Plan de test

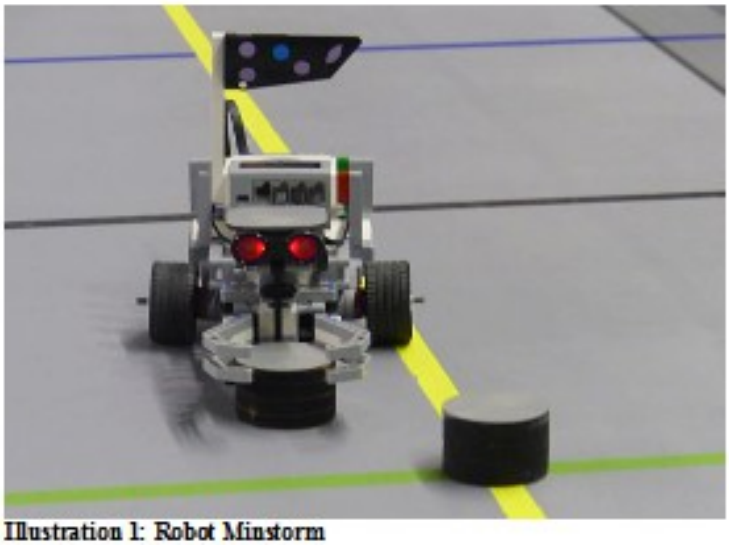

Illustration 1: Robot Minstorm

Initiation à l'intelligence artificielle

EveStandUp !

CATHELIN MARGAUX

DE BELLIS CLÉMENCE

PERRIN MATHIEU

L3 MIASHS

Table des matières

Introduction

Ce document à pour objectif d'énoncer les différents tests réalisés dans l'élaboration de notre projet afin de veiller au bon fonctionnement de chacune des méthodes et de leurs relations les unes et les autres .

Afin de programmer au mieux le robot, nous avons construit plusieurs tests d'intégrations qui avaient pour but de le mettre d'observer son comportement en condition de compétition. Avant cela, de nombreux tests unitaires ont été effectués pour expérimenter les différentes méthodes avant de les inclure dans le programme final.

Nous allons tout d'abord présenter les tests unitaires de chaque méthodes, les puis nous aborderons les tests d'intégrations.

1.Tests unitaires

Il faut savoir que tous les tests unitaires sont réalisés dans des classes à part qui n'apparaissent pas dans le code final.
Ces tests sont nécessaires au bon fonctionnement du code car de nombreuses méthodes font appel à d'autres. Il est donc impératif que les méthodes fonctionnent sans erreur.

1.1.Classe «DuoDeRouesSynchro»

1.1.1.Méthode setspeed(int vitesse)

Cette méthode permet de changer la vitesse des deux moteurs, afin qu'elle soit la même pour les deux (nécessaire pour la synchronisation des roues). Cela permet aussi de régler une vitesse pour les situations appropriée (il faut être moins rapide lors de la recherche du palet et plus rapide pour aller le chercher).

La phase de test consiste à varier la vitesse pour voir si la véhicule était plus ou moins rapide en fonction, d'abord en changeant la valeur de l'attribut SPEED, puis en utilisant la méthode setspeed().

1.1.2.Méthode avancer()

Cette méthode permet de faire tourner les deux moteurs sans fin de manière synchronisée (de manière à faire avancer le robot) afin de pouvoir l'arrêter dans le programme principal lorsque nécessaire.

Le test consiste à faire avancer les deux moteurs et de vérifier s'ils démarrent bien en même temps. Une méthode prédéfinie a d'abord été utilisée, mais étant infructueuse, elle a était remplacée par les méthode startSynchronization() et stopSynchronization().

Les tests ont pour objectif de vérifier le bon enchaînement des instructions.

1.1.3.Méthode stop()

Cette méthode permet d'arrêter les deux moteurs de manière synchronisée, afin de rester le plus droit possible.

