Projet de robotique

Plan de test

DESCOTILS Juliette
ESPINAL Miguel
GATTACIECCA Bastien
LATIFI Arita

Année universitaire 2020-2021

Sommaire

1 - Introduction

Ce document a pour objectif de garantir le fait que le logiciel respecte le cahier des charges. Pour cela, une description méthode par méthode du logiciel a été réalisée en décrivant les tests à effectuer afin de s'assurer que les méthodes remplissent bien les attendus définis au préalable. Ainsi, pour chaque méthode la description, le but, la procédure et les paramètres du test à mettre en œuvre pour vérifier le bon fonctionnement de la méthode sont renseignés dans ce document. Tout d'abord sont décrits les tests d'intégration, servant à tester les méthodes de niveau élevé, puis les tests unitaires, servant à vérifier que les méthodes de bas niveau fonctionnent.

Globalement, ce plan de test s'inscrit dans une suite de documents ayant pour finalité la documentation du projet d'intelligence artificielle. Ce projet consiste à initier les étudiants à l'intelligence artificielle en programmant un robot LEGO qui devra évoluer par lui-même dans un environnement et remplir certaines tâches. Par groupe de quatre, leur but sera de développer les fonctionnalités du robot pour qu'il soit le plus performant possible. Tout au long du projet les étudiants seront amenés à implémenter et tester de nouvelles stratégies sur le robot. Ce projet a pour finalité de confronter les robots respectifs de chaque groupe lors d'une compétition.

1.1 - Objectifs et méthodes

Dans l'ensemble, ces méthodes permettront de mettre en place une stratégie pour la compétition qui opposera deux robots. Le principe étant de ramasser le plus de palets possible pour chaque robot, il faut que le robot soit rapide et sache exactement où il se trouve sur le plateau afin de maximiser le temps de recherche des palets restants. De plus, lorsque le robot aura détecté un palet, il devra le récupérer entre ses pinces et se déplacer avec afin d'aller marquer le palet dans le camp adverse.

Voici les objectifs de chaque classe :

- La classe sample :

Un Sample peut être traduit par échantillon ou vecteur. Il s'agit des trois quantités de R G B. Un Sample utilise un MeanFilter pour moyenner le Sample perçu avec N précédents. On définit un nom pour un sample, c'est important (pour les couleurs principales) d'avoir un nom type String pour pouvoir les différencier ou les comparer entre eux facilement ensuite.

- La classe Calibreur :

Classe qui contient son propre main. Elle sera exécutée avant la compétition pour permettre à notre capteur de couleur de se calibrer sur les couleurs du plateau "le jour J". Les données sont des sample (des échantillons, ou des vecteurs) qui contiennent chacun trois valeurs flottantes : la quantité de R, G, et B. On détermine ainsi toutes les couleurs que le robot est censé pouvoir être capable de détecter en jeu. Il y en a 7 : bleu, rouge, vert, gris, jaune, blanc, noir. Le capteur de couleur renvoyant des quantités de RGB assez "peu probables" (ex : 48/47/16 pour du blanc, là où on attend 255/255/255), on utilise ma méthode des scalaires, qui consiste à déterminer la couleur actuellement perçue comme étant celle qui est la "plus proche" d'une des couleurs de la liste. Pour cela on calcule le scalaire entre la couleur perçue et chacune des couleurs de la liste, et on dit que la couleur renvoyée est celle pour qui son scalaire était le plus petit avec la couleur perçue.

- La classe Perception :

Classe qui gère les trois capteurs du robot. Le capteur tactile, qui renvoie true ou false selon que le balancier appuie dessus ou non. Le capteur de couleur qui détermine quelle couleur parmi une liste de couleurs définies est la plus proche de celle que le capteur perçoit ; on utilise pour cela la méthode des scalaires précisée plus bas. Le capteur à ultrasons qui renvoie une distance.

