Rapport POO

Choix de conception

- Tous les robots doivent se situer sur la case de l'incendie pour pouvoir déverser leur eau, ce qui explique pourquoi le robot à roues ne se déplace pas sur la carte `MushroomOfHell` et `SpiralOfMadness` tous les feux étant situé sur des cases forêt, inaccessibles pour ce robot.
- L'échelle temporelle choisie est de **1 pas = 1 seconde**, il est conseillé d'accélérer le temps lorsque vous lancerez les test.
- Un incendie éteint est toujours considéré comme un incendie mais son attribut `nbL` est égal à 0.
- L'algorithme de calcul de plus court chemin choisie pour le déplacement des robots est **l'Algorithme de Dijkstra** (voir section complexité).
- Différents assets (images) sont utilisés pour les cases de forêt, rocher, ou eau en fonction de la case adjacente, pour un meilleur confort visuel.
- Le remplissage d'eau est une initiative propre aux robots, le chef pompier ne leur ordonne pas.
- Une image de la carte sans robot ni incendie est générée au lancement de la simulation et sert de fond pour le dessin de chaque étape de la simulation. Cela permet de ne pas générer plusieurs fois la carte.

Stratégies Choisies

Les stratégies élémentaire et avancée du sujet ont été implémentées et sont décrites dans les javadocs (voir les fichier `ElementaryFirefighterChief` et `AdvancedFirefighterChief`). Par ailleurs nous avons créé un nouvelle stratégie, mise en oeuvre par la classe `ImprovedFirefighterChief` qui consiste a assigner à chaque robot l'incendie le plus proche à éteindre à l'inverse de la stratégie avancée qui cherchait le robot le plus proche d'un incendie selectionné, cela nous semblait plus cohérent, car sur la stratégie avancée, des robots pouvaient se rendre sur un incendie très éloigné alors qu'il y en avait un juste à côté d'eux.

Remarque

Dans la stratégie élémentaire, il n'est pas spécifié que les robots doivent aller se remplir ils ne le font donc pas et le seul robot capable d'éteindre les incendies est le robot à pattes, équipé d'un réservoir de taille infinie.

Test effectués

Des tests peuvent être effectués, toutes les commandes sont spécifiées dans le **MAKEFILE** et expliquées dans le **README**. Parmi ceux-ci on retrouve les tests suivants :

- Le `Scenario0` déplace le drone 4 fois le drone vers le Nord dans la carte `carteSujet.map`, il essaye de faire sortir le drone hors de la carte et génère donc une exception.
- Le `Scenario1` fait les étapes suivantes :
  1. Déplacer le $2^{\text{ième}}$ robot (à roues) vers le nord, en (5,5).
  2. Le faire intervenir sur la case où il se trouve.
  3. Déplacer le robot deux fois vers l'ouest.
  4. Remplir le robot.

5. Déplacer le robot deux fois vers l'est.
6. Le faire intervenir sur la case où il se trouve.
7. Le feu de la case en question doit alors être éteint.

- Les différentes stratégies implémentées sont testées dans `Strategies.java` où chaque classe du fichier correspond à une stratégie (`StrategieElementaire` correspond au test de la stratégie élémentaire, `StrategieAvancee` au test de la stratégie avancée et `StrategieMieux` à la stratégie la plus optimale).

Remarque

Tous les tests fonctionnent et peuvent être effectués sur un terminal ou les IDEs mentionnés grâce aux commandes spécifiées dans le **Makefile**.

Complexité

On cherche ici à déterminer la complexité de l'opération la plus coûteuse de chaque stratégie : la répartition des robots sur les feux. On effectue nos calculs pour une carte carrée de $M = N \times N$ cases, avec i incendies et r robots.

La complexité de l'algorithme de Dijkstra est de $\Theta\left((a+n)\log(n)\right)$ avec n le nombre de sommets graphe (dans notre cas, le nombre de cases sur la carte, soit N^2) et a le nombre d'arêtes (dans notre cas, une arête = une case reliée à une autre, soit $2N(N-1) = \theta(N^2)$). La complexité en fonction des données du problèmes est donc de $\Theta\left(N^2\log(N^2)\right) = \Theta\left(M\log(M)\right)$.

▷ Stratégie Elementaire : Le chef pompier va boucler sur les incendies et affecter à chaque robot le premier feu. On appelle donc l'algorithme de Dijkstra une fois par robot pour une complexité finale en $\Theta\left(rM\log(M)\right)$.

▷ Stratégie Avancée : Le chef pompier va boucler sur les incendies et affecter pour chacun le robot le plus proche. On appelle donc l'algorithme de Dijkstra au maximum une fois par robot pour chaque feu (dans le cas où les robots ne pourraient pas atteindre le feu). La complexité finale serait alors de $\Theta\left(irM\log(M)\right)$.

