Projet Informatique – Sections Electricité et Microtechnique

Printemps 2021 : *Tchanz* © R. Boulic & collaborators

Rendu2 (1 mai 23h59)

Objectif de ce document : *Ce document utilise l'approche introduite avec la série théorique du Topic1 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 9a).

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).

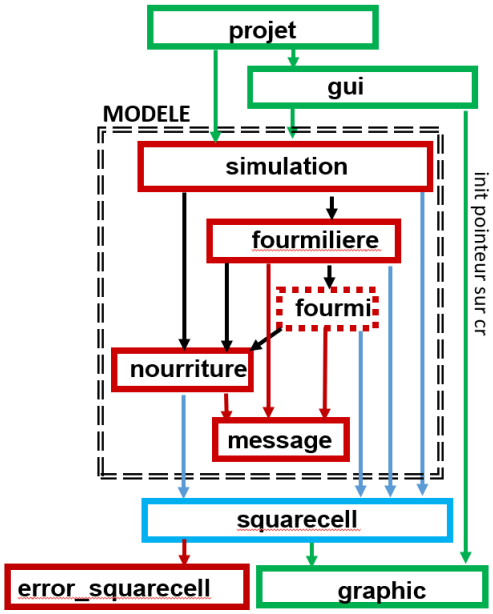
Un rapport devra décrire les choix de structures de donnée en *anticipant* comment elles seront utilisées pour le rendu final.

Lancement et comportement attendu:

Le programme sera lancé selon la syntaxe suivante : `./projet test1.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.

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.



Donnée Fig 9b

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 (section 2.2.2), 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.

2. Architecture du rendu2 (Fig donnée 9b)

Dès le rendu2 il est demandé de mettre en place *une hiérarchie de classes* (deux niveaux suffisent) pour gérer les différents types de fourmis. 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** fourmis : 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, Fourmilière, Nourriture) ?
  - Où et comment sont mémorisés les différents ensembles que doit gérer la simulation ?
- **Brève description des types mis en œuvre dans squarecell**.

On ne demande pas de décrire votre algorithme de simulation 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**.

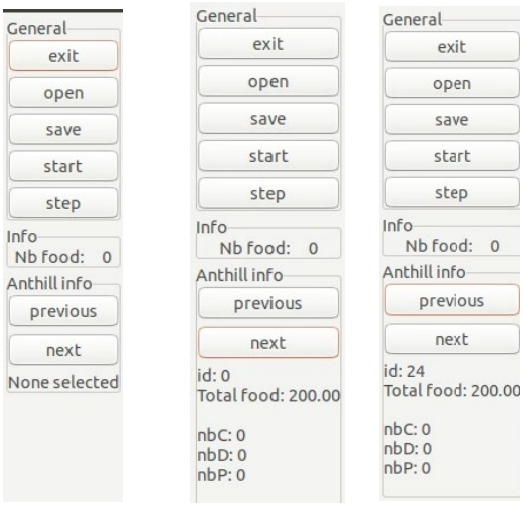
2.2 Module gui

Le module **gui** crée l'interface avec les éléments visibles dans la Figure 7 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 du nombre d'éléments de nourriture disponibles de la simulation
- Affichage des données de la **fourmilière courante** (initialement associée à aucune fourmilière)
- « **previous** » et « **next** »: ces deux boutons modifient la valeur de la **fourmilière courante** entre les valeurs suivantes: "*pas de fourmilière courante",* indice compris entre 0 et nombre de fourmilière-1. **next** change la valeur dans l'ordre ascendant et **previous** dans l'ordre descendant.
    - Exemple lié au fichier de test public **no_error_neighbours_anthill.txt** avec 25 fourmilières:
        - a) **état initial sans aucune fourmilière sélectionnée** comme fourmilière courante
        - b) si, dans l'état initial, on clique sur **next**: on sélectionne la fourmilière d'indice 0 qui devient la fourmilière courante. Si on re-clique on obtient la fourmilière d'indice 1, etc… jusqu'à la fourmilière d'indice 24, et enfin on repasse dans l'état sans fourmilière sélectionnée
        - c) si, dans l'état initial (aucune fourmilière sélectionnée), on clique plutot sur **previous** on sélectionne la fourmilière de plus haut indice (ici 24). Si on re-clique on obtient celle d'indice 23, etc.. jusqu'à 0, et enfin on repasse dans l'état sans fourmilière sélectionnée.



a b c

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 enc ours 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).

