Série 9 :
Robot Aspirator

Le but de cette série est de stimuler la mise en oeuvre des *principes d'abstraction et de ré-utilisation* pour un programme écrit en C et tenant dans un seul fichier. Ce projet implique la plupart des concepts vus dans le cours jusqu'à maintenant: les entrées/sorties, les instructions de contrôle, les fonctions, la notion d'algorithme, éventuellement les machines à états finis. Il vous laisse une part de liberté créative qui nous permettra d'établir un classement des projets en termes de performances sur des scenarios de tests que nous fournissons.

Spécifications : il s'agit d'écrire un programme qui contrôle un robot aspirateur. Pour ce projet le robot est très simplifié. Il occupe **une case** d'une grille **de N x M cases** dont certaines autres cases sont occupées par des obstacles. Ces informations sont obtenues au lancement du programme par un dialogue avec l'utilisateur ou par redirection de fichiers de tests. Ensuite le programme fonctionne de manière autonome en gérant une boucle de mise à jour du robot jusqu'à atteindre l'une des conditions de sortie suivantes :

- **Succès** : Toutes les cases ne comportant pas d'obstacle ont été visitées au moins une fois
- **Echec** si sa batterie de 5 200 unités d'énergie est épuisée
- **Timeout** si plus 5 000 mises à jour ont été effectuées

Pseudocode du programme :

```
Lecture de la taille de la grille : N et M
Lecture de la case initiale du robot et de son orientation
Boucle de lecture des cases « obstacles »

Boucle de mise à jour jusqu'au TimeOut
    Analyser l'état actuel (perception des 8 cases voisines)
    Décider du déplacement autorisé
      - déplacement
      - mise à jour de l'énergie, trajectoire, statistiques
      - décision de sortie en cas de succès ou d'échec
Fin_boucle

Affichage Final des performances
```

Modélisation de l'espace et de l'orientation du robot (Fig. 1):

Le robot occupe seulement une case de la grille N x M. Les coordonnées de la case occupée par le robot sont données comme les indices de ligne et de colonne d'un tableau à deux indices en C. Lorsqu'on fournit cette information en entrée ou lorsqu'on

l'affiche en sortie, on doit d'abord indiquer **l'indice de ligne** entre **0 et N-1** puis **l'indice de colonne** entre **0 et M-1**. Les obstacles sont donnés par une liste de cases appartenant aussi à la grille.

Pour comprendre le codage de l'orientation il faut savoir que le robot peut aussi bien avancer ou reculer mais pas bouger sur le coté, exactement comme le fait une chaise roulante (en Robotique on dit qu'il est « non-holonome »). Le robot peut aussi tourner sur place dans les deux sens. Pour simplifier ce projet, on suppose que tourner et avancer/reculer peut seulement être fait séquentiellement.

Sachant cela, il suffit de coder 4 directions de l'espace pour pouvoir atteindre les 8 cases voisines de l'endroit où se trouve le robot. L'orientation du robot est donc identifiée par une valeur comprise entre 0 et 3. La valeur **0** indique une orientation Nord-Sud, c'est-à-dire verticale par rapport à notre tableau, ensuite l'indice d'orientation augmente d'une unité en tournant de 45° dans le sens des aiguilles d'une montre.

Structure des entrées :
On fournit dans l'ordre :

- La taille de la grille avec deux nombres strictement positifs pour N et M
- Les coordonnées initiales du robot : indice de ligne, de colonne, d'orientation
- La liste des cases (indices de ligne et de colonne pour chaque case) terminée par une valeur négative. On peut fournir plusieurs paires d'entiers sur une ligne puis continuer sur la ligne suivante, etc… les fichiers de test seront organisés de cette manière. Il est possible qu'il n'y ait pas d'obstacle.

Exemple de phase d'initialisation:

- 10 20
- 4 11 2
- 0 0 2 2 4 13 5 2
- 3 6 -1

Cette initialisation déclare un espace de 10 x 20 cases. Le robot se trouve initialement dans la case d'indice ligne 4, d'indice colonne 11, et l'indice d'orientation de 2 indique une orientation **Est-Ouest**. Ensuite 5 cases obstacles sont déclarées. La valeur -1 fait sortir de la boucle de lecture des obstacles.

Les erreurs suivantes devront être détectées dans les entrées :

- N ou M négatif ou nul
- Des indices de robot ou de cases en dehors des intervalles autorisés
- Une case d'obstacle à la même position que le robot

La figure 1 illustre les données de l'exemple ci-dessus. La barre horizontale montre le robot dans la direction est-ouest. Les X montrent la position des obstacles.

