TP Java sur Eclipse – découverte du simulateur de robotique Simbad

- partie 2 -

Objectifs du TP

- Etudier la création de l'environnement dans lequel évolue les robots. Automatiser cette création par lecture d'un fichier de configuration.
- Réaliser et programmer des comportements d'anticollision des robots avec les objets de l'environnement.

Travail à réaliser

Préparation d'un nouveau package

- Créer un nouveau package tp2 dans votre projet
- Dupliquer (copier-coller) les fichiers java du premier tp (package **ex1**) dans le package **tp2** ; ainsi vous pourrez modifier ces fichiers dans le nouveau package
- Quelle action a effectué Eclipse durant la copie ?
- Changer le nom de la classe 'Exemple1' du tp1 en 'Exemple2' dans 'Tp2' en utilisant la commande 'Refactor/Rename' du menu contextuel qui apparaît par clic droit sur le nom du fichier.

Création d'un environnement

- Dans le tp précédent, il n'y avait pas de murs pour délimiter l'espace de déplacement. On va créer des murs qui délimitent l'espace et un cube en plus dans l'environnement.
- Dans la classe **MyEnv**, rajouter les instructions suivantes pour rajouter ces objets :

```
Wall w1 = new Wall(new Vector3d(9, 0, 0), 19, 1, this);
        w1.rotate90(1);
      add(w1);
Wall w2 = new Wall(new Vector3d(-9, 0, 0), 19, 2, this);
      w2.rotate90(1);
      add(w2);
Wall w3 = new Wall(new Vector3d(0, 0, 9), 19, 1, this);
      add(w3);
Wall w4 = new Wall(new Vector3d(0, 0, -9), 19, 2, this);
      add(w4);
Box b1 = new Box(new Vector3d(-6, 0, -3), new Vector3f(1, 1, 1),
            this);
      add(b1);
```

  A quoi doit correspondre la variable this dans le constructeur de Wall ? Où doit-on placer ces lignes de code ?

- Grâce à la présence de sonars sur le Robot, réaliser un comportement simple d'évitement d'obstacle pour l'évitement d'obstacles.
  Utiliser pour cela la méthode **oneHasHit()** de la classe **RangeSensorBelt** utilisée par les sonars pour la détection d'obstacles, et les méthodes de la classe MyRobot

setTranslationalVelocity(x) où x désigne une valeur de vitesse qui peut être positive ou négative, ainsi que la méthode **setRotationalVelocity(y)** où y désigne un angle de rotation positif ou négatif exprimé en radians (par exemple 'Math.PI / 4' qui correspond à 45°) qu'effectuera le robot par seconde de simulation.

Utilisation du débogueur d'eclipse

- Eclipse possède un débogueur intégré qui vous permet de suivre la pile d'exécution et la mémoire de vos variables pas par pas. (cf. diaporama du cours).
- Utiliser ce débogueur en plaçant un point d'arrêt puis en avançant pas à pas dans l'exécution à partir de ce point. Explorer le contenu des variables et la référence sur les objets pour vérifier leur création.

Utilisation d'un fichier de configuration pour générer l'environnement

On se propose ici de générer les éléments fixes de l'environnement (murs, boîtes, arches) via un fichier de configuration qui comporte le type d'objet (Wall 'W', Box 'B', Arch 'A'), le positionnement de l'objet (les trois valeurs de coordonnées (x, y, z) avec y à 0) et sa taille (diffère selon le type d'objet).
Il faut préciser que l'environnement est un carré de 20x20 dont l'origine est au centre.

- Créer un nouveau dossier **donnees** dans le package de votre projet tp2 (dans le menu contextuel : /New/Folder) où vous placerez votre fichier de configuration **myenv.txt** défini par la suite...
- Proposer un fichier type avec une ligne par enregistrement. Définir le format de chaque type d'enregistrement (correspondant à chaque type d'objet). Composer un premier fichier simple **myenv.txt** en reprenant les données d'environnement définis plus haut (murs d'enceinte et une boîte) dans le constructeur de la classe. On rajoutera une donnée en fin de ligne pour préciser la rotation de l'objet à effectuer après sa création (on utilisera 0 ou 1 pour spécifier qu'il n'y a pas de rotation ou une rotation de 90° respectivement : **w.rotate90(1)**)

  Ex : pour les murs, on doit avoir une ligne d'enregistrement du type (premier mur de l'environnement précédent) --le caractère de séparation est un espace blanc (ici on tourne de 90°)-- :

```
W 9 0 0 19 1 1
```

- Compléter le constructeur de la classe **MyEnv** qui doit lire le fichier de configuration de l'environnement pour construire celui-ci. Ainsi on aura ensuite qu'à compléter/modifier ce fichier de configuration pour avoir divers espaces d'évolution de nos robots.

