Corentin PLUMET et Eliot GONCALVES MT-BA2

RENDU 3 DE PROGRAMMATION

Notre stratégie repose sur plusieurs algorithmes décisionnels imbriqués dans une boucle infinie.
Le premier d'entre eux, "**update voisin**" vise à constituer la *liste d'adjacence* de chaque robots, c'est-à-dire les autres robots avec lequel il se trouve en communication directe.

Celui-ci repose sur un parcours des listes de robots de toutes les instances de base présentent et sur un test de distance entre ces robots ayant pour condition d'être inférieur au rayon de communication. Si la condition est vérifiée, chaque robot est pushback dans la liste d'adjacence (*"link"*) de l'autre robot.
Le second algorithme, "**connexion",** utilise la liste d'adjacence calculée précédemment afin de *fixer le mode de contrôle de chaque robot* (c'est-à-dire s'il est connecté à la base ou non). Une méthode de base appelle une fonction récursive en commençant par le robot de communication situé au centre de la base. Cette fonction parcourt la liste de voisin des robots en veillant avec un booléen visité à éviter les doublons. Cet algorithme constitue un vecteur contenant tous les robots en connexion avec la base.

L'algorithme appelé ensuite dans la boucle est celui de "**maintenance"**. Il réalise simplement une opération de maintenance automatique sur tous les robots revenant à la base. Il vérifie dans un premier temps que le robot se trouve bien au centre de la base puis dans un second temps, la distance parcourue par le robot concerné est remise à zéro tandis que l'on retire dp*cout_reparation aux ressources de la base.

L'algorithme suivant, celui de "**création de robots"**, s'occupe de créer les nouveaux robots de chaque base. Il ne peut en créer au maximum 3 par actualisation. La stratégie de création est la suivante : tout d'abord couvrir l'ensemble de la planète avec des robots de communication, puis envoyer des robots de prospection aller chercher les gisements. Ainsi, pendant les 17 premières actualisations, les 49 robots de communication nécessaire sont créés pour quadriller la carte, puis les 12 robots de prospection sont créés. Les robots de forage sont ensuite créés si un robot de prospection a détecté un gisement, que ce dernier est suffisamment intéressant (rapport distance/quantité positif) et qu'un robot de forage n'est pas encore présent/en route vers ce gisement. Dès que la base est au courant de son arrivé, elle créera un/des robots de transports selon si il faut plus de 2 aller retour pour le vider avec le nombre de robots de transport alloué à ce forage. Le nombre de robot de transport créé est plafonné à 5, afin d'avoir un bon rapport vitesse d'exécution/coût de construction.

Les algorithmes suivant sont appelés dans les fonctions "update_remote" et "update_autonomous" selon que le robot soit en connexion ou non avec la base (cf "connexion").

