Université Grenoble Alpes - L3 MIASHS

Projet Intelligence Artificielle

Documentation interne & Code source

Théo Laigle Délyan Zergua

Année 2022-2023

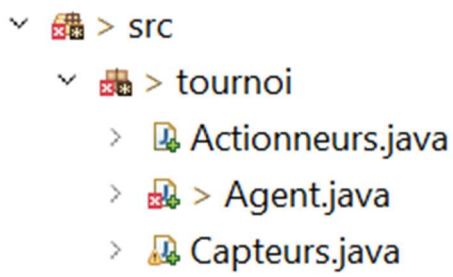
Table des matières

Introduction

Dans ce fichier, nous présentons une documentation de notre code java pour la programmation du robot Lego EV3 mindstorms.

Cette documentation vise à décrire notre projet en java, dont l'objectif est de programmer un robot en vue d'une compétition.

Tout d'abord, nous avons 3 classes dans notre programme : Agent, Capteurs et Actionneurs. Elles sont toutes définis dans le même package que l'on a appelé « tournoi ». Voici la structure que nous avons défini :



Ces classes sont conjointement construite à partir des classes prédéfinies de Lejos. Dans ces classes nous avons un certain nombre de méthodes prédéfinies qui nous sont utiles pour faire fonctionner les moteurs et les capteurs qui nous permettrons par la suite de construire les fonctionnalités du robot. Pour récupérer ces méthodes prédéfinies, nous avons réalisé un import des classes qui les contiennent. Il y a : EV3LargeRegulatedMotor, EV3ColorSensor, EV3TouchSensor et EV3UltrasonicSensor.

Pour chacune des 3 classes, nous allons offrir une description complète qui offre :

1) Rappel des objectifs de le classe
2) Relations d'utilisation avec d'autres classes
3) Lister les modules utilisés par ce module et ceux utilisant ce module
4) Définitions de types : Lister les définitions de type ou les attributs de l'objet associé au module
5) Procédures externes : Lister les procédures visibles par les modules utilisant ce module
6) Variables externes : Lister les variables visibles par les modules utilisant ce module

Description des classes

Prenons les classes dans l'ordre de leur création. Nous commençons alors par la classe Actionneurs.

La classe Actionneurs

Le but de la classe est de faire fonctionner les moteurs du robot. Puis de définir des méthodes qui pourrons être réutilisées dans la classe Agent et donc dans l'automate final.

La classe Actionneurs n'a pas de lien avec la classe Capteurs. En revanche, elle a un lien avec la classe Agent qui a besoin d'une instance de la classe Actionneurs pour fonctionner.

Les modules utilisés sont les suivants :

```
import lejos.hardware.motor.EV3LargeRegulatedMotor;
import lejos.hardware.motor.EV3MediumRegulatedMotor;
import lejos.hardware.port.MotorPort;
import lejos.hardware.port.Port;
```

Dans la classe, il y a 2 modules qui ont été utilisé :

- lejos.hardware.motor : pour importer les classes prédéfinies des différents moteurs du robot.
- lejos.hardware.port : pour importer les classes qui permettent d'établir la liaison entre les moteurs du robot et notre programme java.

Les attributs de l'objet créée :

```
protected static EV3LargeRegulatedMotor roueDroite;
protected static EV3LargeRegulatedMotor roueGauche;
protected EV3MediumRegulatedMotor clamp;
```

Ils définissent les 3 moteurs du robot. On a respectivement la roue droite, la roue gauche et les pinces. Les 3 attributs sont définis comme « protected » de manière que la classe Agent puisse les réutiliser.

