Rapport de POO : Simulateur de robots pompiers

Cyril Dutrieux, Mathias Biehler, Mahmoud Bentriou

20 novembre 2014

Table des matières

1 Présentation du programme

Le but de ce TP libre est de programmer un simulateur de robots pompiers. Ces derniers doivent éteindre tous les incendies d'une carte, affichée par une interface graphique qui nous est fourni. Le simulateur doit afficher les actions des robots dans le temps, et les décisions à propos des actions des robots doivent être prises en temps réel. Tout cela doit être développé selon les concepts de la programmation orientée objet.

Pour executer le programmer on utilise un script bash qui laisse le choix du manager à utiliser ainsi que la carte :
./simulation.sh [numéro manager] [MAP]
où [numéro manager] est le numéro du manager à utiliser (2 ou 3) et [MAP] est le nom d'un fichier carte situé dans le dossier cartes.

2 Choix d'implémentations

2.1 Paquetages

La source est décomposée en plusieurs paquetages :
- simulation : regroupe les fichiers propre à la simulation : données, événements élémentaires et l'affichage
- managerPack : regroupe les différents types de Manager
- elements : regroupe les éléments de la carte : robots, incendie ainsi que algorithme de plus court chemin
- environnement : Regroupe dans quel environnement les éléments évoluent (Carte etc..)
- IHM : Paquetage d'affichage graphique fournie en bytecode (dans le fichier bin)

2.2 Collections

Au niveau du choix des collections, nous avons choisi d'utiliser TreeSet<> pour stocker les événements du simulateur à executer : cette collection allie à la fois facilité d'implémentation dans notre cas (quand on ajoute un événement, il est déjà trié dans la liste) ainsi qu'un cout algorithmique acceptable (en $O(log(n))$).

L'algorithme de plus court chemin A* a besoin d'une file de priorité, nous avons donc utilisé la collection PriorityQueue<>.

2.3 Classes et héritage

On a voulu respecter au mieux les différentes dépendances entre classes selon le graphe UML du sujet. La classe DonneesSimulation possède ses éléments : on crée des instances d'objets quand on lit le fichier de la carte, puis on instancie à nouveau les objets par constructeur de copie pour les données de la classe DonneesSimulation, pour que l'agrégation forte (composition) demandée par le sujet soit respectée. Par contre, les événements stockés dans la classe Simulateur ne sont pas, contrairement à ce qui est demandé par le graphe UML, (ré)instanciés par cette dernière : ils sont crées par la classe Robot. Dans le sujet, on peut considérer par exemple que le robot "peut de lui-même aller se remplir au point d'eau (ou la berge) le plus proche" (partie 4.2). Ainsi, nous n'avons pas jugé utile que tout événement soit obligatoirement instancié dans la classe Simulateur (un événement n'a d'existence ici que pour être stocké dans la liste d'événements du simulateur).

3 Attentes vis à vis du produit livré

Le programme s'execute correctement. Il affiche les déplacements des robots, les incendies et les incendies éteints, et indique par le biais d'un point d'interrogation quand le robot est libre. Le Manager2 est le manager naif et Manager3 est un manager moins naif qui envoie chaque robot sur l'incendie le plus proche. Chaque carte donnée par le sujet est bien lue et les robots se comportent de manière cohérente.