Projet NFA032: Les robots

2016

1 Présentation du sujet

On veut dans ce programme simuler le comportement de robots *programmables* par l'utilisateur dans un environnement minier. Il s'agit de réaliser un jeu où le joueur écrira un programme P destiné à être exécuté par un robot, dans un langage dont nous allons donner les grandes lignes. Le logiciel que vous devez écrire aura en charge de simuler le comportement du robot pendant qu'il exécute le programme P.

Le monde dans lequel le robot se déplace est un damier, où le robot peut se déplacer d'une case à chaque tour, horizontalement ou verticalement. Les cases peuvent être vide, contenir de la roche (auquel cas le déplacement sera impossible), contenir du minerais (ce que cherche le robot), ou contenir de l'eau (si le robot pénètre dans une telle case, il coule).

Le terrain de jeu sera visualisé comme un damier :

```
****************************************
*       $           ***                *
**                                       *
* *              A                       *
*         #                 **           *
* $  **                     #            *
*                                        *
*         # #                            *
*                       $                *
*             $                    ** *
****************************************
```

légende :

$ un minerai ;

***** de la roche ;

de l'eau ;

A le robot.

Le programme et le terrain seront écrits par l'utilisateur et sauvegardé dans des fichiers textes. Supposons que votre logiciel se nomme `JeuRobots`, on lancera le système en tapant :

```
java JeuRobots terrain.txt fichierProgrammeRobot.txt
```

où `terrain.txt` est un fichier texte qui contient une représentation du terrain initial, et `fichierProgrammeRobot.txt` contient les lignes du programme P que le robot doit exécuter.

2 Affichage du terrain : l'interface AfficheRobot

Le projet utilise l'interface java suivante. Nous vous fournissons une implémentation de cette interface au moyen d'une fenêtre graphique qui affiche des images. Vous devez faire un projet qui marche avec cette classe mais vous devez aussi écrire une autre implémentation de AfficheRobot qui affiche le terrain dans un terminal, sous forme de caractères. Autrement dit, votre programme marchera de la même façon en mode texte ou en mode graphique.

```
public interface AfficheRobot{
    void initialiser(char[][] tab) throws CharInconnu;
    void changerCase(int x, int y, char c) throws CharInconnu;
    void afficher();
    void terminer();
}
public class CharInconnu extends Exception{}
```

Passons en revue les méthodes de cette interface :

- initialiser comme son nom l'indique sert à initialiser l'affichage avec le terrain dans sa configuration initiale. Cette méthode affiche cette configuration initiale.
- changerCase change le caractère contenu dans une case du terrain. Cette méthode n'affiche pas le résultat du changement.
- afficher affiche le terrain dans sa situation actuelle.
- terminer est appelée pour terminer proprement un affichage avant de quitter le programme.

Pour ce qui est de l'affichage graphique, initialiser crée une fenêtre graphique et terminer détruit cette fenêtre graphique. En plus des méthodes de l'interface, la classe IGRobot qui implémente AfficheRobot fournit un constructeur sans paramètre et une méthode associerCharImage qui permet d'associer un caractère à un dessin ou si vous préférez, qui explique quelle image doit s'afficher quand un caractère donné est trouvé dans une case. Cette méthode est à appeler (éventuellement plusieurs fois pour plusieurs caractères différents) avant d'appeler initialiser. Il existe à la création de l'objet l'association initiale entre 5 images fournies et les 5 caractères prévus dans l'énoncé du projet `(' ','#','A','$','*')`. La méthode associerCharImage permet de redefinir les images associées à ces 5 caractères et à associer des images à d'autres caractères (par exemple pour représenter d'autres types de terrain). Le premier paramètre de associerCharImage est un caractère (type char), le second est le nom d'un fichier contenant l'image. Le programme cherchera le fichier dans le dossier nommé `images`. Ce fichier doit être au format png et avoir une dimension de 48x48 pixels.

Un fichier d'exemple d'utilisation de la classe IGRobot est fournit avec le code de cette classe et l'interface AfficheRobot.

3 Le langage des robots

Nous donnons ici une suggestion pour le langage des robots. Il pourra être étendu si vous voulez. Un programme pour les robots est un fichier comportant une liste d'instructions, une par ligne. Les instructions disponibles sont :

3.1 Instructions de programmation

Dans le texte qui suit : VARIABLE désigne un nom de variable, EXP désigne un nom de variable ou une valeur entière.

set *VARIABLE EXP*

fixe la valeur de VARIABLE à EXP.
Exemple

```
set compteur 0
```

add *VARIABLE EXP*

ajoute EXP à VARIABLE

sub *VARIABLE EXP*

retranche EXP à VARIABLE

mult *VARIABLE EXP*

multiplie EXP par VARIABLE

div *VARIABLE EXP*

divise VARIABLE par EXP

random *VARIABLE EXP*

place une valeur aléatoire entre 1 et EXP dans VARIABLE.
Exemple

```
random jet 6
```

Sauts : les boucles, conditionnelles, etc... utilisent les labels. Un label est introduit par un instruction spéciale :

label *nomLabel*

permet d'identifier une ligne du programme pour pouvoir y venir au moment de l'exécution d'une instruction de saut ou d'une instruction conditionnelle (voir ci-dessous).

