Arthur Chansel #324265
Benoît Vignon #327737

Rapport Rendu 2

Description des éléments du projet

L'exécution se fait à partir du module projet. Celui-ci est en haut de la hiérarchie composée de 8 modules en tout. Le projet interagit seulement avec le module Simulation.

Le module Simulation comprend essentiellement la classe Simulation. Celle-ci comporte comme attributs principaux les ensembles de gisements, Egis, et de bases, Eb. Ce sont des vecteurs de pointeurs sur des gisements et des bases respectivements. L'attribut état correspond à la manière utilisée pour interpréter les données lues. Simulation a également la méthode de lecture qui permet de lire et formater le fichier en prenant les lignes unes par unes. La lecture envoie les lignes à une autre méthode décodage. Celle-ci crée des pointeurs de gisements, de bases, ou des robots en fonction de la variable état en leur allouant dynamiquement de l'espace mémoire grâce à new.

La classe Gisement est le seul objet du module du même nom. Le gisement a comme attributs un cercle range définissant sa surface et un double cap pour sa capacité. Le constructeur de gisement appelle une méthode de décodage qui lui est propre et qui stocke les informations du fichier dans ses attributs. Le décodage ajoute également le gisement à l'ensemble Egis, passé par référence.

La classe Base est gérée par le module base. Ces objets reprennent des éléments de Gisement, comme le range et le stock qui est analogue à la capacité. Une base comporte en outre un ensemble de pointeurs sur des robots qui lui est propre. Le constructeur de la base appelle également une méthode de décodage, qui est similaire à celle des gisements, mais qui en plus lit le nombre de robots de chaque type. Les bases créent aussi les robots à travers des méthodes uniques.

Le module Robot s'occupe de la super-classe Robot et de chaque sous-classe associée à un type de robot, prospecteur, foreur, transport et communication. Un robot a comme attribut un point position, un point but, des doubles uid, dp et un booléen atteint. Le prospecteur a retour et found comme booléens supplémentaires, ainsi que les informations d'un gisement s'il en a trouvé. Le robot de communication et le foreur n'ont pas d'attributs spécifiques, tandis que le transporteur a un booléen de retour. Les constructeurs des sous-classes appellent les méthodes de décodages respectives qui les initialise et les ajoute à l'ensemble des robots des bases correspondantes.

Les tests sont effectués à différents endroits dans le code. Les tests de superposition reprennent la fonction d'intersection de cercle inclue dans geomod. Ils sont effectués lors de la création des gisements et des bases. Le test d'uid est fait dans le module robot lorsqu'on le crée. Le test du robot de communication est executé dans le module simulation après l'initialisation de tous les robost de ce type de chaque base.

Une simulation correspond à un point de vue global sur la planète, ce qui lui permet de connaître les ensembles de gisements et de bases. Cependant, l'ensemble des robots est propre à chaque base, c'est pourquoi il se trouve dans le module base.

Pseudocode update_voisin():

update_voisin (méthode de la classe simulation)
Entrée: base1, base2
parcours(base1) parcours(base2) Si intersection_cercles(base1.range, base2.range) ajouter(reseau.base1, reseau.base2) ajouter(reseau.base2, reseau.base1)

parcours (méthode de la classe base)
Entrées modifiées: base (liste_rob, réseau, visited)
Si taille(liste_rob) != 0 recursivité(réseau, visited, 1, liste_rob)

récursivité (méthode de la classe base)
Entrée: index, (liste_rob)
Entrée modifiée: (réseau, visited)
visited(index) ← vrai Ajouter (réseau, liste_rob(index)) Pour i de 1 à taille(réseau) Si visited(find_index(liste_rob, réseau(i))) = faux recursivité (réseau, visited, réseau(i).find_index(liste_rob), liste_rob)

find_index (méthode de la classe robot)
Entrée: liste_robot, robot
Sortie: index
Pour i de 1 à taille(liste_rob) Si liste_rob(i).uid = uid Sortir i

//Puisque ce sont des méthodes, nous avons accès aux attributs directement, ils sont notés entre parenthèses dans les entrées/sorties pour faciliter la compréhension du pseudocode