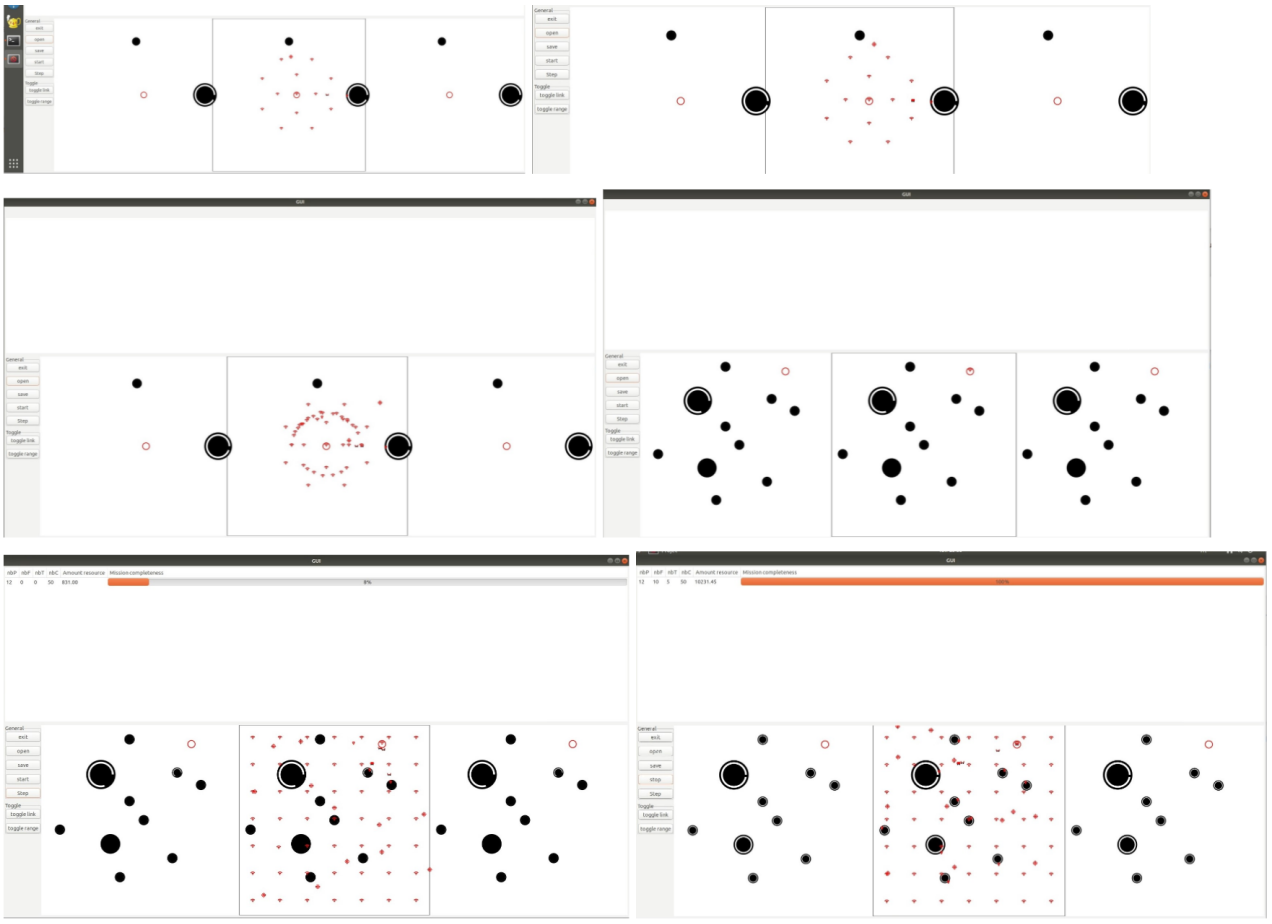
algorithme de prospection : *"action_p","action_p_autonomous"*
Cet algorithme détermine le mouvement des robots de prospection. Ce mouvement est un "zigzag" dont les branches sont plus larges à mesure qu'on s'éloigne de la base. A chaque fois que son but est atteint, si le robot est en mode "remote", il vérifie s'il se trouve sur un gisement, si oui il garde l'information en mémoire tant que la base ne l'utilise pas, sinon il continue son chemin. Si il est en mode "autonomous", le robot rentrera à la base dès que son but sera atteint, qu'il ait trouvé un gisement ou non.

algorithme de forage: *"action_f","action_f_autonomous"*
Cet algorithme détermine le mouvement des robots de forage et l'actualisation de la quantité de ressource contenu dans un gisement. Lorsqu'il est créé un robot de forage a pour but le point du gisement le plus proche de la base. Une fois sur site, il "lit" les ressources disponibles du gisement. Cette information est disponible ou non à la base selon qu'il soit en "remote" ou en "autonomous". Un seul robot de forage est alloué à chaque gisement découvert (cf "creation") et il ne se déplacera plus une fois sur place.

algorithme de transport: *"action_t","action_t_autonomous"*
Cet algorithme détermine les buts des robots de transport et leur déplacement. Le but (gisement) d'un robot de transport est donné par la base et est déterminé en fonction de l'attractivité d'un gisement (ressource, distance) et du nombre de robots de transport l'ayant déjà pour but. S'il faut plus de 2 aller retour aux robots de transports alloué à un gisement pour le vider, ou s'il n'y en a aucun, un robot de transport pourra avoir le robot de forage de ce gisement comme but. Une fois ce but atteint, le robot de forage remplit le robot de transport, si les ressources du gisement le permettent, dont le nouveau but sera donc la base où il videra sa cargaison une fois arrivé. Dans le cas où 2 robots de transport arrivent lors de la même actualisation sur le robot de forage, c'est celui arrivant en premier dans l'exécution du programme qui sera chargé.

algorithme de communication: *"action_c","action_c_autonomous"*
Cet algorithme détermine le mouvement des robots de communication. Lors de leur création, chaque robot de communication reçoit un but etant un maillon du quadrillage de la planète. Il se déplacera donc jusqu'à ce que ce but soit atteint en empruntant le chemin le plus court.

