Simulateur de robots pompiers

Programmation Orientée Objet

Groupe 54

Ce projet a été réalisé dans le cadre des cours de programmation orientée objet à L'ENSIMAG.

12.11.2018

Sommaire

1 Implantation 4

1.1 Etat du logiciel 4

1.1.1 Le lecteur de données 4

1.1.2 La classe Drawable 4

1.1.3 Les données de simulation 5

1.1.4 Les robots 5

1.1.5 Les évènements 5

1.1.6 Simulateur 6

1.1.7 PathFinder 6

1.1.8 Stratégie 6

1.1.9 Le main 7

1.2 Tests effectués 7

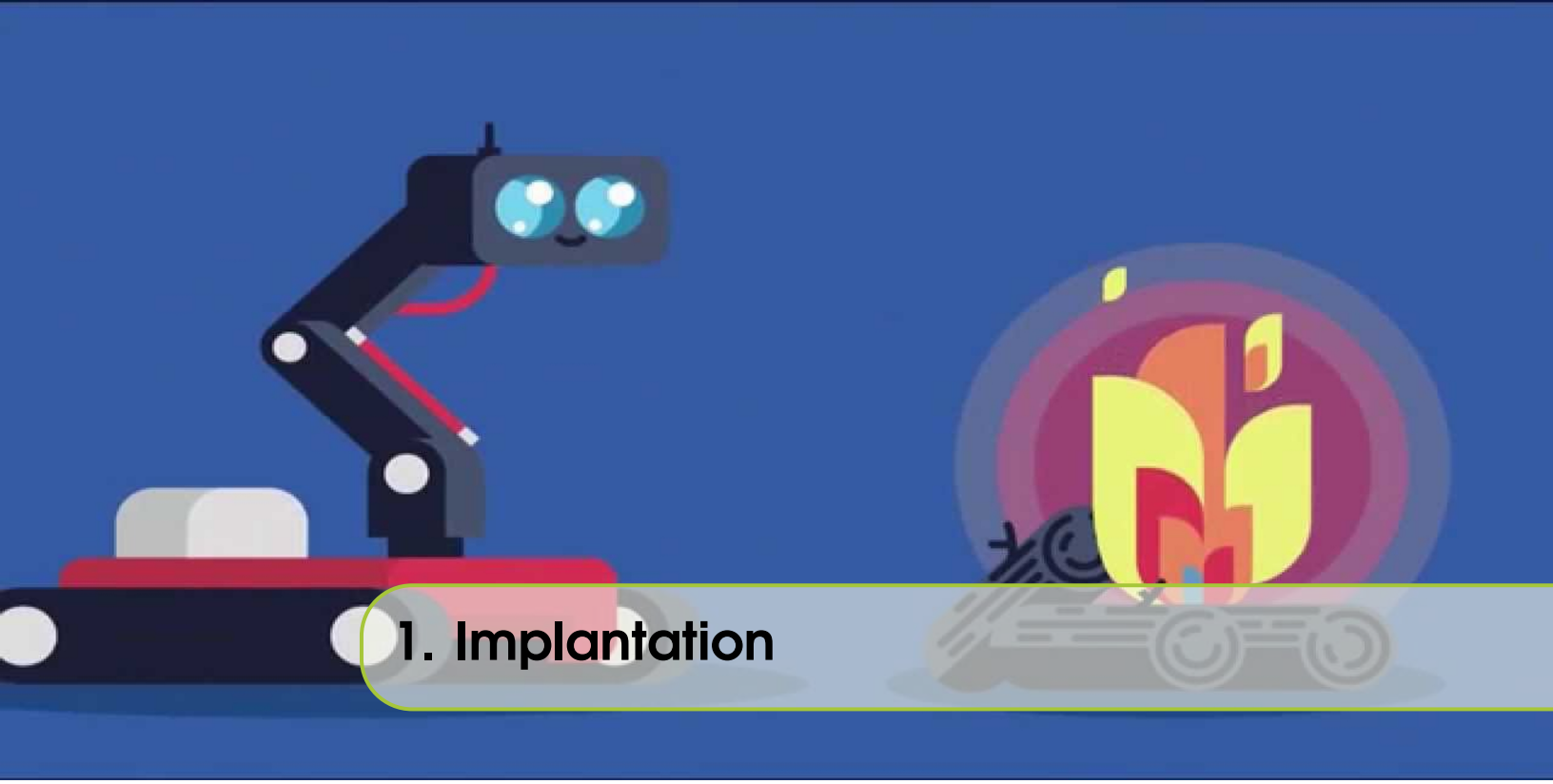
1.2.1 Tests unitaires 7

1.2.2 Tests d'intégration 7

1.2.3 Tests de performance 8

1.3 Conclusion 8

2 Annexes 9



1. Implantation

L'état actuel du projet est satisfaisant, celui-ci répond au cahier des charges explicité dans le sujet[1].
Toutes nos sources compilent selon les instructions du fichier build.xml, voir le README.md.
Si vous avez réussi à récupérer ce pdf, vous avez réussi à décompresser l'archive, et vous n'êtes plus très loin de pouvoir tester notre programme ! La lecture de ce document ainsi que du README vous indiquera comment compiler et exécuter nos sources...
La documentation complète de l'API au format HTML est également disponible (Javadoc).

1.1 Etat du logiciel

Nous sommes satisfaits de l'état d'avancement de notre logiciel, celui-ci répond au cahier des charges explicité dans le sujet. Toutes nos sources compilent selon les instructions du fichier build.xml, voir README.md. Les sources sont organisées en plusieurs packages. Dans la suite, les choix de conception des différentes parties du projet seront détaillés.

1.1.1 Le lecteur de données

Package poog54.io

Notre lecteur de données permet essentiellement via la méthode dataFromFile() de retourner une instance des données de simulation initiales, lues dans un fichier .map. Sa structure est similaire au lecteur de données fournis par les enseignants.

1.1.2 La classe Drawable

Package poog54.io

