Projet Informatique – Sections Electricité et Microtechnique

Printemps 2023 : *Mission propre en Ordre* © R. Boulic & collaborators

Rendu2 (30 avril 23h59)

Objectif de ce document : *Ce document utilise l'approche introduite avec la série théorique sur les méthodes de développement de projet qu'il est important d'avoir faite avant d'aller plus loin.*

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 9b).

1. Buts du rendu2 : mise au point de l'interface graphique et lecture/écriture

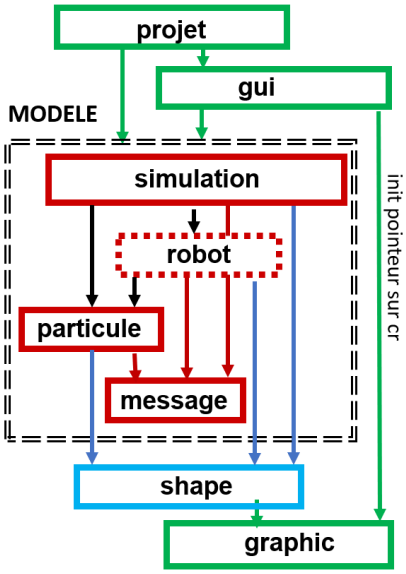
Ce rendu construit les structures de données et affiche l'état initial avec GTKmm (sections 5 et 6 de la donnée).

Un rapport devra décrire les choix de structures de donnée et la répartition des responsabilités des modules : qui fait quelle tâche ?

Lancement et comportement attendu:

Le programme sera lancé selon la syntaxe suivante : `./projet t01.txt`

Le programme construit l'interface graphique (section 5 de la donnée) et ouvre immédiatement le fichier fourni sur la ligne de commande en lecture.



Donnée Fig 9b

En cas de succès de la lecture, le programme affiche l'état initial de la simulation (dessin) et attend des commandes sur les widgets ou les touches du clavier.

En cas d'échec, c'est-à-dire dès la première erreur détectée à la lecture du fichier, le message d'erreur est affiché (c'est suffisant de l'afficher dans le terminal comme pour le rendu1). De plus les structures de données sont effacées, l'affichage du dessin est un Monde vide (espace blanc encadré par le carré indiquant sa frontière), puis **le programme attend** qu'on utilise l'interface graphique pour ouvrir un autre fichier. On ne doit PAS quitter le programme quand on détecte une erreur.

La sauvegarde de l'état courant de la simulation dans un fichier sera aussi testée.

Du point de vue de l'affichage des dessins de la simulation, on testera que le changement de la taille de la fenêtre graphique n'introduit pas de distorsion : les carrés restent des carrés et les cercles restent des cercles.

Le test de la simulation sera limité à la désintégration des particules au cours du temps en agissant sur les boutons **start** et **step** (les robots ne doivent pas bouger). La sauvegarde de l'état de la simulation après désintégration sera aussi testée.

2. Architecture du rendu2 (Fig donnée 9b)

Dès le rendu2 il est demandé de mettre en place *une hiérarchie de classes* pour gérer les différents types de robots. Nous vous demandons de fournir **un rapport de maximum 2 pages** (une feuille recto-verso) décrivant votre organisation du code du Modèle, en particulier :

- **La hiérarchie de classe des** robots : Quels attributs/méthodes sont gérés par la superclasse et par les classes dérivées. Quelles méthodes sont virtuelles afin de mettre en œuvre le polymorphisme ?

- **La structuration des données des autres entités du Modèle :**
  - Quels sont les attributs des classes (Simulation, Particule) ?
  - Où et comment sont mémorisés les différents ensembles que doit gérer la simulation ?

- **Brève description des types mis en œuvre dans shape**.

On ne demande pas de décrire vos algorithmes de simulation (section 2.5) car c'est pour le rapport final mais vous devez l'avoir en tête quand vous faites les choix de l'endroits où vous mémorisez vos données.

2.1 Module projet

Comme pour le rendu1, Le module **projet** contient la fonction **main()** en charge d'analyser si la ligne de commande est correcte. La nouveauté par rapport au rendu1 est la déclaration de l'interface graphique GTKmm depuis **main()**. La classe responsable de l'interface graphique est définie dans le module **gui**.

2.2 Module gui

Le module **gui** crée l'interface avec les éléments visibles ci-dessous et la surface dédiée au dessin.

2.2.1 Partie dédiée au dialogue : les widgets (Fig 7 donnée)

Les commandes suivantes devront être opérationnelles:

- « **exit** »: quitter le programme
- « **open** »: lire un fichier avec GTKmm pour initialiser une simulation avec detection d'erreur
- « **save** »: sauvegarder l'état courant de la simulation (éventuellement vide) dans un fichier
- Affichage des informations suivantes:
  - nombre de mises à jour
  - nombre de particules
  - nombre de Robots réparateurs en service
  - nombre de Robots réparateurs en réserve
  - nombre de robots neutraliseurs en service
  - nombre de robots neutraliseurs en panne
  - nombre de robots neutraliseurs détruits
  - nombre de robots neutraliseurs en réserve

L'image de droite montre l'affichage de la partie gauche du GUI à obtenir pour le fichier public correct t00.txt.
Les 2 premières informations devront être actualisées lorsque la simulation de la désintégration est lancée avec start ou step.

