Projet Informatique – Sections Electricité et Microtechnique

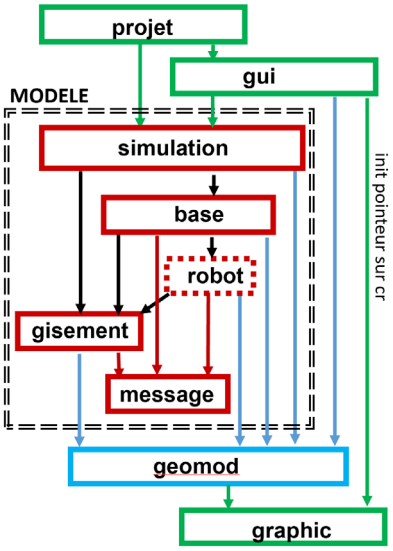
Printemps 2021 :*Planet Donut* © R. Boulic

Rendu3 (dimanche 23 mai)

Objectif de ce document : *comme pour le document des rendus précédents, ce document identifie des ACTIONS à considérer pour réaliser le rendu de manière rigoureuse. Ces ACTIONS 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 6c). A nouveau on s'appuiera sur la série sur les méthodes de développement de projet.*

1. Buts du rendu3 : dialogue avec l'interface graphique et simulation

Votre approche peut évoluer entre ce que vous avez décrit pour le rendu2 en matière de structuration des données et ce que vous mettez en œuvre finalement du moment que vous respectez les responsabilités des différents modules (Fig ci-contre). Ces responsabilités des modules du Modèle restent les même (section 7.2 de la donnée générale et la page 1 de la donnée du rendu2).
Rappel : 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.



Donnée Fig 6c

Les responsabilités des nouveaux modules sont ébauchées dans les sections 7.1 et 7.4 de la donnée générale et complétées ici :

- Le module **projet** contient la fonction **main**() en charge d'analyser si la ligne de commande est bien conforme aux deux syntaxes acceptées pour ce rendu (section 1.2). Le reste est seulement le lancement de l'application GTKmm (voir séries). Ce module reste très petit.

- Le gros module est **gui** qui crée l'interface graphique (Fig 4 de la donnée), gère la réponse aux événements d'interaction avec les boutons, effectue une mise à jour de la simulation à la fois quand il n'y a aucun événement à traiter (méthode **idle()**), et affiche le dessin / actualise les informations sur les bases avec la méthode **on_draw**() (voir séries).
  - Ce module **gui** exploitera l'interface du module **simulation** pour déléguer les commandes de lecture et sauvegarde de fichier lancées à partir des boutons. Même chose pour le dessin. Comme pour le rendu précédent, le module **simulation** délègue aux autres modules du Modèle les tâches de gestion de leur structure de données (lecture, sauvegarde, dessin). Le module **gui** peut aussi exploiter le module **geomod** si nécessaire.

- Le module **graphic** se charge de dessiner les figures géométriques simples qui lui sont demandées par le module **geomod** (ce dernier module doit être complété par des fonctions/méthodes de dessin car il est responsable de demander plusieurs affichages pour qu'on voit par exemple la forme d'un cercle des deux cotés du domaine quand il est sur une limite du domaine, comme sur la Fig1).
  - La gestion des couleurs est décrite dans la section 6 de la donnée générale.

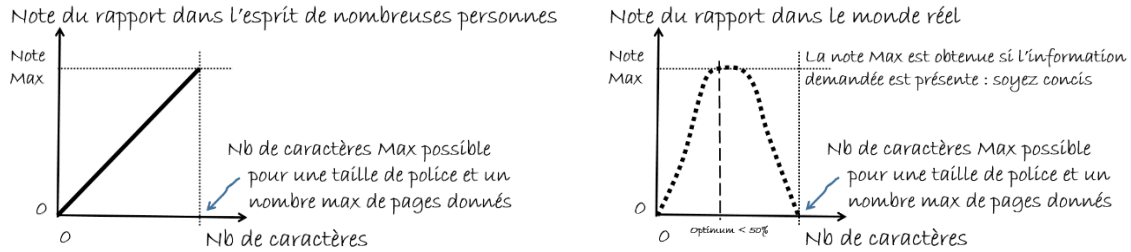


Figure 1 : un mauvais rapport dilue l'information utile jusqu'à atteindre le nombre maximum de caractères possibles sur la page (fig gauche) : en fait ce type de rapport sera pénalisé parce qu'il est peu lisible ; un bon rapport est celui qui fournit les informations demandées avec concision avec une mise en page aérée et lisible (fig droite)