Arthur Chansel #324265
Benoît Vignon #327737

Rapport Rendu 3

Description des algorithmes

Algorithme de connexion

Cet algorithme est appelé à chaque mise à jour de la simulation par la méthode *update_sim*. Pour chaque base de la simulation, après la méthode *update_voisin* nous appelons la méthode *connexion*. Elle prend en paramètre l'index du robot par lequel l'algorithme récursif commence ; le premier appel commence par le robot se trouvant au centre de la base.
À chaque appel, la méthode passe l'attribut *linked_* du robot i à 1 pour indiquer qu'il a déjà été visité par l'algorithme, puis l'adresse du robot est ajoutée au tableau *Reseau_tot_*.
On répète cela de manière récursive pour tous les robots de la base qui se trouvent à une distance *rayon_comm* du robot i et qui n'ont pas encore été visités.

Prospection

Les robots de prospection sont créés systématiquement, à une certaine fréquence. Cette production s'arrête si on atteint le nombre maximal défini arbitrairement. Ces paramètres changent si on est en état d'urgence. Nous avons remarqué qu'à la limite du nombre de prospecteurs, la carte était largement couverte et les gisements tous détectés.
La décision d'un nouveau but se fait sur des cercles centrés sur la base. Nous établissons une liste de points à visiter sur chaque cercle, de sorte à couvrir toute la surface avec un minimum de prospecteurs. On visite d'abord les régions autour de la base, puis on étend progressivement le rayon d'exploration. Le tout a une géométrie hexagonale. Mais si un prospecteur n'a pas assez d'essence pour aller au prochain point puis rentrer à la base, alors son but devient la base, dans l'optique d'effectuer une maintenance.
La maintenance est systématique lorsqu'ils rentrent à la base puisqu'elle est peu coûteuse.

Afin d'interagir, les robots de forage et transport ont un système d'acolytes. Le foreur admet au maximum 5 acolytes et le transporteur un seul.

Forage

Lorsqu'un prospecteur transmet les informations d'un gisement à la base, celle-ci attribue une note au gisement. La note dépend de la distance, de la capacité du gisement et du nombre de foreurs propre à la base avec ce gisement comme but. Suivant la durée écoulée et la quantité de ressources accumulée, le seuil de décision de création d'un foreur évolue.
Le but attribué à un foreur est le point le point le plus proche d'un gisement connu. On ne réattribue pas de but par la suite car le foreur reste fixe une fois arrivé, même quand le gisement visé est entièrement exploité.
Il n'y a pas de maintenance puisque les foreurs ne reviennent jamais à la base.

Transport

Des robots de transports sont créés lorsqu'un gisement est découvert et si aucun autre transporteur n'attend d'ordres à la base. La limite de création est fixée arbitrairement. Nous ne pouvons pas créer de transporteur si son acquisition entraînerait un état d'urgence.

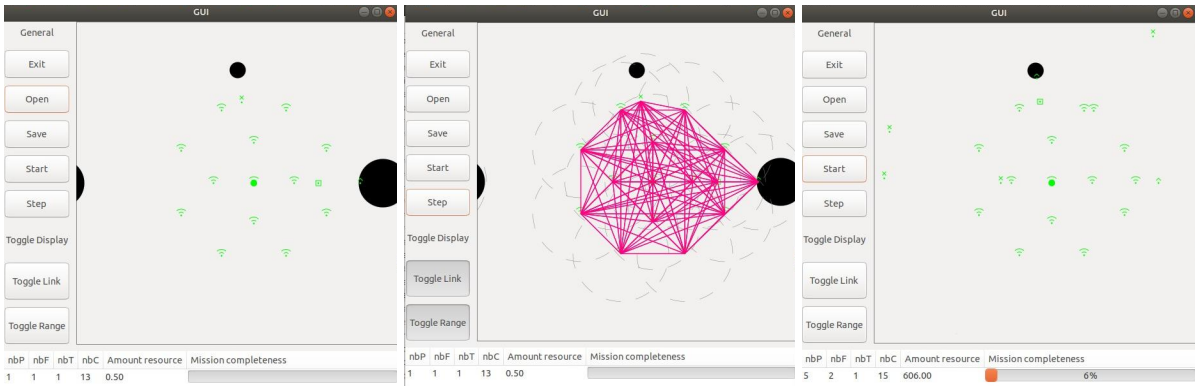
Le but d'un transporteur est la base s'il est chargé en ressources. Sinon, s'il n'a pas d'acolyte alors il cherche un foreur actif ayant un but qui deviendra le sien (il faut aussi que le foreur puisse accepter un nouvel acolyte). Et si ces conditions ne sont pas remplies, alors il rentre à la base et est disponible pour une redéfinition de but immédiatement. On redéfinit un but lorsque le robot est arrivé à son but précédent.
La maintenance est systématique lorsqu'il rentre à la base puisqu'elle est peu coûteuse.

