Rapport Projet POO

Simulation d'une équipe de Robots Pompiers

L'Equipe Projet
M. Ben-rejeb Iyed
M. Luu Duc-Anh
M. Nabé Mamady

Tuteur : *M.* Sylvain Bouveret

Promo 2018

Sommaire

Introduction

Ce projet s'articule autour du développement d'une application Java permettant de simuler une équipe de robots pompiers évoluant de manière autonome dans un environnement naturel . Il s'inscrit dans la mise en pratique de nos connaissances en programmation orientée objet que nous avons pu étudier en cours. Les notions comme l'héritage, l'encapsulation, l'abstraction et les collections sont notamment abordées. Ces notions sont utilisées pour réaliser une application concrète en Java pour simuler une équipe de robots pompiers.

0.1 Implémentation des données du problème

Cette partie consiste à implémenter toutes les classes nécessaires à la génération des cartes de simulation. Elle constitue la mise en place des classes pour définir les données du problème et à l'implémentation de l'interface graphique.

Classes	**Attributs**	**Méthodes**
Carte	private int tailleCases private Case[][] cases	public boolean voisinExiste(Case src, Direction dir) public Case getVoisin(Case src, Direction dir) public Graphe construireGraphe(Robot robot)
Case	private int ligne private int colonne private NatureTerrain nature	public boolean equals(Object o)
Direction	NORD SUD OUEST EST	"Aucune"
NatureTerrain	EAU FORET ROCHE TERRAIN_LIBRE HABITAT	"Aucune"
TypeRobot	DRONE ROUES CHENILLES PATTES	"Aucune"
Incendie	private Case position private int litreEauNec	
DonneesSimulation	private Carte carte private List<Incendie> incendies private List<Robot> robots	public void addIncendie(Incendie incendie) public void addRobot(Robot robot) public boolean removeRobot(Robot robot) public boolean removeIncendie(Incendie incendie)

Dans ce premier tableau, on recense l'ensemble des classes du package donnees sans les robots.
On regroupe dans le tableau suivant les classes utilisées dans le package robot

Classes	Attributs	Méthodes
Robot	protected Case position protected int volumeEau protected double vitesse protected boolean estDisponible	public void deplacerPCC(Case destination, Carte carte, Simulateur simu) public long tempsPCC(Case destination, Carte carte) public long tempsDeplacement(Case c1, Case c2, Carte c) public long tempsDeversement(int volumeEau) public long tempsRemplissage() abstract public boolean isPlaceable(Case position) public boolean EteindreFeu(Incendie incendie) abstract double getVitesse(NatureTerrain nat) abstract public void deverserEau(int vol) abstract public void remplirReservoir()
Drone	Hérite des attributs de Robot	Implémente les méthodes abstraites de Robot
Chenilles	Hérite des attributs de Robot	Implémente les méthodes abstraites de Robot
Roues	Hérite des attributs de Robot	Implémente les méthodes abstraites de Robot
Pattes	Hérite des attributs de Robot	Implémente les méthodes abstraites de Robot

Dans la classe mère Robot, on réunit les attributs et les méthodes communes à tout type de robot. Puisque les tâches vont être effectuées par les robots, nous avons fait le choix de créer une méthode *EteindreFeu(Incendie)* dans la classe Robot qui contiendra l'essentiel du code des actions successives à faire pour éteindre un feu.

0.2 Implémentation des évènements

Dans cette partie, nous avons mis en place les différentes classes permettant de gérer différents évènements associés à des actions élémentaires sur la carte de la simulation ; comme déplacer un robot d'une case à une autre, le faire intervenir sur une case, le faire déplacer d'une case de départ à une case destination en empruntant le plus court chemin etc. Ces classes vont permettre dans la suite de simuler différents scénarios.

Classes	Attributs	Méthodes
Evenement	private long date private Robot rob	Abstract public void execute()
EvenementDeplacerElementaire	Hérite des attributs de Evenement	Implémente la méthode abstraite de Evenement
EvenementDeplacerSurDirection	Hérite des attributs de Evenement	Implémente la méthode abstraite de Evenement
EvenementDeverser	Hérite des attributs de Evenement	Implémente la méthode abstraite de Evenement
EvenementLibererRobot	Hérite des attributs de Evenement	Implémente la méthode abstraite de Evenement
EvenementRemplir	Hérite des attributs de Evenement	Implémente la méthode abstraite de Evenement
Simulateur	private GUISimulator gui private String fichierDonnees private DonneesSimulation data private DonneesSimulation dataPourGarder private List<Robot> robots private List<Incendie> incendies private int largeurCase private int longueurCase private long dateSimulation private LinkedList<Evenement> listeEvenements private LinkedList<Evenement> listeEvenementsPourGarder private ChefPompier mrBobLeChef	void initSimulateur(ChefPompier) void lireData(String) void dessineRobots() void dessineCarte() void dessineIncendies() void dessineCase(Case) void ajouteEvenement(Evenement) void incrementeDate() boolean simulationTerminee() void update() void next() void restart() void ajouteDeplacement(Robot,Direction) void ajouteDeplacementCase(Robot,Case) void ajouteDeversement(Robot,int,Incendie) void ajouteRemplissage(Robot,Case) void ajouteLibererRobot(Robot)

Pour effectuer la simulation on a implémenté 4 types différents d'événements dont 2 ont le même fonctionnement, pour le déplacement l'un suivant une direction donnée l'autre sur une case passée en paramètre, l'événement déverser pour verser un certain volume d'eau sur un incendie, l'événement remplir pour remplir le réservoir à partir d'une case d'eau et en dernier l'événement Liberer robot qui rend un robot disponible une fois qu'il

a fini sa file d'événement a exécuter. Des instances de ces événements concernant des actions spécifiques de robots de la cartes sont insérées dans la liste d'evenement attribut de la classe simulateur pour ainsi être exécutés une fois que la date de simulation est supérieur ou égale à leur date d'exécution. Pour effectuer l'insertion de ces événements en tenant compte des bonnes dates d'exécution on a implémenté une méthode dans la classe simulateur pour chaque type d'événement exemple ajoute Deversement pour l'événement deverser ces méthodes parcourent la file d'exécution et cherche l'événement de plus grande date associé au même robot pour calculer la date du prochain événement et assurer ainsi une bonne gestion du temps .

