UPMC - UE MIB (MV427)

Écrit, 3 juin 2011

Durée : 2h

Tous documents sur papier autorisés

N.B. Les parties A et B sont indépendantes

On veut modéliser le déplacement d'un insecte à la recherche de nourriture dans un champ.

Partie A

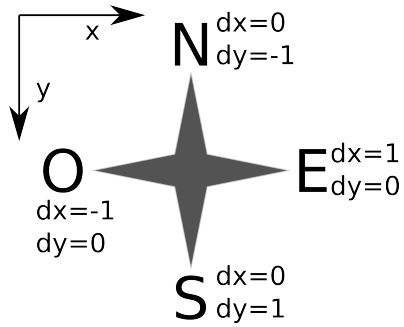
On a commencé à écrire la classe `Insecte` :

```
import java.util.*;

class Insecte {

    private int x,y,dx,dy;
    private Random r = new Random();

    void affiche() {
       System.out.println("Insecte en "+x+";"+y+", direction = ("+dx+";"+dy+")");
    }
}
```



Dans cette classe, les données membres `x` et `y` représentent les coordonnées de l'insecte dans le champ. La direction du prochain déplacement de l'insecte est représentée par `dx` et `dy` (*cf* schéma ci-contre).

Le champ est représenté par la classe `Champ`, qui est la classe exécutable de notre programme. Voici une première version de cette classe :

```
class Champ {

    public static void main(String args[]) {

       int leo = 1000; // longueur est-ouest
       int lns = 1000; // longueur nord-sud

       Insecte bug = new Insecte(leo,lns);
       bug.affiche();

    }
}
```

1) Écrire le constructeur d'`Insecte` qui est appelé par la classe `Champ`, et réalise les opérations suivantes :

- Positionner aléatoirement l'insecte dans le champ.
- Choisir une direction aléatoire qui doit obligatoirement être l'une de ces quatre possibilités : Nord, Sud, Est, Ouest. La direction est spécifiée par les valeurs de `dx` et `dy` (*cf* schéma ci-dessus).

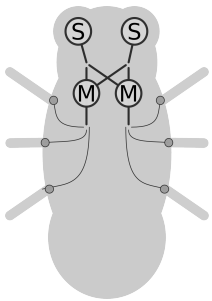
2) Quelles commandes exécutées dans un terminal Unix permettent de compiler puis lancer le programme ? Donner un exemple d'affichage obtenu lors d'une exécution de ce programme.

3) Écrire pour la classe `Insecte` une méthode nommée `tourneAGauche`, qui change la direction de déplacement de 90° vers la gauche (par exemple, si l'insecte était dirigé vers l'Ouest, il est dirigé vers le Sud après exécution de la méthode) en modifiant les valeurs de `dx` et `dy`.

On supposera pour la suite qu'on a écrit sur le même modèle une méthode `tourneADroite`.

4) Écrire la méthode `avance` de la classe `Insecte`, qui déplace l'insecte d'un pas dans la direction définie par `dx` et `dy`. Cette méthode ne doit jamais faire sortir l'insecte du champ.

Partie B



L'insecte dispose d'un système nerveux (*cf* schéma ci-contre) constitué de 2 types de neurones :

- Les neurones sensoriels (S) situés dans les bulbes olfactifs, perçoivent les molécules odorantes et émettent un signal proportionnel à l'intensité du *stimulus* ;
- Les neurones moteurs (M) reçoivent les signaux des neurones sensoriels et selon leur niveau, provoquent une action d'avancement ou de recul des pattes situées de leur côté.

Dans ce système simplifié, le déplacement de l'insecte a lieu en deux phases distinctes : orientation et avancée.

Phase d'orientation :

- Chaque neurone sensoriel perçoit uniquement la nourriture située de son côté de l'insecte (droite ou gauche). Il émet un signal proportionnel à l'inverse de la distance à laquelle se trouve la nourriture détectée (par convention, si la distance est nulle, le signal a une intensité de -1). On suppose que les unités choisies et le mode de calcul permettent de représenter ce signal par un nombre entier.
- Chaque neurone moteur compare les signaux qu'il reçoit des deux neurones sensoriels auxquels il est relié. Si le neurone sensoriel situé du même côté que lui a émis le signal le plus fort, alors le neurone moteur envoie le signal R (recul). Si c'est l'autre neurone sensoriel qui a émis le signal le plus fort, alors le neurone moteur envoie le signal A (avancée). Enfin, si les deux signaux sont égaux, le neurone moteur envoie I (Immobilité).