La classe Drawable est importante à remarquer. Toute classe destinée à être dessinée sur l'écran devra hériter de Drawable. Ainsi, on factorise les méthodes, et les concepts de coordonnées, de chemin du .jpg, et de priorité d'affichage (robots au dessus des incendies, eux mêmes aux dessus des tiles). On peut alors ne traiter que des drawables du point de vue du simulateur lors de l'affichage.

[1] Projet - Simulation d'une équipe de robots pompiers

1.1.3 Les données de simulation

Package poog54.dataclasses

Classes triviales de représentation des données du problème. Les Tiles et incendies sont Drawable. La classe Map est une hash map (voir collections java) de tiles. Une méthode bien pratique de TheMap permet de retourner un itérateur sur tous les drawables de la simulation !

1.1.4 Les robots

Package poog54.dataclasses.robots

La classe abstraite Robot hérite de Drawable. Elle permet de centraliser toutes les caractéristiques communes des pompiers: position, vitesse, capacité de leur réservoir, etc... On note que chaque robot dispose de la carte afin de se déplacer de manière autonome. On note également que la méthode GetPathToPoint(Point) fournit le chemin le plus court entre la position courante du robot et la destination passée en paramètre.

1.1.5 Les évènements

Package poog54.dataclasses.events

Le simulateur possède une liste PririorityQueue d'événements. Cette liste trie automatiquement les événements par ordre chronologique : le critère de priorité est la date des événements discrets.

A chaque pas de simulation, la liste d'événements est dépilée (aperçu, puis suppression en tête) tant qu'il existe des événements dont l'horodatage est antérieur à la date courante maintenue par le simulateur. La classe DiscreteEvent est une classe abstraite qui porte l'horodatage d'un événement et une méthode abstraite Execute(). Elle est dérivée en événements spécifiques (cf annexe Événements Discrets). Chaque robot est géré par une machine d'état dédiée. Au démarrage de la simulation, on considère que tous les robots ont leurs réservoirs pleins. La simulation commence alors par l'affectation d'un feu cible à chaque robot (cf. CHAPITRE STRATÉGIE 1.1.8).

