Mini-projet «Curiosity Revolutions»

TD6 — Structure de base, lecture du terrain

Support enseignant

NB : ce projet utilise le même environnement que l'APP2 «Curiosity Reloaded» de l'UE INF301 ; cependant les problèmes posés et les exercices demandés sont orthogonaux. Il n'est pas nécessaire d'avoir suivi l'UE INF301 pour réaliser ce projet.

1. Contexte

> *« Liquid water on Mars ! » Maintenant, on en est sûrs, il y a de l'eau liquide sur Mars. Petit problème : le plan d'exploration du robot Curiosity est complètement perturbé, il ne faudrait pas qu'il s'embourbe dans de la boue martienne ou tombe au fond d'une flaque par mégarde...*

De nouveaux programmes ont été développés pour ce robot, pour tenir compte de cette nouvelle situation. Vous allez travailler sur un simulateur permettant de tester ces programmes : il vous faudra notamment vérifier que ce simulateur a bien le comportement attendu.

2. Organisation générale

Le programme sur lequel on travaille est un simulateur du robot Curiosity. Ce robot évolue sur un *terrain*, en exécutant un *programme* écrit dans un langage spécifique.

Exercice 1. Réflechissez à la structure globale de ce simulateur : quels peuvent être les modules du programme ? Quels types et/ou fonctions sont définies dans chacun de ces modules ?

> *On ne demande pas ici une spécification précise : de toutes façons pour les TPs une structure sera fournie (et il n'existe évidemment pas de solution unique...). Mais c'est l'occasion de se poser des questions de modularisation (sous forme de brainstorming très informel)...*
>
> *Exemple de découpage en modules :*
> - *module **terrain** :*
>   - *type terrain*
>   - *lire un terrain dans un fichier : le format de terrain donne la position initiale du robot (cf infra), il faut donc soit fournir un «robot» (qui sera «initialisé»), soit des coordonnées en paramètre résultat*
>   - *afficher un terrain*
>   - *éventuellement, des fonctions d'accès : largeur et hauteur d'un terrain, type d'une case d'un terrain en fonction de ses coordonnées...*
> - *module **robot** :*
>   - *type robot*
>   - *fonction d'initialisation d'un robot (coordonnées initiales en paramètres)*
>   - *éventuellement des fonctions d'accès*
> - *un module associant un «terrain» à un «robot» (peut être inclus dans le module «robot», voire dans le module «terrain» !) et dépendant donc des deux modules précédents :*
>   - *fonction d'initialisation (en paramètre : soit un fichier dans lequel on va lire le terrain, soit un terrain lu à l'extérieur du module ainsi que les coordonnées initiales du robot)*
>   - *fonctions d'actions du robot : avancer, tourner à gauche, tourner à droite (modifient les coordonnées du robot, et pour «avancer» renvoie une erreur si l'action n'est pas possible), effectuer une mesure (renvoie le type de terrain où la mesure a été effectuée)*
> - *un module «interprete» pour interpréter le programme du robot (dépend des modules précédents) :*
>   - *fonction d'initialisation : lecture d'un programme dans un fichier*
>   - *fonction d'interprétation : appelle les fonctions d'actions du robot en fonction du programme à interpréter*
> - *enfin, un programme principal qui :*
>   1. *initialise le terrain, le robot, le programme*
>   2. *appelle la fonction d'interprétation*

3. Le robot

Le robot va être amené à se déplacer dans le terrain à partir d'une case initiale pour si possible atteindre une case but, ou bien en sortir (la sortie est l'extérieur du terrain).

Le robot est défini par :

- sa position (`x`,`y`) en coordonnées entières,
- son orientation `o` pouvant prendre 4 valeurs possibles :

NORD : ↑	EST : →	SUD : ↓	OUEST : ←

Le type robot peut donc être défini ainsi :

```
typedef enum { Nord, Est, Sud, Ouest } Orientation;

typedef struct {
  int x, y;
  Orientation o;
} Robot;
```

Le robot est capable des actions suivantes :

- avancer,
- tourner d'un quart de tour à droite sur lui-même,
- tourner d'un quart de tour à gauche sur lui-même,
- effectuer une mesure.

