Simulation d'une équipe de robots pompiers

Equipe 51: Imad Ez-zejjari, Haytham Ait Bahessou, Khalil Leachouri

November 2020

1 Introduction

L'objectif de ce TP est de développer en Java une application permettant de simuler une équipe de robots pompiers évoluant de manière autonome dans un environnement naturel. Au fur et à mesure de notre travail, il fallait respecter plusieurs contraintes liées aux robots: vitesse de déplacement, nature des terrains et capacités d'extinction ...etc. afin de programmer des robots qui peuvent intervenir de façon autonome pour atteindre les incendies d'une façon optimisée en calculant le plus court chemin, on a dû passer par quatre étapes principales:

1. Implémenter les quatre classes principales propres à l'application: Case, Carte, Incendie et Robot puis programmer un test qui charge un fichier de données et affiche la carte correspondante, les robots et les incendies à l'aide de l'interface graphique fournie;

2. Créer un gestionnaire d'évènements et l'ajouter au simulateur tout en implémentant quelques tests (contenant une méthode main) permettant d'exécuter quelques scénarios;

3. Mettre en place les classes permettant de calculer le plus court chemin d'un robot vers une destination, et le traduire en évènements de déplacement, qu'on l'ajoute au simulateur.

4. Créez un ensemble de classes permettant chacune de représenter un chef pompier de stratégies différentes (élémentaire, évoluée...).

2 Les données du problème

Dans cette partie, nous avons implémenté les classes principales avec leurs méthodes nécessaires: *Case* repérée par sa ligne, sa colonne et sa nature de terrain. *Carte* qui contient une sorte de matrice des cases et les méthodes nécessaires pour accéder à la case voisine. *Incendie* défini par la case sur laquelle il se situe et le nombre de litres d'eau nécessaires pour l'éteindre. Enfin la classe *Robot* est définie comme abstraite. à l'aide de l'héritage, on définit les sous classes: drône, robot à roues, à chenilles et à pattes. La drone lève une exception *InvalidVitesseException* si la vitesse entrée n'est pas juste.

la figure suivante illustre bien le résultat obtenu:

3 Simulation de scénarios

Dans cette partie, on a implémenté une classe abstraite Évènement qui est responsable aux mouvements des robots: déplacement, remplissage d'eau et son versement.

1. *EvenDeplacementRobot* comme son nom l'indique, translate un robot vers une case tout en levant une exception *CaseInexistanteException* dans le cas où la case est erronée.

2. *EvenRemplirReservoir* comme son nom l'indique, remplit le réservoir en respectant bien les conditions de remplissage de chaque robot, et lève une exception *RemplissageRobotException* sinon. Et une autre *RemplissageRobotPatteException* si on essaye de remplir le robot à patte.

3. *EvenVerserleau* et *EvenVerserleauFin* permettent respectivement de diminuer l'intensité de l'incendie, et de l'éteindre quand l'intensité devient nulle.

4. *EvenRobotArrive* se charge de vérifier l'arrivée du robot et lève une exception *RobotNonArriveException* sinon.

Chacun de ces évènements a un temps d'exécution, que sert une base pour le simulateur afin d'intervenir pour organiser ces dates. En effet, on lui a donné en paramètre une **SortedMap<Long, ArrayList<Evenement>>** un dictionnaire ayant pour clés les dates des évènements triées par ordre croissant, et pour valeurs les liste des évènement exécutés dans une date donnée. Le choix de cette structure modélise bien les événements et permet d'exécuter ceux qui ont une date précise, ainsi il est plus simple de parcourir ses clés étant déjà triées.

4 Calculs de plus courts chemins

Afin de calculer les meilleurs itinéraires permettant à un robot de se rendre sur une case pour éteindre les incendies, on a utilisé l'algorithme de **Djikstra** qui calcule le plus court chemin entre deux cases données. On a essayé d'adapter cet algorithme à notre situation: les vitesses des robot différents sur chaque type de terrain ainsi qu'il y' a des robots qui ne peuvent pas se déplacer sur certains terrains. Pour atteindre cet objectif, on a crée trois classe essentielles: **Sommet**, **Arete** et **Graphe** qui correspondent respectivement aux classes des sommets, caractérisées par les positions des cases, d'arêtes, qui pendent en paramètre deux cases, et du graphe constitué en utilisant les sommets et les arêtes crées. Ainsi,la classe Graphe contient la méthode *shortestDistancesAndPath* qui applique l'algorithme de **Djikstra** et renvoie le plus court chemin entre le sommet donné en paramètre et tout le autres sommets du graphe et renvoie une **Map<Sommet, ArrayList<Sommet>>** où il donne à chaque sommet son plus court chemin avec le sommet en paramètre qui est composé d'une liste des sommets.

