Projet POO - Rapport

1 Présentation du projet

Le but de ce projet était de réaliser une application permettant de simuler des robots pompiers évoluant dans un environnement naturel. Pour réaliser cette application, nous avions à notre disposition une archive permettant d'instancier une interface graphique de simulateur. Ce code permettait de créer et d'afficher une fenêtre d'interface graphique et de dessiner différentes formes ou images sur cette interface. Il permettait aussi de contrôler la simulation via les différents boutons disponibles (Lecture, Suivant, Début, Quitter).

Pour réaliser cette application, nous avions donc besoin de créer les différentes instances visibles sur la carte, un robot ou un incendie par exemple. Il fallait également implémenter un gestionnaire d'évènement afin de gérer la temporalité des actions et il fallait utiliser l'archive fournie pour afficher le tout.

2 Organisation des packages et présentation de la structure du code

Nous avons rangé nos différentes classes dans des packages pour améliorer l'organisation de notre code. Nous allons donc présenter chaque package avec les différentes classes qu'il contient. Nous tenons à préciser que l'on a également créé de nouvelles exceptions pour être précis sur les types erreurs qui peuvent se produire. Ces exceptions se trouvent dans le package `nouvellesException`.

2.1 `package carte`

Dans ce package se trouvent toutes les classes qui ont un rapport avec l'organisation spatiale du programme, on y trouve :

- `class Case` : Cette classe définit une instance case qui représente une partie de la carte, toutes les cases n'ont pas la même nature du terrain, certaines contiennent de l'eau, d'autres des forêts, d'autres des habitations...
- `class Carte` : Cette classe définit une instance carte. Cette instance connait toutes les cases de la carte, et connait aussi la taille de carte.
- `class Chemin` : Cette classe définit une instance chemin. Un chemin permet de décrire les déplacements d'un robot dans le temps. Il permet de décrire l'ensemble des cases par lesquels passer pour relier deux cases entre elles. Il représente en fait une LinkedList d'étapes.
- `class Etape` : Cette classe définit une instance étape. Une étape est constituée d'une case et d'une date, il s'agit en fait de la date où le robot va terminer son déplacement sur cette case. Ainsi un objet chemin est capable de savoir pour n'importe quelle case du chemin à quelle date le robot sera sur la case.

2.2 package incendies

On ne trouve qu'une seule classe dans ce package, il s'agit de la `class Incendie`. On y caractérise l'objet incendie qui possède donc une position sur la carte (grâce à l'objet case) et la quantité d'eau nécessaire pour éteindre l'incendie.

2.3 package robots

On y trouve une classe abstraite permettant de définir l'objet Robot de façon générale. Mais on y trouve également les classes représentant précisément les différents types de robot. On y trouve :

- `abstract class Robot` : Cette classe définit un objet Robot avec de nombreux attributs propres au robot : sa vitesse, sa position, la capacité de son réservoir... Cette classe définit également de nombreuses méthodes utiles au robot. Ici une liste non exhaustive des différentes méthodes implémentées :
  - `calculPlusCourtChemin` : Cette méthode va utiliser l'algorithme de Dijkstra pour calculer le coût en temps pour se déplacer sur une case précise de la carte. Cela permet donc de trouver le chemin le plus court vers la case où l'on veut aller.
  - `caseEauPlusProche` : Cette méthode va permettre de trouver la case d'eau la plus proche du robot et de renvoyer un chemin vers cette case. Cette fonction est utilisée lorsque le robot souhaite remplir son réservoir.
  - `eteindreIncendie` : Cette méthode va ajouter au gestionnaire d'événements les évènements nécessaires pour que le robot se déplace jusqu'à l'incendie et pour qu'il déverse de l'eau sur la case de l'incendie. Si le robot contient trop d'eau, il verse juste l'eau nécessaire pour éteindre l'incendie, sinon il vide la totalité de son réservoir. Enfin, on programme un événement qui appelle vaSeRemplirOuPas après avoir terminé le déversage.
  - `vaSeRemplirOuPas` ; Cette méthode indique au robot d'aller se remplir si son réservoir est vide. Elle est appelée après avoir déversé de l'eau sur une case. Elle programme donc les évènements nécessaires au robot pour se déplacer vers la case d'eau la plus proche et pour se remplir. Elle programme ensuite soit un événement qui permet de retourner éteindre l'incendie auquel le robot est affecté, soit un réveil du chef pompier si cet incendie est maintenant éteint.
- Les autres classes sont en fait des classes filles de la classe Robot et implémentent des méthodes spécifiques à chaque type de robot.

2.4 package evenement

On trouve dans ce package tous les évènements que nous avons implémenté et que nous utilisons dans nos fonctions. On y trouve notamment des évènements de base comme `Deplacement`, `Remplissage` et `Deversage` qui vont simplement effectuer une action donnée à une date spécifiée. De plus, il existe des événements utilisés pour appeler à une date précise des fonctions plus complexes :

- `EteintIncendie` Cet évènement va exécuter la fonction `eteindreIncendie` qui est décrite plus haut.
- `IncendieEteintOuPas` Cet événement va appeler la fonction `vaSeRemplirOuPas` décrite ci-dessus.