1.1 Rapport final (MAX 2 pages):

Le Rapport ne contient PAS de page de titre, ni de table des matières. Il est écrit avec une police 11 au minimum et 14 au maximum, interligne simple. Le rapport est écrit en français ou en anglais ; une orthographe ou une grammaire défaillante peut induire les correcteurs en erreur. Le Rapport ne duplique pas le précédent. Il contient :

- **la description de vos algorithmes** mettant en œuvre votre **stratégie** qui doit respecter l'ordre des opérations indiquées par le pseudocode 1 de la donnée générale.
    - Algorithme de connexion (2.5.2) : mettez-vous en œuvre une structure de donnée à ce stade ?
    - Prospection (2.4.1), Forage (2.4.2), Transport (2.4.3), Communication (2.4.4).
        - Préciser vos choix vis-à-vis de :
            - La création d'un robot (sous quelles conditions ?)
            - La définition ou redéfinition du but du robot
            - La maintenance des robots (inutile ? systématique ?)
    - Préciser en quoi votre approche est ***robuste*** : c'est-à-dire comment vous pensez qu'elle permet au plus grand nombre de bases de devenir *autonomes* indépendamment de leur nombre et de leur répartition vis-à-vis des gisements, eux-mêmes pouvant être répartis de manières très variées.
- **Captures d'écran de plusieurs pas de votre simulation** pour un fichier de test fourni (rendu3_image.txt) et pour un autre fichier de votre choix.
- **Méthodologie et conclusion** : comment avez-vous organisé votre travail à plusieurs, indiquer la personne responsable de chaque module et comment vous avez organisé le travail au sein du groupe (par quels modules avez-vous commencé, comment les avez-vous testés, comment le feriez-vous maintenant avec le recul.
    - Quel était le bug le plus fréquent, pourquoi ? Quel est celui qui vous a posé le plus de problème et comment a-t-il été résolu, ...).
    - Pour conclure fournissez une brève auto-évaluation de votre travail et de l'environnement mis à votre disposition (points forts, points faibles, améliorations possibles).

Le rapport final doit être inclus dans le fichier archive du rendu final (en format pdf).

1.2 Syntaxes du lancement de l'exécutable pour ce rendu (sections 4.3 et 8.3 de la donnée) :
A partir du rendu3, ces deux syntaxes doivent être opérationnelles :

```
./projet nom_fichier.txt
```
ou
```
./projet
```

Avec ces deux syntaxes on lance le programme avec l'interface graphique GTKmm :

- La première syntaxe ouvre immédiatement le fichier fourni sur la ligne de commande et, en cas de succès, affiche le dessin de l'état initial de la planète Donut. En cas d'échec, c'est-à-dire dès la première erreur détectée à la lecture du fichier, le message d'erreur est affiché dans le terminal, les structures de données sont effacées (aucun affichage graphique n'est visible) puis le programme attend qu'on utilise l'interface graphique pour ouvrir un autre fichier.

- Avec la seconde syntaxe, le programme crée l'interface graphique et attend qu'on l'utilise pour indiquer un fichier à ouvrir. Le comportement est le même que ci-dessus pour le traitement de la lecture.

1.3 Evaluation du rendu3 : en plus d'évaluer les critères sur la qualité du code (ne pas oublier de corriger tout warning obtenu aux rendus précédents), nous effectuerons une évaluation manuelle de votre programme comme suit :

- lancement avec les 2 nouvelles syntaxes
- types de séquence de test de lecture-écriture de fichier **SANS relancer le programme** :
  - lecture avec succès, suivi d'une sauvegarde immédiate, suivi d'une lecture différente avec succès, suivi de la lecture du fichier sauvegardé, etc…
  - même chose avec plusieurs pas de simulation entre la lecture et la sauvegarde.
  - lecture avec échec, lecture avec succès, lecture avec échec, etc…
    Dans le cas d'une lecture avec échec les structures de données doivent être détruites pour ne pas perturber la lecture suivante. On demande de rafraichir l'affichage après cette destruction pour en avoir une confirmation visuelle (écran blanc).

- un Clic sur **Start** doit le faire passer à **Stop,** et vice-versa, et activer l'appel de la méthode idle() qui effectue une mise à jour de la simulation à chaque appel et donc cela donne l'impression d'une simulation en continu.

- un Clic sur **Step** doit faire une seule mise à jour de la simulation. Elle n'a d'effet que si la simulation est stoppée. Plusieurs fichiers de tests seront évalués avec cet outil.

