Simulation d'une équipe de robots pompiers

Anas Mejgari, Youssef Saad, Mohamed Aziz Zeroual.

Novembre 2020

1 Conception et objectifs

Le dessein de ce TP est de réaliser une application en Java qui met en évidence une équipe de robots pompiers évoluant sur une carte de manière autonome, et qui ont la possibilité de se déplacer sur les cases d'une éventuelle carte pour accomplir plusieurs tâches. Au fûr et à mesure de notre travail, il fallait passer par une multitude d'étapes; respecter des contraintes de déplacements et de la nature du terrain, des contraintes de la nature du robot et de sa capacité de réservoir, envisager le recours au plus court chemin pour pouvoir éteindre rapidement l'incendie, puis organiser les interventions des différents robots présents sur une carte pour pouvoir éteindre les incendies d'une façon fructueuse et rapide.
Pour garantir un travail complet, il fallait articuler notre conception, toute en gardant une certaine gradation au cours de la réalisation. Le travail était donc subdivisé en 4 majeures parties:
- Réussir à implémenter les classes mises en jeu, soit les classes Carte, case et la classe abstraite Robot et les classes qui en hérite (Drone, RobRoue, RobPattes et RobChenille), ainsi de coder toutes les méthode relatives à ces classes. A cette étape, comme on va le voir en détail dans la partie suivante, on était capable de générer l'interface graphique fournie, ainsi que les robots et les incendies. C'est une étape de visualisation de situation.
- Construire un géstionnaire d'évenements et l'ajouter au simulateur, puis créer des évements élementaires et des scénarios qui seront exécutés. Á ce stade, ce n'est plus une phase de visualition, mais les robots peuvent se déplacer dans les cases voisines.
-Mettre en exergue les classes susceptibles de réaliser le calcul du plus court chemin en utilisant l'algorithme de Ford, en essayant de l'adapter au évenement de déplacements du simulateur. Il s'agit dans cette étape d'optimiser les déplacements élementaires du robot. Il atteint donc la case souhaitée en ayant emprunter le chemin le plus court.
-Créer un ensemble de classes permettant chacune de modéliser un chef pompier de stratégie dissemblantes. Il s'agit dans un premier temps d'une stratégie élementaire simple sur la répartition des robots, puis d'une stratégie plus évoluée dans un second temps, tout en assurant la communication entre les robots d'une même équipe.

1.1 Première partie : les données du problème

Nous avons commencé par coder les classes Case repéré par sa ligne, sa colonne, et la nature du terrain qu'elle occupe. et Carte qui contient une matrice de case ainsi que toute les méthodes nécessaires pour la réalisation des prochaines étapes, ensuite nous avons programmé la classe Incendie, en essayant d'implémenter des fonctions pour la lecture et la gestion de l'eau pour éteindre une éventuelle incendie. Puis la classe Robot qui est la majeure partie de cette première approche, nous avons implementé les actions pouvant être accompli par un robot donnée, puis des classes relatives à chaque type de robots; drône,robot à roues, à chenilles ou à pattes.
nous avons commencé par une implémentation des attributs et méthodes donnés en énoncé, ainsi que

tous les accesseurs et mutatteurs nécessaires. De plus, nous avons implémenté la classe *DonneesSimulation* qui contient les différents éléments de la simulation (Cartes, Robots et Incendie). Et, nous avons fini par introduire certaines modifications sur les méthodes de la classe par *LecteurDonnees* en faisant appel aux méthodes de la classe DonneesSimulation, afin qu'on puisse lire et stocker les élements de la simulation au même temps. Le résultat de l'interface graphique à ce stade est :

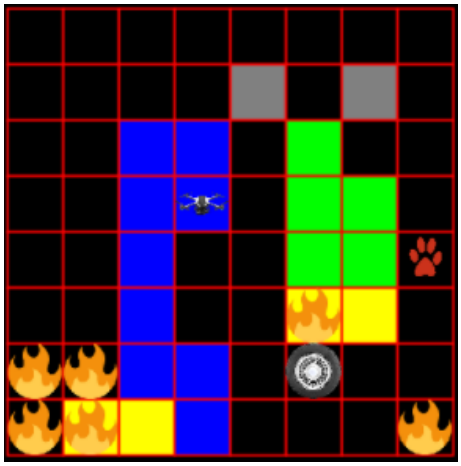

Figure 1: Exemple de simulation de l'interface graphique.

