un code assez brouillon et fourni au début, et qui, au final, s’avérait non fonctionnel. Après réflexion, nous avons trouvé une solution élégante et beaucoup plus courte (et donc moins coûteuse en mémoire et ligne de code).

Une autre classe sujette à beaucoup d’heures de réflexion et de Travail a été la QueenAnt. Pour galvaniser les abeilles nous avons d’abord utilisé un système utilisant la méthode getClosestAnt() dérivée de getClosestBee() avant de revoir radicalement notre façon de voir les choses. Après de multiples tests nous sommes parvenu à code épuré et parfaitement fonctionnel, ne faisant intervenir que des méthodes existantes. Nous avons du par la suite implémenter une nouvelle classe héritant de Place nommée QueenPlace, qui allait nous permettre d’ajouter une fonctionnalité au jeu : celle d’aboutir à un game over si une abeille atteint la QueenAnt (et pas seulement le fond de la fourmilière). Cela était conceptuellement difficile à comprendre et à implémenter mais nous avons fini par réussir. Le tout était corsé bien entendu par le fait que la reine soit protégée ou non par une BodyGuard ne changeait rien à l’issu de la partie. En revanche l’implémentation d’une interface implémentée par la QueenAnt (et 2 autres bonus) ayant pour effet d'empêcher de supprimer la QueenAnt depuis l’interface du jeu a été plus simple, nous avions modifié la mauvais méthode (renvoyant des Exception au niveau de leavePlace()), puis rapidement corrigé le problème. Nous nous sommes inspiré du patron de conception (Singleton), comme préconisé dans le sujet, pour gérer l’unicité de la QueenAnt. Cependant réussir à lui donner le comportement exigé dans le sujet n’a pas été une mince affaire. Cette classe est celle qui nous a probablement demandé le plus de temps, de travail et de réflexion.

Bon nombre d’exceptions ont été levées dans des classes lourdes (comme core.AntGame), à mesure de l’élaboration du code et la modification de ces classes, il était très difficile alors de trouver d’où venait la problème (réellement).

Au fur et à mesure de l’implémentation de nouvelles classes et amélioration dans le jeu, ce dernier devint de plus en plus fourni et de ce fait il est devenu difficile de s’y retrouver au seins du code. Voir le diagramme UML en Annexe 2 pour illustration.

Axes d'améliorations

Nous avons apporté de nombreuses améliorations à la version finale exigée:

-Les bonus conseillés : Quatre nouvelles fourmis Thrower (Stun, Slow, Short et Long). Nous avons dû apporter quelques modifications dans l'action des abeilles avec la Stun et la Slow.

-Un nouveau type d'abeille à l'armure double (Strongbees, vulnérable seulement pour certaines fourmis !). Pour effectuer les test des chaque fourmi il est déconseillé d'utiliser les niveaux autres que Easy, puisqu'il y a des StrongBee (la partie est plus corsée).

-Du son : Un background et des effets sonores en fonction de l'action réalisée. Nous avons pour cela dû implémenter un lecteur audio basique sachant lire des .wav. À noter que les fichiers audio doivent être placés dans /bin/Audio pour que cela fonctionne. Autrement les exceptions empêcheraient le jeu de fonctionner correctement (Eclipse nettoie régulièrement ce dossier donc bien veiller à sauvegarder les fichiers audio ailleurs pour ne pas les perdre).

-Un menu, à l'aide de Windows Builder qui différencie les niveaux de jeux et permet de choisir la "ruche" contre laquelle nous nous battons.

-Cinq nouvelles fourmis implémentant diverses fonctionnalités améliorant le gameplay. La NukeAnt est considérée comme une "super arme" (tue toutes les abeilles présentes dans la Colony), son coût en nourriture devrait être raisonnablement ajusté autour de 40, pour pouvoir être débloquée peu de fois et au bout d'un certain temps, mais il est placé à un petit chiffre dans le code pour pouvoir la tester. Les caractéristiques de la Larve sont les suivantes : coût en nourriture et armure ajustés à 1, la Larve ne fait rien pendant n tours (dans le code n=4 mais il faudrait l'ajuster à la difficulté du jeu), puis évolue en une fourmi au hasard parmi 10 (11% de chance d'obtenir une fourmi classique et 1% de chance d'obtenir une NukeAnt.). Les 3 autres fourmis sont des hybrides de celles existantes, entre la Wall la Harvester et la Scuba, implémentant diverse interfaces et autres fonctionnalités du jeu. Leur description est en commentaire dans leur code.

-Un nouveau type de Place (implémenté tardivement, non terminé) ayant des effets bénéfiques (HoneyPlace). Elle galvanise les abeilles dessus et double la nourriture amassée par la HoneyAnt. Cette place est désactivée par défaut (encore en cours de développement).

Pour ce faire nous avons consulté beaucoup de tutoriels (que vous trouverez en Annexe 3)

Le jeu pourrait encore être amélioré avec différentes nouvelles abeilles et fourmis, ainsi que de nouvelles places, de nouveaux effets afin de densifier et d'enrichir le gameplay du jeu.

