Partiel de robotique
16 mai 2014, session 1

1 Lecture de programmes (facile)

1.1 Dans le couloir

Pippo vient d'écrire un programme qui lui permet de résoudre la question du déplacement probabiliste d'un robot dans un couloir. Plus précisément le problème résolu est le suivant.

> Un robot se déplace dans un couloir de N cases, en avançant case par case dans l'une ou l'autre des deux directions. À chaque fois que le robot se déplace d'une case il y a une probabilité `p[0]` qu'il ne se soit pas déplacé (il reste sur sa case d'origine), une probabilité `p[1]` qu'il se soit déplacé d'exactement une case dans la bonne direction, une probabilité `p[2]` qu'il soit allé trop loin d'une case, etc. Le nombre de possibilités d'erreurs peut varier, mais la somme des probabilités fait toujours 1. Écrire un programme qui simule le déplacement du robot à l'aide d'actions au clavier lui ordonnant de prendre l'une ou l'autre des deux directions, et pour chaque case afficher sa probabilité de présence en cette case. Les deux bouts du couloirs sont fermés, ainsi le robot reste nécessairement dans le couloir.

Pippo a construit un programme complet qui fonctionnait presque mais il y avait un bug étrange et il a effacé une partie du code de la fonction `deplacer_ouest`. Voici son programme.

```
 1  # Paramètres ####
 2  N = 10
 3  p = [0.2, 0.6, 0.1, 0.1]
 4  K = len(p)
 5
 6  def un_char():
 7      # Cette fonction récupère un caractère auprès de l'utlisateur sans
 8      # qu'il soit nécessaire que celui-ci tape <entrée>.
 9      import sys, tty, termios
10      fd = sys.stdin.fileno()
11      old_settings = termios.tcgetattr(fd)
12      try:
13          # passage en mode caractère
14          tty.setraw(sys.stdin.fileno())
15          ch = sys.stdin.read(1)
16      finally:
17          # retour en mode ligne
18          termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
19          return ch
20
21  def deplacer_est(t): # une case vers la droite ->
22      res = [0 for i in range(N)]
```

```
23      for i in range(N):
24          for j in range(0, min(i + 1, K)):
25              res[i] += t[i - j] * p[j]
26      for i in range(0, K): # cumul des cas où le robot atteint le bord Est
27          for j in range(i + 1, K):
28              res[N - 1] += t[N - 1 - i] * p[j]
29      return res
30
31  def deplacer_ouest(t): # une case vers la gauche <-
32      res = [0 for i in range(N)]
33      for i in range(N):
34          for j in range(0, min(N - i, K)):
35              ...
36
37  def main():
38      t = [0 for i in range(N)] # terrain (un couloir)
39      t[N/2] = 1. #position initiale
40      continuer = True
41      while (continuer):
42          affichage_joli = [round(e, 2) for e in t]
43          print affichage_joli # affichage avec moins de décimales
44          c = un_char() # saisie utilisateur
45          if ((c == 'j')  or (c == 'J')):
46              t = deplacer_ouest(t)
47          if ((c == 'j')  or (c == 'L')):
48              t = deplacer_est(t)
49          if ((c == 'q') or (c == 'Q')):
50              continuer = False
51
52  main()
```

Comportement attendu du programme :

```
 1  RobotiqueL1$ python couloir.py
 2  [0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0]
 3  <l'utilisateur tape J (déplacement vers l'Ouest)>
 4  [0.0, 0.0, 0.1, 0.1, 0.6, 0.2, 0.0, 0.0, 0.0, 0.0]
 5  <l'utilisateur tape J (déplacement vers l'Ouest)>
 6  [0.03, 0.13, 0.16, 0.4, 0.24, 0.04, 0.0, 0.0, 0.0, 0.0]
 7  <l'utilisateur tape J (déplacement vers l'Ouest)>
 8  [0.21, 0.19, 0.3, 0.23, 0.07, 0.01, 0.0, 0.0, 0.0, 0.0]
 9  <l'utilisateur tape Q>
10  RobotiqueL1$ python couloir.py
11  [0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0]
12  <l'utilisateur tape L (déplacement vers l'Est)>
13  [0.0, 0.0, 0.0, 0.0, 0.0, 0.2, 0.6, 0.1, 0.1, 0.0]
14  <l'utilisateur tape L (déplacement vers l'Est)>
15  [0.0, 0.0, 0.0, 0.0, 0.0, 0.04, 0.24, 0.4, 0.16, 0.16]
16  <l'utilisateur tape L (déplacement vers l'Est)>
17  [0.0, 0.0, 0.0, 0.0, 0.0, 0.01, 0.07, 0.23, 0.3, 0.39]
18  <l'utilisateur tape Q>
```

1.1.1 Mettre en rapport le code et le comportement (1.5 pt)

Pour chacune des questions suivantes répondre à l'aide du comportement attendu et en donnant les numéros de ligne du programme de Pippo impliquées (le bug de Pippo n'intervient pas ici).

1. De combien de cases est composé le couloir ?
2. Dans quelle case se trouve initialement le robot ?
3. Dans quelle case y-a t'il le plus de chances de trouver le robot après un déplacement de trois cases vers l'Est ? Pourquoi ?

