

# Simulation d'une équipe de robots pompiers

## TP de Programmation Orientée Objet

*Matthias Bouderbala, Fieschi Philémon et Alexis Bigé*

## 1 Introduction

Ce compte-rendu permet de résumer le contenu du code du projet. Le code a été écrit par les trois membres du groupe. Le code s'organise de la manière suivante dans le dossier `src` :

- le package `data` : les classes de données (carte, robots, incendies...)
- le package `gui2` : le simulateur et son scénario, liés à `gui.jar`
- le package `events` : les classes d'évènements
- le package `chemin` : la classe qui définit un chemin
- le package `io` : la lecture et creation de données
- les tests de base

## 2 Choix de conception

### 2.1 Le choix des classes de données

Bien guidés par le sujet, nous avons implémenté les différentes classes de données : `Carte` `Case` `DonneesSimulation` `Incendie`. Les méthodes ont été ajoutées au fur et à mesure selon les besoins pour les différentes étapes du projet.

### 2.2 Le choix des méthodes des robots

Le package `robot` contient tous les types de robots, qui référence une classe principale `Robot`. Cette dernière contient les méthodes de base pour les robots ainsi que celles, essentielles, pour les différentes actions des robots. Nous avons choisi de laisser le robot gérer ses déplacements, quand il reçoit un ordre : `deplacementCase` `ordreRemplissage` `ordreIntervention`, il ajoute ensuite lui-même les évènements au simulateur en prenant en compte le temps d'action.

Parmi ces méthodes, on pourra remarquer :

- **Dijkstra** : renvoie le plus court chemin selon l’algorithme de Dijkstra, implémenté à partir de la version décrite sur la page [https://fr.wikipedia.org/wiki/Algorithme\\_de\\_Dijkstra](https://fr.wikipedia.org/wiki/Algorithme_de_Dijkstra). Le choix de cet algorithme est réfléchi (Dijkstra est plutôt efficace) mais aussi poussé par notre connaissance de cette algorithme.
- **ordreIntervention** : gère le déplacement (si nécessaire) et l’intervention du robot sur un incendie dans le simulateur
- **intervenir** : gère l’intervention directe du robot sur un incendie (le robot a auparavant géré son déplacement)
- **ordreRemplissage** : gère le déplacement (si nécessaire) et le remplissage du robot dans le simulateur. Il utilise la méthode **choisirCaseEau** qui permet d’obtenir la case où se déroulera le remplissage.

Toutes ces actions sont ainsi ajoutées au simulateur selon le temps que le robot prend, temps calculé selon la nature du robot, sa vitesse, la nature du terrain et en partant du principe que le robot parcourt la moitié de la case où il est et la moitié de la case où il va.

## 2.3 Le choix de la classe **Chemin**

Afin de faciliter la recherche d’un plus court chemin, nous avons implémenté une classe **Chemin** qui contient deux **List<>** :

- **List<Case>** : une liste de cases (ordonnées par construction selon la date du déplacement vers chaque case)
- **List<Long>** : une liste de dates (ordonnées par construction)

Avec des méthodes classiques pour ajouter ou récupérer des éléments dans **Chemin**, nous avons pu utiliser ce type de donnée pour implémenter la recherche d’un plus court chemin dans **Robot**.

## 2.4 Le choix du simulateur et de l’implémentation des événements

Notre classe **Simulateur** contient un attribut de type **Scenario** qui permet gérer la séquence d’évènements qui s’exécutent au cours du temps. Ce **Scenario** contient simplement une **ArrayList<Evenement>** qui permet de garder les événements ordonnés par date et ainsi d’en ajouter continuellement au milieu ou à la suite. Hérités de la classe abstraite **Evenement**, les classes **DeplacementUnitaire** **EvenementMessage** **Intervention** **Remplissage** permettent d’effectuer les actions indiquées au robot. En effet, c’est le robot lui-même qui a ajouté ces événements au simulateur, après avoir reçu des ordres.

## **3 Tests et résultats obtenus**

### **3.1 Premiers tests : la lecture des données et les classes de données**

Nous avons choisi de tester nos classes et méthodes au fur et à mesure. En premier lieu, il fallait s'assurer du bon fonctionnement des méthodes des classes de données, dont la principale : `DonneesSimulation`. Le fichier `TestCreationDonnees.java` contient ainsi nos premiers tests sur ces classes de données. Nous n'avons pas eu de soucis particulier à ce niveau.

### **3.2 L'affichage graphique**

### **3.3 Les évènements**

### **3.4 L'organisation des évènements par les robots**

### **3.5 La stratégie**

## **4 Conclusion**