Nous n'avons pas trouvé de bugs (exception) ou situation indésirable après nos batteries de test une fois le code final écrit.
Pour améliorer l'aspect visuel (graphismes) et l'ambiance nous pourrions inclure plus d'animations chez les protagonistes, utiliser des gifs animés des abeilles/fourmis et même des différentes Place et action des fourmis, puis réussir à les afficher en jeu.

Nous avons réussi à créer quelques gifs mais nous n'avons pas eu le temps de les implémenter (avec le code actuel seule la première trame du gif s'affiche, il n'y a pas d'animations).

Un système de point et de highscore avec sauvegarde des points serait une idée intéressante à exploiter.

Nous pourrions également approfondir l'idée du menu afin d'intégrer une interface plus complexe et plus ergonomique au jeu. Par exemple, nous pourrions ajouter une fenêtre lors du game over (ou du gain de la partie) qui permettrait de rejouer la partie avec un mode de difficulté différent, ainsi que d'enregistrer un score avec un pseudo si l'on associe un système de point au jeu (tel un jeu d'arcade). Ou encore d'intégrer un système de gestion de temps (clock) du jeu: un bouton Pause ou Accélérer par exemple. Cependant cela demanderait beaucoup plus de temps à développer.

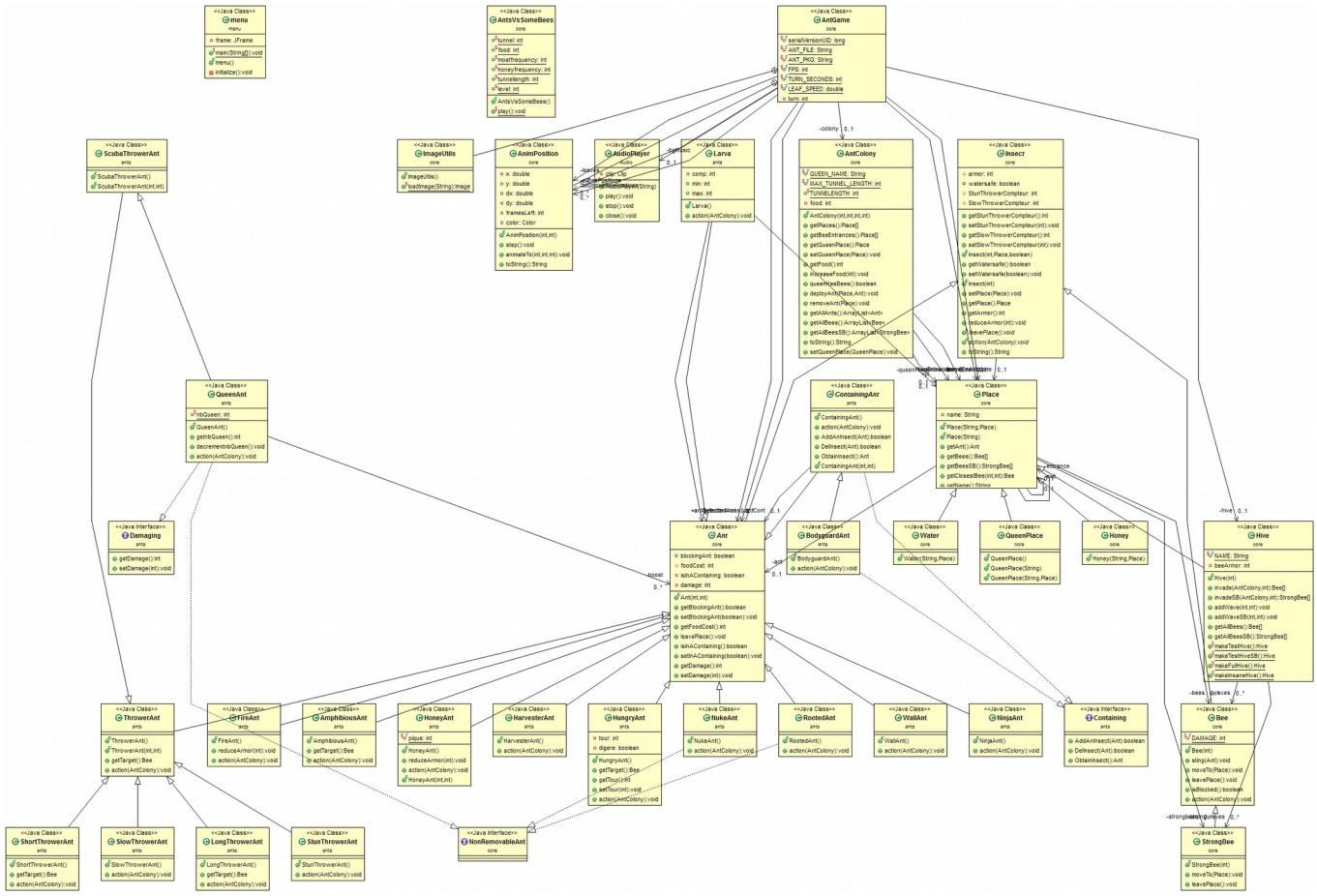
Pour pousser l'optimisation nous pourrions nettoyer profondément le code afin qu'il deviennent parfaitement lisible et intelligible, mais également optimiser les algorithmes afin d'utiliser le moins d'espace mémoire possible. En effet si l'on venait à développer le jeu pour une utilisation commerciale nous ajouterions beaucoup plus de classes et ressources (images, sons, etc) à charger en mémoire au lancement du jeu, ce point deviendrait donc absolument vital pour le jeu.