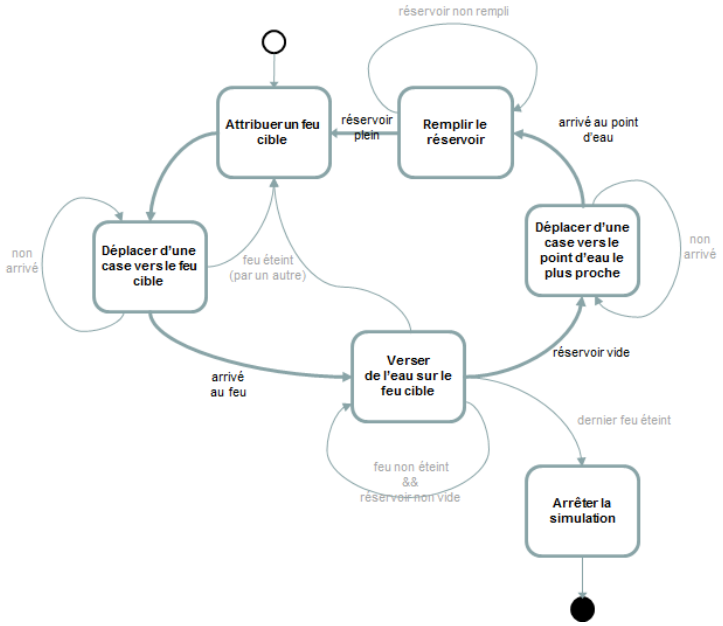


Figure 1.1: Diagramme d'états

Chaque état décrit correspond à l'exécution de l'action associée à un événement discret, via la méthode execute(). On remarque que le déplacement d'un robot vers un feu peut être annulé si celui-ci a été éteint par un autre robot. Pour obtenir cette réactivité, les événements sont créés et ajoutés au fil de la simulation.

1.1.6 Simulateur

Package poog54.io

Le simulateur initie la simulation grâce à un appel à la méthode restart() dans le constructeur. Il contient la gestion des évènements ainsi que la gestion de l'affichage graphique. Concernant ce dernier point, on a en mémoire une hash map, contenant des priority queues de drawables. Ainsi, pour une coordonnée donnée, les drawables sont automatiquement triés par leur priorité d'affichage graphique. Les fonctions d'affichage en sont grandement simplifié !

1.1.7 PathFinder

Package poog54.strategies

Il a été choisi d'implémenter l'algorithme Astar. Astar est un algorithme de recherche du chemin le plus court. Il est une amélioration de l'algorithme Dikjstra qui est, par exemple, utilisé pour le routage internet. Il se différencie l'un l'autre par l'ajouts d'estimateurs pour guider la recherche (Heuristic cost). Le coût heuristique qui a été choisi dans le projet est la distance, elle est garante du bon fonctionnement de l'algorithme. A* a surtout été choisi pour sa rapidité, il est un très bon compromis entre coût de calcul et optimalité de la solution. L'algorithme est calculé à partir d'une liste de cases de type algoTile qui contiennent les informations nécessaires à la résolution de l'algorithme.

1.1.8 Stratégie

Package poog54.strategies.masters

La stratégie permettant d'éteindre au plus vite tous les feux est portée par le chef pompier. Le chef pompier assigne un feu cible à chaque robot présent sur la carte, via la méthode orderRobot-ToFire(Robot, Simulator). Le robot emprunte le chemin le plus court pour se rendre sur les lieux, verse son eau et fait le plein (si nécessaire) au point d'eau le plus proche. Les robots connaissent la carte et se déplacent de manière autonome. Lorsqu'un feu est éteint, tous les robots qui lui étaient affectés demande un nouveau feu cible au chef pompier. Le cycle se répète alors jusqu'à l'extinction de tous les feux. Lorsqu'un robot a terminé de remplir son réservoir, il demande un feu cible au chef pompier.

La classe FiremanMaster (chef pompier) est abstraite. Elle met à disposition quelques algorithmes de recherche de feux cibles pour ses classes filles. Ces dernières implémentent différentes stratégies spécifiques et redéfinissent la méthode orderRobotToFire(Robot, Simulator) Elles sont nommées et identifiées par un grade : première classe, sergent, major... Plus le grade est élevé, plus la stratégie est évoluée. Seule la stratégie Colonel sera détaillée dans ce document, les stratégies de grade inférieur ayant surtout servi à la mise au point de la machine d'état. Tous les résultats sont cependant disponibles en annexe.

