Rapport projet Planet Donut

César Camus-Emschwiller 328075
Marine Moutarlier 310703

1. Description des algorithmes mettant en œuvre la stratégie

Notre programme met à jour dans la boucle de la simulation, les différents algorithmes décris ci-dessous. Ces algorithmes sont des méthodes de Simulation. Ceux-ci sont appelés à chaque mise à jour de la simulation grâce à la méthode idle et permettent au but d'être atteint.

1.1 L'algorithme de connexion

L'algorithme *connexions* met tout d'abord à jour le tableau *tab* qui contient l'ensemble des robots de notre simulation de départ. Pour l'ensemble des robots, il vide la liste des voisins et des liens. Puis rappelle les fonctions *update_voisin* et *update_link* qui remettent à jour les voisins et les liens entre robots. Finalement nous appellons *remote* savoir quels robots sont en mode remote et lesquels sont en mode autonomous.

1.2 L'algorithme de maintenance

Dans la *maintenance* la distance parcourue par un robot est remise à zéro, et il peut ainsi repartir vers son but défini, de récupérer de la ressource.

1.3 L'algorithme de créations de robots, l'élément principal de notre stratégie

Les actions suivantes sont organisées par ordre de priorité. De même les algorithmes traitant de la création de robots ou de leur déplacement sont délégués à des méthodes de Base, pour ne pas briser l'encapsulation.

Les algorithmes *créations* et *update_remot*e sont les plus importants, car au cœur de notre stratégie.

Ils commencent par mettre à jour les gisements, à l'aide des robots de prospection et de forage et ainsi en estimer la quantité.

Ils remettent ensuite le nombre de robots déployés à zéro, puisque la limite de robots maximum par simulation est de trois.

La priorité étant de trouver des gisements et d'envoyer un robot de forage, les algorithmes *test_pros* et *envoyer_forage* sont appelés en premier. Le premier permet de repérer un gisement grâce à un robot de prospection, et *envoyer_forage* envoie directement un robot de forage si un gisement est trouvé, la priorité étant d'extraire de la ressource de ce gisement. Ainsi au début de la simulation infinie, les robots de prospections n'ont pas forcément trouvé de gisement, car ils ne sont pas tous déployés, mais ces actions seront effectuées en premier à la création et au déplacement des robots de prospection.

Par la suite, l'algorithme *envoyer_transport* se charge de déployer un robot de transport vers un gisement dès qu'un robot de forage l'a foré. Celle permet de récupérer la ressource le plus vite possible. Avant d'être envoyé, nous vérifions que la distance que peut parcourir le robot est supérieure à celle du trajet aller et retour entre le gisement et la base. Si ce n'est pas le cas, le robot est systématiquement envoyé à la maintenance.

De même pour l'algorithme de communication, *envoyer_communication.* Pour cela nous avons utilsé une matrice représentant l'espace. Celle-ci est iniatialisée à false partout, puis nous la parcourons pour placer des robots de communication dans chaque entrée. Ceux-ci doivent être séparés par une distance inférieure à rayon_comm, ainsi l'espace doit contenir 7X7 robots de communication pour que celui-ci soit couverts dans son intégralité.

L'algorithme *envoyer_prosepct* est appelé juste après. Les robots de prospection ne sont créés et envoyés qu'un seul par boucle de simulation. Notre stratégie pour quadriller tout l'espace était d'en placer à un intervalle inférieur au rayon minimum d'un gisement. Ainsi ils sont déployés sur l'axe des abscisses d'abord selon cet intervalle, puis ensuite sur les ordonnées selon cette même distance pour pouvoir parcourir tout l'espace et ainsi ne pas oublier de gisement.

Une fonction *forage* vérifie ensuite que les robots de forage soient bien arrivés à leur but ainsi qu'un robot de transport se situe bien au même endroit qu'un forage. si c'est le cas, ils doivent retourner à leur base.

Quand ils sont bien retournés à la base, une fonction *recup_r* se charge de mettre à jour la quantité de ressource que possède la base.

Un algorithme *extract* se charge de faire baisser la ressource du gisement qui vient d'être foré, après l'avoir identifié.

