V. Simulation multi-agents

1. Contexte du projet

Syha est une jeune start-up, qui prévoit de se développer dans les années à venir. Pour le moment, leur priorité est de concevoir un prototype de robot cueilleur fonctionnel.

Mais pour la suite, Syha a l'ambition d'automatiser la récolte de légumes en serre et pour se faire, il faudra que plusieurs robots collaborent et travaillent ensemble sur la même serre. Nous nous sommes donc vu confier la mise au point d'une stratégie de gestion de flotte de robots cueilleurs, ainsi que la réalisation d'un preuve de concept sous forme de simulation multi-agents.

2. Rappel du cahier des charges

Le projet de simulation multi-agents repose donc sur le fait de simuler par ordinateur une flotte de robots cueilleurs collaborant ensemble dans la même serre.
Il faudra ainsi gérer le déplacement de ces robots pour optimiser la récolte tout en gérant les collisions entre les robots et leur autonomie : ils doivent être capables de se recharger tous seuls.

Cette simulation doit être modulable : c'est à dire que les dimensions de la serre ainsi que le nombre de robots et de zones de recharge doivent être paramétrés en entrée pour s'adapter à la serre étudiée.

Cette simulation doit également retourner des données qui vont permettre à Syha d'avoir une idée de l'efficience de la flotte dans la serre.

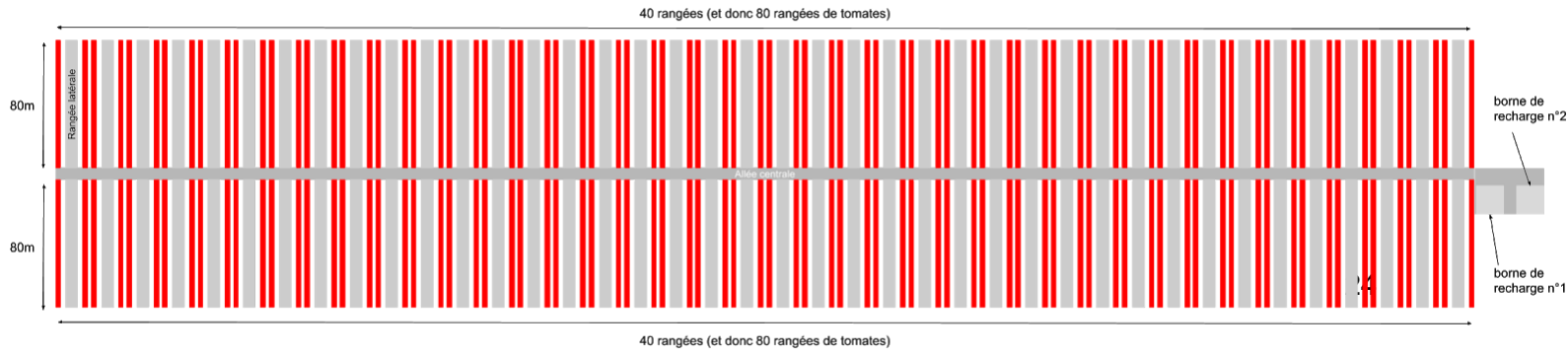
Elle est également un moyen pour Syha de tester plusieurs algorithmes, à savoir différentes manières de gérer les robots cueilleurs et de voir laquelle semble la plus efficace.

3. Environnement de la simulation

Le développement de la SMA se fera sur une serre composée de 80 rangées (40 rangées de part et d'autre de l'allée centrale), avec pour chaque rangée des grappes de tomates de part et d'autre. La longueur des rangées est de 80 "cases". Il y a 2 bornes de rechargement, situées juste à côté de la serre.

Le développement de la SMA a été fait avec 3 robots puis 7 puisque ce nombre est modulable et la simulation doit être fonctionnelle pour n'importe quel nombre de robots.

<u>Voici un schéma qui récapitule la serre sur laquelle nous avons travaillé :</u>



<u>Données :</u>
1 patches = 1 mètre

4. Solutions techniques possibles dans une vraie serre

Il est important de faire le lien entre la simulation et la réalité. En réalité il y aura un master : c'est lui qui recevra toutes les données envoyées par les agents, c'est-à-dire les robots cueilleurs, qui les traitera et qui en fonction de ces données enverra des ordres aux agents.

Ce master peut être soit un serveur installé près de la serre, soit un des agents présents dans la serre.

-> Toutes les informations techniques suivantes ont été fourni par le client.

➢ Comment les agents de la simulation peuvent détecter les autres agents ?

