Projet de simulation d'une équipe de robots pompiers

Novembre 2016

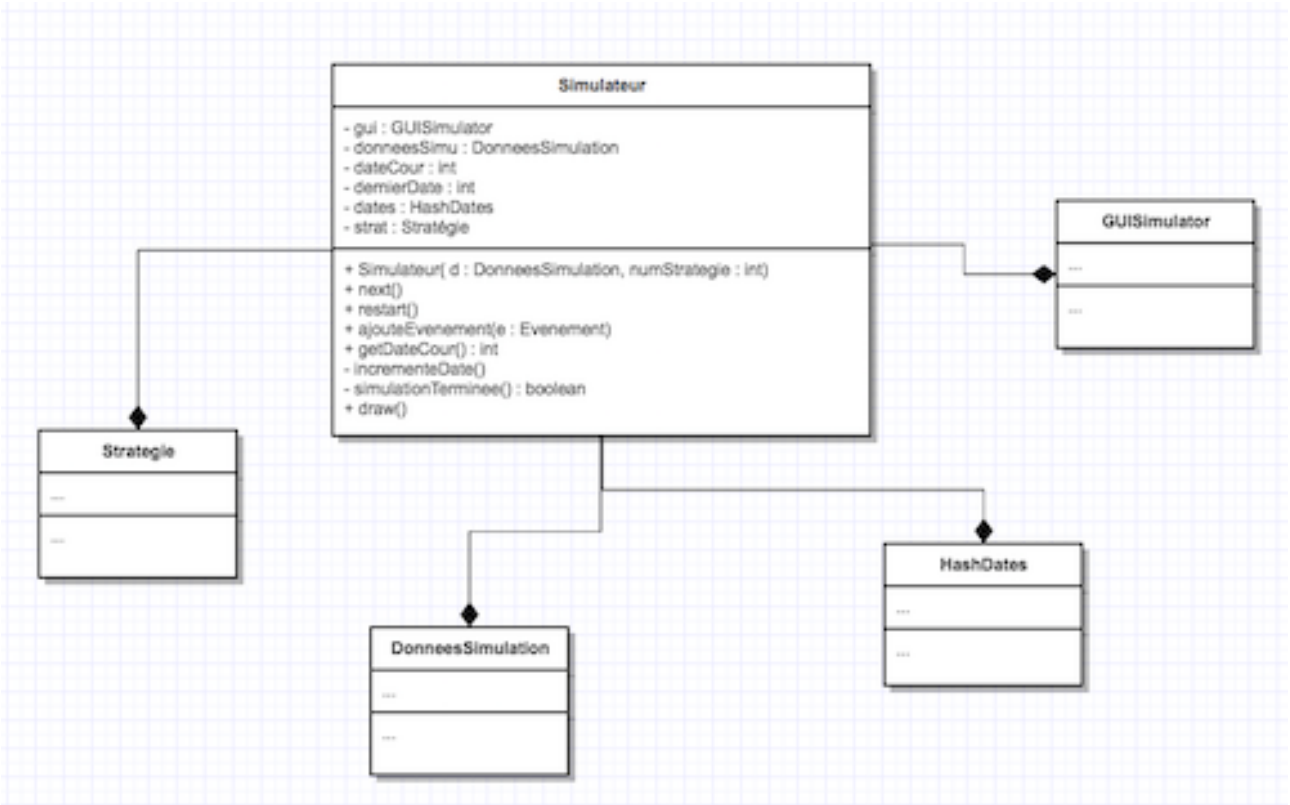
Dans le cadre du projet de simulation d'une éuipe de robots pompiers, nous avons rédigé ce rapport afin de proposer au lecteur une vue d'ensemble de la structure de notre code ainsi que des explications de certains aspects techniques.

Dans une première partie, nous proposons au lecteur plusieurs diagrammes UML permettant de donner la structure globale du code.

Dans une deuxième partie, nous proposons au lecteur un descriptif des deux stratégies utiliées pour résoudre le problème.