Le test consiste à faire stopper les moteurs et vérifier qu'ils s'arrêtent bien en même temps. Une méthode prédéfinie a d'abord été utilisée, mais étant infructueuse, elle a était remplacée par les méthode startSynchronization() et stopSynchronization().

Tout comme les méthodes avancer(), reculer() les tests ont pour principe de contrôler que la méthode fonctionne correctement sans faire varier quoi que ce soit.

1.1.4.Méthode reculer()

Permet de faire tourner les deux moteurs sans fin de manière synchronisée (de manière a faire reculer le robot afin de pouvoir l'arrêter dans le programme principal lorsque nécessaire.

Le test consiste à faire avancer les deux moteurs et de vérifier si les roues commencent bien leur rotation en même temps bien en même temps. Une méthode prédéfinie à d'abord été utilisée, mais étant infructueuse elle a était remplacée par les méthode startSynchronization() et stopSynchronization()

1.1.5.Méthode reculerTemps(int temps)

Permet au robot de reculer pendant un temps donné. Utilisé principalement pour reculer après ne pas avoir détecté un palet, mais un mur ou un autre robot.

Quelques tests ont été faits pour savoir quelle distance était parcourue sur le temps donné.

1.1.6.Méthode rotateAsynchG(int angle, int temps) et Méthode rotateAsynchD(int angle , int temps)

Cette méthode fait tourner de façon asynchrone le véhicule, ce qui lui permet de tourner tout en scannant ce qu'il y a autour de lui. La gauche ou la droite est déterminée en fonction de quel côté de la table de jeu le robot démarre, afin d'être le plus efficace.

Les tests sont principalement faits sur rotateAsynchG(int angle, int temps), afin de pouvoir tester si le robot peut tourner tout en détectant les distances. Lors de ces tests, il faut faire varier les paramètres angle et distance. Les résultats sont ensuite mis en miroirs pour créer la méthode rotateAsynchD(int angle, int temps).

1.2.Classe «Vehicule»

1.2.1.Méthode AvancerTantQue(double distance, LAvue vue)

Fait avancer le robot jusqu'à une distance donnée en paramètre, la paramètre vue étant le capteur à ultrason, pouvant s'actualiser pendant le déplacement du robot. Permet de s'avancer suffisamment vers un mur ou un palet afin de les différencier.

De nombreux tests ont étaient exécutés afin de réussir à en même temps se déplacer et capter la distance avec le capteur à ultrason. Lors de ces tests, il faut faire varier la distance. Les résultats sont que le robot s'arrête à la distance désirée.

1.2.2.Méthode PALET()

Permet de mettre en place toutes les étapes pour détecter un palet**.**

Une première valeur est stockée pour la distance qui le sépare de l'objet à la base, puis le robot va avancer vers l'obstacle. Il va récupérer la nouvelle distance qui le sépare de l'objet. Si la différence de la nouvelle distance et l'ancienne distance est négative cela signifie qu'il s'est rapproché d'un mur, le robot va donc reculer. Si c'est l'inverse, c'est que le robot s'avance vers un palet et qu'en avançant, il sort de son champ de vision ce qui signifie qu'il a bien détecté un palet. Il renvoie True s'il détecte un palet, et False sinon.

Beaucoup de tests ont été effectués pour savoir la distance qu'il fallait parcourir par le robot pour être sûr que différencier un mur ou un autre robot, d'un palet. Il a fallu aussi mettre en place la distance sur laquelle il va reculer en cas d'échec.

1.2.3.Méthode DetectionDunObjetG(captTactile capt, chat cotes) et Méthode DetectionDunObjetD(captTactile capt, chat cotes)

Le robot tourne sur lui-même grâce à la méthode rotateAsynchG(int angle, int temps) ou rotateAsynchD(int angle, int temps). Il va tourner sur lui-même jusqu'à détecter le premier objet à portée. Il va s'avancer vers lui et vérifier si oui ou non c'est un palet avec la méthode PALET(). Il va ensuite le récupérer et retourner true, ou renvoyer false sinon.