Ces attributs permettent de définir le constructeur de la classe :

```
public Actionneurs(Port A, Port B, Port C){
    roueDroite = new EV3LargeRegulatedMotor(MotorPort.A);
    roueGauche = new EV3LargeRegulatedMotor(MotorPort.C);
    clamp = new EV3MediumRegulatedMotor(MotorPort.B);
}
```

Les méthodes principales de la classe :

Le travail précédent a été réalisé dans cette classe dans le but d'écrire des méthodes qui permettent au robot de réaliser des mouvements de bases avec ses moteurs. Il y a notamment :

Les méthodes pour tourner dans un sens ou dans l'autre :

```
public void rotateClockwise(int angle)
{
    roueDroite.rotate(-angle);
}

public void rotateCounterClockwise(int angle)
{
    roueDroite.rotate(angle);
}
```

La méthode pour ouvrir ou fermer les pinces :

```
public void mvt_pince(int angle) {
    clamp.rotate(angle);
}
```

Le robot ouvre ou ferme ses pinces selon que l'on choisisse un angle positif ou négatif.

Les méthodes pour avancer ou reculer :

```
public static void rouler(){
    roueDroite.startSynchronization();
    roueDroite.synchronizeWith(new EV3LargeRegulatedMotor[]{roueGauche});
    roueGauche.forward();
    roueDroite.forward();
    roueDroite.endSynchronization();
}

public void reculer(){
    roueDroite.startSynchronization();
    roueDroite.synchronizeWith(new EV3LargeRegulatedMotor[]{roueGauche});
    roueGauche.backward();
    roueDroite.backward();
    roueDroite.endSynchronization();
}
```

Elles ne prennent pas de paramètres en entrée. Il faut utiliser la méthode Delay.MsDelay(int t) à la suite de ces méthodes pour pouvoir rouler d'un certain temps.

Les classes Actionneurs et capteurs sont conjointement construite à partir des classes prédéfinies de Lejos. Dans ces classes nous avons un certain nombre de méthodes prédéfinies qui nous sont utiles pour faire fonctionner les moteurs et les capteurs qui nous permettrons par la suite de construire les fonctionnalités du robot. Pour récupérer ces méthodes prédéfinies, nous avons réalisé un import des classes qui les contiennent.

La classe Capteurs

Cette classe à pour but de définir et de faire fonctionner l'ensemble des capteurs que possède le robot.

La classe Capteurs a 2 objectifs principaux. Le premier est d'initialiser les capteurs du robot, le deuxième est de faire fonctionner ces capteurs.

La classe Capteurs n'a pas de lien avec la classe Actionneurs. En revanche, elle a un lien avec la classe Agent qui a besoin d'une instance de la classe Capteurs pour fonctionner.

Les modules utilisés sont les suivants :

```
import lejos.robotics.SampleProvider;
import lejos.robotics.filter.MeanFilter;
import lejos.hardware.port.Port;
import lejos.hardware.port.SensorPort;
import lejos.hardware.sensor.EV3ColorSensor;
import lejos.hardware.sensor.EV3TouchSensor;
import lejos.hardware.sensor.EV3UltrasonicSensor;
```

Dans la classe, il y a 3 modules qui ont été utilisé :

- lejos.hardware.sensor : pour importer les classes prédéfinies des différents capteurs du robot.
- lejos.hardware.port : pour importer les classes qui permettent d'établir la liaison entre les capteurs du robot et notre programme java.

- lejos.robotics : Ce module permet d'importer des classes qui facilitent l'utilisation des capteurs, pour prendre des mesures… Il y a notamment SampleProvider qui est utilisée pour prendre des mesures avec chacun des 3 capteurs du robot.

Les attributs de l'objet créée :

```
protected static EV3TouchSensor touch;
protected static EV3UltrasonicSensor ultrason;
protected static EV3ColorSensor color;
```

Ils définissent les 3 capteurs du robot. On a respectivement le capteur tactile, le capteur ultrason et le capteur couleur. Les 3 attributs sont définis comme « protected » de manière que la classe Agent puisse les réutiliser.