Un chef pompier a accès à toutes les données de la simulation (robots, feux, points d'eaux) pour pouvoir élaborer sa stratégie et affecter les feux. Le chef pompier de classe Colonel choisi les feux cibles de manière différente selon le type de robot :

- Robot à pattes : affecté au feu le plus proche de sa position courante. L'objectif est de minimiser ses déplacements. Le robot à pattes est très efficace une fois sur place, mais les temps longs de marche d'approche nuisent à son rendement.
- Robot à roues ou chenilles : affecté au feu le plus proche d'un point d'eau. Ces types de robots effectuent beaucoup d'aller-retours pour remplir leurs réservoirs. Cette distance est alors bien plus grande que celle parcourue lors de l'approche initiale. L'objectif est donc de minimiser la longueur des aller-retours, quelle que soit la distance d'approche à effectuer.
- Drone : affecté à un 'gros' feu, le plus éloigné des autres robots. L'objectif est de mettre à profit leur vitesse de déplacement élevée (sur tout terrain) pour atteindre les feux qui sont les plus inaccessibles pour les robots terrestres. Cependant, leur temps de remplissage étant excessivement longs, le chef pompier veille à affecter un feu dont l'intensité dépasse la capacité du réservoir du drone. S'il ne reste que des 'petits' feux, le drone est affecté au feu le plus isolé.

1.1.9 Le main

Package poog54.dataclasses.runnables

Il n'était d'aucune utilité d'avoir plusieurs classes possédant un main, puisque les différents tests peuvent êtres lancé grâce aux arguments passés au main. Le main a pour fonction de traiter ces arguments, d'instancier un GUI, puis un simulateur dans ce gui.

1.2 Tests effectués

Les tests ont été menés tout au long du projet. Leur nature dépendait de la phase de développement en cours.

1.2.1 Tests unitaires

Ces tests ont été réalisés sur des sous-ensembles de code, principalement au début du développement. Chaque développeur a créé ses propres scénarios de test pour valider les méthodes des classes dont il avait la charge. Les éléments suivants ont été testés selon cette méthode: lecture des données, affichage de l'interface, algorithme du plus court chemin.

1.2.2 Tests d'intégration

Quelques tests ont été réalisés au moment de regrouper les différents modules développés. Ils nous ont permis de s'assurer de la cohérence des interfaces, des liens entre les classes. Le simulateur à événements discrets et la machine d'état de chaque robot a été testé pendant cette phase.

1.2.3 Tests de performance

L'efficacité des différentes stratégies a été testé ici. Ces tests nécessitent que l'application globale soit fonctionnelle. Ils ont donc été réalisés en fin de projet.

L'ensemble de ces tests ne représente pas une campagne d'essais complète. Ils permettent cependant de valider le fonctionnement du simulateur dans les cas nominaux.

La stratégie Colonel donne de bon résultats si les robots présents sur la cartes sont de types différents. La stratégie reste la même quelque soit la composition de l'équipe de robots pompiers. Elle ne s'adapte donc pas si on imagine un scénario comprenant uniquement des robots à chenille. C'est une limite de cette implémentation.

1.3 Conclusion

Le simulateur fonctionne aujourd'hui comme spécifié dans le cahier des charges.

2. Annexes

Cartes \ Stratégies	1st classe	Sergeant	Captain	Major	Colonel
carteSujet.map	24368	23356	21607	18809	15209
desertOfDeath-20x20.map	11497	11336	11446	11518	11433
mushroomOfHell-20x20.map	9778	11976	9747	11225	8349
spiralOfMadness-50x50.map	32212	32101	31542	32008	31516

Table 2.1: Résultats de simulation (Temps en secondes)