Peu de tests ont été effectués pour ces deux méthodes.

1.3.Classe « TestColor »

Tous les tests de cette classe ont dû être réalisés sur la table de jeu.

1.3.1. Méthode colorimétrie()

La méthode de colorimétrie consiste en elle-même à un test. Cette méthode consiste à échantillonner les couleurs du terrain de jeu afin de stocker les valeurs en RVB (Rouge Vert et Bleu) de chaque couleur. Une fois que cette méthode a été faite et que les valeurs sont stockées dans le fichier la méthode n'est plus ré exécuté.

Il n'y a pas de test unitaire dans cette méthode.

1.3.2.Méthode getEchantillon()

Cette méthode consiste à stocker dans un tableau de float la couleur actuellement vue par le robot.

Les tests de cette méthode consistent à afficher en continue (dans une boucle) les valeurs de la couleur captée. Le principe est de vérifier que le robot capte les bonnes couleurs. Les tests de cette méthode ne donnent pas suffisamment d'informations, car la couleur perçue est affichée avec un float. La vérification de la méthode est surtout faite dans la méthode getColor() car les couleurs sont retournées avec une chaîne de caractère.

1.3.3.Méthode getColor(Properties prop , float[]sample)

Cette méthode retourne la couleur que le robot perçoit.

Les tests consistent à afficher dans une boucle, les couleurs obtenues avec cette méthode, par exemple : *System.out.println(TestColor.getColor(colore,tab)) ;*

Les valeurs affichées sont correctes.

1.3.4.Méthode posePaletCamp(Properties prop , Vehicule vehicule, CaptTactile capt, char cotes, Boussole boussole)

Cette méthode a pour objectif de diriger le robot vers le camp afin qu'il y dépose le palet lorsqu'il atteint la ligne blanche. Pour réaliser cette méthode, différents tests sont effectués.

Le premier a été de tester l'arrêt lors de la ligne blanche.

Ensuite, il a fallu tester le fait que le robot s'arrête lorsqu'il voit un obstacle. Si l'obstacle est un mur, c'est-à-dire que la distance ne bouge pas durant une période alors que le robot ne se déplace pas alors il se réoriente pour repartir vers le camp mais si la distance évolue alors il repart dans la même direction afin d'atteindre le camp. Il a faut donc tester la distance à laquelle il doit s'arrêter s'il croise un autre robot. Il a faut ensuite tester le temps d'attente entre les deux prises des distances (l'ancienne puis la nouvelle). Nous testons ensuite la réaction du robot face à un mur ou face à un robot afin de corriger notre code.

Une fois ces petits tests terminés, nous testons la méthode en entier dans différentes circonstances :
- le robot se dirige vers le camp sans aucun obstacle ⇒ le résultat attendu est que le robot dépose correctement le palet dans le camp adverse.
-le robot se dirige vers son camp, mais rencontre un autre robot qui lui dévie sa trajectoire ⇒ le résultat attendu est que le robot attende puis redémarre pour atteindre le but adverse.
-le robot se dirige vers son camp, mais rencontre un autre robot qui ne bouge pas non plus, il est donc assimilé à un mur ⇒ le résultat attendu est que le robot dévie sa trajectoire afin d'atteindre le but adverse.
- le robot se dirige vers un mur ⇒ le résultat attendu est que le robot s'arrête devant le mur attend un court instant avant de recalculer la distance qui le sépare du mur puis se réoriente vers le camp à atteindre.

1.4.Classe « CaptTactile »

1.4.1.Méthode isPressed()

Cette méthode est testée en affichant en continu (dans une boucle tant que) l'état du capteur tactile. Il faut donc parfois l'activer puis le relâcher afin de vérifier qu'il est bien affiché true lorsqu'on presse le capteur.