Ces attributs permettent de définir le constructeur de la classe :

```
public Capteurs(Port S2, Port S3, Port S4){
    touch = new EV3TouchSensor(SensorPort.S2);
    ultrason = new EV3UltrasonicSensor(SensorPort.S4);
    color = new EV3ColorSensor(SensorPort.S3);
}
```

Il y a aussi d'autres attributs :

```
private static float[] bleu = {0.024506f, 0.03412f, 0.0586f};
private static float[] rouge = {0.14334f, 0.0292f, 0.02098f};
private static float[] vert = {0.0604f, 0.10274f, 0.0357f};
private static float[] noir = {0.02296f, 0.02294f, 0.0198f};
private static float[] jaune = {0.27098f, 0.19198f, 0.04724f};
private static float[] blanc = {0.30706f, 0.24374f, 0.19784f};
private static float[] gris = {0.10136f, 0.09136f, 0.07946f};
```

Ce sont des tableaux qui permettent d'identifier le code RVB de chaque couleur présente sur l'aire de jeu. Ces valeurs sont obtenues grâce à des mesures que nous avons réalisé manuellement sur l'aire de jeu. Elles vont permettre de construire la méthode qui identifie les couleurs avec le capteur couleur.

Les méthodes principales de la classe :

```
public boolean isPressed() {
    SampleProvider sp_touch = touch.getTouchMode();
    float [] sample = new float[1];
    sp_touch.fetchSample(sample, 0);
    return (int)sample[0] == 1;
}
```

Cette première méthode ne prend pas de paramètre et retourne un booléen pour savoir si le robot à un objet entre ses pinces.

```
public static String rvb() {
    SampleProvider rvb = new MeanFilter(color.getRGBMode(), 1);
    float[] sample = new float[3];
    rvb.fetchSample(sample, 0);
    double dist_min = Double.MAX_VALUE;
    String color = "";

    if(scalaire(sample, bleu) < dist_min) {
        dist_min = scalaire(sample, bleu);
        color = "bleu";
    }
    if(scalaire(sample, rouge) < dist_min) {
        dist_min = scalaire(sample, rouge);
        color = "rouge";
    }
    if(scalaire(sample, vert) < dist_min) {
        dist_min = scalaire(sample, vert);
        color = "vert";
    }
    if(scalaire(sample, noir) < dist_min) {
        dist_min = scalaire(sample, noir);
        color = "noir";
    }
    if(scalaire(sample, jaune) < dist_min) {
        dist_min = scalaire(sample, jaune);
        color = "jaune";
    }
    if(scalaire(sample, blanc) < dist_min) {
        dist_min = scalaire(sample, blanc);
        color = "blanc";
    }
    if(scalaire(sample, gris) < dist_min) {
        dist_min = scalaire(sample, gris);
        color = "gris";
    }
    return color;
}
```

Cette méthode ne prend rien en entrée et renvoie une chaîne de caractère qui donne la couleur détectée par le capteur couleur.

```
public float dist() {
    SampleProvider sp_dist = ultrason.getDistanceMode();
    float[] sample = new float[sp_dist.sampleSize()];
    sp_dist.fetchSample(sample, 0);
    return sample[0];
}

public static double scalaire(float[] v1, float[] v2) {
    return Math.sqrt (Math.pow(v1[0] - v2[0], 2.0) +
            Math.pow(v1[1] - v2[1], 2.0) +
            Math.pow(v1[2] - v2[2], 2.0));
}
```

Enfin, cette dernière méthode (dist()), ne prend rien en entrée et renvoie la distance de l'objet le plus proche dans un rayon de 20 degrés face au capteur ultrason.

La classe Agent

Le but de cette classe est de définir l'automate que nous allons utiliser pour la compétition. Nous retrouvons exclusivement les méthodes ainsi que l'automate final dans cette classe. Un objet de type Agent est donc simplement un robot qui perçoit et agit dans son environnement grâce à ses capteurs et ses actionneurs.