- La classe Actionneur :

Classe qui contient toutes les méthodes de bas niveau concernant les déplacements. Les moteurs gauche et droit sont des EV3LargeRegulatedMotor tandis que le moteur de la pince est un EV3MediumRegulatedMotor.

On utilise une instance de la classe Chassis ainsi que MovePilot pour calculer tous les mouvements dont nous avons besoin à partir de ces paramètres :

- le diamètre des roues en millimètres (l'unité des distances que nous avons choisie).
- l'offset, c'est-à-dire le décalage des roues par rapport au centre du 'segment' qui relie les deux roues.
- l' "angular speed" qui définit la vitesse du robot dans les méthodes de rotation.
- la "linear speed" qui définit la vitesse du robot dans les méthodes de déplacements avant/arrière
- la "linear acceleration" qui définit la qualité des phases d'accélération en début de mouvement et de décélération en fin de mouvement pour une vitesse moteur spécifiée.

- La classe Agent :

Classe qui utilise les classes précédentes afin d'effectuer des tâches de hauts niveaux.

- La classe AgentStrategy :

Classe qui ordonne les méthodes d'Agent, qui implémente nos différentes stratégies.

1.2 - Documents de référence

Les documents ayant servi lors de l'élaboration du présent document sont :

- le cahier des charges
- le plan de développement
- la javadoc

2 - Tests d'intégration

Décrire les tests (ainsi que leur enchaînement) permettant de vérifier que les différents modules de l'application s'interfacent correctement. On distinguera les interfaces internes des interfaces externes à l'application.

Identification	Description du test Décrire le test de façon externe	But du test Décrire ce que prouve le test	Principe de réalisation Décrire la procédure de test ainsi que les paramètres
Classe Agent			
prendrePalet(double tryDistance, int speed)	Le robot ouvre les pinces et avance à la vitesse "speed" sur la distance "tryDistance" puis les referme lorsque le capteur de toucher détecte un palet.	Le robot capture le palet entre ses pinces s'il se trouve à une distance inférieure à la distance limite donnée.	Placer le robot à 10 cm d'un palet, appeler prendrePalet(200, 200). Le robot doit ouvrir les pinces puis avancer de 10 cm. Une fois le palet détecté, le robot ferme les pinces et capture donc le palet entre ces pinces.
testDistance()	Le robot est placé successivement à différentes distances d 'une paroie puis on appuie sur entrer.	Vérifier que l'appel de cette méthode permet d'obtenir la distance entre le capteur ultrasonique et un objet.	Le robot est placé face à la paroie au croisement des lignes rouges et noires (perpendiculairement à la ligne rouge). On appelle la méthode testDistance() le robot doit donc afficher 50. Puis on déplace manuellement le robot au croisement entre les deux lignes noires, le robot doit alors afficher 100. On appuie ensuite sur entrer, le robot ne doit plus rien afficher.
testCouleur()	Le robot est placé sur différentes lignes de couleurs les unes après les autres puis on appuie sur entrée.	Le robot affiche le nom de la couleur située sous le capteur de couleur et actualise cette valeur si on déplace le robot sur une autre couleur.	Le robot est placé sur une ligne blanche. On appelle la méthode testCouleur(). puis on déplace manuellement le robot sur la ligne jaune. On appuie ensuite sur entrer.
perpendiculaire()	Le robot s'aligne à la perpendiculaire à un mur du plateau.	Le robot se positionne exactement face à un mur du plateau.	Le robot est positionné au hasard sur le plateau. Le robot commence à tourner jusqu'à ce qu'il soit perpendiculaire au mur. Il continue légèrement au-delà de la perpendiculaire puis revient légèrement en arrière.
resetDirection()	Le robot tourne jusqu'à retrouver son angle initial (angle de départ à 0)	Vérifier que l'appel de cette fonction remet le robot face au camp adverse quelque soit l'angle de départ, pour peu que l'attribut de direction soit initialisé à cet angle de départ.	On positionne le robot à 30° (et on initialise l'attribut de direction à 30), après l'appel de méthode le robot tourne sur la droite de 30° pour se remettre droit. Si l'angle de départ est 190°, alors le robot tourne de 170°.