▷ Stratégie Améliorée : Le chef pompier va boucler sur les robots et leur affecter l'incendie le plus proche. La sélection de l'incendie est réalisée au fur et à mesure de l'algorithme de Dijkstra que l'on appelle ainsi qu'une seule fois par robot. La complexité finale serait donc de $\Theta\left(rM\log(M)\right)$. On retrouve la même complexité que la stratégie élémentaire malgré une répartition nettement plus intelligente des robots.

Remarque

Pour se rendre compte de l'intérêt de la stratégie améliorée vous pouvez lancer les différentes stratégie sur la carte `trueSpiralOfMadness-50x50.map` qui met en évidence les défauts des deux premières stratégies.

Classes et Méthodes

Un diagramme simplifié a été réalisé pour comprendre les classes et leurs méthodes (Vous le trouverez également au format html à la racine du projet).

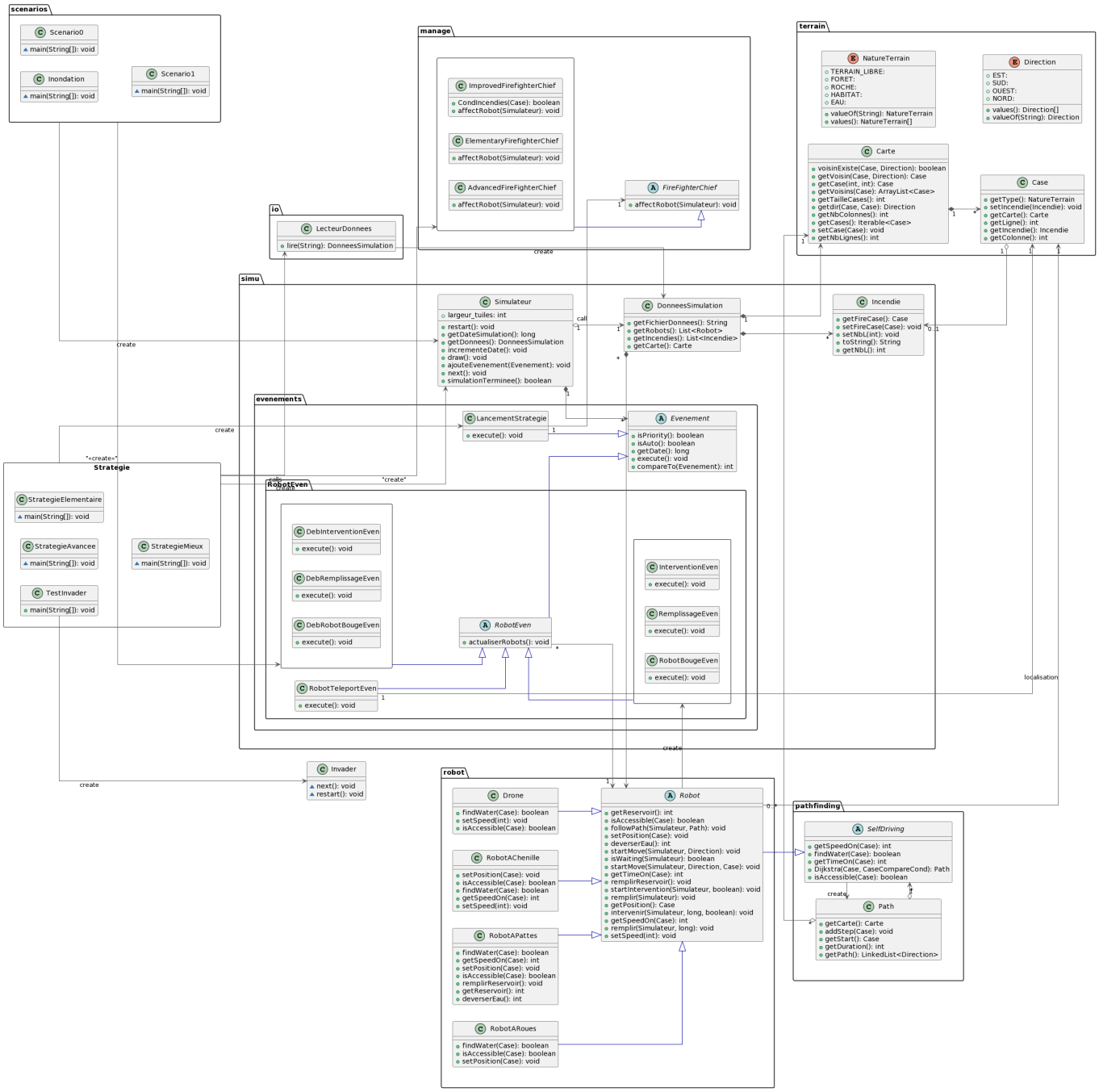

FIGURE 1 – Le graphe UML des différentes classes et méthodes