Plan de tests

https://fr.123rf.com/photo_110580265_bangkok-thailand-august-4-2018-robot-controls.html?vti=o7aeg4akz9luvdjvtz-1-80

Initiation à l'intelligence artificielle
2021/2022

Belaikous
Dong
Parouty
Vallois

Sommaire

I/ Introduction

Ce document présente quelques tests unitaires et d'intégration prévus pour évaluer l'efficacité du code sur les comportements du robot.

II /Tests unitaires

1/Capteurs
Tests relatifs au capteur UltraSonic

getDistance()
Pour ce test, on met le capteur à ultrasons face à des objets à différentes distances et l'on prend connaissance de la valeur retournée grâce à un System.out.print(). Il faut préalablement mesurer (à l'aide d'un mètre ou d'une règle) la distance entre l'objet et le capteur. Cette distance et celle mesurée sont ensuite comparées. L'objectif est d'évaluer l'efficacité à retourner la distance entre le capteur à ultrasons et l'objet le plus proche et la précision de cette mesure.
Le résultat attendu est un float correspondant à une valeur très proche (1 cm de marge) de celle mesurée préalablement entre l'objet et le capteur. Le test est validé si pour au moins 4 tests effectués, la valeur retournée correspond toujours au résultat attendu.
Il n'y a pas de test à effectuer au préalable.

*updateDistanceActuelle(**float** distanceActuelle)*
Pour ce test cet méthode est appelée puis on utilise un System.out.print(getDistanceActuelle()). Ce test est effectué de la manière suivante avec a, b et c trois valeurs distinctes les unes des autres:
updateDistanceActuelle(a) ;
System.out.print(getDistanceActuelle()) ;
updateDistanceActuelle(b) ;
System.out.print(getDistanceActuelle()) ;
updateDistanceActuelle(c) ;
System.out.print(getDistanceActuelle()) ;
L'objectif est la prise en compte de la valeur prise en paramètre pour actualiser (en le remplaçant) l'attribut distanceActuelle. On attend comme résultat les valeurs a b c retournées dans le bon ordre. Le test ne sera validé que si suite à chaque appel de *updateDistanceActuelle(float distanceActuelle)* la méthode *getDistanceActuelle()* permet de comfirmer que l'attribut distanceActuelle est bien actualisé.
Il n'y a pas de test à effectuer au préalable.

shrekFaceAuMur()
Pour ce test le robot est placé face au mur du terrain à 1m, 50 cm, 35 cm, 30 cm, 25 cm, 10 cm de celui-ci. Le robot sera ensuite placé à ces mêmes distances face à un palet (sans qu'il n'y ait de mur derrière le palet et à moins de 30 cm du capteur). A chaque fois que le robot est disposé à la distance correspondante, on utilise un System.out.print(shrekfFaceAuMur()) afin de connaître la valeur retournée par le boolean. L'objectif est de retourner un boolean true si et seulement si *getDistance()* renvoie une valeur inférieure à 30 cm. Ce boolean est utilisé pour détecter la présence de murs à moins de 30 cm en face du robot. Les palets ne sont pas confondus avec des murs car en raison de leur taille , à moins de 30 cm du capteurs ils ne sont pas détectés et la distance renvoyée par *getDistance()* est supérieure à la distance entre les palets et le robot.
La valeur attendue est false lorsque le robot est disposé à une distance supérieure à 30 cm face à un mur et lorsqu'il est face à un palet sans présence de mur (ou autre élément) derrière le palet et à moins de 30 cm du capteur. La valeur attendue est true lorsque le robot est à moins de 30 cm d'un mur. La marge d'erreur acceptée pour le test est de 2 cm.
Avant de réaliser ce test, il est impératif que *getDistance()* ait été testé avec obtention des résultats attendus.

Tests relatifs au capteur de toucher
estTouche()
Le boolean *estTouche()* est appelé 6 fois dans un System.out.println(estTouche). Pour 3 cas, le capteur de Toucher est activé (on appuie dessus pour ça).L'objectif est de vérifier que l'on est bien capable grâce à cette méthode d'identifier si le capteur est en contact avec un objet.
On attend obtenir en sortie « true » pour les 3 cas où le capteur subit effectivement une pression et « false » dans les autres cas. Le test est validé si pour 100 % des essais le résultat correspond à celui attendu. Il n'y a pas de test préalable à réaliser.

