Choix de conception - Simulation de robots-pompiers

Equipe TEIDE 71

Une attention particulière a été portée au projet pour éviter les erreurs de modélisation de classe du problème qui pourrait mener à utiliser des méthodes contraires aux principes de la POO (utilisation de getClass...).

Partie 1 - Modélisation des données du problème

A - Robots & co

Chaque type de robots (***Drone***, ***RobotARoues***...) est représenté comme une sous-classe de la classe mère ***Robot***. Les spécifications de chaque sous classe sont contenues dans des variables de ***Robot***, qui sont initialisées à l'appel des constructeurs des sous-classes. Cette organisation permet de travailler plus facilement avec des listes de ***Robot***.

```
public RobotAChenilles(Case position, double vitesse) {
    super(
        position, //position (Case)

        //vitesseTerrain (EAU, FORET, ROCHE, TERRAIN_LIBRE, HABITAT)
        new double[]{0, vitesse/2, 0, vitesse, vitesse},

        utilisePoudre:false, //utilisePoudre
        remplitSurEau:false, //remplitSurEau
        quant_reservoire:2000, //quant_reservoire (L)
        quant_eau:2000, //quant_eau (L)
        5*60, //duree_remplissage (secondes)
        quant_eau_intervention:100, //quant_eau_intervention (L)
        duree_intervention:8 //duree_intervention (secondes)
    );
}
```

*Constructeur de la sous-classe **Drone***

Pour initialiser la vitesse d'un type de robot pour chaque nature de terrain, on pose une liste ***vitesseTerrain*** indexée par l'ordinal de chaque ***NatureTerrain*** (la vitesse est nulle si le type de terrain est inaccessible).

B - Création des données

On instancie toutes les données des cartes, après lecture, dans une classe ***DonneesSimulation***.

```
public class DonneesSimulation {

    public Carte carte;
    public Robot[] robots;
    public Incendie[] incendies;
```

C - Interface graphique

On dessine dans le ***Simulateur***, une image (dessinée par nos soins) pour chaque instance stockée dans ***DonneesSimulation***.

Au feu les pompiers !!!

Partie 2 - Modélisation des données du problème

A - Evénements

Un ***événement*** est caractérisé par sa date de fin (sa date d'exécution) et sa fonction abstraite ***execute***, qui sera appelé lors de son exécution.

On définit des sous-classes d'événement pour chaque type d'événement (***Déplacement***, ***Remplissage...***). Chaque sous-classe possède une classe statique ***calcDuree***, appelée dans le constructeur qui permet de calculer la date d'exécution de l'événement en lui donnant sa date de début.

```
public Remplissage(long date_debut, Carte carte, Robot robot, int quantEau){

    super(date_debut + calcDuree(robot, quantEau));

    this.carte = carte;
    this.robot = robot;
}
```

On doit ainsi connaître certaines évaluation future des variables du robot : Pour calculer la durée d'un remplissage qui se passera dans 3000 secondes, il faut connaître la quantité d'eau restante du robot dans 3000 secondes (qui ne sera pas forcément la même qu'au moment actuel.

B - Exécution d'événements.

Les événements sont insérés dans une liste chaînée d'événements triée par date d'exécution. A chaque appel de next du ***Simulateur***, on exécute les événements de la date courante et on les enlève de la liste chaînée, puis on incrémente la date courante.

Partie 3 - Plus court chemin

A - Chemin

On définit un chemin par une liste chaînée de ***CaseDuree***, une classe contenant une ***Case***, la ***durée*** et la ***direction*** pour accéder à la prochaine case. Ce sont les informations dont nous aurons besoin pour créer nos ***Déplacements***.

B - Dijkstra

On crée une classe ***PlusCoursChemin*** qui réunit des méthodes statiques utiles à la gestion de chemin.

La méthode principale, ***dijkstra***, renvoie le plus court chemin (en termes de temps à le traverser) à l'aide de l'algorithme de Dijkstra exécuté sur le graphe de cases voisines de la ***carte***.

Ce chemin renvoyé permet l'exécution de ***deplacerRobotChemin***, une méthode renvoyant la file de priorité d'événement à exécuter pour qu'un robot suive un chemin.

Partie 4 - Stratégies

A - Chef Pompier & associations

Les robots ont un ***état*** correspondant à leur action en cours (Disponible, Remplissage, Déplacement, Inutile...).

Un robot est inutile quand il ne peut accéder à aucuns incendies d'une carte, ainsi on évite de recalculer ***Dijkstra*** pour ces robots à chaque étape.

La classe abstraite ***ChefPompier*** permet l'exécution des différentes stratégies. On y définit une méthode abstraite ***jouerStrategie*** appelée à chaque ***next()*** de la simulation. Chaque sous-classe y définit sa stratégie correspondante.

On définit aussi une classe ***AssociationRobotsIncendie***, permettant les association bi-directionelle entre un robot et des incendies affectés (un incendie peut être affecté à plusieurs robots, mais un robot est affecté à au plus un incendie). Une affectation signifie que le robot va s'occuper ou s'occupe d'un incendie.

B - Stratégie élémentaire

Pour tout les robots disponibles :

- Si il n'a plus d'eau, on l'envoie se remplir à l'eau la plus proche et on rompt son affectation avec son incendie.
- Sinon on l'envoie éteindre un incendie accessible, non-éteint et non-affecté. (Et on l'affecte à l'incendie).

Dans ce cas, un incendie est associé au plus avec un robot, et le robot ne va pas chercher l'incendie le plus proche.

C - Stratégie évoluée

On envoie tous les robots prêt à se vider, se vider.
On envoie tous les robots vides se remplir et on supprime leur affectation.

Pour chaque incendie, on cherche le robot disponible, non-affecté, avec de l'eau, le plus proche, on l'affecte et on l'envoie éteindre l'incendie.

Un incendie peut alors avoir plusieurs robots associés, et les distances sont minimisées. Cependant cette méthode est plus lente sur la plupart des cartes : Les robots se répartissent moins bien que pour la stratégie élémentaire, et cela peut mener à plus de distance parcourue.

D - Résultats (en secondes de simulation)

carteSujet :
Stratégie élémentaire : 45761 secondes.
Stratégie évoluée : 37204 secondes

spiralOfMadness-50x50 :
Stratégie élémentaire : 32436 secondes
Stratégie évoluée : 52774 secondes

desertOfDeath-20x20 :
Stratégie élémentaire : 12754 secondes
Stratégie évoluée : 17215 secondes