Les 5 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.

2.2.2 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: cours et série6). 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 **squarecell** de dessiner des carrés de différents styles comme précisé ci-dessous.

- Le module indépendant **squarecell** doit être complété pour proposer une ou plusieurs fonction(s)/méthode(s) pour dessiner ses entités de type "carré". On doit proposer au moins les styles suivants pour dessiner un carré (Figures 2 et 3 de la donnée):
  - **Vide** : seulement le contour du carré avec un trait épais passant par le milieu des cellules constituant la bordure de ce carré.
  - **Losange** : forme de losange inscrite dans le carré de chaque cellule du carré
  - **Uniforme** : carré plein
  - **Diagonale** : quadrillage avec motif en X
  - **Grille** : quadrillage avec motif en +

  squarecell 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** effectue les appels à GTKmm pour définir la couleur et dessiner des lignes, et des carrés avec des motifs demandés. Ce module dessinera avec les couleurs primaires définies dans une table dans l'ordre: rouge, vert, bleu, jaune[1], magenta, cyan.

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 noir puis fait afficher un Monde vide par l'intermédiaire de fonctions de **graphic**:

- les bordures du Monde sont remplies de cellules blanches
- les cellules sont délimitées avec des lignes horizontales et verticales en gris clair

[1] La couleur jaune pure étant peu visible, même sur fond noir, on peut lui ajouter du bleu.

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, c'est à dire entre **0** et **g_max** selon X et Y. 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. On peut ajouter un petit carré de une unité de coté comme dans la Figure 2 de la donnée pour les coordonnées (0,0) pour vérifier que ce carré est bien placé dans le dessin.

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.

Complétez d'abord la méthode **on_draw**() pour qu'elle dessine un Monde vide avec une bordure blanche (épaisseur une cellule). Garder l'affichage des lignes horizontales et verticales à la fin pour qu'elle apparaissent par dessus tout le reste. Ces dessins élémentaires de bas-niveau devraient être délégués à **graphic** (pas besoin de passer par **squarecell**).

Avant de passer au dessin complet du Modèle il est recommandé de tester le dessin des entités carrée de **squarecell** avec les 5 styles décrits plus haut.

Une fois que ces aspects sont maîtrisés au niveau de **gui**, **graphic** et **squarecell**, 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 **squarecell** en lui passant les paramètres de style et d'indice de couleur pour représenter les différentes entités.

3. Evaluation 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 (section 2.2.2) et on doit passer dans le mode où aucune fourmilière courante n'est sélectionnée (2.2.1).

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.

3.1 Barème (20% de la note finale du cours)

L'exécution correcte du rendu2 va compter pour presque la moitié des points du rendu2.

Le reste se répartit entre le respect de l'architecture du projet, l'encapsulation/externalisation, le style du code (seulement 2 méthodes/fonctions au-dessus de 40 lignes sont autorisées pour l'ensemble du code du rendu2) et le rapport.

4. Forme du rendu2

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**[2] 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**
- 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.
- Une approche qui demande un apprentissage supplémentaire serait d'utiliser **github** : cf ce tutorial sur moodle. Attention : il FAUT restreindre l'accès du code aux seuls 2 membres du groupes.

Rappel sur l'outil GDB de recherche de bug :

- Guide utilisateur de GDB avec son code de test ; GDB tutorial in english

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