Pour ce projet, l'interprétation des programmes des robots (fournis dans un langage spécifique — cf INF301) sera fournie.

4. Le terrain

Un *terrain* est un rectangle composé de cases carrées (`L` en largeur et `H` en hauteur), chaque case pouvant être libre (caractère `'.'`), occupée par de l'eau (caractère '~'), ou occupée par un rocher (caractère `'#'`). La case initiale du robot est une case libre, marquée par le caractère `'C'`.

Préconditions pour le terrain :

— les dimensions `L` (largeur) et `H` (hauteur) doivent être inférieures à une dimension maximale `DIM_MAX`,

— il existe un chemin (formé de cases libres) entre la case centrale et l'extérieur du terrain (la sortie).

Un fichier *terrain* est un fichier composé :

1. d'un entier `L`, la largeur du terrain
2. d'un entier `H`, la hauteur du terrain
3. de `H` lignes de `L` caractères dans l'ensemble {`'#','.','~','C'`}.

Exemple de fichier «terrain» :

```
11
9
#..#......#
...........
....C......
..........#
#..#.......
#........#.
......#....
#..##......
..#..#.....
```

Déclaration du type terrain en C :

```c
typedef enum { LIBRE, EAU, ROCHER } Case;

#define DIM_MAX 256

// indexation utilisée :
// 1er indice : abscisse = colonne (colonne de gauche : abscisse = 0)
// 2è me indice : ordonnée = ligne (ligne du haut : ordonnée = 0)

typedef struct {
  int largeur, hauteur;
  Case tab[DIM_MAX][DIM_MAX];
} Terrain;
```

Exercice 2. Écrire une fonction `lire_terrain`, permettant de lire un terrain dans un fichier (on suppose dans un premier temps que le format du fichier à lire est correct et les préconditions remplies).

```
void lire_terrain(FILE *f, Terrain *t, Robot *r) {
  int l, h; // Dimensions du terrain
  int rx, ry; // Coordonnées initiales du robot
  int i, j;
  char ligne[DIM_MAX];
  Case c;

  // Lecture de la largeur
  fscanf(f, "%d", &l);
  t->largeur = l;
  // Lecture de la hauteur
  fscanf(f, "%d", &h);
  t->hauteur = h;
  // Lecture du terrain
  // Lecture du caractè re de retour àla ligne précédant la premiè re ligne
  fscanf(f, "\n");
  for (j = 0; j < h; j++) {
    // Lecture d'une ligne dans le fichier
    fgets(ligne, DIM_MAX, f);
    // Parcours de la ligne
    for (i = 0; i < l; i++) {
      // Initialisation d'une case
      switch (ligne[i]) {
      case '.':
        c = LIBRE;
        break;
      case '#':
        c = ROCHER;
        break;
      case '~':
        c = EAU;
        break;
      case 'C':
        c = LIBRE;
        rx = i;
        ry = j;
        break;
      }
      t->tab[i][j] = c;
    }
  }
  // Initialisation de la position du robot
  r->x = rx;
  r->y = ry;
  fclose(f);
}
```

Exercice 3. Si le fichier n'existe pas ou est incorrect, quelles erreurs peuvent se produire à l'exécution de la fonction `lire_terrain`?

- *le fichier à lire ne peut pas être ouvert*
- *la lecture de la largeur échoue (à cause de la présence de caractères non numériques)*
- *la lecture de la hauteur échoue (à cause de la présence de caractères non numériques)*
- *la largeur est incorrecte (*`L > DIM_MAX` *ou* `L < 0`*)*
- *la hauteur est incorrecte (*`H > DIM_MAX` *ou* `H < 0`*)*
- *les caractères formant le terrain sont différents de* `'#'`, `'.'`, `'~'`, *et* `'C'`
- *une ligne du terrain est trop longue*
- *une ligne du terrain est trop courte*
- *il manque des lignes*
- *il manque la position initiale du robot*
- *(il y a trop de lignes)*

Compléter la fonction `lire_terrain` afin de renvoyer une erreur si le format du fichier n'est pas correct.