L'instruction

goto *LABEL*

saute à la ligne du programme de label « LABEL ».
Exemple

```
label debut
  advance 0
  advance 3
  goto debut
```

Dans ce programme, le robot répète indéfiniment « aller d'une case vers le nord, puis aller d'une case vers l'ouest ».

jumpEqual *EXP EXP LABEL*

saute à label si les deux EXP sont égaux

jumpNEqual *EXP EXP LABEL*

saute à label si les deux EXP sont différents

jumpLess *EXP EXP LABEL*

saute à label si le premier EXP est inférieur strict au second

jumpLessEqual *EXP EXP LABEL*

saute à label si le premier EXP est inférieur ou égal au second.

Ces instructions très simples permettent de simuler la plupart des structures de contrôle d'un langage de programmation avancé. Voici par exemple l'équivalent d'un programme avec une boucle while, qui calcule la somme des dix premiers entiers :

```
  set somme 0
  set cpt 1
label boucle
  jumpEqual cpt 11 fin
  add somme cpt
  add cpt 1
  goto boucle
label fin
```

3.2 Instructions de commande du robot

detect *VARIABLE DIRECTION PORTEE*

utilise le radar du robot.
- DIRECTION vaut 0 pour nord, 1 pour est, 2 pour sud et 3 pour ouest.
- PORTEE est la portée de la détection (qu'on peut donc régler).

DIRECTION et PORTÉE sont soit des entiers, soit des variables.

Le résultat de la détection est un code rangé dans VARIABLE.

Comme il peut y avoir plusieurs objets dans la même direction, `detect` renvoie un code qui décrit *le plus proche objet.* Le code renvoyé est :
- 0 s'il n'y a rien ;
- 1 pour du minerai ;
- 2 pour de la roche ;
- 3 pour de l'eau.

Exemple :
detect resultat 0 10
lance une détection vers le nord, avec une portée de 10 cases. Notez que si dans la première case vers le nord il y a quelque chose, ça sera détecté.

extract

extrait le minerai qui se trouve dans *toutes* les cases environnantes (nord, sud, est ou ouest, pas en diagonale).

advance *DIRECTION*

avance d'une case dans une direction. DIRECTION est un code comme celui utilisé dans detect. Donc advance 0 avance d'une case vers le nord.

4 Règles détaillées

On va simuler le fait que le robot exécute son programme. Évidemment, une instruction comme une affectation ne prend pratiquement pas de temps, alors qu'un déplacement du robot ou l'extraction de minerai est beaucoup plus longue. On vous propose de compter que le déplacement prend une unité de temps, l'extraction dix unités de temps et que les autres instructions prennent un temps négligeable. L'unité de temps est utilisée pour afficher un état du jeu : un affichage est réalisé après chaque unité de temps.

Extraire du minerai le fait disparaître : sa place devient vide. Quand un robot rencontre un obstacle (minerai ou roche), il ne peut avancer (l'instruction de déplacement est sans effet, il continue avec l'instruction suivante). Un robot qui tombe à l'eau est détruit.

Le jeu s'arrête quand le robot tombe à l'eau ou quand le programme du robot s'arrête. Vous pouvez éventuellement prévoir aussi un temps de jeu limité ou un arrêt quand il n'y a plus de minerai à extraire.

5 Réalisation du projet

Dans ce projet, on vous demande impérativement d'utiliser des objets et des exceptions. On vous encourage à utiliser le plus de constructions Java possibles parmi celle qui ont été étudiées en cours : héritage, interface, structures récursives.

Comme d'habitude, votre programme doit avant tout être bien présenté, utiliser des noms de varaibles et de méthodes bien expressifs et avoir des méthodes de taille raisonnable.

Le suivi de projet se fait en deux étape :
- **étape 1 : analyse** à rendre la semaine du 2 mai. Un document vous précisera ce qui vous est demandé précisément.
- **étape 2 : programme** à rendre le 30 mai avec soutenances dans la foulée.

6 Extensions possibles

Voici une liste de possibilités non exhaustive.

- Utiliser comme terrain un planète ronde : quand un robot sort par un côté, il rentre par le côté d'en face.
- étendre le langage de programmation des robots avec d'autres instructions ou en modifiant les instructions proposées. Essayez de respecter l'esprit du langage, avec des instructions simples et courtes.
- augmenter le nombre de choses pouvant apparaître sur le terrain.
- lancer plusieurs robots concurrents ou coopérants sur le terrain.
- autoriser différents types de robots ayant des capacités différentes (déplacements plus rapides ou plus lents, capteurs permettant de détecter les mouvements, ou d'autres types d'objets, robots agressifs, etc).
- ajouter du son au jeu.
- afficher les instructions en cours d'exécution.
- améliorer l'interface graphique qui vous sera proposée.