Conception de notre projet

Organisation des fichiers

Nous avons séparé l'ensemble des fichiers en différentes catégories dans le dossier src :

- enumerator : contenant les enum Direction (NORTH, SOUTH, EAST, WEST) et TypeLand (WATER, FOREST, STONE, FIELD, HABITATION).
- fire : contenant uniquement Fire, un fichier créant les feux et assurant la gestion de ceux-ci, notamment la diminution et la propagation.
- gui : contenant l'ensemble des fichiers pour le bon fonctionnement de l'interface graphique.
- io : contenant la gestion des données de lecture et de dessin pour l'interface graphique.
- map : contenant la définition de la carte (ensemble de Box), l'état actuel de la carte ainsi que la recherche des plus courts chemins avec AStar.
- robot : contenant tous les types de robots ainsi que le robot principal CaptainRobot
- simulation : contenant toute la gestion du lancement de la simulation ainsi que l'avancement des évènements durant celle-ci.
- fichier main pour lancer la simulation et la terminer.

Extension de la classe Feu

Une extension de propagation de feu a été faite :

Bouton d'activation/désactivation :

- Un bouton intitulé "Propagation : OFF" est ajouté à l'interface graphique, permettant de contrôler la propagation du feu.
- Lors de chaque clic, le bouton bascule entre les états "ON" et "OFF", activant ou désactivant la propagation du feu.
- L'état de la propagation est stocké dans une variable booléenne `firePropagationEnabled`, initialisée à `false`.

Conditions de propagation :

- Lorsqu'un incendie est actif et que la propagation est activée (`firePropagationEnabled` est à `true`), chaque feu a la possibilité de se propager vers une case adjacente, respectant certaines règles.
- La propagation est contrôlée par le simulateur via une condition de temps qui déclenche un événement de propagation à intervalles réguliers. Ces intervalles sont calculés dynamiquement en fonction de la taille de chaque case, afin de simuler la diffusion du feu dans un environnement de taille variable.
- Seules les cases adjacentes sans eau et sans robot peuvent être touchées par la propagation.

Cycle de vie des événements de propagation :

- Dans chaque cycle de simulation, le simulateur vérifie si le mode de propagation est activé.
- Lorsque l'intervalle de temps pour la propagation est atteint, le simulateur crée un nouvel événement de propagation `FirePropa` pour chaque incendie actif. Cet événement déclenche la propagation vers une direction choisie aléatoirement parmi les cases adjacentes.
- Si la propagation est désactivée, les événements `FirePropa` ne sont pas créés, et les incendies restent statiques.

Gestion de l'interface utilisateur :

- Le bouton de contrôle de la propagation se comporte de manière interactive : l'affichage du texte bascule entre "Propagation : ON" et "Propagation : OFF" en fonction de l'état actuel.
- Ce bouton permet de démarrer ou d'interrompre la diffusion du feu sans devoir redémarrer la simulation, offrant ainsi une flexibilité à l'utilisateur.

Règles de Propagation

- **Direction** : Le feu se propage uniquement dans une direction aléatoire (Nord, Sud, Est ou Ouest) à chaque cycle d'événement, simulant ainsi une diffusion progressive.
- **Vérifications avant propagation** : Avant de se propager, chaque incendie vérifie que la case adjacente est valide :
  - Elle ne doit pas contenir d'eau (pas de type `WATER`).
  - Elle ne doit pas être occupée par un robot terrestre.
  - Elle ne doit pas déjà être en feu.
- **Réduction de l'intensité** : Lorsqu'un incendie se propage, l'intensité du feu est réduite dans la nouvelle case, simulant un affaiblissement naturel de la propagation du feu.