directionNearestObject()	Le robot doit faire un tour sur lui-même puis il se dirige vers l'objet le plus proche.	Vérifier que l'appel de cette fonction permet de diriger l'avant du robot vers l'objet le plus proche.	Placer le robot sur l'intersection entre les lignes bleue et noire en l'allignant avec la ligne bleue. Placer uniquement deux palets sur le plateau : un au croisement de la ligne bleue avec la ligne rouge et un au croisement des deux lignes noires. Appeler la méthode. Le robot fait un une rotation de 360° et revient donc dans sa position initiale. Puis il fait une rotation de 90° et se retrouve donc face au palet placé au croisement des deux lignes noires, aligné avec la ligne noire.
avancerJusquaColor(String c, double tryDistance, double Speed)	Le robot doit avancer à la vitesse speed jusqu'à ce qu'il perçoive la couleur c passé en paramètre. S'il parcourt la distance tryDistance avant d'avoir détecté la couleur c, il s'arrête.	Ce test prouve que le robot est capable d'avancer jusqu'à une ligne de couleur passée en paramètre sans dépasser une distance limite.	Placer le robot à l'intersection de la ligne verte et jaune (parallèle à la ligne jaune). Appeler la méthode avancerJusquaColor("black",1200,300). Le robot devra avancer et s'arreter sur l'intersection de la ligne noire et jaune. Appeler à nouveau la méthode avancerJusquaColor("black",1200,300) le robot ne détecte pas la coueur noire puisqu'il l'a déjà dépassée, il devra s'arrêter à l'intersection entre les lignes jaune et blanche.
suivreColor(String c, int distance)	Le robot avance tant qu'il perçoit la couleur c passé en paramètre et qu'il n'a pas parcouru la distance "distance"	Ce test prouve que le robot est capable d'avancer sur une distance donnée tout en vérifiant si la couleur qu'il perçoit au niveau du capteur couleur correspond à la couleur passée en paramètre.	Placer le robot sur une extrémité de la ligne rouge. Appeler la méthode suivreColor("rouge", 2000). Le robot devrait avancer jusqu'à l'autre bout de la ligne rouge en s'arrêtant un peu avant la fin.
Identification	Description du test Décrire le test de façon externe	But du test Décrire ce que prouve le test	Principe de réalisation Décrire la procédure de test ainsi que les paramètres
La classe Calibreur			
getNearestSample (Collection<Sample> c,Sample echantillon) *échantillon = couleur perçue actuellement par le robot*	Cette méthode doit renvoyer le sample présent dans la collection des sample principaux dont le scalaire est le plus petit avec le sample en deuxième paramètre de la fonction. Ainsi, on passe en premier paramètre la liste de nos couleurs principales et en deuxième paramètre le sample de la couleur actuellement perçue.	Cette méthode à pour but de retourner le sample de la couleur actuellement perçue par le robot en comparant les scalaires des couleurs qui se trouvent dans la collection c et le sample échantillon	L'attribut sampleList a préalablement été initialisé lors du calibrage des couleurs. Appeler la méthode getNearestSample(sampleList, [0.055,0.247,0.018]) Cette méthode retourne le sample de c qui a le sample le plus proche du deuxième paramètre