En réalité, les robots cueilleurs disposeront soit d'un lidar 360° (comme dans les voitures autonomes par exemple), soit de plusieurs lidar unidirectionnels disposés autour du robot.

On considère donc que les robots cueilleurs sont capables de "regarder" dans toutes les directions.

➢ Comment les agents et le master pourront communiquer ?

La communication se ferait par wifi : le principal avantage étant qu'il est possible de mettre de multiples émetteurs à plusieurs endroits de la serre. On pourrait par exemple décider de disposer de nombreux émetteurs dans l'allée centrale et de faire en sorte que les robots mettent à jour leurs données lorsqu'ils traversent cette allée centrale, à savoir leur position dans la serre, et leur état de batterie.

De plus, le protocole Wifi s'interface très bien avec ROS.

➢ Comment les agents peuvent-ils connaître leur position ?

Les solutions techniques présentées ici ne sont pas très détaillées car aucune étude n'a encore été faite. Ce sont simplement des idées présentées par le client.

Les robots pourront utiliser un codeur sur la motorisation des roues pour connaître leur position dans la rangée. Connaissant le nombre de pas mesuré par le codeur depuis le début de la rangée, le rapport de transmission, et le diamètre des roues, il est facile de calculer la position dans la rangée.

Pour connaître la position dans la serre, l'objectif sera de faire du SLAM (donc de cartographier la serre). Pour cela, on se servira des lidars et des capteurs embarqués des robots qui permettront de faire de l'odométrie.

5. Explications sur la simulation

➢ ETAPE 1 : Décomposition du travail des robots

La serre est composée de 40 rangées sur l'axe horizontal. On attribue équitablement un certain nombre de rangées à chaque robot. Si le nombre de rangées n'est pas un multiple du nombre de robots, on calcule x, le reste de la division euclidienne du nombre de rangées par le nombre de robots, et les x robots qui sont le plus à droite de la serre ont une rangée de plus à faire.

Pour la démonstration de la simulation, nous avons décidé de faire un algorithme où les robots se gèrent automatiquement : chaque rangée est donc attribuée à un robot et est toujours faite par le même robot.

La position initiale de chaque robot est marquée par une case bleue pour que l'on puisse repérer plus facilement les zones de travail de chaque robot.