- **Affichage** :
  - les proportions des formes ne doivent pas avoir de distorsion quand la fenêtre change de taille.
  - les couleurs des bases ne doivent pas changer en cours de simulation quand une base disparait.

2. Organisation du travail

Les ACTIONs suggéreés ici sont de simples propositions ; vous pouvez vous organiser autrement.

2.1 ACTION : interface GTKmm

2.1.1 ACTION : disposition de l'interface GTKmm

Cette action se concentre sur le sous-système de Contrôle ; l'idée est de mettre en place les éléments de l'interface (concept de Box de la série6). On travaille avec les module **projet** et **gui** ; le lien avec le Modèle n'est pas testé ici ; on veut seulement s'assurer que l'apparence de l'interface est correcte et que la reaction des boutons l'est aussi (la série7 détaille les fonctionalités Open/Save de GTKmm). Pour cette action on peut faire afficher un message du type "Ici appel de la méthode X" pour s'assurer que c'est fait au bon moment / en reaction au bon événement.

2.1.2 ACTION : test des boutons Start / Step / Stop

Cette action se concentre sur les boutons qui pilotent la simulation. Le lien avec le Modèle n'est pas testé ici ; on veut seulement s'assurer des points suivants:

- quand on appuie sur Start le bouton montre le texte "Stop" et quand on appuie sur Stop le texte revient à "Start"
- Appuyer sur le bouton Step a seulement un effet quand la simulation n'est pas lancée (on voit le bouton Start). Cet effet est de faire une seule mise à jour.
- Appuyer sur le bouton Start permet l'appel de la méthode **idle**() qui elle-même lance une seule mise à jour de la simulation. On doit pouvoir continuer d'utiliser les autres boutons de l'interface comme OPEN et SAVE même si la simulation est en cours avec les appels de la méthode **idle**().
- Appuyer sur le bouton Stop empêche l'appel de la méthode **idle**().

Pour cette action on peut faire afficher un message du type "Ici appel pour UNE mise à jour de la simulation" pour s'assurer que c'est fait au bon moment / en reaction au bon événement. On peut aussi incrémenter un compteur et le faire afficher pour mieux voir l'effet de Start / Stop et Step.

2.2 ACTION : Transformation du cadrage pour éviter les distorsions du dessin

C'est le cas où on change la taille de la fenêtre pendant l'exécution. Cela produit un appel automatique de **on_draw**. Il est donc important de récupérer dès le début de cette méthode les nouvelles largeur et hauteur puis de faire un ajustement du cadrage [Xmin,Xmax] et [Ymin,Ymax] pour éviter toute distorsion (slide 16 du cours sur le dessin avec GTKmm). La méthode à suivre a été traitée dans l'**exercice 3c de la série6**. Si vous n'avez pas encore mis au point l'affichage du Modèle, ré-utilisez le petit programme qui affiche des cercles, changez la taille de la fenêtre et on doit toujours voir des cercles.

2.3 ACTION : compléter geomod et lien avec graphic

Le role de **geomod** est d'offrir la possibilité de faire afficher des formes géométriques simples (cercle, polygone tel que triangle, carré, losange etc, ou autre: croix, étoile etc) à une position fournie en parametre avec éventuellement un parametre de taille. Le module **geomod** est important car il doit s'assurer que l'affichage reboucle de l'autre coté du domaine si la forme dépasse d'un coté du domaine. Pour chaque demande d'affichage le module **geomod devra faire plusieurs appels au module graphic** pour avoir le rebouclage du dessin.

- Ex: une demande de dessin de cercle de position (950, 0) et de rayon 100 faite au module **geomod** devra produire un appel de dessin de cercle au module **graphic** à cette position (950, 0) mais aussi un autre appel pour un dessin à la position (-1050, 0) pour produire le rebouclement du dessin de l'autre coté du domaine.

Faite un test avec un cercle et généralisez cette approche avec les autres demandes de dessin. Faites ces tests simples indépendamment du Modèle ; les appels à geomod peuvent être lancés depuis la méthode **on_draw** définie dans le module **gui**. Le module graphic effectue les appels à GTKmm pour le dessin lui-même.

2.4 ACTION : gestion des couleurs entre le Modèle, geomod et graphic

La section 6 de la donnée générale precise les couleurs prédéfinies à utiliser par **graphic**. La bordure carrée du domaine de la planète Donut peut être directement affichée depuis la méthode **on_draw** du module **gui** sans passer par le Modèle.