La classe Agent est directement lié avec les 2 autres classes. Elles récupèrent des objets des classes Capteurs et Actionneurs. Dans la classe Capteurs, un objet possède 3 fonctionnalités qui sont ses capteurs. Dans la classe Actionneurs, un objet possède 3 fonctionnalités qui sont ses moteurs.

Les modules utilisés sont les suivants :

```
import java.util.ArrayList;
import lejos.hardware.Button;
import lejos.utility.Delay;
```

Dans la classe, il y a 3 modules qui ont été utilisé :

- lejos.hardware. : pour importer la classe Button afin d'avoir un bouton pour lancer le robot au moment de la compétition.
- java.util: pour importer la classe ArrayList que l'on a beaucoup utilisé pour remplir des tableaux dans lesquels on stockait des mesures.
- lejos.utility : pour importer la classe Delay et permettre d'effectuer certaines actions pendant un temps donné.

Les attributs de l'objet créée :

```
private static Capteurs capteur;
private static Actionneurs actionneur;
```

Ils définissent les 2 atouts du robot, ses capteurs et ses moteurs. Les attributs sont définis comme « private » car, ils n'ont pas besoin d'être réutilisé dans une autre classe.

Ces attributs permettent de définir le constructeur de la classe :

```
public Agent(Actionneurs actionneur, Capteurs capteur) {
    this.actionneur = actionneur;
    this.capteur = capteur;
}
```

Il y a aussi d'autres attributs :

```
public static int speed = 500;
public static int rot_speed = 200;
public static int angleOriente = 0;
private static float dist_critique = 0.326f;
private static boolean palet = false;
```

- speed : pour jouer sur la vitesse du robot
- rot_speed : pour jouer sur la vitesse de rotation des roues
- angleOriente : pour savoir à tout moment ou se situe le robot sur le terrain.
- dist_critique : pour avoir en mémoire la distance à partir de laquelle le robot n'est plus en mesure de détecter des palets.
- palet : pour savoir si le robot à un palet entre les mains ou non.

Les méthodes principales de la classe :

Il y en a trois :

- « AllerVersPalet » qui permet de se rendre vers un palet une fois qu'il est détecté.
- « ChercherPalet » qui permet de faire un scan pour repérer les palets autour de lui et finalement s'orienter vers le palet le plus proche.
- « AllerAuBut » qui permet de se rendre au but une fois que le robot a un palet entre les pinces.

```
public void allerAuBut2() {
        double diff = 0;
    ArrayList<Float> distances = new ArrayList<Float>();
    while(Capteurs.rvb() != "blanc"){
        Actionneurs.rouler();
        float d = capteur.dist();
        if(d < 0.4){
                distances.add(d);
                while(diff < 0.1 && Capteurs.rvb() != "blanc"){
                        Actionneurs.rouler();
                    distances.add(capteur.dist());
                    diff = distances.get(distances.size()-1) - distances.get(distances.size()-2);
                }
                if(Capteurs.rvb() != "blanc") {
                        Actionneurs.stop_roues();
                        Actionneurs.rotateCounterClockwise(200);
                        Actionneurs.rouler();
                        Delay.msDelay(1500);
                        Actionneurs.stop_roues();
                        Actionneurs.rotateClockwise(170);
                        diff = 0;
                }
        }
    }
    Actionneurs.stop_roues();
    // une fois qu'on est sorti du grand while, on ouvre les pinces
    Agent.actionneur.mvt_pince(1500);
    Agent.actionneur.reculer();
    Delay.msDelay(400);
    Actionneurs.stop_roues();
}
```

La méthode AllerAuBut2 utilise 2 sens du robot. Son capteur couleur pour avancer tout droit temps que la ligne blanche n'est pas franchie. Puis son capteur ultrason pour vérifier qu'il n'a pas d'objet en face (en stockant des valeurs dans un ArrayList). Si un objet est en face du robot, il l'évite en se décalant sur la gauche. Une fois la ligne blanche adverse franchie, le robot ouvre ses pinces, recule un peu et fait une rotation de 90 degrés pour se préparer à scanner le terrain.