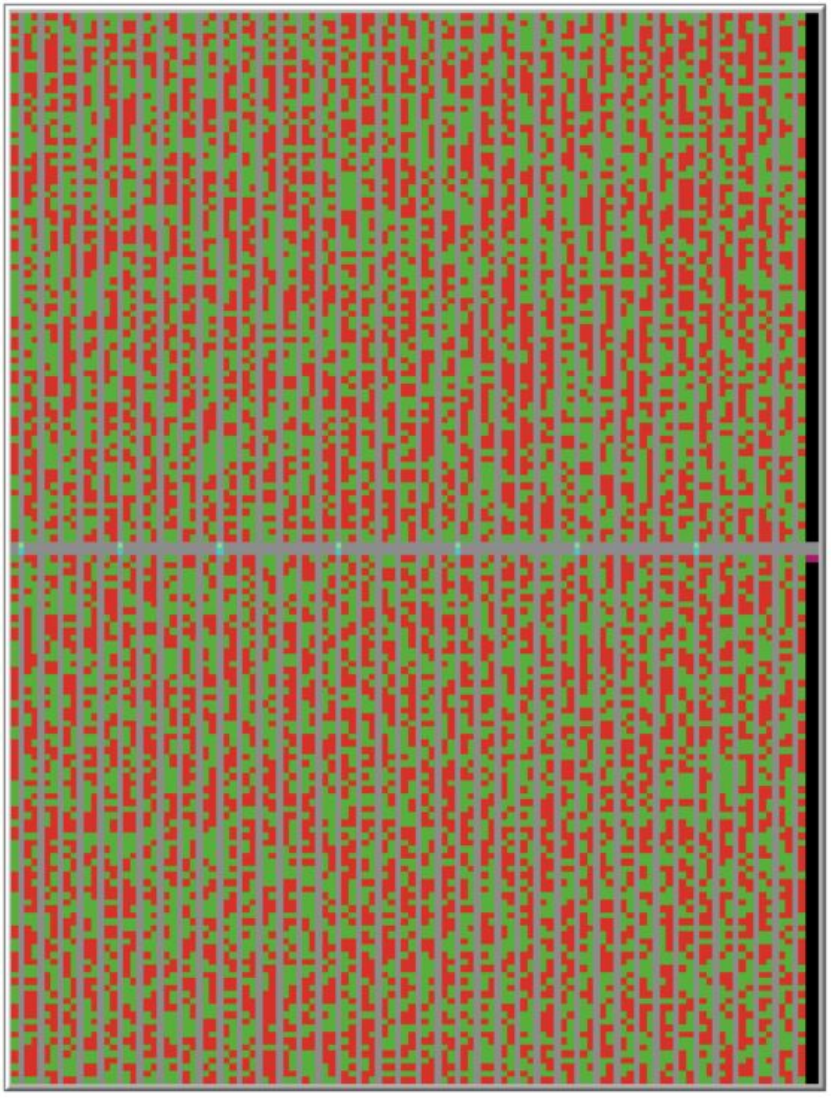


***Figure 31** : Environnement de la simulation avec 7 robots dont les 5 derniers ont une rangée de plus à faire*

Il y a deux zones de recharge situées juste à droite de la serre.

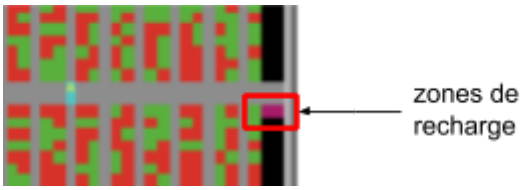


***Figure 32** : Deux zones de recharge*

➢ ETAPE 2 : Mission de chaque robot

Chaque robot fait la rangée du haut, puis la rangée du bas, puis se décale d'une rangée et recommence (rangée du haut, puis du bas).

A chaque fois que le robot avance dans la rangée, il cueille les grappes qui sont à sa droite. Il doit donc faire l'aller-retour pour cueillir de part et d'autre d'une rangée.

➢ ETAPE 3 : Gestion de la batterie des robots

Il existe une variable, nommée *limit_energy_for_recharge*, qui impose le pourcentage de la batterie à partir duquel les robots vont se recharger. Cette variable est un input et peut être paramétrée par l'utilisateur.

Quand 1 ou 2 robots sont en dessous de la limite, ils vont se recharger. S'il y en a plusieurs, on prend les 2 qui ont la batterie la plus faible.

Si plus de 2 robots ont le même niveau de batterie, et que celui-ci est le plus faible, le choix est aléatoire.

Le ou les robots qui ont été choisis pour aller se recharger sont référencés dans la liste d'agents *robot_s_with_min_energy*.

➢ ETAPE 4 : Gestion des collisions

a) Sur les rangées perpendiculaires :

Deux robots ne sont jamais censés se croiser sur les rangées perpendiculaires car il n'y a pas assez de place pour que 2 robots puissent se croiser. Ainsi cette situation n'arrive jamais sauf si les consignes que donnent le master aux agents sont mauvaises.

Néanmoins, si cela arrive, les 2 robots s'arrêtent avant d'entrer en collision et restent sur place.

b) Sur l'allée centrale :

Quand les robots se déplacent sur l'allée centrale, pour aller se recharger par exemple ou pour aller faire une rangée spécifique sous les ordres du master, ils se déplacent toujours sur la partie basse de l'allée (l'allée fait 2 cases de haut, ils se déplacent sur la case du bas).

Quand 2 robots se croisent, par exemple un robot qui va se recharger et l'autre qui en revient, le robot qui va vers la gauche a toujours la priorité et c'est celui qui va vers la droite qui esquive l'autre : il monte d'une case (donc est toujours sur l'allée centrale), se

décale d'une case à droite, puis redescend d'une case. L'autre robot aura continué tout droit puisqu'il a la priorité et les deux se seront évités.

Aussi, lorsqu'un robot est sur la dernière case d'une rangée perpendiculaire, sur le point d'aller sur l'allée centrale, on regarde s'il n'y a pas de robot sur l'allée centrale devant lui et s'il peut donc avancer ou pas. S'il y en a un, alors il attend un tour dans sa rangée avant de pouvoir avancer sur l'allée centrale.

➢ ETAPE 5 : Fonction demandée par Syha

Le client a demandé le développement d'une fonction qui donne l'ordre à un robot spécifique d'aller travailler dans une rangée en particulier. Cette fonction, appelée "*guide-agent-to-specific-lane*", sera utilisée par un master pour commander les agents qui travaillent dans la serre. Ce master peut être soit un serveur, fixe dans la serre, soit un des agents.

6. Plusieurs retours sur le projet

a) La modulabilité de la simulation

Un des points importants à respecter dans le cahier des charges était la modulabilité de la simulation car les futurs clients de Syha auront tous une serre de dimension différente.

Il était donc important de mettre les dimensions de la serre en paramètres d'entrée, au même titre que le nombre de robots et de bornes de recharges.