1.4 L'algorithme de autonomous

Nous avons mis en place un algorithme *update_autonomous*. Celui-ci se charge de faire avancer les robots qui sont en mode autonomous vers leur but défini., après avoir normalisé leur position.

1.5 L'algorithme de déstruction de base

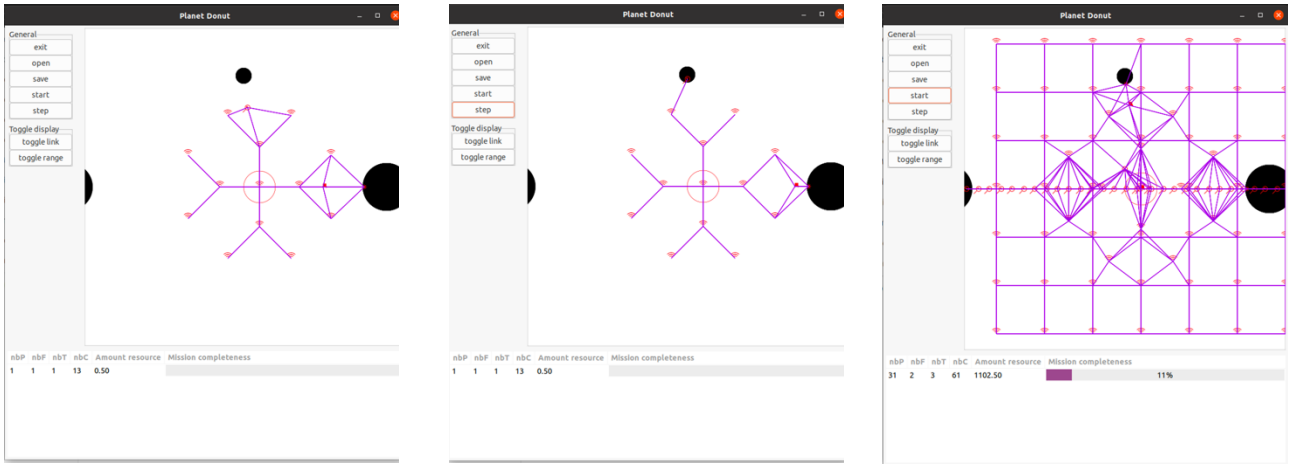
Finalement, *destroy_bases* détruit les bases si celles-ci n'ont plus de ressource.

Notre approche est robuste puisque nos stratégies permettent à la base de quadriller tout l'espace de la planète et ainsi de ne rater aucun gisement à exploiter. Une fois un gisement trouvé elle met en place les fonctions déclarées plus haut, pour pouvoir extraire le plus de ressource possible. Notre stratégie implique le déploiement d'une quantité importante de robot, ce qui coute de la ressource mais qui permet également d'un récupérer le plus. Nous avons décidé de mettre en place une maintenance systématique des robots pour permettre à ceux-ci de repartir à leur but et de ne pas être en mode autonomous. Cette stratégie place tous les robots en mode remote et chacun a son but précis. Chaque base fonctionne de la même manière, le fait qu'il y en ait plusieurs ne rendre pas en compte dans notre algorithme. Ainsi elles ne se font pas concurrence.