Phase d'avancée :

- Chaque neurone sensoriel agit comme précédemment.
- Chaque neurone moteur qui a reçu au moins un signal -1 envoie le signal I. Dans tous les autres cas, il envoie le signal A.

On suppose que la position d'une source de nourriture est communiquée au système sous forme de deux coordonnées (x,y) et que le comportement de l'insecte est affiché à l'écran en deux caractères représentant respectivement les pattes gauches et droites, et suivant la convention : A=Avancer, R=Reculer, I=rester Immobile.

1) Proposer un schéma de classes hiérarchique permettant de représenter les deux types de neurones. Ce schéma doit être donné sous forme de diagramme. On donnera les en-têtes des méthodes (type de retour, paramètres) mais on n'en détaillera pas le code. La hiérarchie doit clairement distinguer les données et méthodes communes aux deux types de neurones, et celles qui sont propres à chaque type.

2) Écrire le code d'une classe exécutable `SystNerv` qui représente l'ensemble du système : déclaration et initialisation des 4 neurones ; localisation de l'insecte et création d'une source de nourriture en deux points aléatoires d'une grille de 1000x1000 points ; appel des méthodes permettant de faire agir l'insecte (afficher les deux caractères qui décrivent le mouvement des pattes) durant une phase d'orientation puis une phase d'avancée.

Partie C

On suppose qu'on dispose d'une classe `Insecte` et d'une classe `SystNerv` fonctionnelles, qui répondent aux spécifications données dans les parties A et B ci-dessus.

Pour finaliser le programme, on veut doter l'insecte d'un système nerveux. Pour cela, on veut modifier la classe `SystNerv` pour qu'elle ne soit plus exécutable, mais qu'il existe une instance de `SystNerv` parmi les attributs de la classe `Insecte`. Ce système nerveux agit sur l'orientation de l'insecte et le fait avancer.

1) Décrire les modifications à apporter à la classe `SystNerv` et à la classe `Insecte` pour réaliser cette opération.

2) Décrire les modifications à apporter à la classe `Champ` pour qu'elle positionne l'insecte et la nourriture, puis que l'insecte de déplace jusqu'à avoir atteint la nourriture.

Barème indicatif sur 60

A1 = 6 ; A2 = 4 ; A3 = 5 ; A4 = 5.
B1 = 12 ; B2 = 10.
C1 = 9 ; C2 = 9.

Les indications de correction ci-dessous ne constituent pas la seule possibilité de réponses justes. De plus, elles sont agrémentées de commentaires (en bleu) qui n'ont pas à figurer dans une copie.

A1 Dans la classe `Champ`, le constructeur d'`Insecte` est appelé par `bug = new Insecte(leo,lns);` et son prototype doit correspondre à cet appel. Pour répondre aux spécifications données dans le texte, on peut avoir :

```
public Insecte(int xm, int ym) {

  // Positionner aléatoirement l'insecte dans le champ :

  x = r.nextInt(xm); // Donne à x une valeur comprise entre 0 et xm-1
  y = r.nextInt(ym); // Donne à y une valeur comprise entre 0 et ym-1

  // Choisir une direction aléatoire (une et une seule des variables
  // dx ou dy doit être nulle, l'autre vaut -1 ou 1) :

  dx = r.nextInt(3)-1;          // dx prend comme valeur -1, 0 ou 1

  if (dx == 0) {                 // Si dx vaut 0,
    dy = r.nextInt(2) * 2 - 1;   //     alors dy prend comme valeur -1 ou 1
  }
  else {                         // Si dx ne vaut pas 0 (il vaut -1 ou 1),
    dy = 0;                      //     alors dy doit être nul
 }
}
```

NB Avec le code ci-dessus, les quatre directions ne sont pas équiprobables (aucune contrainte n'était pas spécifiée à ce propos dans le texte) : Est et Ouest ont une probabilité de 1/3 chacune, Nord et Sud de 1/6 chacune.