5 Résolution du problème

Pour éteindre effectivement les incendies, et exécuter les tâches nécessaires des robots, le sujet propose de coder une stratégie élémentaire au début, et l'améliorer dans une stratégie un peu plus évoluée:

5.1 Stratégie élémentaire

Dans un premier temps, les robots se déplacent vers les incendies, et versent l'eau contenue dans leur réservoir. La méthode suivie parcours tous les incendies, et les robots en essayant de les affecter un par un, mais ne se contente pas de remplir le réservoir du robot une deuxième fois. La principale méthode utilisé pour cette tâche est *eteinte*.
On a pris le choix d'ajouter un booléen: *etat* au robot, pour indiquer s'il est occupé à se déplacer, à éteindre un incendie. Dans le but de gérer les déplacements des robots, et éviter de changer sa case à chaque étape, notre équipe a ajouté au robot une position fictive, ayant comme fonction de suivre le déplacement du robot.

5.2 Stratégie évoluée Beta

Dans un deuxième temps, on a traité toutes les incendies de la carte en entier, en adoptant une méthode plus au moins optimal. En effet, au début on donne à tous les robots un incendie quelconque sans prendre en considération le temps effectué pour éteindre cet incendie. Tout en prenant bien soin de remplir le réservoir à chaque fois il est vidé. Puis, on affecte au robot le plus rapide -celui qui a terminé l'extinction de l'incendie- une autre. Et ainsi de suite, on répète

la démarche avec cet algorithme de recherche de minimum codé dans le *main* de **TestStrategieEvolueeBeta**.
On fait bien attention de remplir le réservoir du robot, à chaque fois qu'on lui propose un incendie à l'aide d'un booléen *remplir*. La méthode phare utilisée pour ces taches est *eteinte* de **StrategieEvolueeBeta** se chargeant de créer les évènements nécessaires pour se déplacer vers l'incendie, verser l'eau, et programmer des aller-retour pour remplir le réservoir et verser l'eau sur l'incendie. On optimise ici, en cherchant la *caseplusproche* pour chercher l'eau, par le biais de l'algorithme du plus court chemin.
Pour mettre en oeuvre cette stratégie, on a utilisé la même idée que dans la stratégie élémentaire, pour distribuer les incendies dès qu'un robot a fini son dernier travail.
On précise ici qu'on avait besoin d'ajouter un attribut pour les robots, appelée *positionFicitive*, pour pouvoir créer les évènements nécessaires pour le déplacement, le versement d'eau... tout en gardant le robot fixe (parce qu'on n'a pas encore commencé la simulation).

5.3 Stratégie évoluée

Étant donnée qu'on a préparé une version Beta (comme l'indique du coup le nom du premier fichier test de la stratégie évoluée), il ne restait que des simples retouches, précisément pour prendre en considération la distance des robots aux incendies, ce qu'on a fait en créant une **HashMap** *closest* pour chaque incendie, qui associe à chaque robot, la date à laquelle il arrivera à l'incendie, compte tenu de sa date de libération, qui se met à jour dans une autre **HashMap** *mapdaterobot* chaque fois qu'un incendie est attribué à un robot. Le reste concernant le calcul du chemin optimal, création des événements de déplacement, de versement d'eau et de remplissage du réservoir (si nécessaire) est le même que la version Beta.

6 Conclusion

Pour conclure, les choix détaillés précédemment nous ont permis d'arriver jusqu'au bout du projet, et d'une manière optimisée réaliser son but principal d'éteindre les incendies par un ensemble de pompiers. Ce sujet de par son importance sera sans doute utile pour plein de projets dans la vie réelle. Sans oublier que ce projet nous a permis de consolider nos compétences en java, et de maîtriser les notions phares et détaillées de ce langage.