Compte rendu TP Java

Le Louer Adrien, Voland Dorian, Maxence Masson

18 novembre 2022

1 Premiere Partie : Conception des classes et de l'architectures

1.1 Architecture du projet

Nous avons fait le choix d'organiser le projet en paquet car c'est une notion clef de java. En faisant cela nous avons une architecture propre qui permet de limiter l'usage des ressources a importer par chaque fichier.

Les paquets sont les suivants

- environment
  - Carte : Classe permettant de créer et de gérer la carte est composé de case
  - Case : Case qui compose la case, sur lequel sont positionné les objets.
  - Direction : Direction cardinal (type Enum) : NORD, EST, SUD, OUEST
  - Incendie : Incendie qui a lieu sur une case.
  - Nature terrain : Type de terrain possible pour une case (type Enum)
- evenement
  - Evenement : Classe abstraite avec trois fonctions à implémenter
- io
  - LecteurDonnees : lit et renvoie un type DonneesSimulation
- robot
  - Une classe par type de robot (drone, robot à chenille, ...) et une classe abstract robot et la classe chef pompier.
- simulation
  - La classe DonneesSimulation et Simulateur
- test
  - Tout les tests relatif au projets

Toute les classe on leur attribut en privée et ne sont accessible qu'a l'aide de getter et de setter.

1.2 Modélisation classe robot et carte

1.2.1 Implémentions des classes robots

Nous avons choisi de mettre un maximum de fonction et méthode dans robots et de ne laisser dans les classes qui en hérite le minimum. Ainsi, seul les methode : getVitesse, getTempsRemplissage, getExtinctionRatio, getMaxCapaciteReservoir et getType sont a redefinir.

Nous avons choisi de les faire sous forme de fonction, plutôt que de simple constante, cela rend la possibilité d'implémenter des fonctions plus complexe si des amélioration sont a prévoir facilement.

De plus, les robots prennent des paramètre en plus, qui permette l'utilisation du chef pompier (chef avec les méthodes setChef et getChef), ou encore du restart de la simulation.

Nous avons fait le choix d'avoir des robots simple, avec des méthodes permettant seulement des déplacements et interactions.

1.2.2 Implémentions de la carte

Pour la carte, nous la composons de case. Chaque case comporte sa position dans la carte, sont terrain et si elle possède un incendie, une référence vers ce dernier. Cela permet d'éteindre l'incendie a l'aide d'un robot car un robot possède une référence vers sa case.

La carte possède une méthode public static getPathToNearerWaterCase, qui permet de calculer le plus cours chemin vers une case d'eau.

La carte possède aussi une liste de toute les cases d'eau, elle nous permet de calculer le case d'eau la plus proche sans avoir relire toute la carte.

2 Deuxieme Partie : Conception du simulateur et des événements

2.1 Conception simulateur

Pour le simulateur, nous avons choisi d'utilisé des images isométriques pour représenter nos objets, c'est image sont stocké dans chacune des instances de leur objet. Le simulateur implemente l'interface simulable

2.1.1 Methode next et affichage

Ensuite le temps est gérer par un long qui s'incrémente de 1 a chaque appuis sur le bouton next. Nous avons choisi le pas : 1 pas de simulation correspond a 1s.

A chaque appuis sur la touche next, tout les evenements ayant une date de simulation inferieur a la date courante, sont exécuter. Si leur date reste inferieur a la date courante il sont supprimé.

Pour la partie affichage, de la carte nous avons des fonctions qui convertissent les coordonnées du tableau en coordonnée affichable. Ensuite a chaque pas, une fois tout les evenments appeler, nous somme obligé de redessiner toute la map pour prendre en compte l'isométrique (superposition d'image).

2.1.2 La méthode restart

Tout les evenements ajouté a l'instant t=0 sont considéré comme les événements initial du programme. Cela veut dire que la simulation doit s'engendrer a partir de ces évènements. Il sont stocké afin de pouvoir restaurer plus tard. Nous stockons aussi les robots et les feux créer a l'instant t=0. Si uterieument, des evenements les fonts disparaitre ou en ajoute de nouveau (propagation feu par exemple), il est facile de retrouver les feux initiaux.

Pour restart, chaque objet, que soit les robots, les incendies, ou les évènements, tous possède une methode reset, qui les remets dans l'état qu'il était $t = 0$;

2.2 Les événements

2.2.1 Les événements élémentaire

Des évènements élémentaires sont utilisé, il s'agit de DeplaceRobot, RemplirRobot et Deverser.

Chacun d'un commence a une date t0 et se termine a une date t1. La modification faites par les évènements n'est prise en compte et affiché qu'a la date t1, c'est a dire une fois l'évènement terminer. Un évènement peut donc modifier sa date pendant sont exécution. Par exemple DeplaceRobot prend le temps de trajet du début à la fin de la case courante.

La date évènement possède l'attribut eventAExecuter. Cette attribut permet d'exécuter un évenement créer dans un évènement sans l'ajouter à la priorityQueue. Il faut utilisé les fonctions LancerEvenementEnAttente() et LancerEvenementPuisLeMettreEnAttente() qui s'occupe de gerer se systeme d'evenement imbriqué.

2.2.2 les événements avancés

Les évènements Intervenir et Remplir utilisent des évènements élémentaire, il sont donc des évènements avancé.

Les évènements élémentaire doivent donner leur temps d'exécutions afin de pouvoir prendre en compte leur exécution dans les évènements avancés.