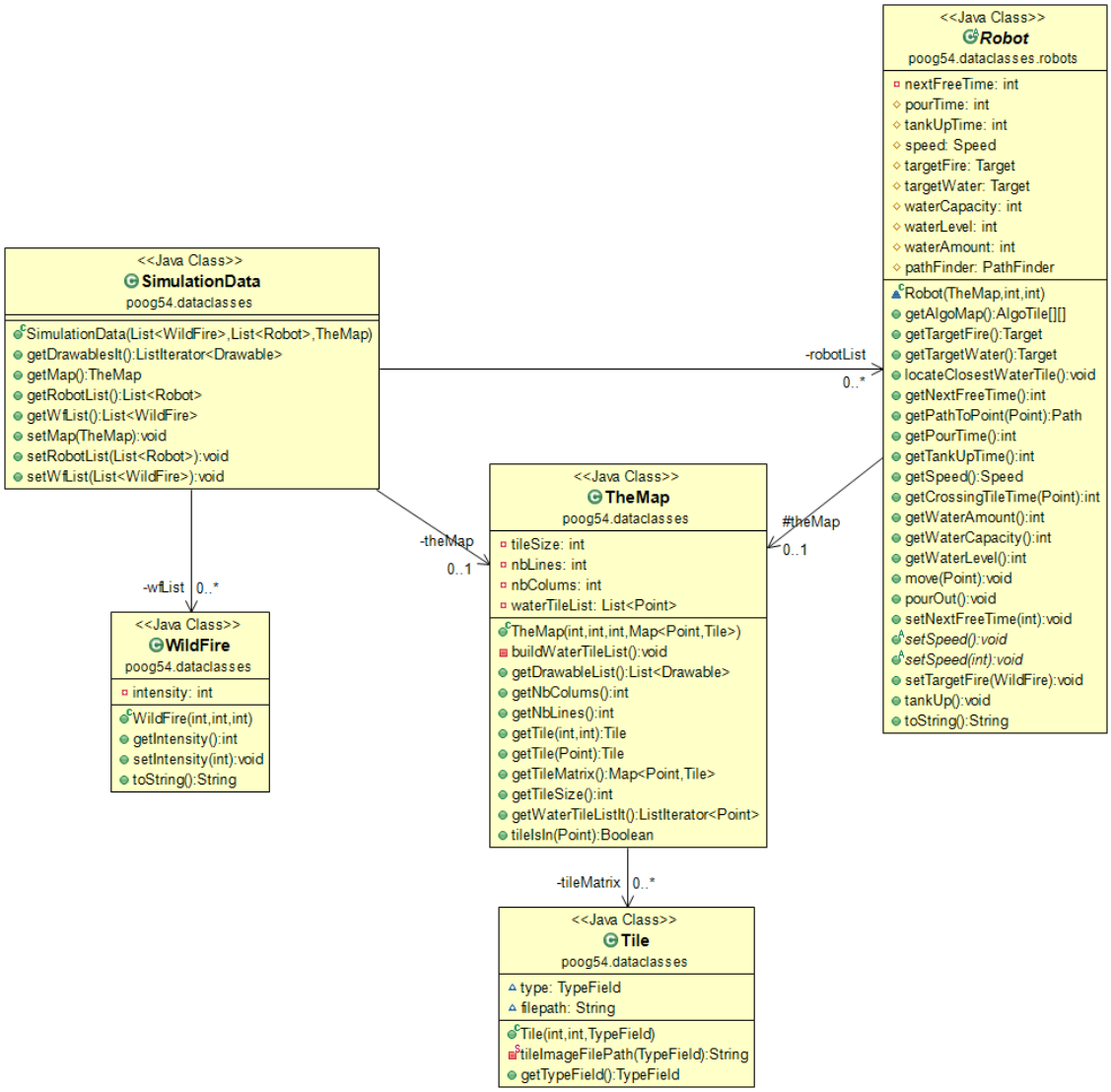


Figure 2.1: Données de simulation

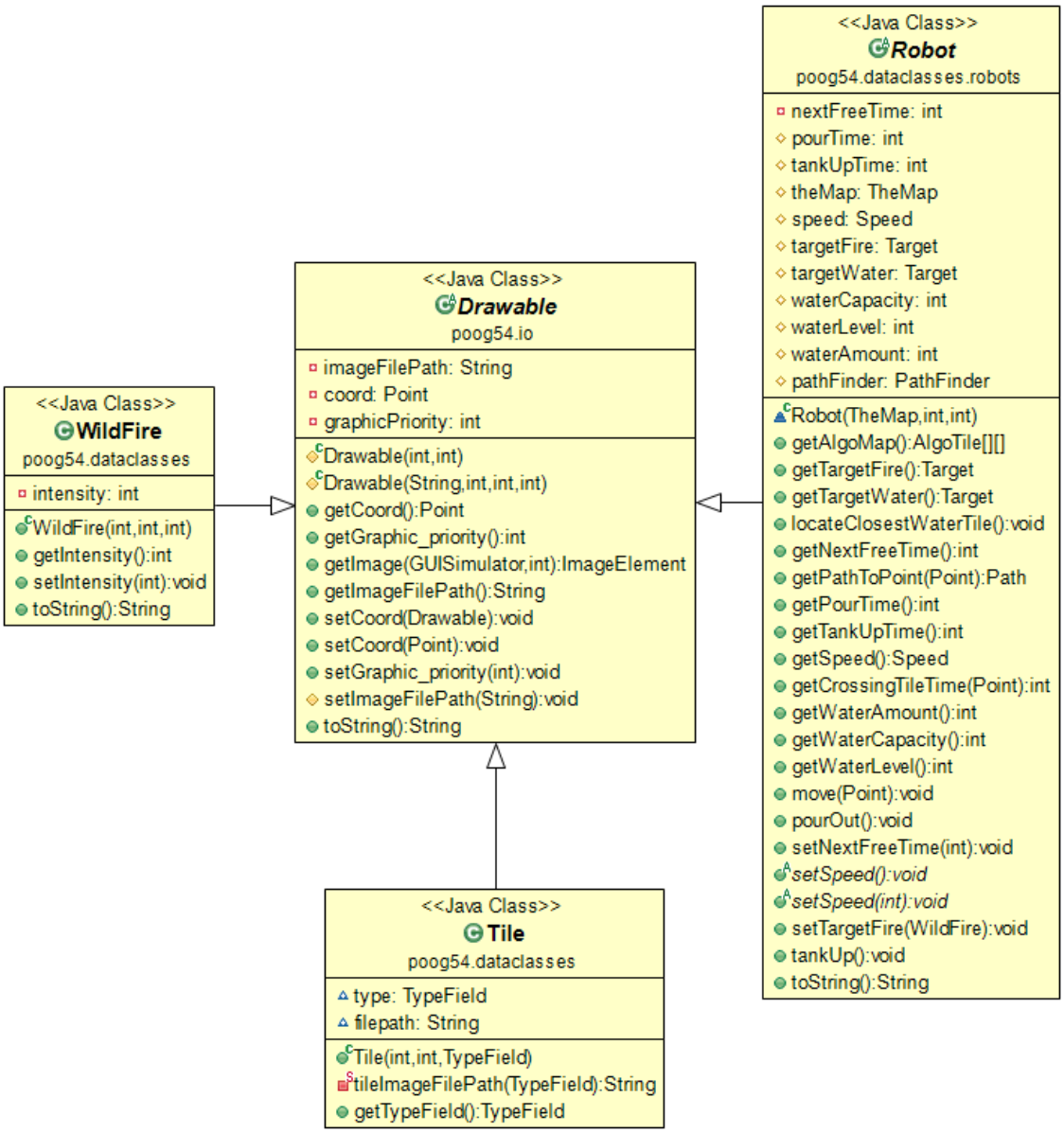