Le test des autres boutons sera limité au comportement suivant:

- « **start** »: le label du bouton devient "**stop**" et un **timer** est lancé qui produit l'affichage d'un compteur entier qui progresse d'une unité à chaque execution du signal handler du timer. Cela permet de simuler l'appel d'une mise à jour de la simulation. Si on re-clique sur le bouton (qui maintenant affiche "**stop**") alors le timer s'arrête et le label redevient "**start**".
- « **step** »: l'action de ce bouton est seulement prise en compte quand la simulation n'est pas en cours d'exécution (c'est à dire quand on voit le label "start" au-dessus du bouton "step"). Dans ce contexte, un clic sur ce bouton produit UNE SEULE mise à jour de la simulation. Cela est simulé en faisant afficher une seule incrémentation du compteur utilisé par le timer (cf bouton start/stop).

L'action des boutons **start** et **step** devra produire la mise à jour du compteur de mises à jour affiché à gauche et simuler (seulement) l'action de désintégration des particules selon la probabilité indiquée dans la section 2.1.4 de la donnée. S'il y a une désintégration le dessin et le compteur de particules doivent être mis à jour.

Les 2 touches de clavier (section 6.1 de la donnée) devront produire le même comportement que les boutons associés.

2.2.1.1 ACTION : test du module projet avec initialisation de l'interface graphique (sans dessin)

Dans cette étape on se contente d'établir le lien entre le module **projet** avec main() et le module **gui** qui initialise l'apparence de l'interface graphique (Donnée Fig 7) sans se connecter au Modèle ni à la partie qui s'occupe du dessin.

Quand la disposition des widgets est correcte, on peut connecter la partie Contrôle du projet à la partie Modèle pour vérifier que la lecture de fichier fonctionne correctement. D'abord avec erreur pour vérifier qu'on ne quitte pas le programme. ~~Ensuite sans erreur pour vérifier les valeurs affichées par les boutons next et previous.~~ Ensuite on peut mettre en oeuvre la sauvegarde de fichier.

A ce stade il faut tester des séquences de lecture/sauvegarde avec et sans erreur car c'est de cette manière que votre projet sera testé.

2.2.1.2 ACTION : test de la generation des probabilités (sans dessin)

Dans cette étape on veut s'assurer d'obtenir la même séquence de nombres aléatoires chaque fois qu'on a ouvert le même fichier (ou chaque fois qu'on ouvre un fichier avec le même nombre de particules). En effet la probabilité qu'on demande de produire est celle du taux de désintégration divisée par nbP, le nombre de particules courant.

Pour verifier cela, il faut ré-initialiser le default_random_engine avec un appel à la méthode **seed**() à chaque lecture de fichier. Nous recommandons que tous les projets utilises la même valeur entière transmise à seed afin de tous obtenir la même sequence de hasard. Pour les besoins de vos tests, vous pouvez changer cette valeur pour observer le changement produit.

Faites afficher dans le terminal la suite des 0 et des 1 des booléens obtenus en vous inspirant de l'exemple de code **`random_seq_reset_avec_seed1.cc`** fourni dans le folder rendu2.

Dans la première phase du test, ne créez pas de nouvelle particule si vous obtenez le booléen vrai car le but est de simplifier au maximum la comparaison.

Une fois cela validé, utilisez l'information de la probabilité de désintégration pour décomposer une particule en ses 4 particules si la condition de taille est respectée. Utilisez le nouveau nombre de particules pour mettre à jour la probabilité. L'exécution après relecture du même fichier doit donner la même suite de 0 et de 1

2.2.2 Conversion des coordonnées dans gui et dessin avec le module graphic

Le sous-système de visualisation apportera une aide précieuse pour la vérification du programme.
Les taches sont réparties comme suit:

- la méthode **on-draw**() de l'interface gérée par le module **gui** effectue la conversion de coordonnées au niveau de ce module **gui**. Ce module doit mémoriser les informations sur la fenêtre (width et height) et sur le cadrage du Modèle (valeurs Min et Max selon X et Y). On mettra en oeuvre l'approche proposée en cours avec les transformation **translate** et **scale** ainsi que l'initialisation du pointeur de context Cairo avec la fonction fournie par le module **graphic**. Enfin cette méthode on_draw doit mettre en oeuvre *l'absence de distorsion* quand on change la taille de la fenêtre (cf semaine6). Dans sa version complète la méthode **on_draw**() demandera au Modèle de se dessiner en appellant une seule méthode offerte par le module **simulation.**

- La méthode de dessin du module **simulation** délègue autant que possible la tâche du dessin aux autres modules du Modèle selon les indications de la *section 6 de la donnée*. Ces modules demandent à **shape**

de dessiner des carrés, cercles, lignes comme précisé ci-dessous. Comme indiqué dans la section 7 de la donnée, l'interface de shape contient aussi l'interface de graphic, ce qui permet de préciser des parametres de style de dessin comme plein/vide et la couleur désirée.