1.4.2.Méthode OuvertureDesPinces() et Méthode FermetureDesPinces()

Elles ont simplement été essayées avec différents délais d'attente, afin d'être en bonne condition pour récupérer le palet et le ramener à la base. .
Les résultats attendus sont que le robot ouvre ou ferme plus ou moins ses pinces en fonction du délai d'attente, mis en place pour pouvoir s'ouvrir et se fermer sur-mesure.

1.4.3.Méthode avancerJusquePalet()

Une fois les tests de la méthode isPressed() effectués, lors de l'appel de cette méthode les tests ont pour but de faire varier les délais d'attente.

1.4.4.Méthode recupPremierPalet()

Cette méthode est créée afin de récupérer rapidement le premier palet. Elle consiste à trouver le premier palet puis le robot dévie sa trajectoire afin de ne pas rencontrer d'autres palets puis s'arrête dans le camp adverse. Cette méthode est liée à d'autres méthodes, les tests sont donc expliqués dans la partie des tests d'intégration.

1.5.Classe « Boussole»

1.5.1.Méthode orienter(int angle)

Cette méthode additionne la valeur de l'orientation avec la valeur passée en paramètre et converti le résultat en angle.

données d'entrées : orientation = 0, angle=90 ⇒ résultat attendu:90
données d'entrées : orientation = 270, angle=180 ⇒ résultat attendu:90

Initialise l'attribut orientation à une valeur et additionne orientation avec la valeur passée en paramètre. Le test vérifie que la méthode calcule bien la somme des angles (valeur entre 0 et 360 degrés) et n'attribue pas à orientation une valeur qui n'est pas en degré. Les premiers tests renvoyaient des sommes sans convertir en angle. Après modification de la méthode avec l'ajout d'un modulo, les tests étaient réussis.

1.5.2.Méthode nouvelAngle(int angle)

Cette méthode met à jour l'attribut orientation avec la valeur passée en paramètre et retourner l'écart entre l'angle de base et la nouvelle valeur.

données d'entrées : orientation = 0, angle=90 ⇒ résultat attendu:90, orientation=90
données d'entrées : orientation = 90, angle=0 ⇒ résultat attendu:-90, orientation=0
données d'entrées : orientation = 180, angle=270 ⇒résultat attendu:90, orientation=270

Vérifie que la méthode retourne la bonne différence entre les 2 angles : angle négatif si le premier est inférieur au second et positif si inversement.

1.5.3. Méthode trouverNord(Avancer t)

Cette méthode fait tourner le robot en direction du nord.

donnée d'entrée : orientation=0 ⇒ résultat attendu: tourne de 90°.
donnée d'entrée : orientation = 180 ⇒ résultat attendu: tourne de -90°.
donnée d'entrée : orientation = 270 ⇒ résultat attendu: tourne de -180°.
donnée d'entrée : orientation = 300 ⇒résultat attendu: tourne de -210°.
donnée d'entrée : orientation = 90 ⇒résultat attendu: ne tourne pas.

Vérifie que le robot tourne bien en direction du nord peu importe la position dans laquelle il se trouve. Le souci est que parfois, il ne choisit pas le chemin le plus rapide, mais les tests sont quand même tous réussi.

1.5.4.Méthode trouverSud(Avancer t)

Cette méthode fait tourner le robot en direction du sud.

donnée d'entrée : orientation=0 ⇒résultat attendu: tourne de 270°.
donnée d'entrée : orientation = 180 ⇒résultat attendu: tourne de 90°.
donnée d'entrée : orientation = 270 ⇒résultat attendu: ne tourne pas.
donnée d'entrée : orientation = 300 ⇒résultat attendu: tourne de -30°.
donnée d'entrée : orientation = 90 ⇒résultat attendu: tourne de 180°.

Vérifie que le robot tourne bien en direction du sud peu importe la position dans laquelle il se trouve. Le souci est que parfois, il ne choisit pas le chemin le plus rapide, mais les tests sont quand même tous réussi.

