Projet de POO

Simulation de robots pompiers

15 novembre 2016

Sommaire

Maxime Deloche & Ludovic Carré & Cyril Carlin

1 Description des packages intéressants

1.1 Chemin

Le package chemin a pour rôle de définir les différentes classes nécessaires à la partie 3 du projet. On a choisi de créer une classe abstraite PlusCourtChemin qui permet une abstraction sur les stratégies de calculs de plus court chemin employées. Nous avons pour l'instant implémenté un seul algorithme, Dijkstra, mais il serait facile de rajouter une classe pour un autre algorithme comme A* sans avoir à modifier le reste du code.

Nous avons hésité entre Dijkstra et A* et finalement choisi Dijkstra car A* ne semblait pas vraiment nécessaire au problème traité. Les cartes sont relativement petites ce qui implique un nombre de noeud assez restreint, 2500 pour la plus grande carte. De plus, il serait difficile de définir une bonne heuristique puisqu'il n'y a pas d'information sur la structure "normale" d'une carte.

1.1.1 Dijkstra avec un hash set

On notera $N = n * m$, où n et m sont les dimensions de la carte, le nombre de noeuds et les complexités sont calculées en pire cas (aucun noeud inaccessible dans le graphe).

— setEnsembleNoeud : Initialise le HashSet avec les noeuds sur lesquels itérer. Coût : $\theta(N)$.

— initDistance : Initialise le tableau de distance à $+\infty$. Coût : $\theta(N)$.

— getMin : Retourne le noeud non exploré le plus proche de la source. Coût : $\sum\limits_{i=0}^{N} N - i = \frac{N(N-1)}{2} = \theta(N^2)$.

— setDistanceVoisins : Met à jour les distances du tableau avec les distances des voisins. Coût : $4N - 4 = \theta(N)$.

Complexité générale de l'algorithme : $\theta(N^2)$.

La Complexité assez élevée et l'on a donc décidé d'améliorer l'algorithme avec une pile de Fibonacci ce qui permet d'atteindre la complexitée optimale en $\theta(4N - 4 + Nlog_2(N))$. Malheureusement, nous avons visés un peut trop haut avec une implémentation générique de la pile de Fibonacci ce qui a beaucoup compliqué le code et l'échéance approchant on a décidé de se reporter sur une solution plus simple mais qui reste très respectable : la file de priorité.

1.1.2 Dijkstra avec une priority queue

Il est possible d'améliorer l'algorithme en utilisant une priority queue comme structure de données. Pour cela il faut rajouter une abstraction Noeud qui contient un poids et une case, la liste est constamment ordonnée tel que l'élément de poids minimale est le premier élément de la liste, ce qui améliore la complexité de l'algorithme car la priority queue est implémentée avec un tas binaire en Java.
De plus, afin d'encore améliorer la rapidité de l'algorithme, nous traitons un ensemble de noeud ne contenant que la source au départ qui se remplit au fur et à mesure des détections de voisins ce qui permet de diminuer les coefficients des différentes étapes de l'algorithme. On note N_i le nombre de noeud à l'iteration i de l'algorithme $= i * 4 - i$.

— stockCarte : La nouvelle version de l'algorithme créé un tableau de noeud constitué des cases de la carte lors de la création de l'objet. Coût : $\theta(N)$.

— setEnsembleNoeud : Il faut initialiser tous les noeuds avec un poids à $+\inf$ et un prédecesseur à null. Coût : $\theta(N)$.
— getMin : La fonction à disparue il s'agit simplement de l'extraction de la tête de la file de priorité. Trouver le min de la file est en $\theta(1)$ puisqu'il s'agit de la tête, cependant il faut déterminer la nouvelle tête de file après l'extraction. Dans un tas binaire, l'opération se fait par swapping jusqu'a obtenir le min en racine. Coût : $\theta(log_2(N)$.

Complexité générale de l'algorithme : $\theta(Nlog_2(N))$.
TODO du coup faut que je me motive pour modifier notre algo.

1.1.3 Reconstruction du chemin avec le tableau de prédecesseur

On définit un chemin comme un hash set de Destination, à chaque objet est associé une position et un temps de trajet pour s'y rendre en partant de la source, et un temps qui est la somme des temps des sous-trajets.
Puisqu'il s'agit d'un hash set, il n'y a pas de notion d'ordre dans un chemin ce qui peut sembler particulier mais puisque les événements sont ajoutés avec un temps d'exécution, l'ordre n'est pas nécessaire.

Algorithme de reconstruction de Dijkstra : L'algorithme consiste simplement à reconstruire le chemin à partir du tableau de prédécesseurs créé lors des mises à jours de distance.
Coût : $\theta(M)$ où M est la longueur du chemin en nombre de case donc $\theta(N-1)$ en pire cas.

2

2.1

2.2

3

3.1

3.2