On peut se servir des mécanismes de gestion d'erreurs des fonctions `fopen`, `fscanf` *et* `fgets` *:*

`fopen` *renvoie* `NULL` *en cas d'erreur*

`fscanf` *renvoie le nombre de valeurs lues, et la valeur* `EOF` *si la fin du fichier est atteinte avant la première valeur lue*

`fgets` *renvoie le pointeur vers la chaîne lue en cas de succès, et la valeur* `NULL` *en cas d'erreur ou si la fin du fichier est atteinte avant d'avoir pu lire un caractère.*

```
1  typedef enum {
2    ERREUR_FICHIER,
3    ERREUR_LECTURE_LARGEUR,
4    ERREUR_LECTURE_HAUTEUR,
5    ERREUR_LARGEUR_INCORRECTE,
6    ERREUR_HAUTEUR_INCORRECTE,
7    ERREUR_CARACTERE_INCORRECT,
8    ERREUR_LIGNE_TROP_LONGUE,
9    ERREUR_LIGNE_TROP_COURTE,
10   ERREUR_LIGNES_MANQUANTES,
11   ERREUR_POSITION_ROBOT_MANQUANTE
12 } erreur_terrain;
13
14 erreur_terrain lire_terrain(FILE *f, Terrain *t, Robot *r) {
15   int l, h; // Dimensions du terrain
16   int rx, ry; // Coordonnées initiales du robot
17   int pos_robot = 0; // Booléen : vrai si la position du robot a été lue
18   int n; // Nombre de valeurs lues
19   char *res; // Résultat de la lecture d'une ligne
20   int lgligne; // Longueur de la ligne lue
21   int i, j;
22   char ligne[DIM_MAX];
23   Case c;
24
25   if (f == NULL) {
26     return ERREUR_FICHIER;
27   }
28   // Lecture de la largeur
29   n = fscanf(f, "%d", &l);
30   if (n == 0) {
31     return ERREUR_LECTURE_LARGEUR;
32   } else if ((l < 0) || (l > DIM_MAX)) {
33     return ERREUR_LARGEUR_INCORRECTE;
34   }
35   t->largeur = l;
36   // Lecture de la hauteur
37   fscanf(f, "%d", &h);
38   if (n == 0) {
39     return ERREUR_LECTURE_HAUTEUR;
40   } else if ((h < 0) || (h > DIM_MAX)) {
41     return ERREUR_HAUTEUR_INCORRECTE;
```

```
  }
  t->hauteur = h;
  // Lecture du terrain
  // Lecture du caractè re de retour àla ligne précédant la premiè re ligne
  fscanf(f, "\n");
  for (j = 0; j < h; j++) {
    // Lecture d'une ligne dans le fichier
    res = fgets(ligne, DIM_MAX, f);
    if (res == NULL) {
      return ERREUR_LIGNES_MANQUANTES;
    }
    lgligne = strlen(ligne);
    if (lgligne < l) {
      return ERREUR_LIGNE_TROP_COURTE;
    } else if (lgligne > l) {
      return ERREUR_LIGNE_TROP_LONGUE;
    }
    // Parcours de la ligne
    for (i = 0; i < l; i++) {
      // Initialisation d'une case
      switch (ligne[i]) {
      case '.':
        c = LIBRE;
        break;
      case '#':
        c = ROCHER;
        break;
      case '~':
        c = EAU;
        break;
      case 'C':
        c = LIBRE;
        rx = i;
        ry = j;
        pos_robot = 1;
        break;
      default:
        return ERREUR_CARACTERE_INCORRECT;
      }
      t->tab[i][j] = c;
    }
  }
  if (!pos_robot) {
    return ERREUR_POSITION_ROBOT_MANQUANTE;
  }
  // Initialisation de la position du robot
  r.x = rx;
  r.y = ry;
  fclose(f);
}
```