1 Architecture du projet

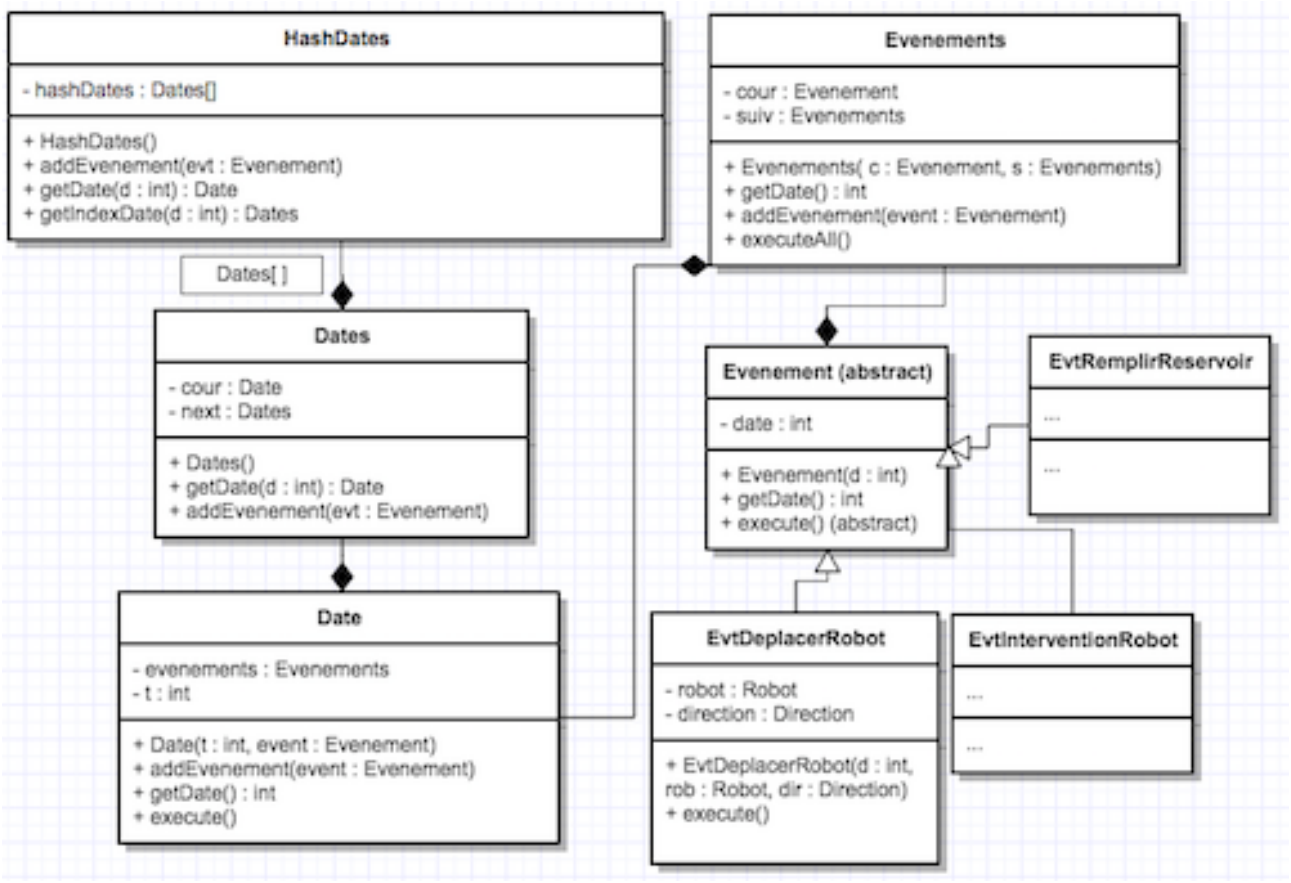
Dans cette première partie, vous trouverez les diagrammes UML permettant de comprendre la structure globale de notre rendu.



1.1 Description

Avec ce premier diagramme, nous remarquons que le code est structuré avec pour socle un objet Simulateur qui est le point d'entrée pour le lancement d'une simulation.

