1.0

Projet Informatique – Sections Electricité et Microtechnique

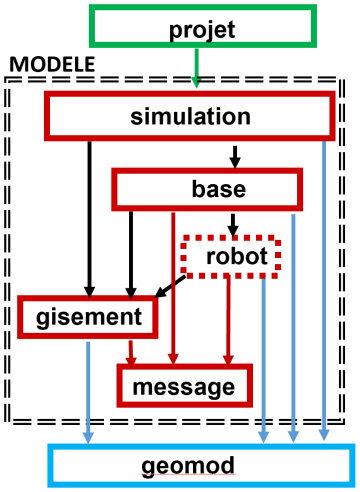
Printemps 2021 :Planet Donut © R. Boulic & collaborators

Rendu2 (dimanche 25 avril)

Objectif de ce document : *en plus de préciser ce qui doit être fait, ce document identifie des ACTIONS à considérer pour réaliser le rendu de manière rigoureuse. Ces ACTIONS sont équivalentes à celles indiquées pour le projet d'automne ; elles ne sont pas notées, elles servent à vous organiser. Vous pouvez adopter une approche différente du moment que vous respectez l'architecture minimale du projet (donnée Fig 6b). La différence principale avec le projet du semestre d'automne est qu'avec la compilation séparée il y a moins de contraintes concernant l'ordre d'exécution des actions, c'est pourquoi elles n'ont pas de numéro. Il y a aussi plus de liberté dans les choix de structuration des données ; vous avez une certaine marge de manœuvre si les options proposées ne vous conviennent pas. Sans surprise on retrouvera l'approche introduite en TP de Topic1 sur les méthodes de développement de projet.*

1. Buts du rendu2 : conception des structures de données et tests de fichiers

Avant toute écriture de code, le travail demandé pour le rendu2 est de **concevoir les structures de données** du Modèle en respectant les responsabilités des différents modules (Fig ci-contre).



Donnée Fig 6b

N'utilise PAS GTKmm

De plus le rendu2 doit déjà être *compatible avec la structuration des données requise pour la simulation même si celle-ci est seulement demandée pour le dernier rendu*. Sachant que les modules du Modèle mettent en œuvre des classes en respectant le *principe d'encapsulation*, il ne sera pas possible pour un module externe d'accéder aux attributs d'un module donné. C'est pourquoi, même si le **Pseudocode 1 de la Donnée** doit être votre guide, celui-ci ne peut pas être traduit directement en code. Une mise à jour de la simulation constitue effectivement une des tâches du module **simulation** mais ce module ne dispose pas d'un accès direct aux attributs des bases car les bases sont gérées dans leur propre module **base**. Il faut donc déléguer la suite de cette tâche au module **base** car cela donne accès à ses attributs.

Les responsabilités des modules ébauchées en section 7.2 de la Donnée sont complétées ici :

- **simulation** : SEUL point d'entrée du Modèle vis-à-vis de l'extérieur (module projet pour le rendu2)
  - gère au plus haut niveau d'abstraction les tâches de (*une*) mise à jour de la simulation, lecture et écriture de fichier.
  - Délégue, en vertu du principe d'abstraction, l'exécution des sous-problèmes de ces tâches auprès des modules qui gèrent les entités de **base** et de **gisement**.

- **base** : gère à son niveau d'abstraction les tâches qui lui sont déléguées par le module **simulation** et délègue la mise en œuvre des sous-tâches aux modules **robot**(s) et **gisement**.

- **robot** : il s'agit *d'une hiérarchie de classes* (deux niveaux suffisent) au choix dans un ou plusieurs modules. Ces modules gèrent les tâches qui leur sont déléguées par le module base. Les robots de prospection et de forage ont besoin du module **gisement** pour détecter la présence d'un gisement ou savoir son état courant.

- Le module **message** est *fourni* pour l'affichage de messages standardisés pour la tâche de lecture.

1.1 Rapport décrivant la structuration des données du Modèle (MAX 2 pages):

Le rapport doit décrire comment sont structurées les classes nécessaires pour le projet. Cela doit se traduire par une brève description des éléments suivant pour chaque module :

- **Quelles sont les données** dont ce module est responsable : préciser les **classes, types, noms** et **buts**

- **Où sont les ensembles de données ?** Sachant qu'une instance d'une classe décrit une seule entité MAIS que la simulation devra éventuellement en gérer plusieurs, où prévoyez-vous de mémoriser ces ensembles d'instances.

    - Plus concrètement, où existera votre ensemble de bases ? Dans une instance du module **simulation** ou dans le module **base** ? De même où existeront les ensembles de robots gérés par chaque base ? Dans les instances de **base** ou ailleurs ?