Figure 2.2: Objets affichables (Drawable

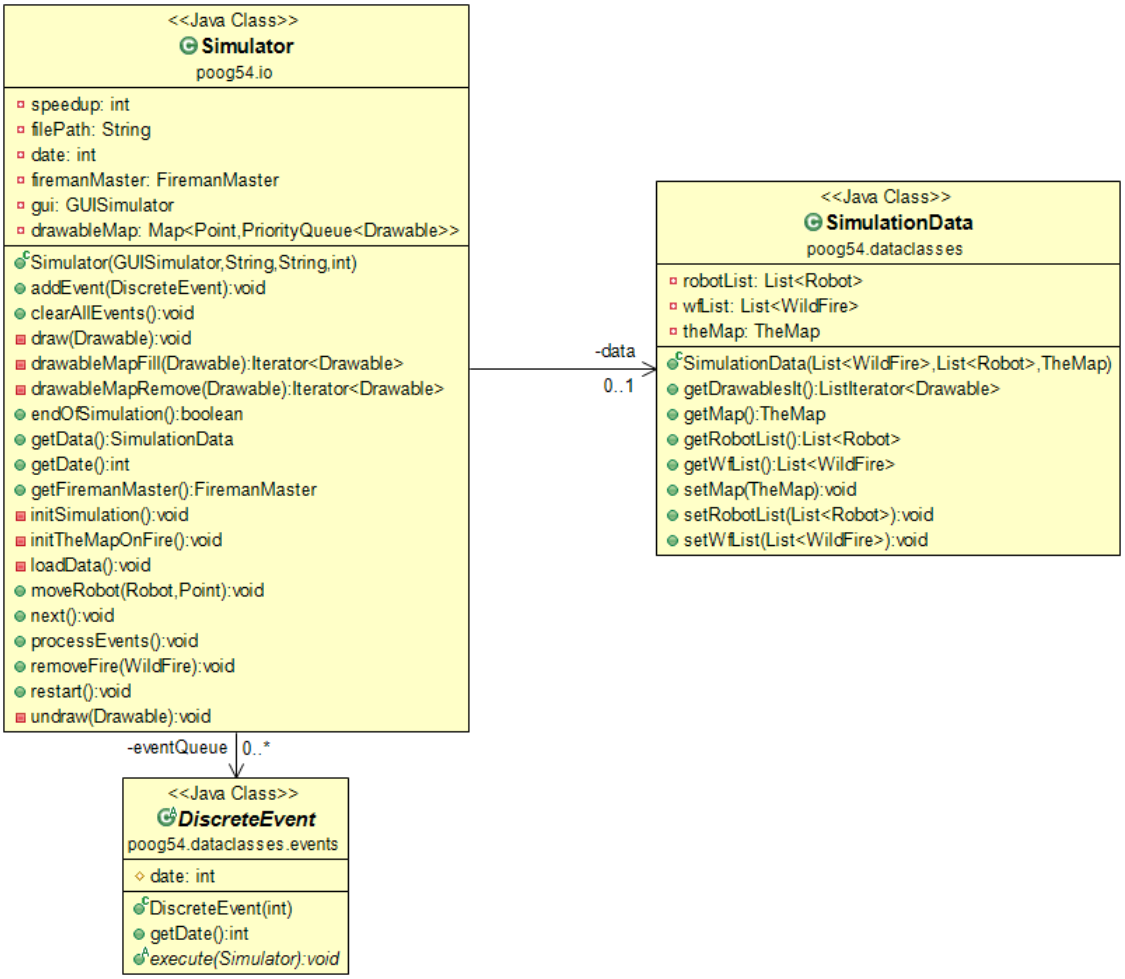