Ce simulateur contient un objet Strategie qui décrit la stratégie employée (dans notre projet : stratégie élémentaire ou stratégie évoluée), un objet DonneesSimulation qui regroupe l'ensemble des données propres au problème (carte, robots, incendies, etc) et un objet HashDates qui agit comme calendrier pour l'exécution des divers évènements.



1.2 Description

Ce deuxième diagramme permet de détailler la classe HashDates, le calendrier pour l'ensemble de nos évènements.

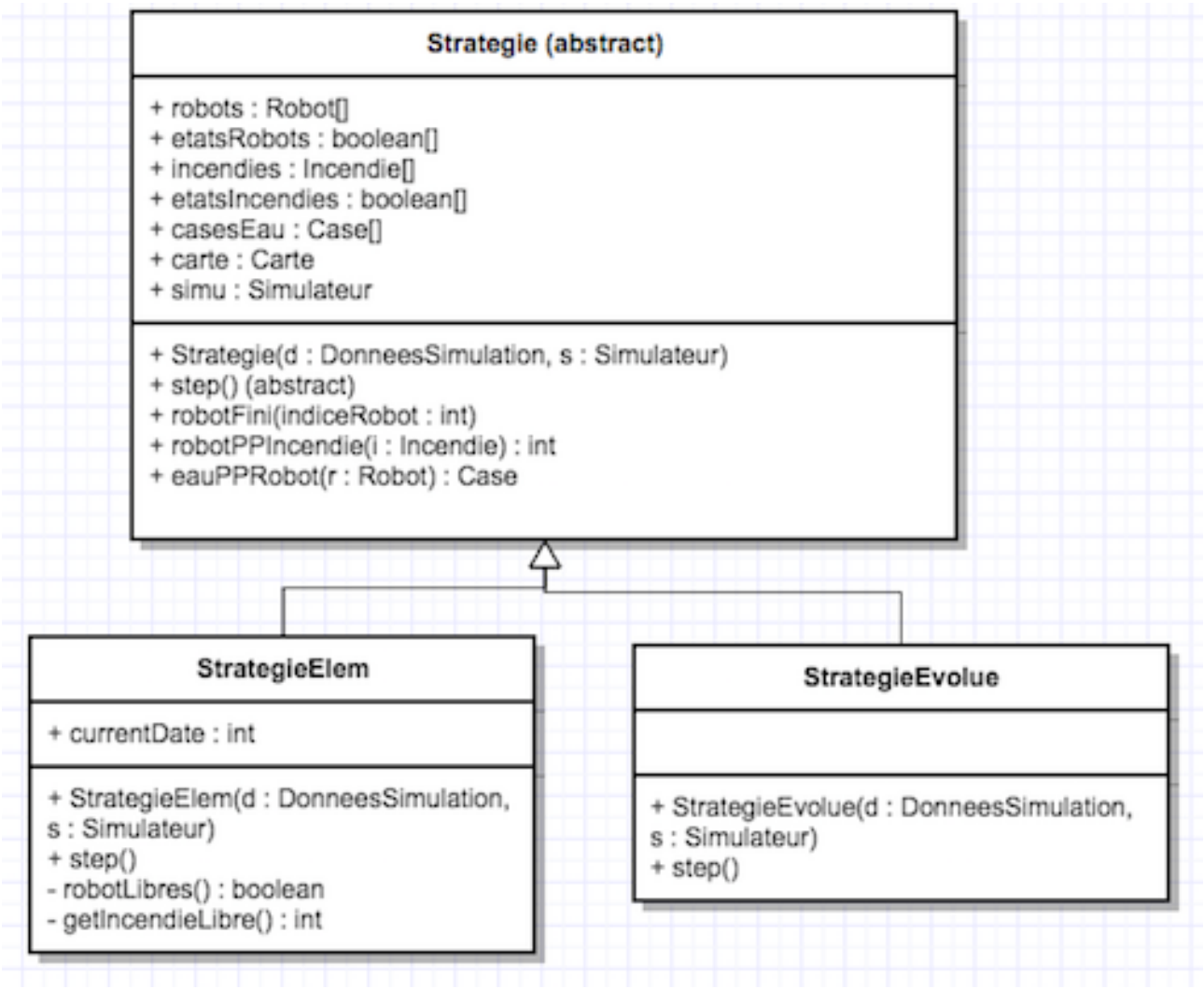
Pour créer ce calendrier, nous avons fait le choix d'utiliser une sorte d'array (de taille 100) de listes chainées contenant des objets Dates.