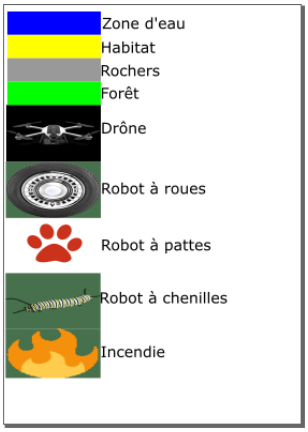

Figure 2 : Légende de la carte.

Á ce stade, on affiche l'interface graphique qui contient tous les données du sujet, représentés par des fichiers de type png.

1.2 Deuxième partie :Les événements

Nous avons crée une classe abstraite *Evenement* qui contient des méthodes abstraites comme **execute** qui s'occupe de l'exécution d'un evenement donée et **tempsEvenement** qui calcule le temps de son exécution, puis nous avons implémenté d'autres classes par héritage, elle représente les éventuels évenements dont on cite: le déplacement élementaire(le déplacement d'une case) par la classe *Deplacement*, le déplacement multiple (de plusieurs cases) par la classe *DeplacementMultiple*, l'action de verser l'eau sur une incendie par un robot à travers la classe *Verser*, et le remplissage du réservoir à travers la classe *Remplir*.

1.3 Troisième partie : Calculs de plus courts chemins

Dans la perspective d'optimiser le déplacement du robot dans les différentes cases de la carte pour éteindre une incendie le plus vite possible, nous avons codé la classe *listCourtChemin* du package courtChemin afin d'exhiber un passage par le plus court chemin, en utilisant l'algorithme de Ford et son implémentation en Java. Cet algorithme se base sur l'utilisation de la matrice d'adjacence d'un graphe seulement, sans avoir recours à la création des classes spécifiques pour manipuler des graphes, il est de complexité $O(n^2)$, où n est le nombre de noeuds d'un graphe. Pour une bonne implémentation de cet algorithme, on codé pour chaque type de robot une méthode *setMatriceAdjacence* qui récupére les données de la carte et le type du robot afin de créer la matrice d'adjacence du graphe de déplacement du robot dans le graphe (les noeuds sont les cases et les arcs contiennent le temps du déplacement du robot d'une case à une autre (avec une modélisation d'un déplacement impossible) par un coût infini. Pour tester l'algorithme du plus court chemin, nous avons pris un exemple de graphe du cours de la recherche opérationnelle, et nous l'avons appliqué dans la méthode Main de la classe *pluscourtchemin*.

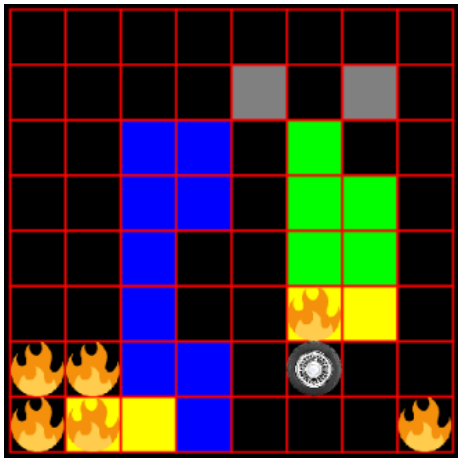

Figure 3 : Position initiale du robot.

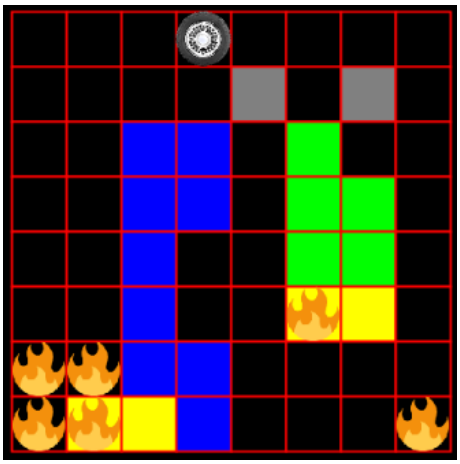

Figure 4 : Position du robot au cours de la simulation.