2.5 `package simulation`

Ce package contient les informations essentielles au fonctionnement de la simulation. On y trouve `DonneesSimulation` qui contient toutes les informations sur les robots, les incendies et la carte. Une instance de `DonneesSimulation` est obtenu grâce à la classe `LecteurDonnees` qui parcourt l'ensemble du fichier source de la carte et réunit les différentes informations dans les ArrayList correspondantes.

On y trouve également `ChefPompier` qui s'occupe d'affecter les incendies aux robots. Pour cela il utilise une stratégie plutôt basique. Il va parcourir la liste des robots et pour chaque robot non affecté, il va chercher un incendie qui n'est pas déjà affecté. Il va ensuite demander au robot si il peut se déplacer sur l'incendie, si c'est le cas il affecte l'incendie au robot sinon il passe à l'incendie suivant. De plus, à chaque fois que le robot a fini d'éteindre un incendie il va réveiller le chef pompier qui va l'affecter à un nouvel incendie.

Enfin, nous avons également un simulateur qui s'occupe à la fois de gérer l'affichage de la carte et de la position des robots mais qui gère aussi le flot d'exécution des évènements. On peut souligner quelques points intéressants :

— Le stockage des évènements s'effectue dans une pile à priorité, cela permet donc d'ajouter un évènement à la pile et que celui-ci soit immédiatement placé au bon endroit dans la pile (c'est à dire à la bonne date). Ainsi il suffit de parcourir la pile dans l'ordre et d'exécuter les évènements en haut de la pile (date la plus petite) en premier à chaque fois.

— La fonction restart va s'occuper de replacer les robots au bon endroit, de rénitialiser le nombre de litres d'eau pour éteindre l'incendie et changer l'affectation des robots. Il va redessiner la carte puis il appelle ensuite la fonction du chef pompier `affecteIncendies` qui va s'occuper de préparer les déplacements des robots pour le moment où l'on va appuyer sur lecture.

3 Tests effectués et résultats obtenus

Nous avons écrit 4 tests différents que nous allons vous présenter, ainsi que les résultats que nous avons obtenus à ces tests.

3.1 `TestAffichage`

Ce test permet de vérifier le bon fonctionnement de l'affichage de la carte à l'aide de l'interface graphique. Nous avons donc pu remarquer que pour toutes les cartes, celles-ci s'affichaient toutes correctement.

Ce test permettait également de vérifier la bonne position initiale des robots et des incendies sur la carte.

Nous avons donc pu remarquer que notre programme fonctionnait bien et que la taille des cases affichées s'adaptait à la taille de la carte.

3.2 `TestScenario0`

Le but premier de ce test est de savoir si on observe une erreur quand un robot essaye de sortir de la carte. Toutefois, ce test va aussi permettre d'effectuer un premier test des déplacements, puisque le robot va essayer de sortir de la carte.

Les résultats de ce test sont satisfaisants puisque l'on peut voir que le drone se déplace bien vers le nord comme on le veut mais aussi parce que le programme lève une exception créée pour l'occasion quand celui-ci essaye de sortie de la carte.

3.3 `TestScenario1`

Ce test va permettre de tester les évènements de base, c'est-à-dire les évènements `Déplacement`, `Remplissage` et `Deversage`. Pour cela, on prévoit toutes les actions du robot avant de lancer le déroulement temporel du programme. On ajoute donc les évènements nécessaires pour se déplacer, déverser l'eau, et remplir le réservoir.

On peut voir que toutes les actions s'effectuent correctement, le robot se déplace bien avec la bonne vitesse, il déverse bien son eau à la bonne vitesse et se remplit également à la bonne vitesse. A la fin de toutes les actions, l'incendie est éteint. Le test est donc réussi car tout s'est déroulé comme prévu.

3.4 `TestEteintTout`

Ce test représente la finalité du projet, c'est-à-dire la réalisation d'une application permettant de simuler des robots pompiers évoluant dans un environnement naturel. On peut tester le programme sur les différentes cartes et on peut voir que quelle que soit la carte choisie, les robots sont bien tous affectés à un feu différent. Ils s'occupent tous d'éteindre leur feu et de réveiller le chef pompier dès qu'ils ont terminer pour se voir attribuer un nouvel incendie.

Notre application est donc fonctionnelle et permet d'éteindre tous les incendies sur l'ensemble des cartes proposées.

4 Conclusion

Nous avons apprécié ce projet, il était très interactif et on peut facilement observer notre avancement au fur à mesure en regardant dans un premier temps la carte s'afficher, puis les robots bouger, et enfin en automatisant la gestion des robots.

Néanmoins, avec plus de temps, il aurait été assez facile d'optimiser certaines fonctions encore un peu trop coûteuses. De plus, on pourrait aussi facilement créer d'autres stratégies plus évoluées afin d'éteindre les feux plus rapidement.