A2 On compile le programme avec :

```
javac Champ.java
```

Comme la classe `Champ` utilise une instance d'`Insecte`, la compilation de `Champ.java` déclenche si nécessaire la compilation d'`Insecte.java`. Il n'est donc pas utile de compiler ce dernier avec une commande spécifique (mais si on le fait, cela ne produit pas d'erreur). La compilation produit deux fichiers : `Insecte.class` et `Champ.class` (la classe exécutable).

Pour exécuter le programme, la commande est :

```
java Champ
```

On obtient un affichage du type :

```
Insecte en 108;342, direction = (-1;0)
```

Naturellement, les valeurs numériques pour la position de l'insecte peuvent être des entiers quelconques entre 0 et 999. La direction est l'une des quatre possibles : (-1;0), (1,0), (0,-1), (0,1).

A3 Voici un exemple de code simple pour la méthode `tourneAGauche()` :

```
public void tourneAGauche() {

    // Puisqu'au départ, l'insecte est tourné vers N, S, O ou E,
    // il suffit de tester l'une des deux variables pour connaître
    // l'orientation initiale.

    if (dx == -1) {      // si l'insecte est tourné vers l'Ouest
      dx =  0;
      dy =  1;
    }

    else if (dx ==  1) { // si l'insecte est tourné vers l'Est
      dx =  0;
      dy = -1;
    }

    else if (dy == -1) { // si l'insecte est tourné vers le Nord
      dx = -1;
      dy =  0;
    }

    else if (dy ==  1) { // si l'insecte est tourné vers l'Ouest
      dx =  1;
      dy =  0;
    }

}
```

On peut aussi écrire une version plus élégante :

```
public void tourneAGauche() {
    int tx = dx;
    dx =  dy;
    dy = -tx;
}
```

A4 La méthode `avance()` peut être très simple :

```
public void avance() {

    x = x + dx;
    if (x >= 1000)
        x = 999;
    if (x < 0)
        x = 0;

    y = y + dy;
    if (y >= 1000)
        y = 999;
    if (y < 0)
        y = 0;

}
```

Cependant, cette version présente un défaut : les limites de l'espace sont arbitrairement fixées à 1000. Or, le constructeur d'`Insecte` permet à la classe qui crée l'objet (ici `Champ`) de fixer ces limites, et il les utilise pour le positionnement initial. Il serait préférable que les autres méthodes d'`Insecte` tiennent compte des mêmes limites que celles utilisées lors de la construction de l'objet. Pour ce faire, il faut enrichir la classe `Insecte` de deux variables supplémentaires, dont les valeurs sont fixées lors de la construction, et utilisées par les autres méthodes. Il faut donc :

- Ajouter, parmi les variables membres de la classe `Insecte` :

```
int xmax, ymax;
```

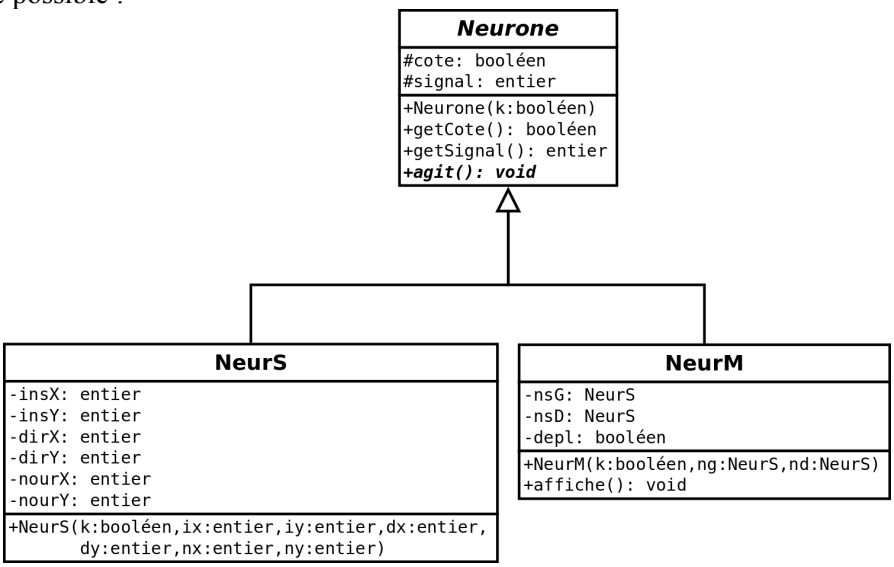
- Ajouter dans le constructeur d'`Insecte` :

```
xmax = xm;
ymax = ym;
```