1.5.5.Méthode trouverOuest(Avancer t)

Cette méthode fait tourner le robot en direction de l'ouest.

donnée d'entrée : orientation=0 ⇒ résultat attendu: ne tourne pas.
donnée d'entrée : orientation = 180 ⇒ résultat attendu: tourne de 180°s.
donnée d'entrée : orientation = 270 ⇒ résultat attendu: tourne de -270°.
donnée d'entrée : orientation = 300 ⇒ résultat attendu: tourne de -300°.
donnée d'entrée : orientation = 90 ⇒ résultat attendu: tourne de 90°.

Vérifie que le robot tourne bien en direction de l'ouest peu importe la position dans laquelle il se trouve. Le souci est que parfois, il ne choisit pas le chemin le plus rapide, mais les tests sont quand même tous réussi.

1.5.6.Méthode trouverEst(Avancer t)

Cette méthode fait tourner le robot en direction de l'est.

donnée d'entrée : orientation = 0 ⇒ résultat attendu: tourne de 180°.
donnée d'entrée : orientation = 180 ⇒ résultat attendu: ne tourne pas.
donnée d'entrée : orientation = 270 ⇒ résultat attendu: tourne de -90°.
donnée d'entrée : orientation = 300 ⇒ résultat attendu: tourne de -120°.
donnée d'entrée : orientation = 90 ⇒ résultat attendu: tourne de 90°.

Vérifie que le robot tourne bien en direction de l'est peu importe la position dans laquelle il se trouve. Le soucis est que parfois, il ne choisit pas le chemin le plus rapide, mais les tests sont quand même tous réussir.

2.Tests d'intégration

Après avoir expérimenté les différentes méthodes une à unes, il a fallu réaliser différents tests afin de vérifier les relations entre méthodes.

2.1.Tests des liens entres méthodes

Les tests de la méthode recupPremierPalet() est présent dans cette section étant donné qu'elle est le résultat de différentes méthodes. En effet cette méthode est un ensemble d'autres méthodes.

Il faut vérifier que les méthodes s'enchaînent sans difficulté.
Une fois la méthode crée les tests ont pour but que le robot soit le plus précis possible. Par

exemple, un des tests principal est de faire varier la distance dans la méthode AvancerTantQue() afin de voir si la distance entre le robot et le palet ou entre le robot et le mur est la plus optimale. Ensuite, il a fallu tester les rotations tout d'abord sans boussole étant donné qu'au départ la boussole n'était pas créée. Ces tests avaient pour but de déterminer la valeur de ces rotations. La boussole a ensuite été rajoutée et par manque de temps seulement un test pour s'assurer de la bonne intégration de la boussole dans la méthode.

2.2.Tests du programme principal

L'objectif du programme principal était de faire appel à toutes les méthodes créé ultérieurement.

Celui-ci prendra en compte le côté de départ du robot, afin de pouvoir s'adapter aux différentes situations.

Le programme commence par la récupération du premier palet, qui est fixe puisque l'on peut choisir notre point de départ, et son déplacement jusqu'à la zone d'en-but adverse. Une fois cela fait, le robot va commencer à détecter ce qu'il y a autour de lui, en choisissant son sens de rotation en fonction du côté de départ. S'il croise trop de mur (au bout de 3) il change de sens de rotation afin de changer son approche. S'il croise assez de palet, il s'adapte en changeant de sens de rotation, ayant ramassé les palets à portés dans ce sens-là.

La plupart des tests ont était fait sur la coarticulation de toutes ces méthodes, et du calibrage avec la boussole. Le robot devait effectuer les bonnes rotations en fonction du côté qu'il avait comme point de départ, du nombre de palets récupérés et du nombre de murs rencontrés, tout en évitant de rentrer dans les autres robots.

3.Bibliographie

Illustration 1 :Campus de Grenoble. (2016). *Challenge PersyCup 2016 | Place Gre'net* [Photo]. Consultéà l'adresse https://www.placegrenet.fr/2016/05/22/challenge-persycup-2016-bataille-de-robots-campus-degrenoble/90349/p1080627_opt