    - L'important est d'indiquer brièvement POURQUOI vous avez fait votre choix en vous appuyant sur la manière dont vous anticipez que ces ensembles seront utilisés pour la simulation. L'élément suivant du rapport est un premier pas dans cette direction.

Décrivez dans la section finale du rapport le **pseudocode** de la tâche du calcul des voisins **`update_voisin(…)`** qui est appelé en début de chaque mise à jour de la simulation (**Pseudocode 1 Donnée**). A la différence du Pseudocode1 le vôtre doit respecter le *principe d'encapsulation.* C'est pourquoi les paramètres de la tâche sont seulement deux bases. Précisez comment cette tâche se décompose au niveau des modules **base** et **robot**(s). vous pouvez faire apparaître les types, fonctions ou methodes que vous avez défini dans votre module **geomod** pour ce pseudocode.

Remarque : cette tâche du calcul des voisins devra aussi être automatiquement appelée immédiatement après la lecture d'un fichier pour le rendu final car la simulation est une tâche indépendante de la lecture de fichier ; l'idée est de pouvoir dessiner le réseau des voisins avant de lancer la première mise à jour de la simulation.

1.2 Tests de la lecture de fichiers de configuration

1.2.1 Module projet

Pour ce rendu le code de scaffolding de main() correspondant aux scenarios de test du module **geomod** doit être enlevé car on considère que ce module est validé. De même le module **message_geomod** n'est plus utilisé.

Le module **projet** contient la fonction **main()** en charge d'analyser si la ligne de commande est bien conforme à la syntaxe suivante : **`./projet test1.txt`**

Où **`test1.txt`** est un fichier de configuration (Donnée **Section 4**).

Nous ne testerons pas l'absence de l'argument. Pour le rendu2, votre programme doit se terminer en cas d'absence d'argument. Si l'argument est présent nous supposons qu'il correspond à un fichier de test présent dans le répertoire courant. Ce nom de fichier doit être transmis à une fonction ou méthode de **lecture** du module **simulation.**

Deux stratégies sont autorisées concernant **l'instance** de la classe Simulation qui pilote le Modèle :

- Elle existe dans le module **projet** et une méthode lecture est appelée sur cette instance.
- Elle est cachée dans le module **simulation** (car elle est unique) et une fonction lecture du module simulation est appelée sans avoir besoin de préciser en paramètre sur quelle instance elle travaille.

Nous ne testerons pas l'absence du fichier / l'échec de son ouverture ; si cela arrive la lecture doit se terminer avec exit().

Le programme cherche à initialiser l'état du Modèle ; il **s'arrête** dès la ***première*** erreur trouvée dans le fichier.
Il **s'arrête** aussi après la lecture du fichier s'il n'y a aucune erreur ; dans ce cas, il y a affichage d'un message indiquant le succès de la lecture.

Remarque sur l'organisation des données après la lecture : la Donnée insiste à plusieurs endroits sur le fait que les ***ensembles d'instances doivent être organisés dans le même ordre que dans le fichier*** (et pas dans l'ordre inverse par exemple). Cela est important pour le rendu3 pour comprendre l'exécution séquentielle de la simulation et pour le codage de la couleur de l'affichage.

1.2.2 Liste des erreurs à détecter (Donnée section 8.2):

Voici la liste finale des points à vérifier pendant la lecture

- Absence de collision entre gisements
- Absence de collision entre bases
- Absence de collision entre gisements et bases
- Absence de duplication d'uid de robots au niveau de chaque base
- Présence d'un robot communication au centre de chaque base

Nous ne testerons pas vos projets sur la longueur des lignes de fichier ni sur d'autres éventuelles incohérences. De votre coté il faudra être vigilant pour ne pas introduire d'erreur au moment de l'écriture du fichier (rendu3) ou de l'édition manuelle de fichiers de test. Par exemple, les paramètres d'un robot de forage doivent être cohérents avec l'état courant du gisement sur lequel il est actif. Autre valeur à surveiller : la ressource d'un gisement est bornée par la valeur donnée par l'équation 1 (Donnée section 2.2 p3).

Nous vous fournissons un module **message** comme détaillé maintenant.

1.2.3 Utilisation du module message :

Le module **message** est fourni dans un fichier archive sur moodle ; il ne doit pas être modifié.
Son interface **message.h** détaille l'ensemble des fonctions à appeler.

