Rapport:
Projet Pacman

Thomas Bianchini - Nathanael Couret - Antoine Valette - Clement Taboulot

2 mars 2015

Table des matières

1 Organisation du projet

Étant un groupe de quatre collaborateurs, nous avons commencé par nous organiser afin que chacun connaisse son rôle dans le but de faire avancer le projet correctement.
Tout d'abord, la base de notre travail repose sur la possibilité pour chacun des membres de disposer des ressources nécessaires afin d'avancer ses tâches. Pour cela, nous avons mis en place un repository git (hébergé sur GitHub). Ce choix se justifie car la mise en place est simple et l'équipe avait les connaissances suffisantes d'utilisation.
Ensuite nous avons décidé de réfléchir tous ensemble sur le sujet afin d'ébaucher une architecture pour notre application qui sera expliquer en détail dans la partie structure de l'application. Puis nous nous sommes mis d'accord sur les tâches de chacun :

- Antoine : algorithme minimax
- Clement : algorithme du plus court chemin
- Nathanël : deplacement aleatoire, gestion des cartes pourries, gestion des victoires/défaites IHM
- Thomas : parser de fichier de map, mis en place de la structure, gestion IHM
- Clement - Thomas : deplacement des fantomes
- Groupe : code review, javadoc, rapport

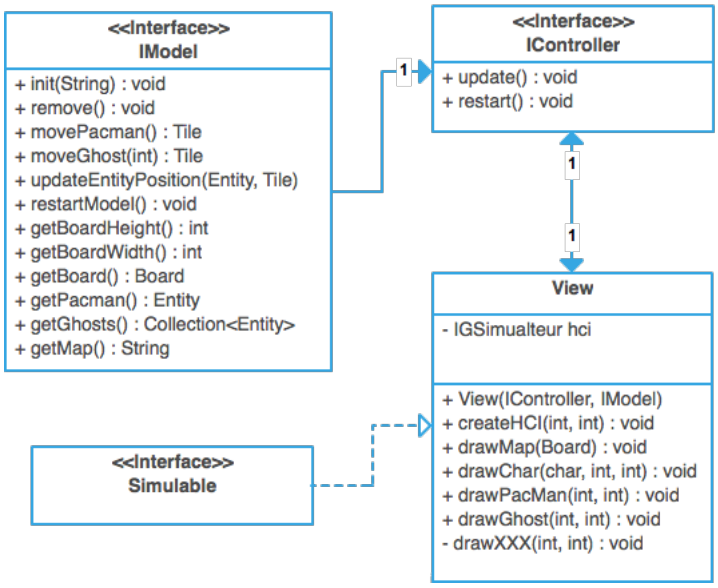
2 Structure de l'application

2.1 Le Modele - Vue - Controlleur

La première idée que nous avons eu a été d'introduire un desgin pattern afin de structurer l'architecture globale de l'application.
Le pattern MVC permet en effet de découper l'application en trois grandes parties :

- l'affichage de données
- la sauvegarde et manipulation de données
- les interactions de l'utilisateur

Nous avons donc adapté le MVC à notre application dont voici le diagramme de classe :