L'utilisation de se système permet de considérer les évènements Intervenir et Remplir comme de simple évènements pour la simulations. De plus, cela simplifie leur comportements.

3 Troisième Partie : Calcul du plus court chemin

Nom de classe : CalculPlusCourtChemin

Attributs : Carte carte, Robot robot, Simulateur simulateur, Case caseDepart, Case caseArrivee, double timetotravel, ArrayList<Case> chemin

Nous avons décidé de déléguer le calcul du plus court chemin à une classe tierce pour rendre le code plus modulaire et rendre le débogage/la lecture plus aisée.

Principe :

Pour le calcul du plus court chemin, nous avons utilisé le principe de l'algorithme de Dijkstra avec quelques changement. :

1. L'algorithme de Dijkstra calcul le plus court chemin pour aller à n'importe quelle point du graphe, ici nous ne voulons que le plus court chemin pour aller entre deux cases ainsi nous faisons un parcours en Breadth-First Search pour ne visiter que les cases qui sont réellement nécessaire au calcul.
2. Dans Dijkstra à l'initialisation, tous les points du graphe sont rangés dans une liste et on les enlève une fois qu'elles sont visitées. Or ici, à cause du parcours Breadth-First Search, nous sommes obligés d'avoir une liste contenant les cases à visiter et celles déjà visitées. Nous avons choisis de les modéliser par des HashSet car on ne veut pas de doublons.
3. La présence de cases qui ne peuvent pas être traversées par certains robots nous oblige dans l'initialisation à mettre les cases d'eau comme "case visitée" puisque le robot ne pourra et ne doit jamais aller dessus.

Modélisation :

Pour modéliser la table des coûts et les prédécesseurs, nous avons utilisés une HashMap pour pouvoir accéder facilement aux informations relatives à une case.

Il découle donc comme méthodes principales pour réaliser la partie 3 :

1. Obtention de la case parmis les cases pas encore visitées minimisant le temps à parcourir dans le sous-graphe courant
2. Calcul du temps pour parcourir une case
3. Initialisation des différentes listes
4. Implémentation de Dijkstra
5. Calcul du chemin et du temps de déplacement grâce à l'HashMap des prédécesseurs et de la table des coûts

Il est à noter que dans la classe qui calcul le plus court chemin, nous n'avons pas de méthode qui ajoute les évènements nécessaire au déplacement du robot, ce travail sera effectué dans l'évènement Intervenir et Remplir.

Tests et Resultats :

Pour tester, nous avons pris des cases aléatoires et avons fait déplacer les robots dessus pour voir si les chemins étaient les bons et si ils respectaient bien les contraintes d'environnements. Les résultats étaient concluant, les chemins étant les bons et les contraintes respectées.

4 Quatrieme Partie :

Nom de la classe : ChefPompier

Attributs : Simulateur simulateur, Carte carte, Robot[] robots, Incendie[] incendies

Principe :

A l'initialisation, pour chaque robot, le chef pompier envoie le robot sur l'incendie le plus proche de lui, qu'il peut atteindre. Si il n'existe pas, cela veut dire que le robot ne peut accéder à aucun incendie et donc il ne sera plus utilisé par la suite. Ensuite, à chaque fois que un robot finit d'intervenir sur un feu, il regarde si son réservoir est vide. Deux scénarios sont possibles : -Soit il est vide et donc il notifie le chef pompier qu'il va remplir son réservoir. Si il ne peut pas le remplir, il ne sera plus utilisé par la suite, si il peut, une fois qu'il a finit, il appelle le chef pompier pour avoir sa prochaine affectation. -Soit il est non vide et donc il appelle le chef pompier pour avoir sa prochaine affectation.

Lorsqu'un robot demande une affectation au chef pompier, le chef pompier regarde si il reste des incendies à éteindre et si le robot possède un incendie proche atteignable. Si il y en a un, alors il envoie le robot sur l'incendie, sinon, le robot ne sera plus jamais utilisé.

Modélisation :

Les évènements Intervenir et Remplir sont nécessaires. L'évènement Intervenir réalise le déplacement puis le déservage d'eau si une fois sur place l'incendie est encore allumé. Une fois fini, le robot regarde l'état de son réservoir et en fonction, fait l'appel qu'il faut au chef pompier. L'évènement Remplir regarde si c'est possible, si oui il réalise le déplacement et le remplissage du réservoir, si non le robot ne fait aucun appel au chef et donc ne sera plus utilisé.

Il découle donc comme méthodes principales pour réaliser la partie 4 :

1. Regarder si il reste des feux à éteindre
2. Obtenir le feu le plus proche du robot
3. Appel au chef pour lui demander si le robot appelant doit encore intervenir ou pas
4. Appel au chef pour lui signaler que le robot appelant va se remplir

Tests et résultats :

Avant de passer a la suite, chaque parti est testé afin de vérifier quelle fonctionne correctement. Cependant nos tests notre batterie de test ne couvre pas tout les cas.

Pour tester les évènements, on effectue trois scenario, 1 test la sorti de map comme cas limite. Un autre test toutes les directions de deplacements possibles et les evenements se produisant en meme temps. Enfin on test les déplacements avancés, remplir et déverser (fonctionne que si évènements élémentaire fonctionnent).

Pour tester le chef pompier, nous avons pris toutes les cartes données et avons lancé le chef pompier dessus. Les robots éteignaient bien tous les feux en passant par les bons chemins et ils remplissaient leurs réservoir une fois vidé. Cependant, pour la carte spiralOfMadness, la simulation est un peu longue. Il reste donc des optimisations à faire.