updateStatus()
L'attribut statut est initialisé à false. Puis dans une boucle while qui prend en condition le fait que estTouche() renvoie false, la méthode updateStatus est appelée. On appelle ensuite un System.out.println(touchdanslePasse()) pour lire la valeur de statut. L'opération est répétée 5 fois. L'objectif est de vérifier que cette méthode actualise l'attribut statut lorsque le capteur de toucher subit une pression. Cela permet d'utiliser la valeur de status pour savoir si le capteur a été activé dans le passé.
On attend de ce test que pour chacun des 5 essais, dès que la pression sur le capteur est appliquée le robot affiche « true ». Le test est validé si c'est le cas pour chaque essai. Il faut préalablement avoir testé *estTouche()* avec obtention des résultats attendus.

setDirection0()
La méthode *setDirection0()* est appelée puis on utilise un System.out.print(this.getD()). L'objectif est de s'assurer de pouvoir réinitialiser la direction connue du robot à 0.

On attend lire la valeur 0 suite au System.out.print. Le test ne sera validé que si le résultat est systématiquement égal à 0. Il n'y a pas de test à effectuer préalablement .

setDirectionRecherche()
La méthode *setDirectionRecherche()* est appelée puis on utilise un System.out.print(this.getD()). L'objectif est de s'assurer de pouvoir réinitialiser la direction connue du robot à 180.
On attend lire la valeur 180 suite au System.out.print. Le test ne sera validé que si le résultat est systématiquement égal à 0. Il n'y a pas de test à effectuer préalablement .

updateDirection(double angle)
La méthode updateDirection est appelée 8 fois différentes en prenant en paramètre 8 configurations différentes avec d la direction connue par le robot (retournée par getD) et angle l'angle pris en paramètre. Pour un des tests on aura angle et d >0,pour un autre un autre test angle et d < 0, un autre angle<0<d, pour le suivant d<0<angle, ainsi que d=0< angle , d=0>angle, puis angle=0>d, et enfin angle=0<d. Suite à chaque appel de la méthode, on utilise un System.out.print(this.getD()). L'objectif est de s'assurer qu'il est possible d'actualiser la direction du robot.
On attend comme résultat la somme entre angle et d. La marge d'erreur est de 0 degré. Pas de tests préalable à réaliser.

Test relatif au capteur de couleurs
getColor()
Le robot est placé de sorte que son capteur de couleur soit au dessus d'une zone du terrain d'une couleur c définie. L'opération est répétée 3 fois pour chaque couleur et est effectuée pour toutes les couleurs présentes sur le terrain (couleurs des lignes et gris du terrain). On utilise un System.out.println(getColor()) à chaque configuration. L'objectif est de s'assurer d'identifier les différentes couleurs situées en dessous du capteur lors du tournoi et ainsi de distinguer le blanc des autres couleur. Cet indice est utilisé pour détecter les zones d'embut.
On attend obtenir pour chaque System.out.println(getColor()) un entier qui ne varie pas pour les 3 tests réalisés au dessus d'une même couleur, mais qui varie entre les tests effectués au dessus de zones de couleurs différentes. Le test est validé si et seulement si l'entier renvoyé au dessus de zones blanches ne varie jamais et que pour toutes les zones non blanches l'entier renvoyé est systématiquement différent de celui renvoyé au dessus des zones blanches.

2/Moteurs
Tests relatifs aux mouvements des pinces
shrekPincesEtat()
Pour tester le boolean shrekPincesEtat(), la valeur du boolean etatPinces est définie true ou false puis on utilise un System.out.print(shrekPincesEtat()). L'objectif est de vérifier que l'on peut avoir accès via une méthode publique à l'état des pinces après actualisations de celui-ci.
La valeur attendue doit correspondre à etatPinces (false si etatPinces=false, true sinon). Le test est validé si pour 100 % des appels, le résultat obtenu correspond à celui attendu.

rotate(float f)
Pour cette méthode, le test consiste à appeler la méthode *rotate(f)* avec f une valeur. Le test est effectué avec au moins 5 valeurs différentes de f. Le degré d'ouverture des pinces est ensuite mesuré. L'objectif est de parvenir à ouvrir et fermer (lorsque f est négatif) les pinces du robot en choisissant le degré d'ouverture(ou de fermeture).
On attend obtenir une ouverture des pinces correspondant à la valeur entrée en paramètre.