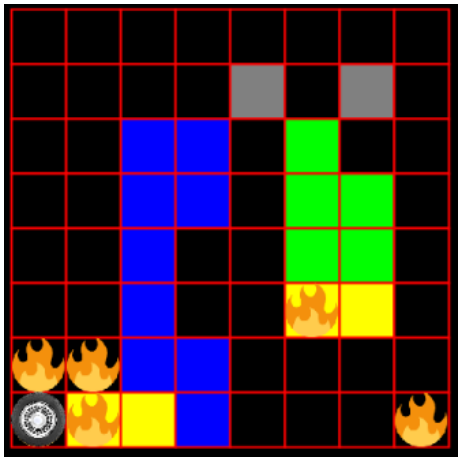

Figure 5 : Position finale du robot.

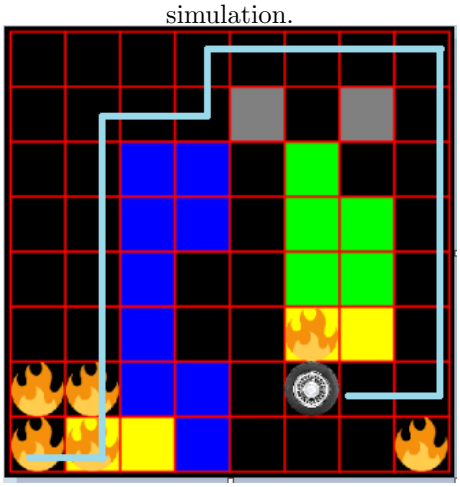

Figure 6 : Le plus court chemin emprunté par le robot d'après l'algorithme de Ford.

1.4 Quatrième partie : Résolution du problème

Cette partie consiste à proposer une stratégie d'affectation de tâches, d'organiser les déplacement et les interventions des robots pour éteindre toutes les incendies d'une carte donnée, dans un temps minimal. Nous avons implémenté une classe abstraite *ChefPompier* contenant des méthodes abstraites, comme stratégie qui détermine la stratégie à adopter, et SimuTerminee qui vérifie la fin d'une éventuelle simulation. Nous avons opté par la suite à deux stratégie, une première élementaire simple qui consiste á envoyer les robots - sous les ordres d'un chef-pompier- pour intervenir et éteindre les incendies, sans recherche d'optimisation du processus d'évolution. Puis une seconde stratégie plus avancée, voire plus sophistiquée au niveau des interventions et des décisions prises par le chef-pompier, dans laquelle on choisit le robot le plus proche à une incendie afin de dimunier l'intensité de cette dernière, voire l'éteindre.

1.5 Dernière partie : le programme principal

Le programme principal *testRobot* consiste à :
-Demander à l'utilisateur d'entrer 1 pour une simulation à stratégie élementaire, et 2 pour une autre évoluée, on s'assure que l'entier entrée appartient á la paire 1,2, sinon on redemande à l'utilisateur de renouveler l'opération.
-Lecture des données, et stockage d'un objet de type *DonneesSimulation* accessible par la méthode statique *getDonnees* de la classe *lecteurDonnees*.
-Simulation et affichage de l'interface.
-Retour du message simulationterminnee à la fin.

2 Conclusion

En ce qui concerne la répartition des tâches et la méthodologie de travail, nous avons opté pour un stratagème d'évolution collective; nous avons essayé d'avoir la même cadence au sein de l'équipe, en tentant de progresser d'une maniére simultanée et assurée. Nous avons évolué en masse, c'est à dire que nous avons privilégié la résolution complète de chaque partie.
Il existe d'autres méthodes plus évoluées, et propres à des cas particuliers, par exemple dans le cas où le nombre d'incendies est inférieur à celui des robots, dans cette situation, on va parcourir les robots tout en cherchant l'incendie la plus proche.