Gestion des erreurs et tests

Nous avons, tout au long du projet, ajouté des tests afin d'éviter tout problème :

- Vérification que l'on reste dans la carte en aval d'un déplacement (lance une erreur si un robot sort de la carte).
- Vérification des fichiers donnés en argument lors du lancement de la simulation (lance une erreur si les arguments sont incorrects).
- Vérification des terrains lors des déplacements des robots (certains ne peuvent pas traverser des terrains spécifiques).
- Vérification de la décroissance de l'intensité des feux au fil du temps.
- Vérification de la propagation du feu lorsque celle-ci est activée.
- Vérification des chemins empruntés par les robots (si ceux-ci sont bien optimaux)

Optimisations

Nous avons voulu optimiser certaines parties de notre code afin d'alléger le nombre de calculs lors de la simulation :

- Lorsqu'un robot n'a aucun chemin vers le moindre feu (il est donc inutile), nous le laissons de côté et ne faisons plus de calcul sur celui-ci.
- Nous avons préféré implémenter A Star pour calculer les plus courts chemins au lieu de Dijkstra pour gagner en complexité temporelle; notre heuristique H est la suivante:

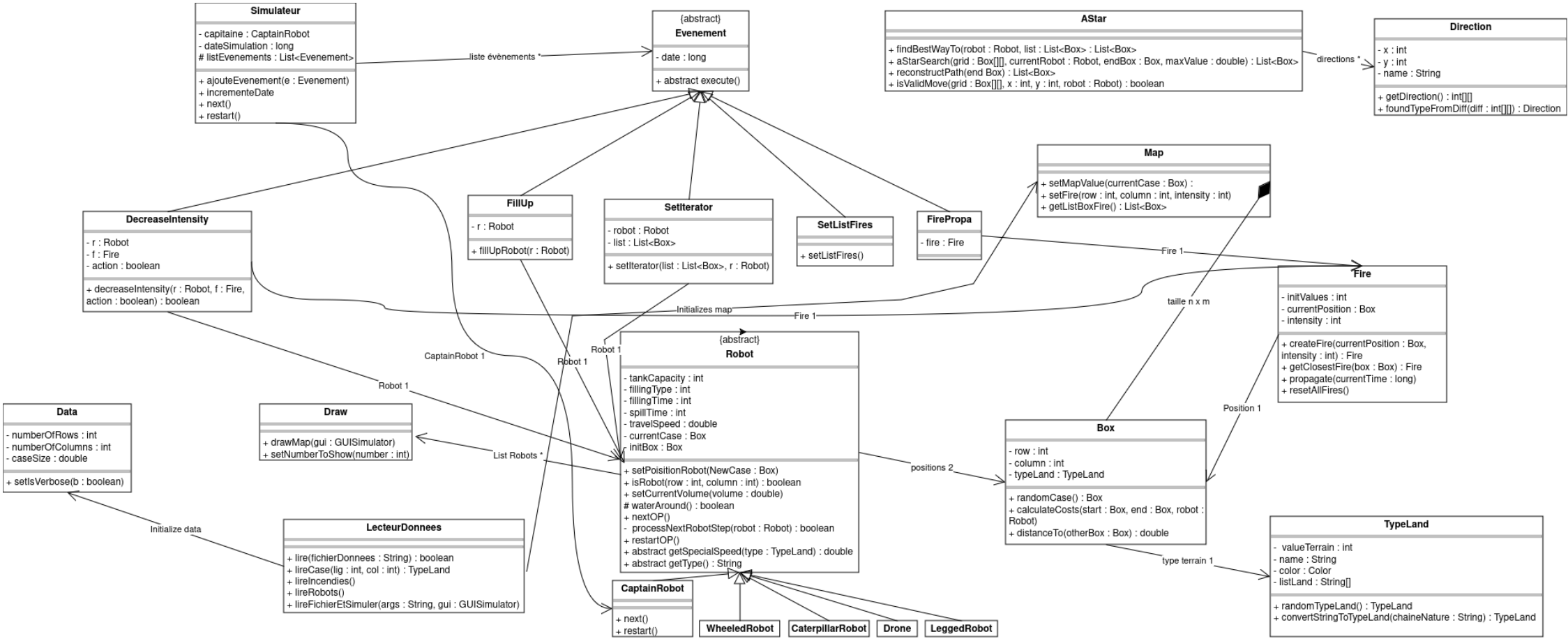
  $H_r(x,y) = \frac{|abs(r) - x| + |ord(r) - y|}{v(r)}$

  avec (x, y) les coordonnées de la case visée, $abs(r)$ et $ord(r)$ respectivement l'abscisse actuelle du robot r et l'ordonnée actuelle de r, et $v(r)$ la vitesse actuelle de r. Nous avons utilisé cette formule car nous avions besoin d'une valeur homogène à un temps.
- Les robots font un nouveau calcul du feu le plus proche lorsque l'un est éteint pour éviter tout trajet inutile si un autre robot se dirige vers ce feu.
- Nous avons volontairement enlevé les déplacements des robots du type Événement puisqu'ils ajoutent trop de calculs lors de la simulation.