- Modifier la méthode `avance()` en remplaçant `1000` par `xmax` ou `ymax`, par exemple :

```
if (x >= xmax)
    x = xmax-1;
```

B1 Voici une hiérarchie possible :



D'autres choix sont possibles sur plusieurs points. Par exemple :

- L'attribut `signal` est utilisé par le neurone pour stocker le signal qu'il émet. Le fonctionnement d'un neurone consiste à donner une valeur à cette variable lorsque la méthode `agit()` est appelée. Cette valeur est ensuite accessible en utilisant la méthode `getSignal()`. On aurait aussi pu faire émettre le signal sous forme d'une valeur de retour de la méthode `agit()`, et ne pas le stocker dans une variable membre de la classe `Neurone`.

- Le neurone moteur dispose de deux variables : `nsG` et `nsD`, qui sont les références des neurones sensoriels gauche et droit. Il peut ainsi interroger ces neurones, pour connaître le signal qu'ils émettent. Il est possible d'inverser le dispositif : chaque neurone sensoriel contient les références des deux neurones moteurs, et leur envoie la valeur du signal qu'il émet. Les neurones moteurs n'ont alors pas besoin de contenir les références des neurones sensoriels.
- Les neurones sensoriels utilisent la position (`insX` et `insY`) et l'orientation (`dirX` et `dirY`) de l'insecte, ainsi que la position de la source de nourriture (`nourX` et `nourY`) pour calculer leur signal. Ici, ces valeurs sont données lors de la construction du neurone sensoriel. Une autre option aurait consisté à donner ces valeurs après la construction, au moyen de méthodes spécifiques. La méthode `agit()` du neurone sensoriel aurait alors commencé par vérifier que les données avaient bien été communiquées, avant d'effectuer ses calculs.

B2 Avec la hiérarchie présentée ci-dessus, la classe `SystNerv` peut être :

```
import java.util.*;

class SystNerv {

    public static void main(String args[]) {

        Random r = new Random();

        // Position de l'insecte
        int xi = r.nextInt(1000);
        int yi = r.nextInt(1000);

        // Orientation de l'insecte
        int orx[] = {-1, 0, 1, 0};
        int ory[] = { 0, 1, 0,-1};
        int or = r.nextInt(4);
        int xd = orx[or];
        int yd = ory[or];

        // Position de la nourriture
        int xn = r.nextInt(1000);
        int yn = r.nextInt(1000);

        // Création des neurones
        Neurone[] neur = new Neurone[4];
        neur[0] = new NeurS(true, xi,yi,xd,yd,xn,yn);
        neur[1] = new NeurS(false,xi,yi,xd,yd,xn,yn);
        neur[2] = new NeurM(true, (NeurS)neur[0],(NeurS)neur[1]);
        neur[3] = new NeurM(false,(NeurS)neur[0],(NeurS)neur[1]);

        // Afichage de la situation initiale
        System.out.println("Insecte en "+xi+";"+yi+", direction = ("+xd+";"+yd+")");
        System.out.println("Nourriture en "+xn+";"+yn);

        // Fonctionnement des neurones, phase d'orientation
        for ( int n=0; n<4; n++ ) {
          neur[n].agit();
        }

        // Affichage des impulsions envoyées aux pattes gauche et droite
        System.out.println(((NeurM)neur[2]).affiche()+" "+((NeurM)neur[3]).affiche());

        // Fonctionnement des neurones, phase d'avancée
        for ( int n=0; n<4; n++ ) {
          neur[n].agit();
        }

        // Affichage des impulsions envoyées aux pattes gauche et droite
        System.out.println(((NeurM)neur[2]).affiche()+" "+((NeurM)neur[3]).affiche());

    }

}
```

NB L'orientation de l'insecte est réalisée par une méthode différente de celle utilisée dans la partie A. L'algorithme écrit ici présente l'avantage de donner la même probabilité à toutes les directions. Bien sûr, cette méthode aurait pu remplacer celle qui est présentée pour la question A1, et à l'inverse, on aurait pu reprendre ici la méthode donnée en A1.

