TP 3 - PROGRAMMATION ROBOTIQUE - INFO2

YANN CHEVALEYRE & PIERRE BOUDES

Ce TP est en deux parties.

La permière partie vous initiera à la **plannification** (c'est à dire la recherche de trajectoire). Pour simplifier, on supposera qu'on connait la position et l'orientation du robot à tout instant (grace à un GPS et un magnétomètre), et qu'on connait aussi le plan (c'est à dire la position des obstacles). Il faudra simplement établir la trajectoire jusqu'au but. Le robot commencera en haut à gauche et devra atteindre le bas à droite de l'écran, en évitant les obstacles.

Dans la seconde partie (à venir), nous lèverons l'hypothèse de connaissance du plan, et le robot devra explorer de lui-même son environnemetn pour recréer la position des obstacles. C'est la phase de **mapping.** Dès lors que le plan est suffisament bien connu, on peut après faire de la plannification.

1. Prise en main du simulateur

En python, le simulateur se lance ainsi:

```
from robosim import *
init()
```

C'est un simulateur ultra-simplifié ou l'on peut commander un robot. Voici une partie des commandes disponibles:

```
# Commandes de navigation
avance, obstacle, oriente, tournegauche, tournedroite
# Capteurs et relevés
telemetre, telemetre_coords, position, obstacle_coords
# Dessin
line, circle, efface
```

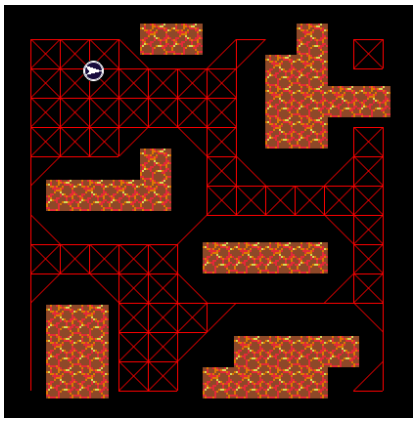
Dans Python, lisez l'aide de ces commandes (par exemple en tapant help(avance))

(1) Faites un petit programme qui tire un angle au hasard, oriente le robot selon cet angle, et fasse avancer le robot pixel par pixel pendant 10 pas de temps
Pour tirer un angle au hasard, on peut utiliser ceci:
```
from random import randint
print randint(0,360)
```
(2) Améliorer le programme précédent ainsi:
au lieu d'avancer pendant 10 pas de temps, le robot avance jusqu'à ce qu'il heurte un obstacle. Ensuite, le robot doit tirer une nouvelle orientation au hasard et recommencer.
(3) Coder une fonction *aller(x,y)* qui déplace le robot jusqu'en x, y. On fera les hypothèses suivantes:

(a) On suppose que le point aux coordonnées x, y se situe sur la droite du robot, ou en dessous du robot, ou en bas à droite. Autrement dit, si cx, cy sont les coordonnées du robot, on supposera que $x \geq cx$ et $y \geq cy$.
(b) On ne déplacera le robot que horizontalement, verticalement, et en diagonal.
(c) Le robot doit atteindre le point (x, y) le plus vite possible

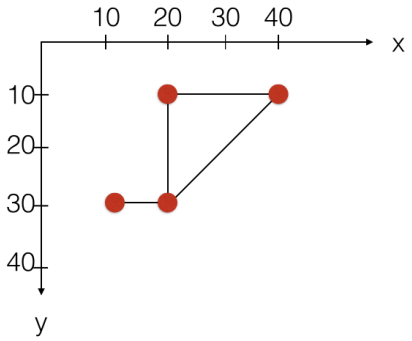
2. La librarie de graphes NetworkX

Pour modéliser les trajectoires possibles jusqu'au but (le coin en bas à droite de la fenêtre), on va utiliser un graphe, et calculer le plus court chemin. Sur la figure ci-dessous, le graphe est représenté par des traits rouges verticaux, horizontaux et diagonaux, et le plus court chemin dans ce graphe mène clairement au but sans toucher d'obstacle.



Votre but sera donc de construire ce graphe et de l'utiliser. Dans les graphes que vous manipulerez, chaque noeud sera une paire de coordonnées (x, y).

Par exemple, le graphe ci-dessous a trois noeuds, identifiés par $(20, 10), (40, 10)$ et $(20, 30)$.



Vous utiliserez la librairie python *networkx* pour créer des graphes. Ainsi, le graphe ci-dessus se créer ainsi en Python:

```
import networkx as nx
```

```
from math import sqrt

g = nx.Graph()
g.add_edge( (10,30),(20,30),weight=10.0 )
g.add_edge( (20,10),(40,10),weight=20.0 )
g.add_edge( (20,10),(20,30),weight=20.0 )
g.add_edge( (20,30),(40,10),weight=20.0*sqrt(2) )
```

Les autres fonctions de networkx utiles pour nous sont les suivantes:

```
>>> nx.shortest_path(g,(10,30),(40,10),weight='weight')
[(10, 30), (20, 30), (40, 10)]
# renvoyer la liste des aretes
>>> g.edges()
[((20, 10), (40, 10)), ((20, 10), (20, 30)),
((10, 30), (20, 30)), ((20, 30), (40, 10))]
# renvoyer la liste des noeuds
>>> g.nodes()
[(20, 10), (10, 30), (20, 30), (40, 10)]
# supprime une arete
>>> g.remove_edge( (40, 10), (20, 30) )
# recalcul du plus court chemin
>>> nx.shortest_path(g,(10,30),(40,10),weight='weight')
[(10, 30), (20, 30), (20, 10), (40, 10)]
# supprime un noeud et toutes les aretes associees
>>> g.remove_node( (20,10) )
>>> g.edges()
[((10, 30), (20, 30))]
>>> g.nodes()
[(10, 30), (20, 30), (40, 10)]
```

Donc, pour parcourir l'ensemble des noeuds d'un graphe g, on peut faire par exemple:

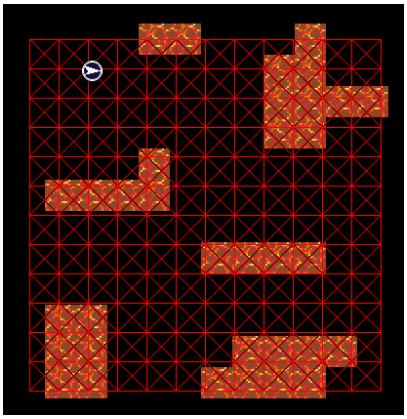
```
for (x,y) in g.nodes():
    print "voici le noeud ",x,y
```

Et pour parcourir l'ensemble des aretes d'un graphe g, on peut faire par exemple:

```
for (x1,y1),(x2,y2) in g.edges():
    print "le noeud ",x1,y1," est relie au noeud ",x2,y2
```

3. Planification

Création du graphe en forme de Grille. Dans un premier temps, vous créerez un graphe recouvrant l'ensemble de l'environnemnent comme dans la figure suivante:



Le coin en haut à gauche a pour coordonnées $(80, 80)$, et le coin en bas à droite $(440, 440)$. L'écart entre deux noeuds est de 30.

Elagage du graphe. Ensuite, vous devez parcourir chaque noeud du graphe, tester s'il est à proximité d'un obstacle avec la fonction *obstacle_coords*, et supprimer ce noeud si c'est le cas.

Plus court-chemin. Ensuite, vous devrez lancer le calcul du plus court chemin de votre position actuelle vers le bas du graphe. Ce calcul renverra une liste de noeuds. Votre robot devra aller de noeud en noeud en suivant cette liste, grâce à la fonction *aller(x,y)*