Cette approche à été pensée pour faire tendre vers l'autonomie les bases qui la mettent en place. Le choix du quadrillage de la zone par des communications découle de leur faible coût et du gain de temps que représente l'état remote sur l'état autonomous. La création des transports et des prospections est plafonnée à respectivement 5 et 12 robots pour limiter les dépenses de création élevée de ces types de robots. La trajectoire de ces mêmes prospections a été pensée pour couvrir une large zone et optimiser les chances de découverte d'un gisement. Pour inclure les cas de compétition inter base, les forages et les transports partent sur les gisements jugés les plus intéressants au vues des ressources encore disponibles.
Exemples d'exécution :



Corentin PLUMET et Eliot GONCALVES MT-BA2

Trouver un équilibre de répartition des tâches n'a pas été tout de suite évident. Les rendus 1 et 2, contrairement au rendu 3, ne présentaient pas vraiment deux grandes taches/modules distincts.
Pour la partie geomod, les différentes fonctions du module furent réalisées et corrigées par les deux membres du groupe d'une manière relativement homogène.

Le rendu 2 fut plus fastidieux à réaliser, à cause d'une mauvaise compréhension de la donnée, nous avons réalisé une partie des fonctions du rendu 3, partiellement les constructeurs de chaque type présents sur la map par Eliot et le début des algos décisionnelles par Corentin.
Ensuite, Corentin s'occupe en majeure partie de l'automate de lecture et Eliot du rapport et des fonctions de test d'erreurs (sur les intersections etc..).

Dès le début du rendu 3 nous avons décidé de nous partager les tâches de la manière suivante :
La partie algorithme décisionnelle pour Corentin avec l'écriture des algos dans le modules simulation et de toutes les fonctions, méthodes nécessaires pour le bon fonctionnement du modèle dans les modules robots, bases, gisement et la partie interface graphique pour Eliot avec le module gui, graphic, les nouvelles fonctions de geomod et l'écriture des fichiers.
Dans l'optique de vérifier le bon fonctionnement des algorithmes décisionnels en les voyant s'exécuter. Un affichage simple des instances de chaque classe sur la carte fut privilégié sans s'occuper des toggles, de l'ouverture des fichiers depuis le terminal, de l'ide ou de l'écriture.
Pendant ce temps, Corentin écrivait les algorithmes décisionnels, anticipant leurs partielles révision après l'épreuve de l'affichage.
Dans le rendu 3 apparut le bug qui nous prit le plus de temps (4 jours) celui-ci reposait sur un segmentation fault lié au lieu d'appel des fonctions de dessin apparaissant lors de la mise en commun du module graphique et du reste des modules. Ce problème fut réglé à la suite d'attentives recherches dans le programme, à des questions posées sur discourse et à l'aide directe apportée par le professeur sur la question.
Mis à part ce gros bug, le bug le plus fréquent fut les segmentation fault, dûs à l'importante présence de vector et de pointeurs dans le programme
Une fois ce problème réglé, nous pouvions observer l'exécution du modèle sur la carte, comme prévu, certaines erreurs apparurent.
Vient ensuite l'ajustement des algorithmes pour favoriser l'accumulation des ressources et le bon déplacement des robots.
Un problème apparu suite à des discussions avec d'autres membres de la section quant aux algo de décisions qu'il fallut modifier. Ceux-ci nécessitent maintenant plus de coûts calcul provenant de l'utilisation de *update voisin* et *connexion* et fournissent donc un affichage plus lent que les versions précédentes mais suivent plus explicitement les directives de la donnée.
De même, pour le module graphique, certaines choses manquaient comme les vérifications qu'une simulation existait ou la prise en charge des échecs de lecture, des ajustements furent donc réalisés.

En conclusion, le projet rendu rempli la grande majorité du cahier des charges, les seuls points faibles notables pourrait être la vitesse d'exécution du programme relativement faible dû à un important coût calcul et la présence d'arguments supplémentaires dans certaines classe par rapport à la donnée, arguments néanmoins nécessaires pour le bon fonctionnement du modèle tel que conçu par le groupe.

L'usage de discourse et de l'assistanat via discord nous a été très utile pour nous débloquer ou nous éclairer sur certains sujets.
Les points qui pourraient être améliorés seraient peut-être selon nous quelques précisions et approfondissement sur les tâches à effectuer dans les données.