Dans la partie B, la classe `SysNerv` se comporte comme un système nerveux isolé, qu'on aurait prélevé sur l'insecte. Elle ne fait qu'afficher les impulsions envoyées aux muscles, mais aucun mouvement n'est vraiment effectué. En particulier, la direction de l'insecte n'est pas réellement modifiée par l'action des neurones moteurs lors de la phase d'orientation. En conséquence, les neurones sensoriels reçoivent les mêmes *stimuli* lors de la phase d'avancée qui suit, ce qui ne serait pas le cas si l'insecte tournait en réponse aux impulsions musculaires. Mais le résultat de la phase d'avancée est le même quelque soit l'orientation de l'insecte. Dans la mesure où l'on ne fait fonctionner ce système que pendant un cycle, les signaux émis sont les mêmes que ceux qu'on obtiendrait si le système nerveux agissait effectivement sur l'insecte. Cette interaction fait justement l'objet de la partie C.

C1 Pour que le système nerveux soit intégré à l'insecte, il faut :

- que la classe `SystNerv` ne soit plus exécutable. Elle n'a donc plus de méthode `main()`. Elle possède comme attribut le tableau de quatre neurones. Ses instructions sont séparées en deux méthodes :
  - un constructeur, qui initialise la position de l'insecte, son orientation et la position de la nourriture à partir de paramètres, et qui crée les quatre neurones ;
  - une méthode `fonctionne()`, qui actionne le système nerveux sur une phase, en appellant les méthodes `agit()` de tous les neurones (*cf* les boucles de la version exécutable) ;
- que la classe `Insecte` contienne une instance de `SystNerv`. Pour l'utiliser, on crée une méthode `bouge()`, qui appelle la méthode `fonctionne()` du `SystNerv`, et selon son résultat, appelle
  - `tourneAGauche()` si le `SystNerv` renvoie AR,
  - `tourneADroite()` si le `SystNerv` renvoie RA,
  - `avance()` si le `SystNerv` renvoie AA,
  - rien dans les autres cas (en particulier si le `SystNerv` renvoie 00).
- que la classe `SystNerv` puisse prendre connaissance des rotations et déplacements de l'insecte (pour les transmettre aux `NeurS`). Il y a au moins deux possibilités pour obtenir ce résultat :
  - Ajouter dans `SystNerv` une référence vers l'insecte dans lequel il se trouve. Dans ce cas, la classe `SystNerv` ne possède plus de variables pour la position et l'orientation, mais interroge l'`Insecte` lorsqu'elle a besoin de connaître ces données ;
  - Ajouter à la classe `SystNerv` une instruction de mise à jour des informations de position et d'orientation. Dans ce cas, il faut que les méthodes de rotation et de déplacement de l'insecte fassent systématiquement appel à l'instruction de mise à jour après avoir modifié l'une des données.

  Dans les deux cas, il faut que la classe `Insecte` possède deux attributs de plus : les coordonnées de la source de nourriture (qu'elle transmet à son `SystNerv`). Comme noté auparavant, les valeurs de ces deux coordonnées peuvent être affectées au moment de la construction de l'`Insecte`, ou en utilisant une méthode spécifique.

C2 Les modifications de la classe `Champ` sont plus restreintes. Il suffit de :

- créer la nourriture en un point aléatoire et transmettre ces données à l'insecte ;
- appeler la méthode bouge() de l'insecte tant que sa position n'est pas identique à celle de la nourriture.

Voici ce que cela peut donner :

```
import java.util.*;

class Champ {
    public static void main(String args[]) {
      Random r = new Random();
      int leo = 1000;
      int lns = 1000;
      int xn  = r.nextInt(leo);
      int yn  = r.nextInt(lns);
      Insecte bug = new Insecte(leo,lns,xn,yn);
      System.out.println("Nourriture en "+xn+";"+yn);
      while ( ! ( (bug.getX()==xn) && (bug.getY()==yn) ) ) {
          bug.bouge();
          bug.affiche();
      }
  }
}
```