ShrekFermePincesSansPalai()
etatPinces est initialisé à false, la méthode *ShrekFermePincesSansPalai()* est appelée, puis on utilise un System.out.print(shrekPincesEtat()). L'objectif est de s'assurer d'avoir une méthode qui permette de fermer les pinces actualisant etatPinces.
On attend comme résultat la fermeture des pinces ainsi que la valeur true présentée par le System.out.println. *rotate()*et *etatPinces()*doivent avoir été préalablement testés avec obtention des resultats attendus.

shrekFermePincesAvecPalai()
etatPinces est initialisé à true, la méthode *ShrekFermePincesAvecPalai()* est appelée, puis on utilise un System.out.print(shrekPincesEtat()). L'objectif est de s'assurer d'avoir une méthode qui permette d'ouvrir les pinces en actualisant etatPinces.
On attend comme résultat la fermeture des pinces ainsi que la valeur false présentée par le System.out.println. *rotate()et etatPinces()*doivent avoir été préalablement testés avec obtention des resultats attendus.

shrekOuvrePincesAvecPalai()
etatPinces est initialisé à false, la méthode *ShrekOuvrePincesAvecPalai()* est appelée, puis on utilise un System.out.print(shrekPincesEtat()). L'objectif est de s'assurer d'avoir une méthode qui permette d'ouvrir les pinces en actualisant etatPinces.
On attend comme résultat la fermeture des pinces ainsi que la valeur true présentée par le System.out.println. *rotate()et etatPinces()*doivent avoir été préalablement testés avec obtention des resultats attendus.

shrekFermePincesAvecPalai()

etatPinces est initialisé à true, la méthode *ShrekFermePincesAvecPalai()* est appelée, puis on utilise un System.out.print(shrekPincesEtat()). L'objectif est de s'assurer d'avoir une méthode qui permette de fermer les pinces en actualisant etatPinces.
On attend comme résultat la fermeture des pinces ainsi que la valeur false présentée par le System.out.println. *rotate()et etatPinces()*doivent avoir été préalablement testés avec obtention des resultats attendus.

closeWhile(int angle)
La méthode closeWhile est appelée 5 fois en prenant en paramètre 5 valeurs différentes dans une boucle while. Un System.out.print() est appelé en même temps. L'objectif est de s'assurer que le robot peut être à la fois en train de fermer les pinces et d'effectuer une autre action.
On attend observer une fermeture des pinces correspondant au paramètre *angle,* simultanément à l'autre action appelée pour chacun des 5 tests. *rotate()* doit avoir été testée avec obtention des résultats attendus.

openWhile(int angle)
La méthode openWhile est appelée 5 fois en prenant en paramètre 5 valeurs différentes dans une boucle while. Un System.out.print est appelé en même temps. L'objectif est de s'assurer que le robot peut être à la fois en train d'ouvrir les pinces et d'effectuer une autre action.
On attend observer une ouverture des pinces correspondant au paramètre *angle,* simultanément à l'autre action appelée pour chacun des 5 tests. *rotate()* doit avoir été testée avec obtention des résultats attendus.

Tests relatifs aux déplacements du robot

shrekAvance(double distance)
La méthode est appelée et prend en paramètre un double correspondant à une distance. L'opération est répétée 5 fois pour 5 distances différentes. La distance parcourue par le robot à la suite de cette instruction est mesurée. L'objectif est de s'assurer que l'on est capable de déplacer le robot d'une distance d choisie par nous.
On attend obtenir que le robot se déplace de la distance prise en paramètre (pour d= 0.3, il doit avoir avancé de 30 cm par exemple). La distance mesurée doit correspondre à celle entrée en paramètre La marge d'erreur pour valider le test est de 1.5 cm. Il n'y a pas de test à effectuer au préalable de celui-ci.

shrekTourneGauche(double angle)
La méthode est appelée et prend en paramètre un double correspondant à un angle. L'opération est répétée 5 fois pour 5 angles différents. L'angle de rotation effectuée par le robot vers sa gauche est ensuite mesuré. L'objectif est de s'assurer que l'on est capable de commander une rotation du robot vers la gauche en choisissant l'angle de cette rotation.
On attend obtenir que le robot opère une rotation vers la gauche correspondant à l'angle entré en paramètre. La marge d'erreur pour valider le test est de 5 degrés car la vitesse de rotation peut biaiser la rotation de l'angle. Il n'y a pas de test à effectuer au préalable de celui-ci.

