

Rapport de Simulation d'une équipe de robots pompiers

Groupe X

11 novembre 2020

1 Données du problème

1.1 Carte

Une carte est représentée par une table de hachage non-modifiable dont les clés, représentées par des entiers et correspondant aux cases, sont associées à la nature du terrain correspondant (type énuméré). Plus précisément, une case (i, j) est identifié par l'entier $i * largeur + j$ où *largeur* représente la largeur de la carte. La classe définissant une carte est disponible dans le fichier *Carte.java* du package *Game*.

Notons que les complexités des opérations d'une table de hachage sont identiques à celles d'une matrice 2D dans le cadre de notre problème. Par ailleurs, précisons qu'on s'affranchit de tous espaces mémoires inutiles dans le cas où dans un futur proche, nous souhaiterions gérer le cas de cartes non-rectangulaires : le problème serait plus simple à gérer, à raison de modifier le système de coordonnées.

1.2 Incendies

Les incendies sont caractéristiques des données d'une simulation. Ceux-ci sont représentés par une table de hachage, attribut de la classe *DonneesSimulation*, dont les clés sont des entiers (position de l'incendie) associées à des entiers (intensité de l'incendie). Une des forces de ce stockage est de pouvoir itérer simplement sur tous les incendies, sans avoir besoin de parcourir toute la carte. Ainsi, on a une connaissance à tout instant des incendies courants.

1.3 Robots

Pour représenter les robots, nous proposons une représentation basé sur un *pattern design type-object*. Cette représentation propose des avantages multiples, notamment elle permet de modifier ou d'ajouter de nouveaux types sans avoir à recompiler ou à changer du code (contrairement à l'héritage qui nécessite de créer de nouvelles classes). Néanmoins, ce design-pattern n'inclut pas l'ajout de nouvelles fonctionnalités propres à un robot, mais dans le cadre de notre problème, ce choix est intéressant.

Un robot est caractérisé par son type, objet instancié dans une classe non-modifiable *MyRobotTypes* contenant la déclaration de tous les types de robots nécessaires à la simulation. La classe *RobotTypes* contient le constructeur et méthodes nécessaires pour l'instanciation des robots dans *MyRobotTypes*. Pour plus de simplicité sur la gestion des événements, un robot est identifié par un nombre entier et par une position initiale sur une case de la carte. Toutes les classes définies pour les robots se situent dans le package *Game.Robots*.

1.4 Interface Graphique

Nous disposons d'une classe *TileImg* implémentant *GraphicalElement* pour représenter les images. Une image est caractérisée par un fond, sur lequel on superpose plusieurs motifs qui peuvent être les robots et/ou les incendies. L'interface graphique en lui-même est défini par une classe *GraphicsComponent* contenant un buffer des images pour ne pas avoir besoin de les recharger à chaque affichage, les données de simulation du problème et des méthodes définies pour normaliser, placer les robots et incendies. Toutes les classes définies pour l'interface graphique se situent dans le package *Game.Graphics*.

2 Simulation de scénarios

2.1 Données des simulations

Une simulation est caractérisée par une carte, une table de hachage pour les incendies (comme expliqué en 1.2) et une table de hachage pour les robots dont les clés sont des entiers (positions) associées à une liste chaînée des identifiants des robots (plusieurs robots peuvent se trouver sur une même position). Notamment, une simulation représentée par une classe *DonneesSimulation* possède deux méthodes déterminant le temps nécessaire pour se déplacer d’une case à une autre et le temps nécessaire pour un robot de déverser l’eau sur l’incendie.

2.2 Simulateur

Un simulateur est une entité exécutant une simulation donnée. Notamment, un simulateur, à partir de données de simulation, ordonnance les événements des entités sur la carte. Ce simulateur est ajustable dans le choix où une stratégie est donnée. La méthode *next()* exécute le prochain événement, soumis ou non à une stratégie donnée. Quant à la méthode *restart()*, le simulateur sauvegarde toujours une copie des données de simulation.

Une action est une entité agissant sur les données d’une simulation. Représentée par une classe abstraite, une action doit nécessairement, pouvoir être exécutée et avoir une durée. Un robot peut effectuer plusieurs actions, chacune représentées par des classes filles :

1. ActionEmpty : déverser de l’eau sur un incendie.
2. ActionFill : se remplir si le robot est sur de l’eau ou à côté.
3. ActionMove : se déplacer vers une case voisine si c’est possible.

Un événement est décrit par une date et une action. Tous les événements sont gérés par une entité *EventManager* les ordonnant selon leur date respective. Plus précisément, nous disposons d’une file de priorité basée sur un comparateur des dates des événements.

3 Calculs de plus courts chemins

4 Résolution du problème

5 Expérimentations

5.1 Tests

5.2 Résultats

6 Conclusion