Ces fonctions sont prévues pour afficher un message vers `cout` (et pas ailleurs) comme ceci :

```
if(une détection d'erreur est vraie)
      cout << message::appel_de_la_fonction(paramètres éventuels);
```

Le message affiché par ces fonctions se termine par un passage à la ligne. Il n'est pas nécessaire d'en ajouter un. Une seule fonction n'est pas liée à une détection d'erreur ; elle doit être appelée quand la lecture de fichier et toutes les validations ont été effectuées avec succès :

```
cout << message::success();
```

2. Organisation du travail

Comme indiqué dans les objectifs de ce rendu, les ACTIONS suivantes ne sont pas notées, elles servent à vous organiser et parfois à travailler indépendamment pour les deux membres d'un groupe. Vous pouvez adopter une approche différente du moment que vous respectez l'architecture minimale du projet (Donnée Fig 6b). La différence principale avec le projet du semestre d'automne est qu'avec la compilation séparée il y a moins de contraintes concernant l'ordre d'exécution des actions, c'est pourquoi elles n'ont pas de numéro.

2.1 Module projet

2.1.1 ACTION : test du module projet / de la ligne de commande

La fonction main() peut très rapidement être testée pour savoir si le bon nombre d'argument est fourni. Si c'est le cas, transmettre la chaine de caractère du nom de fichier à un *stub* de la fonction/méthode du module **simulation** en charge de lire un fichier. A ce stade le module **simulation** peut être réduit à une interface très incomplète ; seule compte d'avoir la fonction/méthode utile pour la lecture pour faire ces tests.

2.2 Le sous-système du Modèle

2.2.1 Module simulation

Ce module gère une classe Simulation dont ce programme va utiliser une seule instance.

2.2.1.1 ACTION : tests du module simulation

Au stade du rendu2, seul le constructeur et la fonction/méthode de lecture de fichier sont nécessaires. Dans ce module l'opération de lecture met en place *l'automate de lecture* (cours Topic3) qui filtre les lignes inutiles du fichier et délègue l'analyse fine de lecture d'une ligne aux autres modules plus spécialisés (gisement, base, robot(s)) qui ont la responsabilité de faire les vérifications nécessaires. L'automate peut être testé avec des *stubs* de ces fonctions/méthodes plus spécialisées de lecture qui renvoient toujours true pour indiquer que le décodage de la ligne de fichier s'est bien passé.

Certaines vérifications impliquent d'avoir déjà lu des ensembles d'éléments différents (ex : gisement et base). Dans ce cas il faut prévoir d'appeler la fonction/méthode de vérification supplémentaires après la fin de la lecture elle-même.

A ce stade l'intégration avec le module projet est immédiate puisqu'il suffit de remplacer le stub de la fonction/méthode de lecture du fichier par un appel de celle mise au point pour le module simulation.

2.2.2 Module gisement

Ce module gère une classe Gisement relativement autonome dans le sens où d'autres modules ont besoin de lui mais lui n'a besoin de personne (seulement les types élémentaires de **geomod** et le module **message**).

2.2.2.1 ACTION : tests du module gisement

Au stade du rendu2, seul le constructeur et la fonction/méthode de décodage d'une ligne lue dans un fichier sont nécessaires.

Sachant qu'il faut mémoriser un ensemble E_g d'instances de Gisement, il est possible d'écrire un peu de code de scaffolding qui ajoute un nouveau gisement G à un tel ensemble seulement s'il n'intersecte pas un Gisement déjà présent dans E_g. Une telle vérification est effectuée plus efficacement par une méthode de la classe Gisement car elle a accès à tous les attributs private de cette classe.

Une fois effectué cette validation est possible d'intégrer ce module avec celui de Simulation ; c'est à vous de décider où l'ensemble E_g existe : est-ce un attribut de Simulation ou est-ce un ensemble caché dans l'implémentation de Gisement ?

2.2.3 Module base

Ce module est un intermédiaire entre la classe Simulation et la hiérarchie de classes des robots. Ainsi, le décodage d'une ligne de fichier contenant un robot doit être partiellement traité par le module **base** comme suit : le module **base** délègue aux modules **robots** pour le décodage de la ligne lue dans un fichier pour initialiser un robot et ensuite il décide où mémoriser le robot ainsi créé.

Rappel : toute base doit obligatoirement avoir un robot de communication positionné au centre de la base. Cela se détecte dès le décodage des données d'une base puisque les nombres de robots y sont fournis.

2.2.3.1 ACTION : tests du module base

Au stade du rendu2, seul le constructeur et la fonction/méthode de décodage d'une ligne lue dans un fichier sont nécessaires. On mettra en place des stubs pour simuler le décodage d'une ligne de robot. Sachant que chaque base gère ses robots il semble logique de prévoir la mémorisation des ensembles de ces entités[1] comme des attributs de la classe Base.

De plus sachant qu'il faut mémoriser un ensemble E_b d'instances de Base, il est possible d'écrire un peu de code de scaffolding qui ajoute une nouvelle Base à un tel ensemble seulement si elle n'intersecte pas une Base déjà présent dans E_b. Une telle vérification est effectuée plus efficacement par une méthode de la classe Base car elle a accès à tous les attributs private de cette classe.