shrekTourneDroite(double angle)
La méthode est appelée et prend en paramètre un double correspondant à un angle. L'opération est répétée 5 fois pour 5 angles différents. L'angle de rotation effectuée par le robot vers sa droite est ensuite mesuré. L'objectif est de s'assurer que l'on est capable de commander une rotation du robot vers la droite en choisissant l'angle de cette rotation.
On attend obtenir que le robot opère une rotation vers la droite correspondant à l'angle entré en paramètre. La marge d'erreur pour valider le test est de 5 degrés car la vitesse de rotation peut biaiser la rotation de l'angle. Il n'y a pas de test à effectuer au préalable de celui-ci.

shrekRotateWhile(double angle)
La méthode *shrekRotateWhile(double angle)* avant une boucle while prenant en condition le boolean isMoving() et contenant un System.out.println(). L'opération est répétée 5 fois avec 5 angles différents en paramètres. Il y a au moins 20 degrés de différence entre chaque angle. L'angle tourné est ensuite mesurée. L'objectif est de vérifier que le robot est capable de tourner en fonction d'un angle entré en paramètre en effectuant une autre action simultanément.
On attend que le robot affiche le contenu du *System.out.println()* tout en se déplaçant de manière simultanée. Le test sera validé si affiche et tourne simultanément, et si la rotation correspond à l'angle entré en paramètre avec une marge d'erreur de 5 degrés, car la vitesse peut biaiser les mouvements du robots.Il n'y a pas de test à effectuer au préalable.

shrekMoveWhile(double distance)
La méthode *shrekMoveWhile(double distance)* avant une boucle while prenant en condition le boolean isMoving() et contenant un System.out.println(). L'opération est répétée 5 fois avec 5 distances différentes ayant au moins 10 cm d'écart les unes avec les autres. La distance effectivement parcourue est ensuite mesurée. L'objectif est de vérifier que le robot est capable de se déplacer d'une distance entrée en paramètre tout en effectuant une autre action.
On attend que le robot affiche le contenu du System.out.println() tout en se déplaçant de manière simultanée. Le test sera validé si affiche et avence simultanément, et si le robot se déplace de la distance entrée en paramètres avec une marge d'erreur de 2cm, car la vitesse peut biaiser les mouvements du robots.Il n'y a pas de test à effectuer au préalable.

shrekStop()
La méthode *shrekMoveWhile(double distance)* est appelée en prenant en paramètre une distance de 3 mètres avant une boucle for qui prend en condition le boolean shrekIsMoving et qui au bout de 15 itérations appelle la méthode *shrekStop() :*

```
shrekMoveWhile(3) ;
for (int i=0 ; shrekIsMoving ; i++) {
System.out.print(i+ «  ») ;
Delay.msDelay(900) ;
if (i==15){
shrekStop() ;
Delay.msDelay(900) ;
}
}
```
L'opération est répétée 4 fois.
L'objectif est de s'assurer que l'on peut arrêter le robot pendant ses déplacements, même si la distance prise en paramètre n'a pas encore été entièrement parcourue.
On attend de ce test que le robot se déplace et s'arrête avant d'avoir fini de parcourir la distance prise en paramètre et après avoir affiché 1 2 3 4 5 6 7 8 9 10 11 12 13 15. Le test sera validé si le robot avance d'une certaine distance identique à 1 cm près pour chacun des 4 tests, si cette distance est inférieure à celle entrée en paramètre et si l'arrêt a lieu tout de suite après avoir affiché le 15. *shrekMoveWhile(double distance)* doit avoir été testée au préalable et avec obtention des résultats attendus.

setAngularSpeed(double speed)
Après avoir utilisé une rotation avec rotate(a) (a un angle défini), on utilise un System.out.println(getAngularSpeed()) pour connaître la vitesse actuelle du robot, puis on appelle *setAngularSpeed(double speed)* avant de réeffectuer rotate(a) et System.out.println(getAngularSpeed()). L'opération est effectuée 2 fois avec comme paramètre une vitesse plus grande que celle obtenue avec le System.out.println(getAngularSpeed()) initial puis une plus petite. On choisit une vitesse au moins 2 fois plus grande ou deux fois plus petite que la vitesse initiale. L'objectif est de vérifier que l'on est bien capable de modifier la vitesse de rotation du robot en entrant en paramètre la vitesse souhaitée.
On attend obtenir après le premier System.out.print une valeur initilale, puis la valeur speed rentrée en paramètre avec le deuxième System.out.print. Si celle-ci est plus petite que la vitesse initiale, alors on souhaite observer une rotation du robot plus lente pour le deuxième rotate(a) et inversement. Le test ne sera validé que si pour chaque essai le résultat souhaité est obtenu. Il faut avoir testé *rotate(double angle)* en obtenant les résultats souhaités avant ce test.

