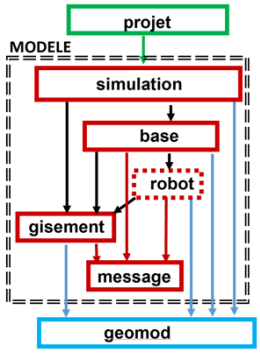
325978 (Da Silva Daniel Justino)
328434 (Georg Schwabedal)

Rendu 2 Rapport Pdf

1. Mise en place du code et explication de la construction générale du projet.



Comme décrit lors de la donnée du projet nous avons reconstruit le squelette décrit dans le schéma ci-contre. On peut dès lors faire la liste des modules que nous avons repris : geomod et message, et la liste des modules crées lors de ce rendu : gisement, robot, base, simulation et projet (tous sauf projet sont .cc et .h).

Tout d'abord nous allons décrire le module **geomod**. De manière très objective le rendu final (Planète Donut) n'est qu'un ensemble de cercles en interactions. D'où la création d'une classe : *Cercle*. Un cercle est caractérisé par un rayon et un centre d'où le choix de créer une classe : *Point*. C'est dans cette dernière qu'est utilisée la matrice de points d'adjacence pour le principe de retournement de l'espace. Enfin nous avons créé une classe *Vecteur* avec deux points comme arguments. Cette classe est plutôt fonctionnelle et a comme rôle les calculs mathématiques avec les deux classes précédentes. Elle a cependant aussi été créée dans la possibilité d'avoir une instance de mémorisation d'un déplacement pour le futur peut être.

Nous avons repris le module **message** tel que donné lors de la consigne du rendu 2 et nous l'avons appliqué pour afficher les messages d'erreurs souhaités.

Nous allons maintenant traiter le module de **gisement**. Il possède une unique classe : *Gisement*. Elle se charge de représenter un gisement au travers d'un cercle et rassemble ses caractéristiques telles que sa capacité de ressources dans ses attributs. La totalité de ces instances sont conservées dans un tableau (vector) de Gisement. Il suffit d'une seule méthode (ajout_Gisement) qui vas ajouter le nouveau gisement et le rendre ainsi repérable seulement si les tests de vérification sont passés. (Nous viendrons sur la localisation du tableau plus tard).

Dans le module **robot**. On a créé une superclasse *Robot* qui rassemble les variables communes à tous les types de robots. Chaque Robot n'est en fait rien d'autre qu'un Cercle en déplacement et a donc une telle instance dans ses attributs. Nous avons exploité le principe d'héritage pour la création des Robots particuliers à travers de sous-classes. Leurs noms sont : *Prospecteur*, *Forage*, *Transport* et *Communication*. Chaque sous-classe a sa propre liste d'attributs propres décrite dans la donnée du projet. Les robots sont conservés dans un tableau comme nous le verrons dans un instant.

Venons-en au module de **base**. Ici comme pour gisement il n'a y pas beaucoup plus qu'une classe *Base*. Cette dernière est un cercle possédant une liste de Robots qui lui sont attribués avec des caractéristiques propres telles que ses ressources. Chaque base a ses propres robots de tous les types. Ceux-ci sont stockés dans un vector de Robots dans leur base respective. C'est ici que se trouve la méthode de création de robots. Elle permet de parcourir le tableau et d'effectuer au même moment les tests de vérification, un robot n'est inscrit que si tous les tests ont été un succès. De manière analogue aux Robot et aux Gisement les Base sont stockés dans un troisième tableau qui effectue également les tests qui s'imposent lors de l'ajout de la base.

Enfin le module **simulation**. Les tableaux de Gisement et de Base sont stockés ici. La fonction de lecture de fichier également. Cela permet un accès plus direct lors des tests inter base-gisement. Et de manière générale aux instances de chaque classe.

L'idée général de ce squelette est de faire un maximum de tests inter-modulaires au niveau du module simulation. Cette procédure est choisie parce qu'elle permet de réserver les modules

uniquement à leurs classes correspondantes. La stratégie étant ici de minimiser les actions entre modules et rassembler les actions dans un seul lieu : le module simulation.

Le module **projet** rassemble uniquement la fonction main() et se contente d'appeler la fonction lecture() du module simulation.

De manière générale nous tâchons de privilégier au maximum la simplicité. Pour ce rendu. C'est aussi la raison pourquoi nous nous sommes décidés à faire des tableaux d'instances (pour les tableaux Gisement, Base, Robot) et pas encore des tableaux de pointeurs pour rester flexible par rapport à ce qui nous attend dans le rendu 3.

2. La fonction update_voisins().

Le but de cette fonction est de mettre à jour un tableau de tous les robots voisins de chaque robot d'une base. Il faut donc juste comparer les cercles qui décrivent chaque robot et regarder s'ils sont en intersection. Il semble donc logique que chaque robot ait son propre tableau de robots-voisins dans lequel on conservera liste les tableaux voisins. On va ici référer aux robots directement sans passer par le billet de pointeurs mais il va de soi qu'il faudra utiliser ces derniers dans le code réel. On peut se poser la question quand est ce que les robots d'une même base se comparent ? Tout simplement quand on prend comme argument deux fois la même base comme arguments. Cependant ce cas implique que l'on doit vérifier que le robot ne se prenne pas lui-même comme voisin. Ainsi on utilise une dernière boucle if qui rajoute le robot seulement si la position des bases des robots comparés sont différentes ainsi que leurs uid.

update_voisin(Base A , Base B)
Pour i allant de 1 à taille du tableau de robots de A : **Vider** le tableau des voisins du robot i de la base A **Pour** j allant de 1 à taille du tableau de robots de B : **Si** la norme (centre robot i de la base A, centre le robot j de la base B) < rayon - epsilon **Si** égalité (Centre de la base A , Centre de la base B)== faux **and** uid du robot i de la base A != Uid du robot j de la base B Ajouter le robot j de la base B au tableau des robots voisins du robot i de la base A

Tableau explicative des fonctions utilisées précédemment

norme (Point A , Point B) :	Cette fonction redonne la distance entre deux points sous la forme d'un double
Egalite(Point A , Point B) :	Fonction qui vérifie si deux points se superposent