Ecole Nationale Supérieure d'Informatique et Mathématiques Appliquées

ensimag 4mpoo

Projet POO : Robots-Pompiers

Numéro d'équipe : 31

Contents

1 Choix de Conception et Classes utilisées

Le projet est constitué de quatre parties focntionnelles, la première représente les objets matériels (Carte,Cases, Robots, Feu), la deuxième est la partie visuelle qui affiche une image de la carte et ses constituants à des instants donnés, la troisiéme est le gestionnaire d'évènements qui execute ceux-ci selon leur date d'execution et leur durée d'execution. Enfin il y a la partie stratégie qui gère le choix des évènements de manière optimisée.

1.1 Carte

La carte est présentée sous forme d'un objet. Elle est constituée d'une ArrayList de cases de même taille organisées en lignes et colonnes.

La classe *Carte* contient plusieurs méthodes permettant de connaître l'environnement des cases et de vérifier si certains déplacements de robots sont possibles.

1.2 Case

Une case est une sudivision d'une carte permettant de réduire l'espace de manoeuvre du programme et ainsi de le simplifier. Elle permet aussi de prendre en compte les caractéristiques des zones de la carte et de faciliter le calcul du plus court chemin.

Une *Case* a un numéro de *ligne*, un numéro de *colonne* et une *nature* appartenant à un type énuméré *NatureTerrain*.

1.3 Robot

L'entité Robot est une entité abstraite qui permet la création de robots de plusieurs sous-types (Robot à roues, Robot à chenilles...) qui interviennent dans la carte de manières différentes selon leur type afin d'éteindre le feu.

1.4 Evénements

La classe abstraite événément contient une date, un robot et une fonction abstraite d'exécution d'événement. On définit à partir de cette classe trois classes héritées, *EvenementDeplacer*, *EvenementVerserEau* et *EvenementRemplir*. Cela permet aux robots d'effectuer les trois actions principales décrites dans le sujet, de se déplacer, de verser de l'eau et de remplir leur réservoir.

La liste des événements est un *SortedSet* qui est triée selon la date de chaque événement. Cela permet d'exécuter les événements lors de la simulation de façon chronologique, même si le calcul de la liste des événements dans les stratégies ne se fait pas dans l'ordre.

1.5 Simulateur

Le simulateur est une classe qui permet de lancer la simulation des évènements selon leur date (en millisocondes) et d'afficher la carte au cours de l'exuction.

Le simulateur contient une méthode d'affichage *draw()* qui, à partir des données de *DonnéesSimulation* récupérées sur la carte affiche les cases de la carte, les robots et les incendies sous forme de *gui.ImageElement*.

Le simulateur permet aussi de modifier les dates de simulations grâce à la méthode *incrementeDate()*.

Le méthode *next()* exécute tous les évènements qui ont lieu avant la date de simulation. Ces évènement sont stockés sous forme d'un *SortedSet Evenement* en fonction de dates d'exécution. Le simulateur parcourt l'itérateur sur les évènements, si leur date est inférieure à la date de simulation ils sont acceptés puis exécutés dans l'ordre.

La méthode *restart()* remet les données de la carte à leur état initiale et remet la date à l'instant initial.

2 Les Stratégies utilisées

2.1 Calcule de deplacement

Le déplacement des robots ce fait après le choix du chemin optimal à parcourir. Ceci est fait par la classe *CalculeChemin*.

Pour qu'un robot puisse se déplacer d'une casse à une autre, il doit calculer le chemin le plus court grâce à l'algorithme de Dijkstra, dans notre cas on remarquera qu'au lieu de prendre la distance comme facteur de décision on a opté pour le temps de déplacement, ce qui peu engendrer un chemin plus long mais qui est parcouru plus rapidement par le rebot en question. Les méthodes utilisées sont: *CalculeRoute(Case caseActuelle, Case caseDestination, Carte carte)*, *construireRoute(Distance distanceActuelle, Case caseActuelle)*, *calculDeplacements(long date, SortedSetEvenement evenements, int tailleCase)*, *calculDeplacements(long date, int tailleCase)*.

Pour calculer la séquence des déplacements, nous avons opté pour une pile de cases : lorsqu'on calcule la séquence des cases que le robot doit suivre pour se rendre à la case destination, on rajoute à l'entête notre séquence de déplacements. La pile nous permet d'enléver l'entête de la séquence à chaque fois. Ainsi le calcul des événements de déplacement se fait dans le bon ordre.