La méthode ne prend rien et entrée et se renvoie rien, elle se contente d'exécuter des instructions.

Ici, nous avons présenté AllerAuBut2, mais nous avions une version AllerAuBut qui prenant en compte l'angle orienté pour se situer sur le terrain.

```
public void chercherPalet(){
ArrayList<Float> distances = new ArrayList<Float>();
ArrayList<Float> distCritique = new ArrayList<Float>();
ArrayList<Integer> angleCritique = new ArrayList<Integer>();
double diff = 0;
distances.add(capteur.dist());
int angle = 0;
while(angle < 750) {
        Actionneurs.roueDroite.backward();
        distances.add(capteur.dist());
    if(diff < -0.3 || diff > 0.3) {
        distCritique.add(distances.get(distances.size()-1));
        angleCritique.add(angle);
    }
    diff = distances.get(distances.size()-1) - distances.get(distances.size()-2);
    angle = angle + Actionneurs.roueDroite.getLimitAngle();
}
Actionneurs.stop_roues();
double min = distCritique.get(0);
int angleMin = 0;
for(int j = 1; j<distCritique.size(); j++)
     if(distCritique.get(j) < min) {
        min = distCritique.get(j);
                angleMin = angleCritique.get(j);
     }
while(angle > angleMin) {
        Actionneurs.roueDroite.forward();
        angle = angle + Actionneurs.roueDroite.getLimitAngle();
        Actionneurs.stop_roues();
        System.out.println("angle limited : " + Actionneurs.roueDroite.getLimitAngle());
        Button.ENTER.waitForPressAndRelease();
}
}
```

Il y a ensuite, la méthode chercherPalet, qui consiste à effectuer un scanne du terrain pour détecter tous les palets que le robot peut voir depuis sa position initiale. Un premier balayage de 180° est réalisé, durant ce balayage, le robot prend un maximum de mesure de la distance. D'une mesure à l'autre, s'il détecte des sauts de valeurs, il récupérer la valeur la plus petite des 2, on dit alors que l'on a une distance critique. Une fois son balayage de 180° terminé, le robot en effectue un second dans l'autre sens de 180° aussi. Puis il s'arrête une fois qu'il voit la plus petit distance critique qu'il a mesuré.

La méthode n'a aucun paramètre et n'a aucune sortie, elle exécute seulement des instructions.

```
public void allerVersPalet2(){ // deuxieme version beaucoup plus simple
        while(!capteur.isPressed()) {
            System.out.println("palet detecte");
            Actionneurs.rouler();
        }
        Actionneurs.stop_roues();
        Agent.actionneur.mvt_pince(-1500);
        palet = true;
    }
```

Enfin, nous avons la méthode AllerVersPalet. Elle consiste tout simplement à avancer tout droit temps que le capteur tactile n'a pas été pressé.

La méthode n'a aucun paramètre et n'a aucune sortie, elle exécute seulement des instructions.

Nous avons aussi fait des méthodes pour tourner de 90 et de 180°. Ces méthodes nous ont été très utile dans notre stratégie finale.

```
public void tourneD90(){
        Actionneurs.rotateClockwise(400);
        }

public void tourneG90(){
        Actionneurs.rotateCounterClockwise(420);
        }


public void tourne180(){
        Actionneurs.rotateClockwise(800);
}
```

Notre automate :

Notre automate ne trouve dans le « main » de la classe Agent. Nous avons réalisé 2 automates différents qui réalise en fait les mêmes tâches mais sont codés différents selon si l'on commence la compétition à droite ou à gauche.

