TP

I. Présentation

Le but de ce TP est de donner le meilleur comportement possible à des robots dans une usine. Ce TP simule le travail de robots qui doivent aller chercher des boites dans une zone puis les ramener dans une seconde zone. Tous les robots présents dans cette usine sont identiques mais ils ont deux états différents :

- Ils cherchent une boite
- Ils ramènent une boite

Les robots ont connaissances de la position des zones des boites (méthode « distanceToClaim » et « distanceToRelease »), ainsi que de l'état des autres robots (méthode getIsCarrying », par contre ils ne connaissent pas les déplacements des autres robots. Le robot n'a que 4 possibilités de déplacement : devant, derrière, droite et gauche.

La spécificité de l'environnement dans lequel le robot évolue est qu'un grand mur sépare les deux zones et seules deux couloirs ayant une largeur d'un seul robot permettent la communication.

Les robots ont un champ de vision définis par « VIEW_RADIUS ».

Le but est donc de trouver le comportement optimal des robots afin que ceux-ci ne s'inter bloquent pas dans les couloirs.

Ce comportement sera décidé dans la méthode « onDecide ». Le but sera de trouver l'aire à mettre dans « targetArea ». Celle-ci sera atteint par le robot dans la méthode « onAct ».

La méthode « checkFreedom » permet de savoir si une aire est possible d'accés.

II. Comportement nominal

Ce premier comportement est le comportement basique du robot. Il devra aller, selon son état, chercher une boite ou la ramener.

Le résultat devrait être que les robots se bloquent dans un couloir.

III. Comportement aléatoire

Ce comportement utilise le tirage de Monte Carlo. Le but ici est que si le robot ne peut pas aller vers son objectif à cause d'un obstacle, celui-ci utilisera un déplacement pris au hasard dans tous ses déplacements possibles.

Les valeurs dans le tirage seront :

- Le retour en arrière vaut 0, (la dernière aire où le robot était et la direction en arrière)
- On calcul la distance entre chaque aire possible (sauf en arrière) et l'aire de l'objectif.
    - La direction ayant la distance plus élevé aura 15
    - La deuxième aura 10
    - La dernière aura 1
- Si seul la direction en arrière est possible alors celle-ci sera choisie

Le résultat devrait être que certains robots arrivent à passer mais ce sera long.

IV. Comportement aléatoire avec mémoire de la direction

Pour éviter que le robot ne change constamment de direction quand il est face à un obstacle il serait bien que celui-ci grade la première direction choisie aléatoirement en mémoire, ceci en excluant la dernière zone où le robot était.

Le résultat devrait être que les robots accèdent plus facilement aux couloirs mais peine à passer.

V. Comportement coopératif

Nos robots ont donc un souci lorsqu'ils se rencontrent dans un couloir, il faudrait donc que l'un des deux revienne en arrière. On propose d'obliger un des deux robots à revenir en arrière si celui en face de lui est plus proche de son objectif. De ce fait un robot sera en train de revenir en arrière et les robots dans le même état que lui doivent savoir cela pour éviter qu'ils restent bloqués.

De ce fait tous les robots dans le couloir qui sont dans le même état vont revenir en arrière pour laisser passer les autres.

Le résultat devrait être une meilleure fluidité des passages dans le couloir, les robots doivent se laisser passer.

VI. Mémoire

Maintenant que nos robots sont polis, il serait intéressant que ceux-ci évitent d'aller dans des zones où ils ont déjà rencontrer des problèmes. C'est pourquoi on leur intègre une mémoire. Chaque robot aura deux mémoires, une pour chaque état. Cette mémoire est limitée par la « MEMORY_SIZE », lorsqu'on voudra ajouter une aire il faudra utiliser la méthode « addToQueue ».

A chaque fois que le robot est bloqué et doit retourner en arrière il mettra dans sa mémoire la première aire qui lui offrira un nombre de choix supérieur à 1. Il devra aussi marquer l'aire où il a changé de direction.

Avec cette mémoire le robot devra éviter certaines zones, pour ce faire il devra altérer son tirage de Monte Carlo. On propose pour cela une vision de mémoire de 3 fois la taille de son champ de

vision. Chaque aire ayant été marquée aura une influence sur les probabilités. Le tableau suivant résume cela :

1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	2	1	1	1	1	1
1	1	1	1	2	3	2	1	1	1	1
1	1	1	2	3	4	3	2	1	1	1
1	1	2	3	4	5	4	3	2	1	1
1	2	3	4	5	Robot	5	4	3	2	1
1	1	2	3	4	5	4	3	2	1	1
1	1	1	2	3	4	3	2	1	1	1
1	1	1	1	2	3	2	1	1	1	1
1	1	1	1	1	2	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1

On voit 4 arcs, chacun correspondant à une direction, les lignes vertes correspondent à deux directions différentes. A chaque fois qu'une aire marquée est rencontré, on ajoute la valeur du tableau aux trois autres directions sauf vers la dernière aire et le retour arrière.

Dans le cas d'un mémoire trop importante ou un environnement avec peu d'obstacle le robot devra avoir un mécanisme d'oubli sinon il risque de ne jamais vouloir aller dans les couloirs.

On propose donc que lorsque l'on intègre une aire celle-ci a une valeur de 10 et toutes les 10 actions du robot, il réduit la valeur de toutes ses aires de 1. Une fois qu'elle est à 0 il l'enlève.