Une fois effectué cette validation il est possible d'intégrer ce module avec celui de Simulation ; c'est à vous de décider où l'ensemble E_b existe : est-ce un attribut de Simulation ou est-ce un ensemble caché dans l'implémentation de Base ?

2.2.3.2 ACTION : test d'intersection gisement-base

Si vous avez intégré les modules **gisement** et **base** au module **simulation**, il est possible de détecter l'intersection des gisements avec les bases. Ici se pose une question : doit-on introduire une dépendance entre les modules **gisement** et **base** pour effectuer ce test ?

Réponse : en fait pas du tout si on réfléchit à la nature du test : seule l'information des cercles est nécessaire. Voici une manière de faire ce test : une méthode du module **simulation** met en place une simple boucle qui demande la position de chaque Base et qui la transmet à une fonction/méthode du module **gisement** qui se charge de tester si le cercle de la Base dont on transmet le centre intersecte l'un des gisements de la simulation.

2.2.4 Module robot : hiérarchie de classes (deux niveaux suffisent)

Vous pouvez créer un module par classe ou utilise un seul module pour les classes de cette hiérarchie.

2.2.4.1 ACTION : tests du module robot puis méthode de lecture puis intégration avec le module base

Commencez par mettre en place la hiérarchie en identifiant les attributs de la classe de base et ceux des classes dérivées. Utilisez autant que possible les types fournis par le module **geomod**. Ecrivez du code de scaffolding pour initialiser des instances des différentes classes et vérifier la valeur des attributs avec une méthode d'affichage dans le terminal.

Ensuite seulement écrivez la méthode qui décode la ligne du fichier pour chaque classe dérivée (section 4). Testez cette méthode en transmettant une ligne avec la syntaxe du fichier et vérifiez avec la méthode d'affichage que le décodage s'est bien passé.

Une fois que la lecture est validée à l'échelle de la hiérarchie de classes, vous pouvez passer à l'intégration avec le module **base** dans un premier temps. Vous pouvez remplacer les stubs de lecture des robots par les méthodes que vous venez de valider. Le module **base** devrait ensuite être testé avec une méthode d'affichage qui affiche l'ensemble des robots d'une base après plusieurs appels de la méthode de lecture de Base.

2.2.4.2 ACTION : tests d'uid unique à l'échelle de chaque base

Nous avons indiqué que les ensembles de robots étaient des attributs des bases. C'est seulement à ce niveau que l'unicité des uids des robots doit être vérifiée. Des robots appartenant à des bases différentes peuvent avoir le même uid.

---

[1] Sachant ce qui a été dit en cours sur la representation des graphes et sur que nous verrons sur le Polymorphisme, nous vous recommandons de mémoriser plutôt des ensembles de ***pointeurs*** sur les instances des classes robots.

3. Forme du rendu2

Pour chaque rendu **UN SEUL membre d'un groupe** (noté **SCIPER1** ci-dessous) doit télécharger un fichier **zip** sur moodle (pas d'email). Le non-respect de cette consigne sera pénalisé de plusieurs points.
Le nom de ce fichier **zip** a la forme : `SCIPER1_SCIPER2.zip`

Compléter le fichier fourni **mysciper.txt** en remplaçant 111111 par le numéro SCIPER de la personne qui télécharge le fichier archive et 222222 par le numéro SCIPER du second membre du groupe.

Le fichier archive du rendu2 doit contenir (**AUCUN répertoire**) :

- Fichier texte édité **mysciper.txt**
- Votre fichier **Makefile** produisant un executable **projet**
- votre code source (.cc et .h) y compris le module **message**

*On doit pouvoir produire l'exécutable **projet** à partir du* Makefile *après décompression du contenu du fichier zip. La commande* **make** *ne doit faire AUCUN déplacement de fichier ; tout reste dans l'unique répertoire créé par la décompression du fichier archive.*

Auto-vérification : Après avoir téléversé le fichier zip de votre rendu sur moodle (upload), récupérez-le (download), décompressez-le et assurez-vous que la commande **make** produit bien l'exécutable et que celui-ci fonctionne correctement.

Exécution sur la VM: votre projet sera évalué sur la VM d'automne (version du 23 septembre).

Backup : Vous êtes responsable de faire votre copie de sauvegarde du projet. Il y a un backup automatique seulement sur votre compte myNAS. Sur la VM, vous devez activer vous-même le backup (icone « engrenage » en haut à droite, choisir system settings, choisir backup et activer cette fonction en précisant les paramètres). Une alternative est de s'envoyer la dernière version du code source par email.

Outils possibles (mais pas obligatoires) pour la gestion du code au sein d'un groupe :

- vous pouvez envisager d'utiliser **gdrive.epfl.ch** pour définir un répertoire partagé par les 2 membres du groupe. Cependant, il n'y a pas d'éditeur de code en mode partagé.