1.1.2 Expliquer le comportement attendu (1.5 pt)

Dans le comportement attendu, le tableau après un déplacement de deux cases (ou plus) vers l'Ouest (ligne 6) n'est pas symétrique au tableau après un déplacement de deux cases (ou plus) vers l'Est (ligne 15). Expliquer pourquoi.

1.1.3 Le bug de Pippo (2 pt)

Le bug de Pippo se manifestait de la façon suivante, avant qu'il ne perde la fonction `deplacer_ouest` qui était correcte. Lorsque Pippo demandait le déplacement du robot vers l'ouest en tapant `"j"`, celui ci n'avançait pas vraiment mais se dispersait dans les cases alentours. Pouvez-vous expliquer cela et corriger le bug (en supposant que vous disposez de la fonction `deplacer_ouest` correcte) ?
Comportement observé :

```
RobotiqueL1$ python couloir.py
[0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0]
<Pippo tape j>
[0.0, 0.0, 0.02, 0.08, 0.19, 0.42, 0.19, 0.08, 0.02, 0.0]
<Pippo prend sa tête à deux mains (facepalm)>
```

1.1.4 Reconstituer le programme (2.5 pt)

Aidez Pippo en reconstituant le code de `deplacer_ouest`.

1.1.5 Au bout du couloir le robot chute et disparaît ! (1.5 pt)

Le couloir dans lequel se déplace le robot de Pippo est en fait posé sur une table. On retire les murs à chacune des deux extrémités. Si le robot dépasse la dernière case il tombe et disparaît. On veut représenter la nouvelle probabilité de présence du robot de Pippo en chaque case du couloir. Pour cela vous devez expliquer quoi modifier dans le programme de Pippo. La somme des probabilités de présence sur les cases du couloir doit-elle nécessairement être 1 ?

1.2 Python-nxt (2 pt)

Commenter chaque ligne du programme suivant en expliquant succinctement sa fonction. Que signifie le fait que `weak_turn` est *non bloquant* ?

```
1  b = find_one_brick()
2  mgauche = Motor(b, PORT_A)
3  mdroite = Motor(b, PORT_B)
4  mgauche.weak_turn(100, 360) # /!\ non bloquant
5  lux = Color20(b, PORT_3)
6  while (1):
7      intensite = lux.get_reflected_light(nxt.sensor.Type.COLORNONE)
```

```
 8      if (intensite > 100):
 9          mgauche.brake()
10          break
```

2 Élaboration de petits programmes

2.1 NXT (4 pt)

Vous disposez d'un robot NXT équipé de deux roues motorisées reliées en A et B et d'un capteur de lumière dirigé vers le sol entre ses deux roues et branché sur le port 3. Ce robot se trouve actuellement sur une tâche sombre (intensité < 100). Droit devant lui se trouve une zone claire puis une nouvelle tâche sombre, à une distance inconnue.
Écrire un programme pour le robot qui l'amène et le stoppe exactement à mi-distance entre les deux tâche sombres. Attention le terrain peut-être en pente (mais avec une déclivité constante).

2.2 Boids (2 pt)

M. Tortue écrit un programme de Boids en *Python turtle*. Initialement, il tire au hasard la position $P_0, ..., P_9$ de 10 tortues à l'écran (des vecteurs $\overrightarrow{P_i} = (px[i],\ py[i])$), et il leur attribue une vitesse nulle (un vecteur $\overrightarrow{v_i} = (vx[i],\ vy[i])$) à chacune. Le reste du programme est dans un boucle infinie. M. Tortue applique deux règles pour calculer la nouvelle vitesse de chacune des tortues. L'influence des règles est déterminée par des paramètres globaux $\alpha > 0$ et $\beta > 0$.

Règle 1 Calculer le centre du groupe C de tortue et, pour chaque tortue i, ajouter à sa vitesse une composante $\alpha \times \overrightarrow{P_iC}$ ramenant la tortue i vers le centre.

Règle 2 Calculer la moyenne des vitesses $\overrightarrow{m}$ du groupe de tortue, et pour chaque tortue i, ajouter à v_i un composante $\beta \times \overrightarrow{m}$.

Après quoi la position de chaque tortue est modifiée en lui ajoutant sa propre vitesse.

1. Que va t'il arriver avec ce groupe de tortues ?
2. Pouvez-vous suggérer à M. Tortue une troisième règle (en pseudo-code) qui lui permette de simuler de façon un peu plus réaliste un groupe d'oiseaux ?

3 Problème (plus difficile)

3.1 Damier (4 pt)

Comment faire en sorte que le robot de Pippo se déplace maintenant sur un damier 8x8 ? Vous traiterez les deux cas : sans murs puis avec murs. Il est plus facile de commencer par traiter ce problème en considérant que le damier n'a pas de murs (le robot chute en bord de damier) puis de modifier le programme pour traiter le cas avec murs.
Le tableau des probabilités reste un tableau à une dimension, qui représente quelle que soit la direction choisie, Sud, Est, Nord, Ouest, la probabilité d'avancer de zéro, une, deux etc. cases dans cette direction.
Vous pouvez répondre par un algorithme détaillé en pseudo-code, ou un programme complet ou bien en expliquant les modifications que vous apportez au programme de Pippo.