3 - Tests unitaires

Identification	Mise en oeuvre	Données d'entrées	Résultat attendus	Critère d'évaluation
Classe Perception				
update()	On place le robot sur la ligne noire au niveau du croisement avec la ligne rouge. Le robot a un palet entre les pinces, les pinces sont fermées.	On appelle la méthode update() en ajoutant system.out.println(color + distance + touch).	Retourne "red 50 true".	La valeur retournée est "red 50 true"
setCalibratedSamples()	Le robot doit déjà avoir fait le calibrage au préalable. Il dispose en mémoire du fichier "sample.txt"	Appeler la méthode setCalibratedSample()	La méthode doit lire le fichier source créé par le calibrage "sample.txt" puis créer un tableau de String dont chaque case contient la valeur de R,G et B. Il sera fait de même mais dans un tableau de float en parsant les String. Ensuite chaque sample de couleur sera ajouté dans la liste des samples (sampleList)	La liste de sample sampleList contient les 7 sample des 7 couleurs principales du plateau que le robot doit être capable de discerner.
getCouleur()	On place le robot sur la ligne rouge	Appeler la méthode getCouleur()	La méthode doit retourner red	La valeur retourné doit être red
getTouche()	On place un palet dans les pinces du robot, le palet touche le capteur de toucher	Appeler la méthode getTouche()	Retourne true si le capteur de toucher est sollicité sinon false	La méthode doit retourner true
getDistance()	On place le robot à 50 cm d'un mur	Appeler la méthode getDistance()	La méthode retourne 50,00	La valeur retourné est 50,00 (+/- 2 cm)

Identification	Mise en oeuvre	Données d'entrées	Résultat attendus	Critère d'évaluation
Classe Actionneur				
getMouvement()	Positionner le robot n'importe où sur le plateau.	Appeler la méthode avancer(1000, 50, true). Appeler la méthode getMouvement() sur actionneur.	Retourne un objet de type Move correspondant au mouvement actuel (travel)	Retourne TRAVEL.
avancer(double distance, double speed, boolean nonBloquante)	Placer le robot sur la ligne noire au croisement avec la ligne blanche face au camp adverse.	Appeler la méthode avancer(600, 100, false)	Le robot doit avancer de 60 cm à une vitesse de 100mm/s et bloquer toute autre action.	Le robot est au croisement entre la ligne bleue et la ligne noire 6 secondes après le début du déplacement.
rotation(double angle, double speed, boolean nonBloquante)	Positionner le robot à l'intersection entre les lignes blanche et rouge perpendiculairement à la ligne blanche.	Appeler la méthode rotation(90, 50, false)	Le robot fait une rotation de 90° vers la gauche et se trouve donc perpendiculaire à la ligne rouge en bloquant toute autre action.	Le robot se trouve perpendiculaire à la ligne rouge (+/- 2°).
travelArc(double radius, double distance, boolean nonBloquante)	Positionner le robot au croisement entre la ligne noire et face à la ligne rouge.	Appeler la méthode travelArc(60,377,false)	Le robot avance en tournant vers la gauche, il s'arrête lorsqu'il a parcouru une distance de 377cm.	Le robot revient à sa position initiale
stop()	Positionner le robot n'importe ou sur le plateau et appeler la méthode avancer(6000, 50, true).	Appeler la méthode stop().	Le robot doit s'arrêter lorsque la commande est exécutée.	Le robot s'arrête immédiatement.
ouvrirPinces()	Positionner le robot n'importe ou sur le plateau. Ajouter system.out.print(pincesOuvertes) dans le corps de la méthode.	Appeler la méthode ouvrirPinces().	Le robot doit ouvrir ses pinces et la méthode doit mettre à jour l'attribut pincesOuvertes.	Le robot ouvre ses pinces et affiche true.
fermerPinces()	Positionner le robot n'importe ou sur le plateau. Ajouter system.out.print(pincesOuvertes) dans le corps de la méthode.	Appeler la méthode fermerPinces()	Le robot doit fermer ses pinces et la méthode doit mettre à jour l'attribut pincesOuvertes.	Le robot ferme ses pinces et affiche false.