Nous présentons uniquement notre code java de la stratégie pour un départ côté droit :

```
        public static void main(String[] args) {

                Agent robot = new Agent(actionneur, capteur);

// ____________________ Droite ____________________

                System.out.println("go");
                Button.ENTER.waitForPressAndRelease();

                robot.allerVersPalet2();
                robot.allerAuBut2();
                robot.tourne180();


                while(Capteurs.rvb() != "jaune") {
                        robot.avancerVers("bleu");
                        robot.tourneG90();
                        Actionneurs.rotateClockwise(15);
                        //robot.allerVersPalet2();
                        while(!capteur.isPressed() && Capteurs.rvb() != "jaune") {
                                Actionneurs.rouler();
                        }
                        if(Capteurs.rvb() == "jaune")
                                continue;
                        Actionneurs.stop_roues();
                        actionneur.mvt_pince(-1500);
                        robot.tourneG90();
                        Actionneurs.rotateCounterClockwise(40);
                        robot.allerAuBut2();
                        // au lieu de faire un 180, on fait un 90 a gauche 2 fois
                        // pour reprendre la ligne bleue a la base. et donc pas se faire
                        // avoir par une autre couleur qui superpose le bleu.
                        robot.tourneG90();
                        Actionneurs.rouler();
                        Delay.msDelay(200);
                        Actionneurs.stop_roues();
                        robot.tourneG90();
                }
                Actionneurs.rouler();
                Delay.msDelay(1500);
                Actionneurs.stop_roues();
                robot.tourneD90();
```

On commence par mettre un bouton pour le début de la compétition. Les 3 premières instructions permettent de se rendre vers le premier palet puis de l'amener dans la zone adverse.

Le premier « while » qui ne s'arrête pas tant que le robot n'a pas détecté du jaune permet de récupérer tous les palets de la première ligne de couleur parallèle à la ligne blanche du but adverse.

A la fin du premier « while », on effectue 4 instructions qui vont permettre de se rendre jusqu'à la ligne noir parallèle aux lignes blanches sur le terrain.

```
while(capteur.dist() > 0.25) {
        robot.avancerVers("noir");
        Actionneurs.rouler();
        Delay.msDelay(500);
        Actionneurs.stop_roues();
        robot.tourneD90();
        Actionneurs.rotateCounterClockwise(70);
        while(!capteur.isPressed() && capteur.dist() >=0.25) {
                Actionneurs.rouler();
        }
        if(capteur.dist() < 0.25)
                continue;
        Actionneurs.stop_roues();
        actionneur.mvt_pince(-1500);
        actionneur.tourneD90();
        robot.allerAuBut2();
        // meme probleme
        robot.tourneD90();
        Actionneurs.rouler();
        Delay.msDelay(30);
        Actionneurs.stop_roues();
        robot.tourneD90();
}

robot.tourneG90();
```

Une fois arrivé à ce « while », on se retrouve au niveau de la ligne noir au milieu du terrain. Cette boucle va permettre de récupérer les palets sur la ligne noire du milieu de la même manière que la première boucle. La seule chose qui diffère est que la condition d'arrêt est une que le robot détecte une distance inférieure à 0.25.

```
while(Capteurs.rvb() != "jaune") {
        robot.avancerVers("vert");
        robot.tourneG90();
        while(!capteur.isPressed() && Capteurs.rvb() != "jaune") {
                Actionneurs.rouler();
        }
        if(Capteurs.rvb() == "jaune")
                continue;
        Actionneurs.stop_roues();
        actionneur.mvt_pince(-1500);
        robot.tourneG90();
        robot.allerAuBut2();
        //meme probleme
        robot.tourneG90();
        Actionneurs.rouler();
        Delay.msDelay(30);
        robot.tourneG90();
}
```

Enfin, on a réalisé une dernière boucle « while » qui permet de récupérer les palets sur la 3 lignes de couleurs du terrain qui est parallèle aux lignes blanches. Cette boucle fonctionne de la même manière que la première boucle de l'automate.