Perception locale et Contrôle du robot :
La perception du robot est **locale** : il ne sait rien d'autre sur le monde au-delà des 8 cases voisines. Il peut, d'une part, percevoir la **présence d'obstacle** dans ces 8 cases, et d'autre part percevoir si les 8 cases **ont déjà été nettoyées.** Cette perception locale doit être évalué une seule fois par passage dans la boucle de mise à jour.

Cette perception locale des obstacles et des cases déjà nettoyées sera mise en œuvre avec un **tableau à 2 indices** de type **char** qui doit être initialisé au lancement du programme comme décrit dans la Figure 1. Un caractère '**X**' indique un obstacle tandis qu'un caractère **espace** indique une case non-nettoyée. Le robot peut se ré-orienter et se déplacer d'une seule case par mise à jour. *Après chaque déplacement du robot,* on remplace le contenu de la case de destination par le caractère de son **orientation finale**. Ce n'est pas interdit de repasser plusieurs fois par le même endroit s'il n'y a pas d'obstacle.

Consultation supplémentaire du tableau: à chaque passage dans la boucle de mise à jour, votre programme peut consulter toutes les cases du tableau à 2 indices pour vérifier si le robot a visité toutes les cases autorisées et donc vérifier la décision de sortie avec succès.

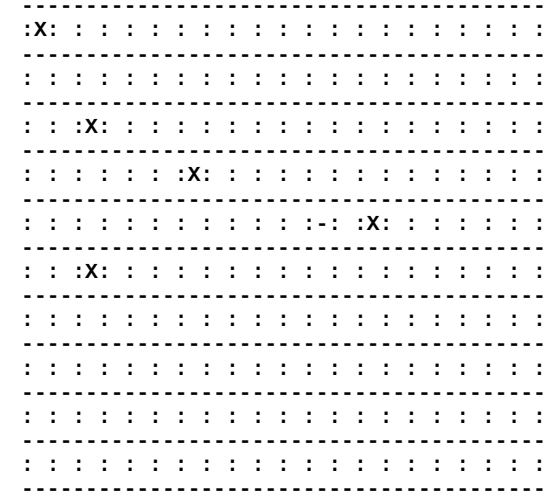

Figure 1 : grille de 10 x 20 cases représentée par un tableau à 2 indices de **char**, situation initiale: le robot est indiqué dans la case d'indices de ligne 4 et de colonne 11, sa direction est Est_Ouest ce qui explique la ***barre horizontale***. On choisira le ***slash***, ***l'anti-slash*** et la ***barre verticale*** pour représenter les autres directions. Les obstacles sont indiqués par un **X**. Ils ne sont pas encore perçus par le robot car ils sont trop éloignés des capteurs. Les cases qui n'ont pas encore été visitées par le robot sont initialisées avec le caractère **espace**.

Pendant une mise à jour, **S'il a assez d'énergie,** le robot peut, dans l'ordre, tourner puis se déplacer d'une case, ou seulement faire l'un ou l'autre. Nous illustrons le cas complet ici :

- mise à jour de la perception locale

- (éventuellement) **tourner dans une direction désirée ou au hasard**

- puis **se déplacer en ligne droite**. Le robot peut avancer/reculer seulement ***sur l'une des 2 cases voisines*** dans la direction actuelle :
  - Si la case de destination est ***dans la grille*** et est ***libre d'obstacle***
  - S'il a la ***place pour passer*** : ce cas est à examiner pour les cases en diagonales. En effet il faut aussi qu'il n'y ait pas d'obstacle sur les deux autres cases en diagonales qui séparent le robot de sa destination (fig. 3).

La consommation d'énergie est simplifiée de la manière suivante :

- **<u>Translation</u> : avancer ou reculer d'une case coûte une unité d'énergie**
- **<u>Rotation</u> : tourner coûte une unité pour chaque huitième de tour** (45°)