Pour les autres éléments du dessin, le module **gui** délègue au **Modèle** l'action de s'afficher avec une méthode de dessin de l'instance de la classe Simulation. Celle-ci délègue la demande de dessin aux autres classes (gisement, base, robots):

- **Gisement**: une méthode spécifique de **geomod** demande à graphic un cercle plein noir
- **Base**: l'indice de la base est transmis en paramètre à une méthode spécifique de **geomod** qui fait suivre à **graphic** ; **graphic** fait un modulo de cet indice avec la taille de sa table prédéfinie des couleurs primaires. Cela garantit qu'on obtient toujours l'ordre : rouge, vert, bleu, (gris-)jaune, magenta, cyan et ça reboucle s'il y a plus de 6 bases.
- **Robot**: l'indice de sa base est utilisé de la même manière pour obtenir la même couleur.

Le dessin des lignes représentant le réseau se fait avec une méthode spécifique de **geomod** qui fait suivre à **graphic** pour obtenir la couleur demandée (violet). Même chose pour les cercles de communication (gris).

Nous sommes consients que cette simplification n'est pas totale satisfaisantee du point de vue de l'architecture MVC mais elle est simple à mettre en oeuvre.

2.5 ACTION : mise au point de la simulation avec l'interface graphique et affichage dans le terminal

Nous recommandons d'avoir fait les autres ACTIONs de mise au point de l'interface graphique avant de tester votre simulation car l'affichage graphique de chaque état successif de la simulation est un puissant outil de mise au point. N'hésitez pas à compléter avec du code de scaffolding qui affiche la valeurs des variables importantes dans le terminal pour compléter vos outils de mise au point. D'ailleurs, en cas de bug, l'affichage dans le terminal devrait être redirigé vers un fichier de texte (cf cours redirection de la sortie) pour pouvoir l'ouvrir avec un éditeur de texte après le crash du programme et analyser l'évolution des valeurs de vos variables.

Nous recommandons d'effectuer les tests les plus simples possibles pour valider progressivement les différentes fonctionalités de la simulation: une base avec son robot de communication. Testez d'abord le comportement de la base avec différents niveau de sa ressource. Est-ce conforme à votre stratégie? Dès qu'un robot supplémentaire est créé le réseau de communication est-il correctement mis à jour / affiché ? la base se comporte-t-elle conformément à connexion de ses robots (connecté ou non-connecté) ?

2.6 ACTION : lecture/écriture de fichier

2.6.1 Test basique avec GTKmm

Pour ces tests, il faut que le module **gui** soit connecté au **Modèle** au moins pour la lecture et écriture de fichiers. Il n'est pas nécessaire que la simulation soit déjà finalisée à ce stade car on utilise seulement les données lues. Lancer le programme avec l'argument (nom de fichier) ; on doit voir l'état initial de la simulation affiché graphiquement avec le réseau de communication. Quitter puis relancer sans argument et utiliser le bouton GTKmm pour ouvrir le même fichier. On doit avoir le même affichage.

Chaque ouverture de fichier doit d'abord détruire la simulation courante avant de lire un fichier. Cela est vérifié comme suit:

- Ouvrez un fichier correct puis ouvrez un autre fichier correct dont l'affichage est différent puis ré-ouvrez le premier fichier correct sans relancer le programme. L'affichage doit être cohérent avec chaque fichier ouvert.

- Ouvrez un fichier correct puis sauvegardez-le avec Save et ré-ouvrez-le avec Open sans relancer le programme. L'affichage doit être le même.

- Test avec un fichier contenant une erreur ; le message doit apparaitre dans le terminal mais on ne doit pas quitter le programme. Celui-ci affiche un écran blanc (pas de dessin) et attend qu'on utilise l'interface graphique pour ouvrir un fichier.

2.6.2 Test avec simulation

Pour ces tests une simulation élémentaire suffit car un petit changement visible suffit. Ouvrez un fichier, lancez la simulation pour avoir un changement visible, puis sauvegardez l'état de la simulation dans un fichier. Ensuite ré-ouvrez le fichier dans l'état initial puis le fichier sauvegardé. L'affichage doit être cohérent avec chaque fichier ouvert.

Le fait d'avoir lu le fichier sauvegardé ne doit pas introduire de changement dans la stratégie de la simulation. Donc la simulation qui commence à partir du fichier sauvegardé devrait se comporter de la même manière que la simulation qui commence à partir de l'état initial.

3. Forme du rendu

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 rendu1 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**
- le fichier pdf du rapport

*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.