Communication
Les robots de communication sont créés systématiquement à une certaine fréquence, elle change si on est en état d'urgence. Cette production s'arrête si on atteint le nombre maximal qui correspond à la couverture quasi-totale de la carte. Il est atteint au bout de 1250 updates.
Les buts sont uniquement définis à la création. Ils sont déterminés en suivant le même raisonnement que les points des prospecteurs.
Il n'y a pas de maintenance puisqu'ils ne reviennent jamais à la base.

Nous estimons notre code robuste car les décisions prises dépendent de plusieurs facteurs qui changent régulièrement. Le système de notation de gisements permet de limiter les dépenses notamment en créant un nombre de robots cohérents avec le contexte. De plus que les foreurs et transporteurs sont particulièrement onéreux. La création de points basés sur une géométrie hexagonale permet de couvrir la carte de manière rapide, pour les communications et prospecteurs. Mathématiquement, c'est la géométrie la plus optimisée pour couvrir une surface avec des cercles. L'état d'urgence assure une stabilité élevée lorsque les stocks sont bas. La redéfinition de but des prospecteurs permet de les maintenir en vie tout au long de la simulation.
Nous avons varié nos paramètres arbitraires afin de prendre des décisions assurant la survie et la victoire des bases autant que possible.

Captures d'écran

rendu3_image.txt



rendu3_t04.txt

Méthodologie et conclusion

Pour ce rendu 3, Benoît s'est principalement occupé des algorithmes de simulation, et Arthur était responsable de l'interface graphique. Nous avons pu développer ces deux parties indépendamment dans un premier temps. Puis lorsque l'interface graphique fut terminée, nous avons pu nous concentrer ensemble sur le bon fonctionnement du projet et son optimisation.
Les algorithmes de simulation étaient difficiles à tester sans l'interface graphique, c'est pour cela qu'il était plus compliqué d'avancer individuellement sur cette partie.
L'interface graphique était plus simple à tester, en affichant des formes standard sur la map et vérifier son rebouclement, la distorsion, etc.
Pour tester le tout, nous avons utilisé des fichiers de lecture afin de voir comment la simulation se déroulait.
Cette méthodologie n'a pas posé trop de problèmes, et nous ferions la même chose avec le recul.

Le plus gros problème que nous avons rencontré serait sans doute les erreurs sémantiques, car il est plus compliqué de trouver d'où cela vient. Pour les résoudre, il nous a fallu du temps et reprendre la logique de nos algorithmes pas à pas, en affichant des valeurs à des endroits stratégiques du code.

Nous avons essayé de fournir un code robuste en exploitant les outils mis à disposition par le cours. C'est là que nous voyons que le formalisme est important et qu'il nous a souvent permis de gagner du temps et d'éviter des erreurs. Notre code n'adopte peut être pas le meilleur des formalismes étant donné que notre vision a évoluée au fil du temps, alors peut-être que nous aurions structuré différemment certaines choses s'il fallait recommencer. (par exemple, utiliser des struct dans geomod aurait été plus simple que des classes).
Nous avons pu constater que notre programme tourne assez rapidement, même avec des fichiers lourds. Cela indique peut-être l'optimisation de nos algorithmes.

L'état d'urgence est un des points forts de notre programme. Lorsqu'une base passe dans ce mode, elle minimise ses dépenses, ce qui lui permet de survivre encore longtemps et d'avoir le temps de trouver des gisements à exploiter.