Dates contient lui-même l'élément de "base" Date ainsi qu'une référence au prochain objet Dates de la liste chainée.

L'objet de "base" Date contient alors une liste chainée d'évènements à être exécutés à la date (entier) attribuée à cet objet Date. Deux dates (entier) sont différentes si elles diffèrent d'une seconde ou plus.

L'array de liste chainée de Dates est en fait une table de Hachâge, une clé étant la date des objets Date modulo l'indice de l'array.

A titre d'exemple, les objets Date de date 0, 100, 200, etc seront situés dans la liste chainée à l'indice 0 de hashDates.



1.3 Description

Pour l'implémentation des stratégies, nous sommes partis d'une classe abstraite contenant un ensemble de données utiles pour la logique d'une stratégie spécifique : élémentaire ou évoluée.

L'idée principale est la suivante : afin d'avancer dans une simulation, l'objet simulateur possède une méthode next().

Cette méthode next() fait appel à une méthode step(), abstraite dans la classe Stratégie et donc implémentée dans StrategieElem ainsi que dans StrategieEvolue, qui permet de "calculer" les évènements pour chacun des robots en fonction des données du problème. Ces évenements sont ajoutés au "calendrier" hashDates qui seront, le moment venu, exécutés par le simulateur.

1.4 Lancer une simulation

Afin de lancer une simulation, vous pouvez utiliser le Makefile de la manière suivante :

make nom-de-stratégie CARTE=nom-de-carte

A titre d'exemple :

make strategieElem CARTE=desertOfDeath-20x20

2 Les stratégies

2.1 Stratégie Elémentaire

La stratégie élementaire fonctionne de la manière suivante :

A chaque pas d'exécution, nous regardons tous les robots, libres ou non:

- si un robot a besoin d'eau, on l'envoit vers la case d'eau la plus proche et le robot ne répond à aucune demande tant que son réservoir n'est pas rempli.
- si un robot a un réservoir plein mais n'a pas d'incendie attribué, on lui attribue un incendie et on lui demande de s'y rendre pour l'éteindre.
- Les autres robots n'étant dans aucun de ces deux cas ont des incendies qui leurs sont attribués

2.2 Stratégie Evoluée

La stratégie évoluée fonctionne de la manière suivante :

Tout d'abord on parcourt l'ensemble des robots "libres", et pour chaque robot "libre" on regarde quel est l'incendie le plus proche du robot. On programme alors une suite d'évènements traduisant le déplacement (le plus court) du robot vers cet incendie.

Ensuite, à chaque pas d'exécution, on parcourt tous les robots qui sont dans un état "occupé".

- on regarde alors en premier lieu si le réservoir de ce robot est vide. Si c'est le cas, on programme une suite d'évènements permettant à ce robot de se diriger vers le point d'eau le plus proche, si ceci n'a pas été encore fait auparavant. Si jamais le robot est arrivé à "destination" alors il remplit son réservoir.

- Sinon, si un robot dit "occupé" a encore de l'eau c'est soit qu'il est en cours de déplacement vers l'incendie qui lui est attribué auquel cas on ne fait rien, soit il a atteint cet incendie. Il regarde alors si l'incendie n'a pas été éteint entre temps. Si c'est le cas, le robot est déclaré "libre" et se voit attribuer un autre incendie, sinon il intervient sur cet incendie, en versant une certaine quantité d'eau à chaque "tour" jusqu'à ce que soit l'incendie soit éteint ou soit que son réservoir soit vide. A noter que plusieurs robots peuvent intervenir en même temps sur le même incendie.

Enfin, on regarde si l'intervention du robot a permis de venir à bout de l'incendie. Dans ce cas là, le robot est déclaré "libre" et il peut alors intervenir sur un autre incendie si besoin. Sinon, c'est que le robot n'a plus d'eau alors le prochain tour permettra au robot de programmer les évènements pour remplir son réservoir.