2.2 Stratégie Elémentaire

Dans *ChefPompierElementaire* la méthode *strategieElementaire* parcours tous les incendies de la carte. Pour chaque incendie, elle prend le premier robot non occupé, calcule le plus court chemin pour ce robot pour atteindre l'incendie. Si le plus court chemin existe, le robot est déplacé (*EvenementDeplacer*) vers la case de l'incendie pour l'éteindre grâce à l'évènement *EvenementVerserEau*. Si le chemin n'existe pas on passe au robot non occupé suivant. Quand le feu est éteint on passe au feu suivant.

2.3 Stratégie Avancée

Dans la méthode *strategieElvoluee* de *ChefPompierEvoluee*, la méthode *trouverIncendie* permet de trouver le premier incendie de la carte non encore éteint. Tant que le résultat de cette fonction est non null, pour l'incendie en question, on cherche parmi les robots non occupés à la date donnée le robot qui va le plus rapidement à la case de l'incendie grâce à la méthode *trouverRobot*. Si ce robot existe, il est déplacé vers la case de feu, il verse l'eau. Si le réservoir du robot est vide *trouverCaseEau* retourne l'indice de la case d'eau la plus proche. Si elle existe le Robot se déplace à cette case, remplit son réservoir (à part le robot à pattes qui a un réservoir infini donc ne se remplit jamais). Ce traitement est fait jusqu'à ce que tous les feux soient éteints.

3 Tests effectués et Résultats

Le fichier principal de test, qui arrive à tester notre travail de façon complète est le fichier contenant la classe TestStrategieEvoluee. Nous avons crée plusieurs fichiers de test pour tenir en compte notre progression d'une étape à l'autre.

TestLecteurDonnees: Ce test récupère les données écrites dans le fichier de lecture donnée en argument. L'affichage des données lues montre bien que la lecture se fait correctement.

TestEvenement: prend en argument un fichier de données de la carte et un fichier dévènements. Ce test sert à tester un évènement élémentaire

TestScenario0 et ***TestScenario1***: ce sont deux tests demandés par le sujet et se font sur *carteSujet.map*. Le premier teste le déplacement d'un seul robot, le deuxième teste les évèment de déplacement, de remplissage et de versement pour un deuxième robot. Ces deux tests ont pour but de vérifier le fonctionnement simples des évènements choisis à l'avance. Ces tests fontionnaient bien avant qu'on améliore la gestion des évènements.

TestStrategieElementaire et ***TestStrategieEvoluee***: Ils prennent en argument un fichier décrivant la carte et une stratégie. Les stratégies fonctionnent pour les trois

premières cartes proposées. Le soucis pour la dernière carte c'est qu'il y a un robot qui ne peut rien faire et donv on reboucle.

Nous avons effectué des tests se basant principalement sur les cartes déjà fournies. En même temps, pour arriver à déboguer nos programmes nous avons utilisé des cartes plus simples, dans le but de tester sur des données les plus simples possibles, mais qui contiennent toujours les bugs qu'on cherche à éliminer.

Un problème assez important que nous avons pu détecter, c'est la façon d'ajouter dans un *SortedSet*. Notre set d'événements trié selon les dates contient également un comparateur. Notre comparateur pour quelques temps prenait en compte uniquement les dates. A cause de cela, lors de l'ajout de deux événements qui s'attribuent à deux robots différents, un des deux événements ne s'ajoute pas dans le *SortedSet*. La raison étant le comparateur programmé qui prenait ces deux événements comme étant les mêmes et donc ne rajoutait pas un même élément deux fois. Après changement de la notion d'égalité entre deux événements dans le comparateur, ce but n'est plus présent.

4 Améliorations à venir

Dans la suite, on pourrait envisager des autres stratégies pour éteindre des feus dans une carte. Par exemple, on pourrait penser de prendre en compte l'inténsité de chaque incendie, ainsi que la capacité de réservoir de chaque robot, afin de mieux assigner un robot à une incendie. Ainsi, on pourrait attribuer à la plus grande incendie, un robot à pattes qui a un réservoir qui se remplit jamais. Aux incéndies moins importantes, on pourrait également leur attribuer des robots les réservoirs desquels sont plus petits.