Figure 2.3: Simulateur à événements discrets

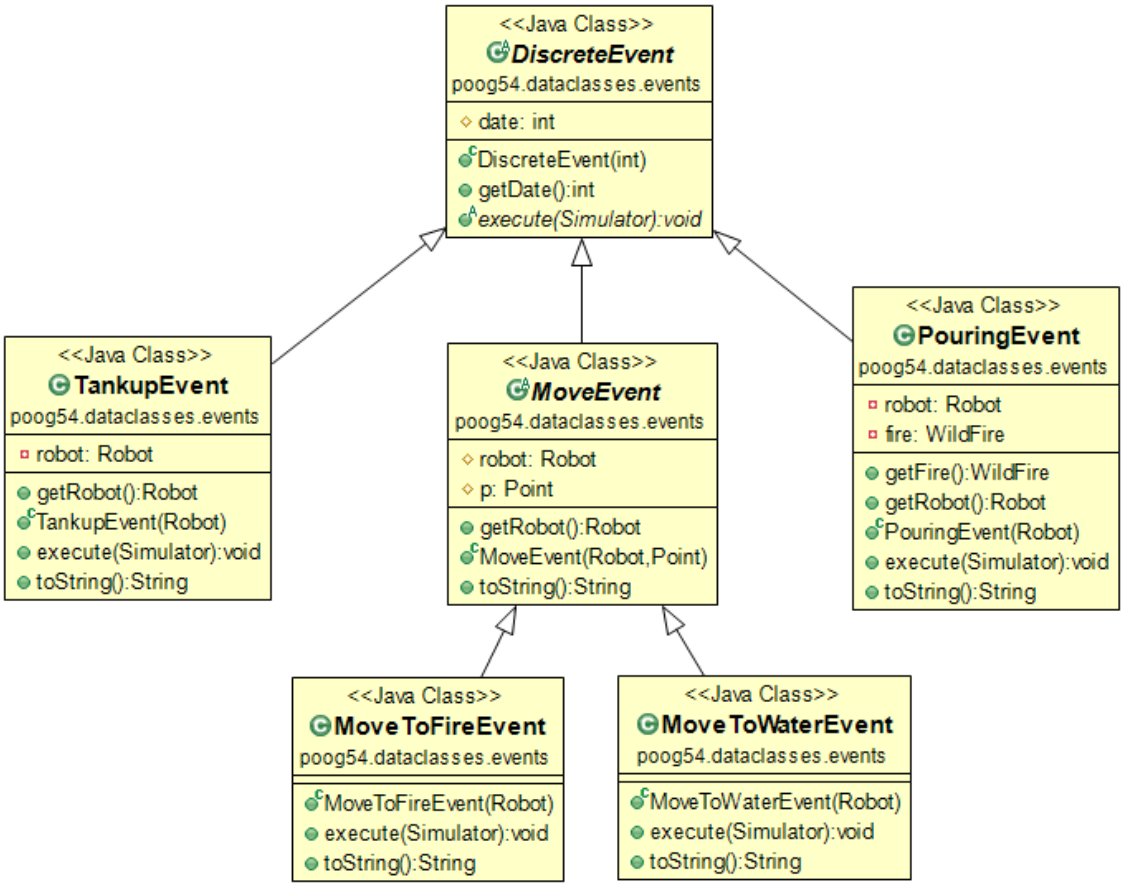


