Lycos, va chercher!

laurent.jospin.59@free.fr, http://jospin.lstl.fr

Lycée Saint-Louis, Paris

En cas d'utilisation de Spyder, commencer par *enregistrer le fichier vide* automatiquement créé sous un nom pertinent (surtout sans apostrophe!) puis régler les options d'exécution (touche Ctrl + F6 du clavier) en : *Exécuter dans un nouvel interpréteur dédié* avec l'option *Intéragir avec l'interpréteur Python après l'exécution*. L'exécution des programmes se fait avec la touche F5. A chaque fois qu'un fichier est enregistré en changeant de destination, les options d'exécution doivent être configurées à nouveau.

En cas d'utilisation de Pyzo, enregistrer également le fichier vide automatiquement créé puis la première exécution doit être faite avec *Exécuter en tant que script* (Ctrl + Maj + E).

Le fichier `lycos.py` doit être téléchargé et placé dans le même dossier que votre programme.

Dans ce sujet, on appelle *procédure* une fonction qui produit un effet mais ne renvoie pas de valeurs, et on réserve par suite le mot *fonction* aux fonctions qui ne sont pas des procédures.

Au delà de l'objectif essentiel de prise en main de la programmation en Python, dans ce sujet on cherche à faire travailler sur la clareté et la concision du code. Ainsi l'évaluation tiendra compte de ces critères. Des fonctions ou procédures écrites dans une question précédente peuvent évidemment être réutilisées dans une question ultérieure. Dans certaines questions, l'efficacité du parcours pourra également être valorisée.

1 Programmation de Lycos

La programmation tient son nom du fait de décider par avance le comportement d'une machine. Nous allons ici programmer un robot virtuel, nommé Lycos pour le besoin de l'activité.

Pour pouvoir accéder au robot, nous utiliserons un module externe créé spécialement pour cette activité, nommé `lycos` les procédures suivantes de contrôle du robot :

- `affiche_etat()` qui affiche la pièce dans laquelle se déplace le robot. Les murs sont symbolisés par `#`, les objets à ramasser par `$`, le robot par `R` s'il transporte un objet et `r` sinon.
- `avance()` qui fait avancer le robot dans la direction dans laquelle il est tournée,
- `pivote_a_droite()` qui fait tourner le robot d'un quart de tour vers la droite,
- `prend_objet()` qui prend un objet situé sur la même position que le robot,
- `pose_objet()` qui dépose un objet transporté par le robot

ainsi que de quelques fonctions :

- `peut_avancer()` qui renvoie un booléen indiquant si la position suivante dans la direction du robot n'est pas un mur,
- `presence_objet()` qui renvoie un booléen indiquant s'il y un objet sur la même position,
- `compte_cailloux()` qui renvoie le nombre de cailloux présents sur la cases sur laquelle le robot est situé.

Quelques procédures permettent de mettre en place le robot et la pièce correspondant aux différentes situations décrites. On pourra notamment utiliser la procédure `initialise0()` pour faire des premiers tests, par exemple le programme minimal suivant devrait produire un affichage compréhensible dans la console.

```
1 from lycos import *
2
3 initialise0()
4 affiche_etat()
5 avance()
6 pivote_a_droite()
7 avance()
8 affiche_etat()
```

1. Ecrire une procédure `demi_tour` et une procédure `pivote_a_gauche` en répétant la procédure `pivote_a_droite`.

2. Ecrire une procédure qui fait avancer de `n` cases, où `n` est un argument.

2 Aller chercher un objet

3. La situation 1, obtenue en appelant la procédure `initialise1()`, est celle du robot face à un objet situé à une distance inconnue, il n'a qu'à avancer pour trouver l'objet. Ecrire une procédure `va_chercher` qui fait avancer le robot jusqu'à trouver l'objet puis prendre l'objet puis revenir au point de départ, puis le poser.

La fonction `verifie1()` renverra le booléen `True` après l'exécution de la procédure si le robot a bien effectué la consigne.

4. Dans la seconde situation obtenue avec `initialise2()`, le robot doit aller chercher des objets situés sur les n prochaines cases, le nombre n n'est pas connu et doit être déterminé par l'absence d'objet sur la case $n+1$. Le robot peut transporter un seul objet à la fois. De même, on pourra tester `verifie2()` après l'éxécution de la procédure.

5. On considère maintenant la situation 3 dans laquelle le robot est dans un couloir sans intersection. Ecrire une procédure `avance_couloir` prenant un entier `n` en argument et qui fait avancer de `n` cases en tournant à droite lorsque le robot ne peut avancer, ou à gauche s'il ne peut tourner à droite.

6. Dans cette situation 3, l'objet est au bout d'un couloir ne contenant aucune intersection mais éventuellement des tournants. Ecrire une fonction qui envoie le robot chercher l'objet puis le ramener et le poser au point de départ.

3 Parcourir un labyrinthe à main droite

La technique de la main droite consiste, dans un labyrinthe, à toujours partir dans la direction la plus à droite à chaque intersection. Si le labyrinthe possède certaines propriétés, on finit par trouver la sortie.

Dans la situation 4, on se situe dans un tel labyrinthe et on cherche à retrouver l'objet.

7. Ecrire une fonction `nombre_directions` qui regarde les 4 directions et qui compte le nombre de directions dans lesquelles on peut avancer. A la fin de cette fonction la direction du robot doit être revenue à la direction antérieure à la fonction.

8. Ecrire une procédure `avance_le_plus_a_droite` qui avance d'une case dans la direction la plus à droite possible (à droite si possible, sinon tout droit, sinon à gauche, sinon demi tour).

9. Ecrire une procédure `trouve_objet_labyrinthe` qui répète la fonction précédente jusqu'à trouver l'objet puis le prend, on ne demande pas de le ramener.

10. Le robot se déplace en déposant un caillou sur chaque position sur laquelle il avance, il peut compter les cailloux à la position où il est grâce à la fonction `compte_cailloux()`.

Ecrire une fonction `essaye_trouver_objet_labyrinthe` qui s'inspire de la fonction précédente mais s'arrête si le nombre de cailloux sur une case indique qu'on ne trouvera jamais l'objet avec cette méthode (nombre de cailloux supérieur au nombre de directions). La fonction devra renvoyer un booléen indiquant si l'objet a été trouvé et ramassé.

4 Rangement d'un couloir

11. Dans la situation 5, des objets sont disposés dans un couloir sur des positions pas nécessairement contigües, le robot est situé à une extrémité et doit ranger les objets en les plaçant sur des positions contigües du couloir. Ecrire une procédure réalisant ceci en essayant de réduire au maximum le nombre de déplacements effectués.