3 Les algorithmes

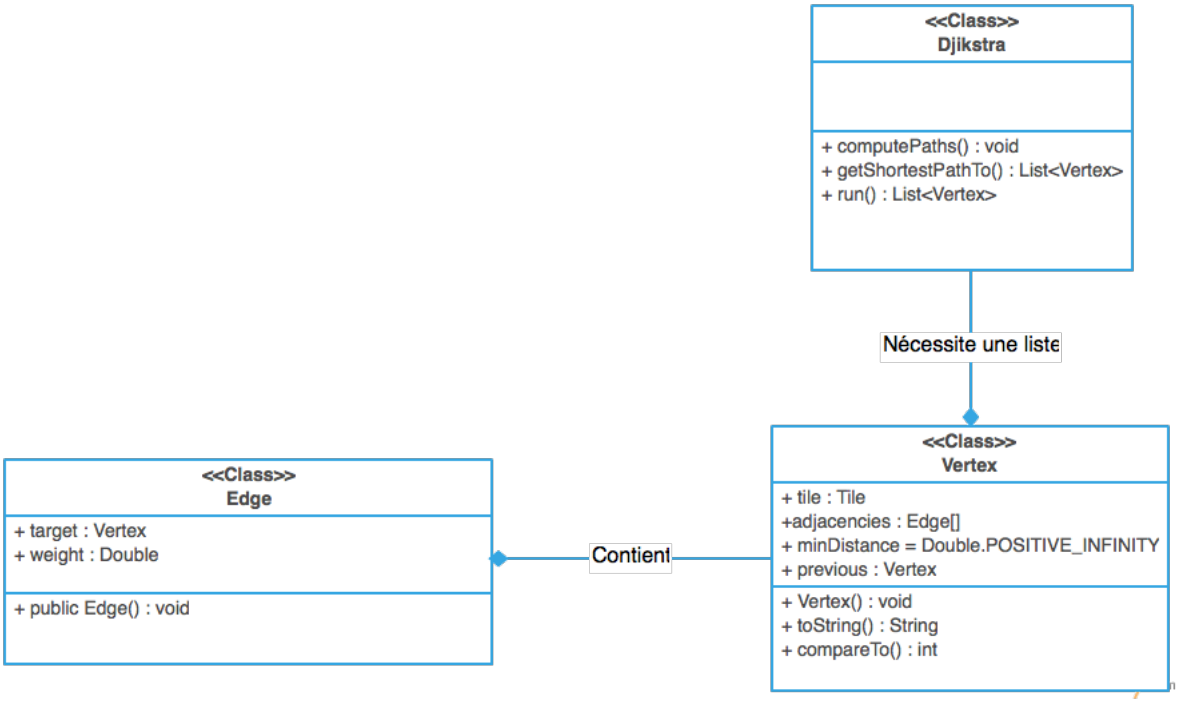
3.1 Déplacements aléatoires

3.2 Algorithme du plus court chemin

3.2.1 Choix de l'algorithme

Pour le calcul du plus court chemin nous avons fait le choix d'appliquer l'algorithme de Dijkstra. C'est un algorithme simple, efficace et que nous avons vu cours de Recherche Operationnelle. Pour l'implémentation nous avons créé un package Djikstra comportant les classes suivantes :

— Vertex : cette classe correspond au noeuds du graphe utilisé pour l'algorithme ;
— Edge : cette classe correspond au arc entre 2 noeuds adjacents ;
— Djikstra : cette classe permet de calculer le plus court chemin entre un source et un objectif.



Le calcul du plus court chemin se deroule en suivant les étapes qui suivent :

— Créer une liste repertoriant tous les noeuds de la carte, c'est-à-dire créer une liste de Vertex ;
— Pour chaque, noeuds déterminer qui sont ses voisins et le poids du passage du noeud A au noeud B (pas defaut 1). Cela correspond a un tableau de Edge ;
— Lancer l'algorithme de Dijkstra avec la methode run() en précisant le point de depart et le point point d'arrivé dans les paramètres.
— Le chemin retourné est une liste de Vertex correspondant au chemin à suivre.

3.2.2 Tests et problèmes rencontrés

Pour pouvoir valider cette fonction de l'application, il fallait qu'elle réponde à 3 critères :

— Si 2 chemins se présentent à Pacman, il doit prendre le chemin le plus court. Le test utilisant la carte Djikstra2.map valide cette fonctionnalité ;
— Si 2 chemins se présentent à Pacman, et que le chemin le plus court contient un obstacle, alors Pacman emprunte un autre chemins. Cette fonctionnalité est validée avec le test utilisant la carte Djikstra3.map ;

— Si la carte contient plusieurs SUPER PAC GUM, alors Pacman mange toutes les SUPER PAC GUM en se dirigeant toujours vers celle qui se trouve la plus proche de lui. Cette fonctionnalité est validé par le test contenant la carte Djikstra1.map ;

Durant l'implémentation de cette fonctionnalité un problème essentiel ont été rencontré.
Il s'agit de la construction du graphe qui permet à l'algortihme de Djikstra de fonctionner. En effet, il devait représenter fidèlement la carte. Plusieurs itérations ont été nécessaire pour obtenir une fonction opérationnelle. Par la suite, pour améliorer cette fonctionnalité nous avons pris en compte qu'il n'était pas nécessaire de recalculer à chaque déplacement un nouveau graphe et un nouveau plus court chemin. En effet, un nouveau chemin n'est calculé que si Pacman vient de manger une super pacgum.
Lorsque Pacman a mangé toutes les super pacgum, son objectif devient alors les simples pacgum. Et à partir du moment où la carte ne contient plus de pacgum alors Pacman n'a plus de réel objectif. Sa statégie passe donc en aléatoire.

3.2.3 Améliorations possibles

Cette fonctionnalité n'est pas parfaite et plusieurs amélioration serait possible. Premièrement si Pacman et les Ghosts ont pour statégie le plus court chemin alors le résultat obtenu n'est pas celui souhaité. De plus, il est possible que Pacman se fasse manger par un Ghost. Quand un chemin est calculé Pacman va suivre se chemin, et l'évaluation de la viabilité de la case suivante n'est pas totalement fonctionnelle. Dans certains cas, la case suivante vers laquelle il va se diriger est aussi une case adjacente à un ghost. Et il arrive que Pacman et un Ghost se retrouvent au même endroit. Une méthode badNextTile() est présente pour évaluer cette viabilité mais son fonctionnement n'est pas opérationnel.
Le calcul de plus court chemin entre un point A et un point B fonctionne. Mais un ajout d'intelligence pour faire fasse à certaines situation pourrait être un axe d'amélioration.

3.3 Minimax