Tests d'intégration

Concernant les tests d'intégration, les fonctions testées s'appuient sur celles décrites plus haut. Pour la liste des tests à effectuer préalablement nous ne les incluons pas car il serait difficile d'être exhaustif. Nous partons du principe que les tests unitaires ont déjà été effectués au préalable.

*quiMarque(**double** x, **double** xdiff)*
La méthode est appelé avec le robot avec les pinces fermées et contenant un palet positionné sur le terrain soit sur une ligne d'embut, soit à proximité et orienté face à celle ci. L'opération est répétée 4 fois (2 fois sur la ligne d'embut, 2 fois avant celle-ci). L'objectif est de vérifier que l'on sait marquer un but en déposant le palet derrière la ligne d'embut.
On attend de ce test que le robot s'il n'est pas en contact avec la ligne d'embut avance vers celle-ci et une fois dessus avance, dépose le palet en ouvrant les pinces, recule de sorte à se retrouver sur la ligne d'embut puis tourne à 180 degrés. Le test est validé si à chaque essai le robot arrive à enchaîner les actions décrites plus haut et si la distance à laquelle il avance pour déposer le palet est suffisement grande pour lui permettre de ne pas entrer en collision avec lors de son demi-tour.

*quiMarquePremierPalai(**double** x, **double** xdiff)*
La méthode est appelée alors que le robot est situé sur le terrain à une position correspondant à celle qu'il aura au début des matchs (sur sa zone d'embut face à la zone d'embut adverse). Les palets sont disposés aux emplacements qu'ils auront le jour du tournoi. L'opération est répétée 3 fois. Pour un des 3 essais, le robot n'est pas positionné exactement sur la ligne d'embut mais presque. L'objectif de ce test est de vérifier que la méthode développée pour attraper le premier palet est efficace.
Il est attendu que le robot avance vers le palet situé en face de lui puis opère une rotation lui permettant d'esquiver les autres palets situés sur sa trajectoire initiale. En suivant sa nouvelle trajectoire il marque puis tourne de sorte à se retrouver exectement en face de sa ligne d'embut (où il est placé initialement). Lorsqu'il n'est pas positionné exactement sur la ligne blanche, il doit pouvoir ajuster sa position. Le test est validé si les actions décrites plus haut sont exécutées par le robot et s'il ne rentre pas en collision avec un mur ou d'autres palets.

testpince3()
La méthode est lancée alors qu'un palet se trouve en face du robot à une distance supérieure ou égale à 50 cm. L'opération est répétée 3 fois. L'objectif est de vérifier que l'on est capable de s'avancer vers un palet et de stopper sa trajectoire une fois le capteur de toucher en contact avec le palet.
On attend du robot qu'il avance en ouvrant les pinces et s'arrête une fois que le capteur de toucher a été en contact avec le palet. Le test est validé si le robot agit comme décrit précédement et si les pinces sont ouvertes suffisement tôt pour éviter une collision entre celles ci et le palet. Le test est validé si le résultat attendu est obtenu pour les 3 essais.

testligne()

Le robot est placé sur le terrain face à la ligne d'embut sans obstacle sur sa trajectoire. L'opération est répétée 4 fois. L'objectif est de vérifier que l'on est capable de commander un déplacement qui s'arrête dès que le capteur de couleurs est situé au dessus de la ligne blanche.
Il est attendu que le robot avance en ligne droite jusqu'à la ligne d'embut puis s'arrête une fois que le capteur de couleur est dessus. Le test est validé si le résultat souhaité est obtenu pour les 4 essais.

rechercheGraal(double angle)
La méthode est lancée avec en paramètre un angle de balayage. Dans le périmètre qui sera balayé par le capteur de distances du robot, plusieurs palets sont disposés. Ils sont situés à 30 cm ou plus du capteur de distances car en dessous ils ne sont pas détectés. L'opération est répétée 4 fois. La disposition des palets est modifiée entre les essais. L'objectif est de vérifier que le robot est capable de détecter le palet le plus proche et de se positionner de sorte à lui faire face.
Il est attendu que le robot opère une rotation à gauche et à droite balayant l'angle entré en paramètres. A la fin de ce balayage il opère une rotation qui lui permet de faire face au palet le plus proche. Le test est validé si le résultat attendu est obtenu pour les 4 essais.