isMoving()	Positionner le robot n'importe ou sur le plateau	Appeler la méthode isMoving()	La méthode doit retourner false car il n'est pas en mouvement	La méthode retourne false
	Positionner le robot n'importe ou sur le plateau	Appeler la méthode avancer(600, 50, true) puis la méthode isMoving()	La méthode doit retourner true car le robot est en mouvement	La méthode retourne true
pincesOuvertes()	Positionner le robot n'importe ou sur le plateau avec les pinces ouvertes	Appeler la méthode pincesOuvertes()	La méthode doit retourner true car les pinces sont ouvertes	La méthode retourne true
	Positionner le robot n'importe ou sur le plateau avec les pinces fermées	Appeler la méthode pincesOuvertes()	La méthode doit retourner false car les pinces sont fermées	La méthode retourne false
setDirection(int direction)	Positionner le robot n'importe ou sur le plateau.	Appeler les méthodes getDirection() puis rotation(90,60,false) puis getDirection() puis setDirection(0) et enfin getDirection().	La méthode doit retourner 0 puis faire une rotation de 90° puis retourner 90 puis retourner 0.	La console affiche : 0 90 0
getDirection()	Positionner le robot n'importe ou sur le plateau	Appeler la méthode rotation(90, 60, false) puis appeler la méthode getDirection().	La méthode doit retourner 90 car le robot à fait une rotation de 90° avec la méthode rotation(90,60,false).	La méthode retourne 90.
updateDirection(double angle)	Placer le robot n'importe où sur le plateau. Afficher le résultat de getDirection() sur actionneur.	Appeler la méthode rotation(380, 50, false). Afficher ensuite le résultat de la méthode getDirection().	Le résultat obtenu lors du deuxième appel à la méthode getDirection() doit afficher le nouvel angle après rotation.	La console doit afficher : 0 20
Identification	Mise en oeuvre	Données d'entrées	Résultat attendus	Critère d'évaluation
Classe Calibreur				

goMessage(GraphicsLCD g)	Placer le robot n'importe ou sur le plateau	Appeler la méthode goMessage(g)	La méthode construit le message de présentation puis le supprimer lorsqu'un bouton est touché.	La console du robot doit afficher les messages suivant : "Calibrage du ColorSensor. Verifier que le capteur de couleur est bien branche sur le Port S2. Appuyer sur entrée pour démarrer le calibrage. Appuyer sur Echap pour quitter. Let's go" Puis dès que l'utilisateur appui sur n'importe quelle bouton, le message disparaît.
scalaire(float[] v1, float[] v2)	Faire un main	Créer deux tableau de float v1=[47,47,18] et v2=[255,255,255] Puis appeler la méthode scalaire(v1, v2)	La méthode doit calculer le scalaire de deux vecteurs de même tailles et retourner un le résultat sous forme de double	La méthode retourne 378.3041104719852
Identification	Mise en oeuvre	Données d'entrées	Résultat attendus	Critère d'évaluation
Classe Sample				
calibrateColor()	Créer un Sample red = new Sample ("red") Puis placer le robot sur la ligne rouge	Appeler la méthode calibrateColor()	La méthode doit afficher sur la bric du robot "ENTRER POUR DETECTER red .." et dès qu'un bouton est enclenché la couleur qui se trouve en dessous du capteur de couleur sera callibrée comme étant du rouge.	L'attribut "sample" sera mis à jour.

4 - Références

Sayegh, C. (2018, 2 mai). Pourquoi rédiger un cahier de recette avant vos mises en production ? CloudNetCare. https://www.cloudnetcare.fr/rediger-cahier-de-recette-avant-mise-en-production/

Rodrigues, G. (s. d.). Créer des plans de tests de charge réalistes-. Developpez.com. https://arodrigues.developpez.com/tutoriels/java/performance/plan-test-realiste/

gestion-projets-informatiques.over-blog.fr. (2010, 10 février). Géstion de projets informatiques : http://gestion-projets-informatiques.over-blog.fr/article-introduction-44670796.html