2. Captures d'écran de plusieurs pas de simulation

Fichier rendu3_image.txt

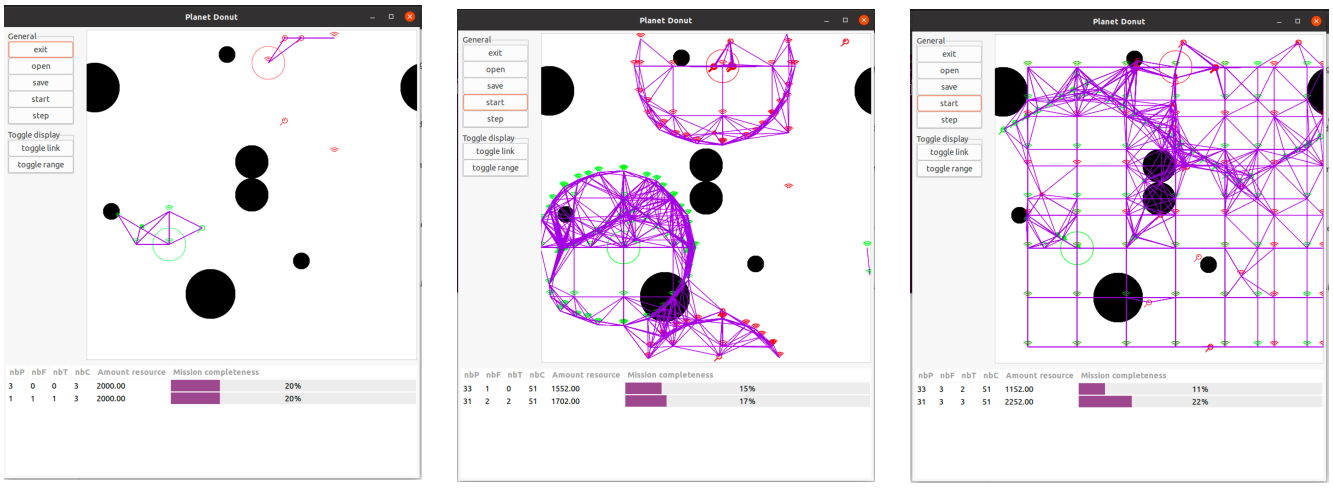
1. Au lancement :
2. Après quelques mises à jour, un robot transport est envoyé chercher de la ressource, un prosepcteur cherche des gisements.
3. Apres une quantité importante de mises à jour, l'espace est quadrillé, la ressource de la base augmente en continu



1. 2. 3.

Fichier test.txt

1. Au lancement :
2. L'espace commence à se quadriller de robots de communication, l'état initial ne permettant pas de récuper de la ressource
3. Apres une quantité importante de mises à jour, l'espace est quadrillé, la ressource de la base augmente en continu



1. 2. 3.

3. Méthodologie et conclusion

Le premier rendu a été fait en grande partie grâce aux heures du vendredi après-midi, ou nous avons travaillé ensemble. Ces heures nous ont permis de bien l'avancer, Cesar a fini ensuite les fonctions géométriques, tandis que Marine l'architecture avec le module projet, et le makefile.

Le deuxième rendu a été commencé également ensemble, mais la charge de travail étant plus importante nous avons plus travaillé seuls. César a mis en place les modules de *Simulation*, *Base* et *Robot*, tandis que Marine s'est occupée de *Proje*t, *Simulation* et *Gisement*.

Pour ce dernier rendu nous avons réparti le travail de la manière suivante : César s'est occupé de la stratégie du projet, de la boucle infinie. Cette stratégie a apporté des modifications sur les fichiers de *Simulation, Base, Robot et Gisement.* Marine s'est chargé de l'interface graphique et a donc créé les fichiers *Gui*, *Graphic* et cela a entrainé des modifications en particulier sur le module *Geomod* mais également *Base, Gisement et Simulation*. Nous nous sommes très régulièrement vus pour pouvoir s'expliquer nos parties et pour les rassembler au fur et à mesure de notre avancement.

Notre problème le plus récurrent durant ce projet était les segmentation fault. Surtout pour le dernier rendu, où de nombreux pointeurs ont été mis en place, créant de nombreux problèmes de ce type. Le plus gros problème concernait la méthode *on_draw* de gui.cc qui appelait des méthodes de simulation. Ces méthodes devaient être appelées avec une instance, qui était celle donnée sur la ligne de commande lors du lancement du projet. Nous avons dû créer un pointeur sur cette simulation de projet.cc pour pouvoir l'appeler depuis gui.cc, ce qui a nous posé de nombreux problèmes de segmentation fault tout au long de gui.cc, la simulation n'étant pas supprimée, le pointeur ne pointait sur rien. Nous avons finalement réussi à régler ce problème après avoir alloué dynamiquement de la mémoire.

Nos qualités ont été notre organisation et notre bonne entente qui nous ont permis de surmonter les problèmes. De plus nous débuggions souvent les problèmes de l'autre, un point de vue externe étant toujours utile pour repérer les problèmes.