- Pour gérer la liste des divers objets constituant l'environnement, on utilisera une collection de type tableau associatif (type HashMap) nommée **envConfig** (qui sera un attribut privé de la classe **MyEnv**) de la façon suivante :
  L'idée est ici d'associer 3 listes (type ArrayList) aux 3 types d'objets.
  NB : on utilisera ici un seul type générique pour définir la liste des objets manipulés dans la HashMap, englobant ces trois types d'objets différents, à savoir le type *BlockWorldObject*, qui est la classe mère des classes *Wall*, *Box*, et *Arch*.
  Les clés (chaînes de caractères, type String) seront donc les suivantes : 'Wall' pour les murs, 'Box' pour les boîtes, et 'Arch' pour les arches), et les valeurs associées seront des listes d'objets de type *BlockWorldObject*.

A la lecture de votre fichier de configuration, vous construirez et remplirez ce tableau associatif par la création des listes et des objets à construire. et à réunir l'ensemble dans une HashMap à trois clés d'entrée 'Wall', 'Box', et 'Arch' associées aux listes correspondantes. Vous pourrez utiliser notamment une syntaxe switch/case pour analyser le fichier, dont on donne un petit rappel d'utilisation ci-dessous avec des chaînes de caractères :

```
int month = 2;
String monthString;
        switch (month) {
            case 1:  monthString = "January";
                     break;
            case 2:  monthString = "February";
                     break;
            case 3:  monthString = "March";
                     break;
            case 4:  monthString = "April";
                     break;
            case 5:  monthString = "May";
                     break;
            case 6:  monthString = "June";
                     break;
            default: monthString = "Invalid month";
                     break;
        }
```

Vous aurez également besoin les méthodes suivantes :

- `public String[] split(String separateur)`
  sépare une chaîne de caractère sur laquelle elle est appliquée selon le caractère 'separateur' en un tableau de chaînes de caractères découpées par ce séparateur ; ici il s'agit d'un espace blanc.
- Double.valueOf(String s):Double
- Float.valueOf(String s):Float
- Integer.valueOf(String s):Integer

A noter que le constructeur Vector3d(double, double, double) attend des double, Vector3f(float, float, float) des Float et que rotate90(int) attend un entier.

Pour finir, vous ajouterez les objets ainsi créés dans chaque liste à l'environnement dans une boucle de lecture de chacune de ces listes à partir du tableau **envConfig**.

Ecrire le code correspondant, tester et déboguer sur Eclipse.

Tester d'autres configurations d'environnement avec le même comportement de robot.

Programmer un groupe de robots

Amélioration de la détection d'obstacles

- Dans le tp précédent, on a utilisé une méthode de détection assez sommaire. Nous allons améliorer cette détection.
- Dans la classe MyRobot, on ajoute des capteurs de type 'sonar' dans le constructeur de la classe (cf. TP1) :

```
        sonars = RobotFactory.addSonarBeltSensor(this,8);
```

- Grâce à la présence de ces sonars sur le Robot, on réalise le comportement suivant pour l'évitement d'obstacles :

```
if (sonars.oneHasHit()){
            double left = sonars.getFrontLeftQuadrantMeasurement();
            double right = sonars.getFrontRightQuadrantMeasurement();
            double front = sonars.getFrontQuadrantMeasurement();

                 if ((front > 0.5)||(left > 0.5)||(right > 0.5)) {
                     if (left < right)
                         setRotationalVelocity(-1);
                     else
                         setRotationalVelocity(1) ;
                 }
             }

if (this.collisionDetected()){
            setTranslationalVelocity(-1.0);
                  setRotationalVelocity(0.5) ; }
```

Où doit-on placer ce code ?
De quelle classe proviennent les méthodes utilisées (utiliser l'aide contextuelle d'Eclipse pour cela)?

Interpréter ce que réalise ce code et compléter la javadoc de la méthode utilisée, en inscrivant '/**' puis retour chariot, une ligne au-dessus de la méthode .

Réalisation d'une flottille de robots identiques

On souhaite mettre un ensemble de robots (10 au maximum) se déplaçant collectivement dans le même environnement, comportant tous la méthode de détection ci-dessus.
Créer une liste de robots (de type ArrayList) dans la classe MyEnv, nommée **listeRobots** dont vous pourrez mettre le nombre total dans le fichier de configuration MyEnv.txt qui définit l'environnement avec une ligne supplémentaire de la forme :

```
R 10
```

Vous utiliserez la méthode statique Math.random() qui renvoie un nombre aléatoire entre 0 et 1, pour positionner les robots de façon aléatoire initialement sans qu'ils ne puissent se superposer, et pour leur donner des angles de déplacement aléatoires avec le temps.

Ecrire le code correspondant, tester, améliorer l'algorithme de détection si besoin et déboguer avec Eclipse.