Figure 2.4: Evénements discrets

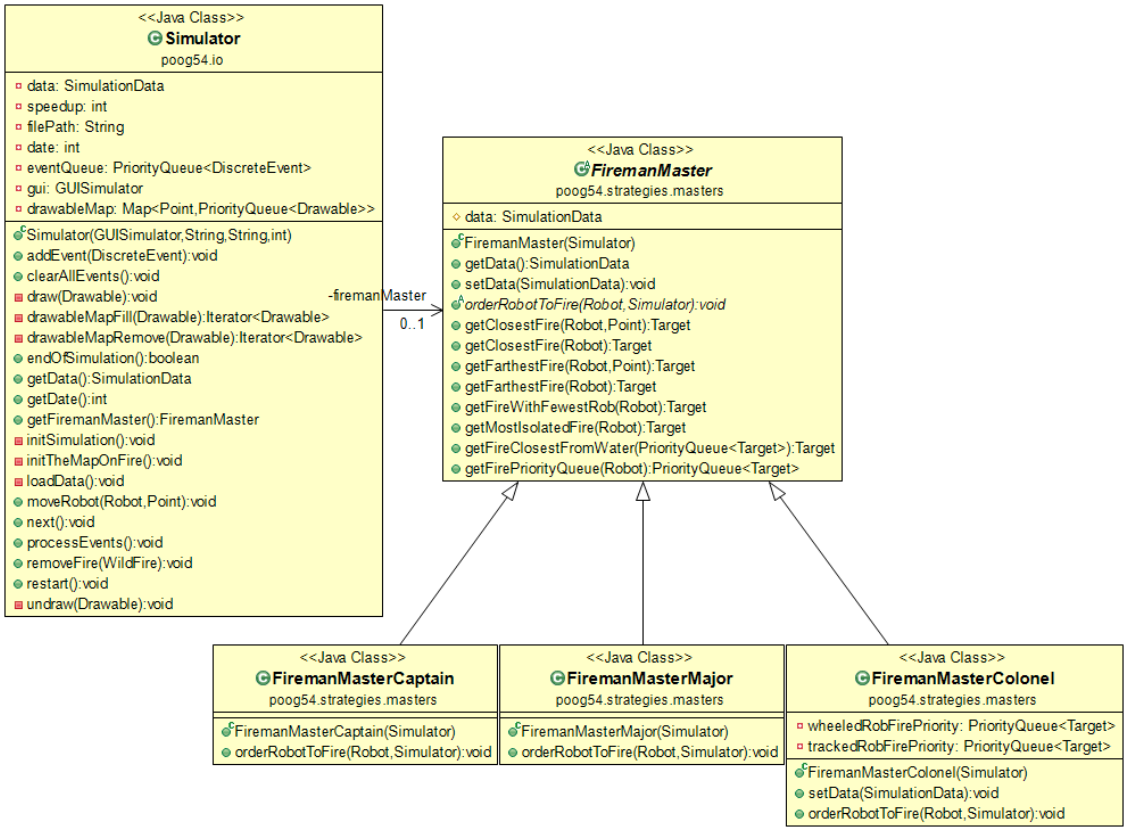


Figure 2.5: Stratégies du chef pompier