Rapport de TP POO : Simulation d'une équipe de robots pompiers

KEDA Pofinet (IF1)
MANSOUR Augustin (IF1)
EL IDRISSI BOUTAHER Mehdi (IF1)

15 novembre 2016

1 Principe de notre programme

L'objectif de ce TP était de développer en Java une application permettant de simuler une équipe de robots pompiers pouvant évoluer de manière autonome dans un environnement naturel. Nous avons veillé à faire différents packages pour rendre notre code plus lisible. Les packages sont les suivants :

- Le package **chemins** contient toutes les classes permettant de créer le plus court chemin.
- Le package **commandant** contient le chef pompier.
- Le package **elements** contient les éléments de base de la carte, et la carte elle même.
- Le package **io** contient le lecteur et créateur de données.
- Le package **robots** contient les différents types de robot.
- Le package **simulation** contient le simulateur.
- Le package **tests** contient tous nos tests intermédiaires.

Nous allons présenté différentes classes que nous pensons être importantes :

- la classe **Robot**, scindée elle-même en deux classes abstraites robots : **Robotsaeriens** et **Robotsterrestres**. La raison de cette distinction est due au fait que les robots aériens ne remplissent pas leur réservoir de la même manière que les robots terrestres.
- La classe **Simulation** : elle récolte les données et le simulateur gère la liste des événements.

2 Algorithme de Dijkstra, recherche de plus court chemin

Nos robots pompiers devant respecter certaines contraintes en fonction de leur type et donc, ne pouvant acceder à n'importe quelle case, il fallait implémenter un algorithme calculant le plus court chemin le plus court permettant à un type de robot de se déplacer vers un incendie ou vers un point d'eau tout en prenant en compte les contraintes liées au terrain.
Nous avons alors créé une classe **Trouverchemin** qui créer un graphe en fonction du robot et se base sur l'algorithme de Dijkstra pour fournir un plus court chemin. Pour se faire, nous avons créé une méthode constructionDuGraphe qui prend en compte les cases de la carte suivant les lignes et les colonnes et qui crée le graphe pondéré sur lequel on va utiliser l'algorithme de Dijkstra grâce à la méthode **calculPlusCourtChemin()**. On a ensuite représenté le plus court chemin renvoyé par cette dernière fonction sous la forme d'une liste chainée et une méthode **seRendreSurUneCase()** permettant à notre robot de se rendre sur les différentes cases pointées par notre liste.

3 Problèmes rencontrés

Nous avons rencontré des problèmes pour l'affichage du déplacement des robots case par case. En effet, le robot se téléportait directement de la case départ à la case destination sans nous dessiner le chemin qu'il empruntait. Nous avons décidé, au départ, d'utiliser des threads pour permettre le déplacement d'une case à l'autre. Graphiquement, nous ne pouvions pas obtenir quelque chose de satisfaisant. Nous avons finalement préféré utiliser une autre manière en nous basant sur le fait que les événements ont une date à laquelle ils doivent être éxécutés.
Dans la partie 4 du projet, nous avons encore un soucis quant à la gestion des robots par le robot chef. En effet, certains robots qui ne sont plus occupés ne vont pas éteindre d'incendie que nous n'avions pas pu débugger par manque de temps.

4 Conclusion

En somme, ce projet nous a permis de revoir les différentes notions de la programmation orientée objet en Java, à savoir :

- l'héritage à travers les différentes classes de robots.
- le polymorphisme.
- les collections des objets.