Ouvertures

Nous avions quelques idées pour optimiser le code et réduire le temps de calcul :

- Garder en mémoire le trajet du robot entre le feu et l'eau si le feu n'est toujours pas éteint afin de ne pas recalculer ce même trajet plusieurs fois de suite.
- Trouver le point d'eau le plus proche du nouveau feu ciblé lorsqu'un robot a éteint un feu et doit se recharger, au lieu de chercher le point d'eau le plus proche de sa propre position.
- Nous avions aussi l'idée de rajouter à l'extension PropraFire un choix de difficulté avec un nouveau bouton sur l'interface, permettant de choisir la difficulté/vitesse de propagation des feux.


Simulateur
- capitaine : CaptainRobot
- dateSimulation : long
listEvenements : List<Evenement>
+ ajouteEvenement(e : Evenement)
+ incrementeDate
+ next()
+ restart()
liste évènements *
{abstract}
Evenement
- date : long
+ abstract execute()
AStar
+ findBestWayTo(robot : Robot, list : List<Box> : List<Box>
+ aStarSearch(grid : Box[][], currentRobot : Robot, endBox : Box, maxValue : double) : List<Box>
+ reconstructPath(end Box) : List<Box>
+ isValidMove(grid : Box[][], x : int, y : int, robot : Robot) : boolean
directions *
Direction
- x : int
- y : int
- name : String
+ getDirection() : int[][]
+ foundTypeFromDiff(diff : int[][]) : Direction
Map
+ setMapValue(currentCase : Box) :
+ setFire(row : int, column : int, intensity : int)
+ getListBoxFire() : List<Box>
DecreaseIntensity
- r : Robot
- f : Fire
- action : boolean
+ decreaseIntensity(r : Robot, f : Fire, action : boolean) : boolean
FillUp
- r : Robot
+ fillUpRobot(r : Robot)
SetIterator
- robot : Robot
- list : List<Box>
+ setIterator(list : List<Box>, r : Robot)
SetListFires
+ setListFires()
FirePropa
- fire : Fire
Fire 1
Fire
- initValues : int
- currentPosition : Box
- intensity : int
+ createFire(currentPosition : Box, intensity : int) : Fire
+ getClosestFire(box : Box) : Fire
+ propagate(currentTime : long)
+ resetAllFires()
Initializes map
Fire 1
taille n x m
Position 1
CaptainRobot 1
Robot 1
Robot 1
Robot 1
{abstract}
Robot
- tankCapacity : int
- fillingType : int
- fillingTime : int
- spillTime : int
- travelSpeed : double
- currentCase : Box
- initBox : Box
+ setPoisitionRobot(NewCase : Box)
+ isRobot(row : int, column : int) : boolean
+ setCurrentVolume(volume : double)
# waterAround() : boolean
+ nextOP()
- processNextRobotStep(robot : Robot) : boolean
+ restartOP()
+ abstract getSpecialSpeed(type : TypeLand) : double
+ abstract getType() : String
Data
- numberOfRows : int
- numberOfColumns : int
- caseSize : double
+ setIsVerbose(b : boolean)
Draw
+ drawMap(gui : GUISimulator)
+ setNumberToShow(number : int)
List Robots *
positions 2
Box
- row : int
- column : int
- typeLand : TypeLand
+ randomCase() : Box
+ calculateCosts(start : Box, end : Box, robot : Robot)
+ distanceTo(otherBox : Box) : double
Initialize data
LecteurDonnees
+ lire(fichierDonnees : String) : boolean
+ lireCase(lig : int, col : int) : TypeLand
+ lireIncendies()
+ lireRobots()
+ lireFichierEtSimuler(args : String, gui : GUISimulator)
CaptainRobot
+ next()
+ restart()
WheeledRobot
CaterpillarRobot
Drone
LeggedRobot
type terrain 1
TypeLand
- valueTerrain : int
- name : String
- color : Color
- listLand : String[]
+ randomTypeLand() : TypeLand
+ convertStringToTypeLand(chaineNature : String) : TypeLand