Le nombre de robots qui travaillent dans la zone est complètement modulable et permet de comparer l'efficacité de ceux-ci en faisant plusieurs simulations et en modifiant leur nombre.

Le nombre de rangées de tomates est également modulable : il suffit de modifier la valeur de *number_of_perpendicular_lanes*.

Il est important de se rappeler que la taille de la "fenêtre graphique" dépend de paramètres définis par l'utilisateur (dans "Settings…", puis *max-pxcor* et *max-pycor*).

Ainsi, lorsque l'on ajoute des rangées de tomates dans la serre (donc qu'on augmente la valeur de *number_of_perpendicular_lanes* par exemple), il faut modifier les dimensions de la fenêtre graphique (en l'occurrence *max-pxcor*) pour les voir.

Pour le dimensionnement de la serre en hauteur, c'est à dire la variable *length_of_rows*, même idée : il faut que *max-pycor* soit égal à 2 fois *length_of_rows* plus la hauteur de l'allée centrale.

En conclusion le dimensionnement de la serre est modulable mais une attention particulière est nécessaire de la part de l'utilisateur car les dimensions de la fenêtre graphique (c'est-à-dire les valeurs de *max-pxcor* et *max-pycor* dans "settings") doivent être définies en fonction de *length_of_rows* et *number_of_perpendicular_lanes*.

Un des problèmes est que nous nous sommes beaucoup servis des primitives *world-height* et *world-width* de NetLogo dans le code, donc il faut bien définir *max-pxcor* et *max-pycor* pour que la simulation fonctionne. A noter que la position initiale des robots dépend aussi de *max-pycor*.

Cependant, le nombre de bornes de recharges, c'est à dire *number_of_loading_zones* n'est pas modulable.

b) Gestion des collisions

Lorsqu'un robot se trouve sur la dernière case d'une rangée de tomates et s'apprête à aller sur l'allée centrale, il regarde devant lui pour savoir s'il y a un robot et donc s'il peut avancer ou non.

Cependant, ce sont les cases en diagonale qu'il devrait vérifier. En effet, un robot sur la case devant aura bougé au tick suivant, mais un robot en diagonale pourrait être un danger.

De plus, 2 robots qui se rencontrent sur une allée perpendiculaire évitent la collision en s'arrêtant et ne bougent plus. Nous n'avons pas géré les déplacements après cela car dans la démonstration de notre simulation ce cas n'arrive jamais. Mais il peut très bien arriver lorsque les agents sont gérés par un master et que celui-ci donne de mauvais ordres. Il faudrait donc pouvoir gérer ce genre d'événements.

c) Structure du code et utilité d'un master

A cause d'une planification insuffisante, le code de déplacement des agents manque de clarté. Si le temps le permettait il aurait été intéressant de reprendre le code pour le clarifier.

Cela ne gêne pas le fonctionnement du code mais représentera une perte de temps pour le client, car la finalité est de coder l'algorithme du master en se servant de la dernière fonction *guide-agent-to-specific-lane* (qui donne l'ordre à un agent d'aller travailler sur une rangée spécifique).

Le principal défaut du code est qu'un robot ne va jamais aider un autre robot, il s'occupe seulement de ses rangées. Pour cette simulation nous avons créé des agents qui se gèrent automatiquement, mais nous n'avons pas implémenté de master intelligent.

d) Recharge des agents

La recharge des agents peut être optimisée. En effet, on envoie des nouveaux robots se recharger seulement lorsque ceux présents sur les bornes sont partis, sans prendre en compte le temps de parcours vers les bornes.

On pourrait donc calculer *robot_s_with_min_energy* plus tôt, quand les robots en cours de recharge sont à 95% de batterie par exemple, voir même en fonction de la distance des nouveaux robots sélectionnés par rapport aux bornes.

Aussi, nous ne prenons pas en compte la distance des agents par rapport aux bornes, mais par exemple si 2 robots ont le même niveau d'énergie et doivent se recharger, il est sûrement préférable d'envoyer celui qui est le plus loin des bornes, car il perdra plus d'énergie en se déplaçant jusqu'aux bornes que l'autre.

e) Finalité de la simulation

Le but de cette simulation est de savoir combien de robots sont nécessaires pour optimiser la récolte de légumes dans une serre, ainsi que le nombre de bornes de recharges optimal et d'avoir des données sur la récolte.

Nous continuons d'y travailler mais par manque de temps nous ne pourrons certainement pas aller au bout de cette démarche et obtenir des résultats détaillés.

f) Nom des variables

Nous nous sommes appliqués à faire un lexique : pour qu'en cas de doute on puisse retrouver plus facilement à quoi sert telle fonction ou variable.