Conclusion

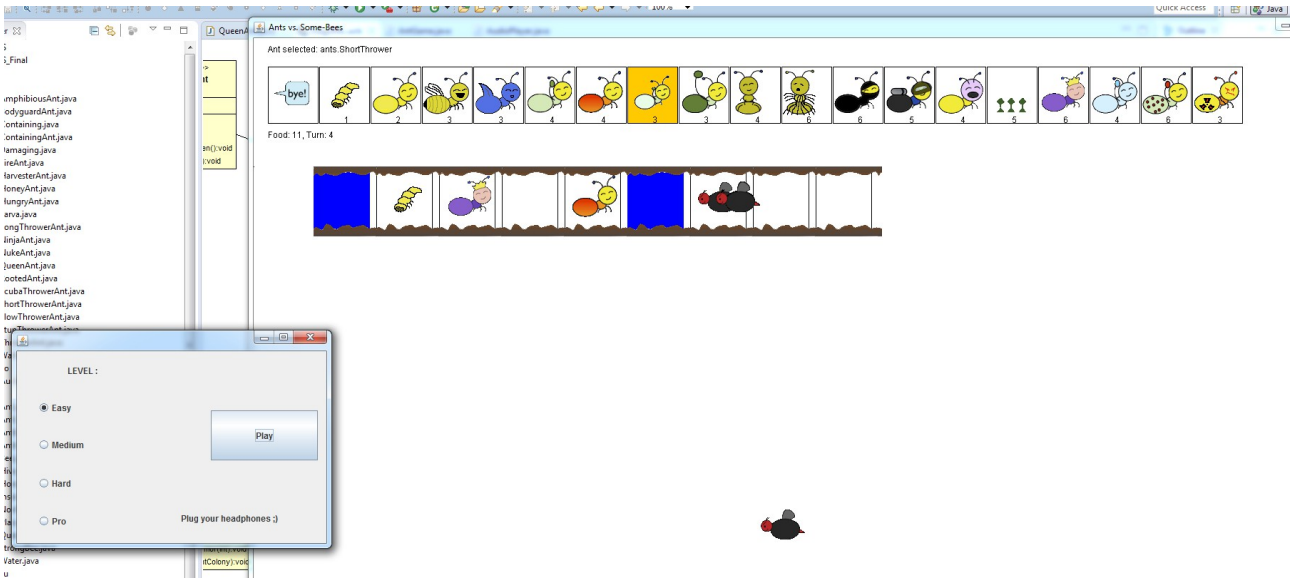
Ce projet nous a permis d'aborder beaucoup de points essentiels du langage Java et de la programmation orientée objet (notamment l'héritage, le polymorphisme). L'aspect ludique et interactif d'un jeu nous a permis de nous pencher sur différents aspects et différents outils. La composition imposée nous a permis de revoir les bases du cours de POO, tandis que les bonus nous ont permis de l'approfondir et d'aussi d'apprendre de nouvelles techniques. Nous n'avons pas eu de réel mal à résoudre nos problèmes, du moment que nous sommes très intéressés par la programmation orientée objet que nous trouvons particulièrement élégante. Nous avons enrichi le jeu le plus possible dans les limites de nos capacités et de notre imagination. Notre expérience de joueur nous l'a permis; ainsi nous avons pu améliorer le gameplay de façon adéquat. Le tableau qui suit représente notre répartition du travail en heure :

	Code imposé	*Code bonus*	*Recherches*	*Rapport*	*Total :*
GARCIA Guillaume	12	17	5	5	39
GHANEM Hakim	14	15	7	2	38

Annexe 1 : Diagramme UML des classes du jeu :



Annexe 2 : Images du Jeu :



Annexe 3 : Sources :

https://www.youtube.com/watch?v=ar0hTsb9sxM

https://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html

https://openclassrooms.com/courses/apprenez-a-programmer-en-java/l-heritage-1

http://www.vogella.com/tutorials/EclipseWindowBuilder/article.html

https://www.youtube.com/watch?v=Z06v0XeqJ0U

https://fr.wikipedia.org/wiki/Singleton_(patron_de_conception)

http://www.journaldunet.com/developpeur/pratique/developpement/12315/comment-generer-un-nombre-aleatoire-random-en-java-compris-entre-deux-chiffres.html