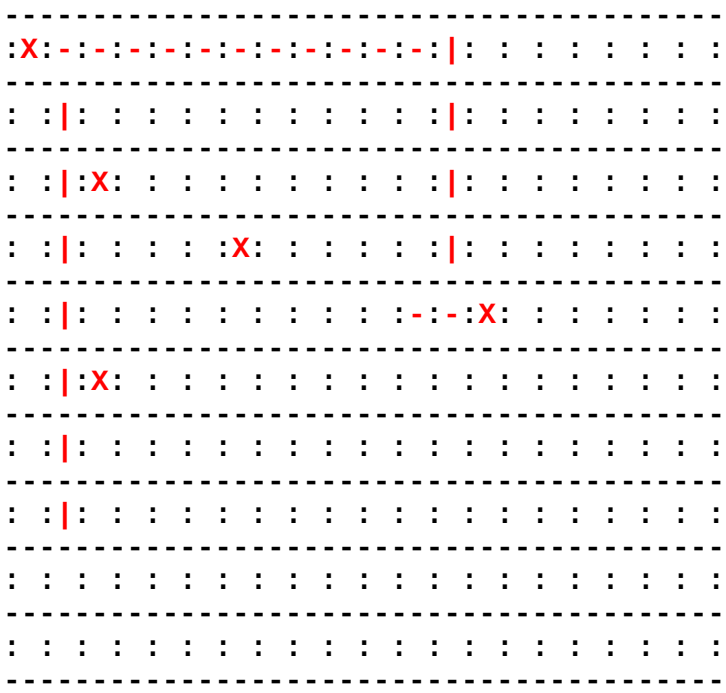

Figure 2 : mémoire de trajectoire partielle

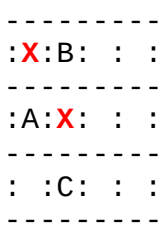

Figure 3 : le robot ne peut pas aller directement de A à B car il ne peut pas passer entre les 2 obstacles X, il faut faire le tour par l'extérieur. On autorise le fait de contourner un obstacle en prenant la diagonale, par exemple on peut aller directement de A à C.

<u>Suite de l'analyse à faire par vous même</u>:
Choisissez une méthode simple pour le choix du déplacement. Considérez d'introduire une part de hasard dans ce choix avec les fonctions **srand** et **rand**.

<u>Affichage en fin de programme pour chaque scénario de test, dans l'ordre suivant :</u>
- La mémoire de trajectoire (=contenu du tableau comme dans les fig 1 et 2)
- le nombre de cases visitées par le robot / le nombre de cases sans obstacle
- le nombre d'unités d'énergie restantes
- nature de l'arrêt : Succès, Echec, Timeout

Codage :

* Votre programme doit être entièrement contenu dans un seul fichier .c
* L'usage de variables globales est **interdit**. On peut seulement travailler avec des variable locales ou **static** locales à une fonction. Cette contrainte est imposée pour vous forcer à travailler sur le passage de paramètres.

Rendu :

Le code source doit être téléchargé sur la database moodle prévue pour cela au plus tard le jeudi 2/12 à minuit. Vous devez aussi fournir aux assistants les éléments **« papier »** suivants, au plus tard en TP le 3/12 à 10h00:

1) résultat de la phase d'analyse :
   a. **Rapport** (max 2 pages) : décrire l'organisation générale du programme en faisant ressortir la mise en oeuvre des principes *d'abstraction* et de *ré-utilisation*. Indiquer la liste des fonctions avec le but de chaque fonction. Préciser l'approche retenue pour le choix du déplacement.
   b. **Dessin** du *graphe des appels de fonctions* sur une page séparée (notes de cours chap 6).
2) code source imprimé avec une **mise en page** sans passage à la ligne parasite (=ajouté par l'impression = wrapping) ni de fin de ligne coupée.

3) **Les conventions de programmations** du cours sont notées

4) Impression de **l'exécution de votre programme sur nos tests** (par redirection de l'entrée et de la sortie)

Les fichiers de test sont téléchargeables sur le site moodle du cours.
Traitez les tests dans l'ordre proposé.

Barème (40 p)

(4p) rapport / point 1a
(4p) dessin / point 1b
(4p) décomposition : **mise en œuvre des principes d'abstraction et de ré-utilisation**
(6p) clarté du listing(indentation, mise en page)
(4p) respect des conventions de programmation du cours (nom des variables, etc…)
(18p) votre programme se termine correctement pour les 6 fichiers de test

Feedback que nous vous donnerons sur le projet :

- Le listing du code source vous sera rendu juste avant les vacances avec des indications de façon à s'améliorer pour le projet de semestre suivant
- la base de données des projets sera mise à disposition sur moodle en fin de semestre
- indépendamment de la note obtenue, un classement des projets sera établi selon les performances du programme sur les fichiers tests (en nb de mise à jour en cas de succès, en carburant restant en cas de timeout).