- Le module indépendant **shape** doit être complété pour proposer une ou plusieurs fonction(s)/méthode(s) pour dessiner des cercles, carré, ligne.

- **shape** doit transférer les demandes de dessin de bas niveau au module **graphic** qui est le seul à avoir le droit de faire des appels directs à GTKmm

- le module **graphic** effecture les appels à GTKmm pour définir la couleur et dessiner des lignes, cercles et carrés.

Notons que, s'il n'y a pas de simulation (cas de lecture de fichier avec erreur ou lancement du programme sans fournir de fichier de configuration) alors la méthode **on-draw**() du module **gui** se charge seulement de ré-initialiser le fond du dessin en blanc puis fait afficher un Monde vide montrant seulement un carré pour matérialiser sa frontière par l'intermédiaire de fonctions de **graphic**

2.2.2.1 ACTION : test du module projet avec initialisation de l'interface graphique (avec dessin)

Dans une première étape, on laisse le Modèle de coté et on se contente de vérifier que la méthode **on_draw**() est capable de mettre en place la conversion de coordonnée et l'absence de distorsion en appellant une fonction du module **graphic** qui dessine un carré ayant la taille du Monde. Sa couleur doit être différente de celle du fond pour les distinguer l'un de l'autre. Ce carré doit rester carré quand on change la taille de la fenètre.

Un exemple GTKmm présenté en cours en semaine7 permet de prendre en compte le fait que la surface de dessin est décalée par rapport à l'origine de la fenêtre.

Une fois que ces aspects sont maîtrisés au niveau de **gui**, **graphic** et **shape**, vous pouvez commencer à écrire et tester la méthode de dessin du Modèle. Le niveau **simulation** délègue aux modules inférieurs la tâche de se dessiner et eux-mêmes délèguent cette tâche au module **shape** en lui passant les paramètres de style et d'indice de couleur pour représenter les différentes entités.

3. Evaluation du rendu2 :

Le barème détaillé est visible à la suite de la donnée générale. Seulement deux méthodes/fonctions au-dessus de 40 lignes (max 80 lignes) sont autorisées pour l'ensemble du code du rendu2.

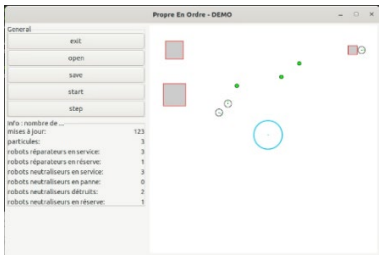
Nous effectuerons une évaluation manuelle de votre programme SANS LE RELANCER :

- lecture avec succès, suivi d'une sauvegarde, suivi d'une lecture différente avec succès, suivi de la lecture du fichier sauvegardé, etc…
- lecture avec succès, suivi d'une sauvegarde, suivi de l'utilisation des autres boutons, suivi de la lecture du fichier sauvegardé, etc…
- 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. L'affichage doit alors montrer un Monde vide avec seulement le carré indiquant le domaine.

Le fonctionnement attendu des boutons est décrit en section 2.2.1.

- **Affichage** :
Les proportions, formes et couleurs des éléments de la simulation doivent être respectées. L'absence de distorsion est à mettre en oeuvre dès ce rendu2 ; voici un exemple avec le fichier t00.txt:



Fenêtre avec la taille initiale

Agrandissement horizontal

Agrandissement vertical

4. Forme du rendu2

Convention de style : il est demandé de respecter les conventions de programmation du cours.
Documentation : l'entête de vos fichiers source doit indiquer le nom du fichier et les noms des membres du groupe avec, **pour les fichiers .cc**, une estimation du pourcentage de contribution de chaque membre du groupe au code de ce fichier.

Rendu : pour chaque rendu **UN SEUL membre d'un groupe** (noté **SCIPER1** ci-dessous) doit téléverser un fichier **zip**[1] 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**) :

- Rapport (fichier pdf)
- Fichier texte édité **mysciper.txt**
- Votre fichier **Makefile** produisant un executable **projet**
- Tout le code source (.cc et .h) nécessaire pour produire l'exécutable.

*On doit obtenir l'exécutable **projet** en lançant la commande* **make** *après décompression du fichier **zip**.*

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 à distance.

Backup : Il y a un backup automatique sur votre compte myNAS.

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 et pas plus. Cependant il n'y a pas d'éditeur de code en mode partagé.
- **Solution plus ambitieuse mais qui demande un apprentissage non négligeable** : créer un compte sur gitlab.epfl.ch . A partir de ce compte vous créez un projet ; .Attention : il FAUT **restreindre** l'accès du code aux seuls 2 membres du groupes sinon il est public par défaut. Ensuite il faut utiliser l'outil **git** avec **gitlab** comme expliqué dans cette video YouTube

[1] Nous exigeons le format zip pour le fichier archive