0.3 Calcul de Plus Court Chemin

Pour réussir cette partie, nous avons créé trois classes :

* **Une classe Point** qui présente un point du graphe à l'aide d'une case et de ses liens avec les autres points du graphe. Elle contient aussi deux attributs pour nous permettre de calculer le plus court chemin.

* **Une classe Arrete** qui lie deux points du graphe. Elle contient aussi un attribut poids qui est en fait le temps de déplacement entre ces 2 points.

* **Une classe Graphe** qui représente un graphe composé d'une liste de points.
  C'est ici que nous implémentons l'algorithme de calcul du plus court chemin. Nous avons choisi pour cela, l'algorithme de Dijkstra qui ne tient pas en compte les poids négatifs (ce qui est effectivement notre cas ici) et qui s'avère être efficace pour notre problème.

Classes	Attributs	Méthodes
Point	private Case casePoint private List<Arrete> destinations private int marque private int definitif	public List<Arrete> getDestinations() public void setMarque(int m) public void setDefinitif(int d) public void addArrete(Arrete arrete)
Arrete	private Point depart private Point arrivee private int poids	public Arrete(Point depart, Point arrivee, int poids)
Graphe	private Point[][] points	public List<Case> plusCourtChemin(Case depart, Case destination) private Point[][] clonePoints(Point[][] points) public static void afficherPCC(List<Case> liste) public void afficherGraphe()

0.4 Implémentation des Stratégies pour résoudre le problème

Pour répondre au problème, deux stratégies de gestion des robots pompiers : une simple et l'autre la plus élaborée. Cette partie donc consistait à l'implémentation des classes nécessaires pour une gestion autonome de notre environnement de simulation ; pour cela, nous avons mis en place trois (3) classes.

Classes	Attributs	Méthodes
ChefPompier	protected Simulateur simul protected DonneesSimulation donnees	abstract public void deploier()
StrategieElementaire	hérite des attributs du chef pompier	public void deploier()
StrategieEvoluee	hérite des attributs du chef pompier	public void deploier()

La classe ChefPompier joue le rôle de manager, on la définit abstraite pour l'utiliser ensuite dans la réalisation des autres classes qui vont implémenter les algorithmes des stratégies. Pour cela, nous avons besoin des données du simulateur, pour lui permettre d'avoir une vision d'ensemble de la situation. Ensuite, tout le code réside dans la méthode *deploier()* où nous exécutons les différentes tâches.

0.5 Tests réalisés

Nous avons effectué des tests de façon incrémentale, après la mise en place des classes de base, nous procédions à des tests élémentaires pour voir les résultats des modifications. Ici, nous nous sommes néanmoins plus concentrés sur les tests des fonctionnalités exigées quant au rendu du projet. Pour cela, nous avons effectué plusieurs tests.

Tests	Objectifs
TestScenario0	Vérifier la fonctionnalité des méthodes de changement de places des robots Vérifier l'affichage des éléments de la carte
TestScenario1	Vérifier la fonctionnalité des méthodes de remplissage et de déversement de l'eau des robots
TestStrategieElmentaire	Vérifier la fonctionnalité de la stratégie élémentaire, pour éteindre tous les incendies

* Dans le test **TestScenario0**, on a par exemple vérifier si un robot ne sortait pas de la carte et l'affichage attendu des différents éléments.

* Dans le test **TestScenario1**, nous avons vérifier si les robots se remplissaient sur la bonne case et s'ils arrivaient à éteindre les incendies effectivement.

* Dans le test **TestStrategieElementaire** nous avons vérifier si les commandes du robot chef pompier s'éxécutait pour éteindre les incendies sur la carte. Il montre la collaboration minimale entre les différents robots pour éteindre les incendies. Toutefois, il faut signaler que le robot Drone étant plus rapide que les autres, il y a des situations où il est le seul qui éteint tous les incendies.

Conclusion

Ce projet a été l'occasion pour nous de mettre en pratique nos connaissances de cours sur le concept de POO à travers le langage Java. Nous avons été guidés tout au long par les principes fondamentaux comme l'encapsulation , l'hérédité, le polymorphisme, la gestion des exceptions et l'abstraction. La liberté des choix dont nous avons bénéficié et l'autonomie demandée pour la réalisation de ce projet sont des éléments qui nous ont permis d'évoluer de façon coordonnée tout en maintenant une bonne dynamique de groupe. L'un des éléments qui nous a permis aussi d'évoluer vite et en n'ayant pas à redéployer les structures abstraites est l'utilisation des collections Java. Bien qu'au début leurs utilisations ne nous apparaissaient pas nécessaires, nous nous sommes vite rendus compte de leur nécessité, qui plus est, il fallait les comprendre de nous mêmes. Malgré les difficultés rencontrées, ce projet fut l'occasion de saisir les dimensions d'un réel projet ; de la conception (bien que beaucoup guidée !) à l'implémentation des solutions, en passant par l'organisation et la communication au sein de l'équipe.