V 1.0

Projet Informatique – Sections Electricité et Microtechnique

Printemps 2024 : *Microrécif* © R. Boulic & collaborators

Rendu2 (28 avril 23h59)

Table des matières :

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 générale 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 générale).

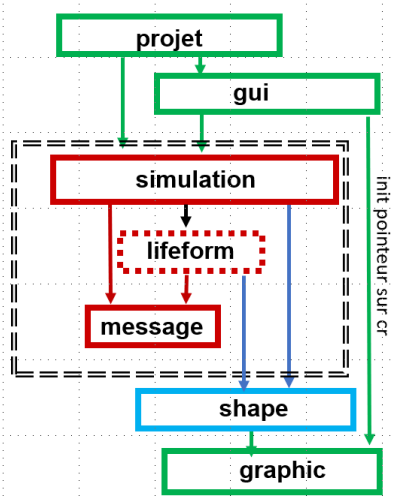
Un rapport devra décrire les choix de structures de donnée et la répartition des responsabilités des modules et des classes : qui fait quelle tâche ? Où sont les ensembles de données ?

Lancement et comportement attendu:

Le programme sera lancé selon la syntaxe suivante :

```
./projet t01.txt
```

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



Donnée générale 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 partie **`Mise à jour des algues`** du **pseudocode1** de la section 2 de la donnée générale. Cela inclut d'abord l'incrémentation de leur age et leur éventuelle disparition si une algue atteint l'age **max_life_alg**. Ensuite, si la checkbox **naissance d'algue** est activée, il y a une création conditionnelle des algues au cours du temps en agissant sur les boutons **start** et/ou **step**.

La sauvegarde de l'état de la simulation après creation de plusieurs algues sera aussi testée.

2. Architecture du rendu2 (Fig 9b donnée générale)

Dès le rendu2 il est demandé de mettre en place *une hiérarchie* de classes pour gérer les différents types d'entités de Lifeform. 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** entités de Lifeform : Quels attributs/méthodes sont gérés par la superclasse et par les classes dérivées. Optionnel : si vous utilisez le polymorphisme, quelles méthodes sont virtuelles afin de le mettre en œuvre ?

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

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

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 Sous-système Modèle

Le module **simulation** reste le point d'entrée obligatoire pour des appels depuis le module **projet** ou depuis le module **gui** responsable de de l'interface graphique. Pour le rendu2, il faut les complements suivants:

- Adapter la **lecture** de fichier pour:
  - ré-initialiser la simulation avant la commencer la lecture de fichier afin de pouvoir lire un nouveau fichier sans être perturbé par l'état des attributs/variables d'état de la lecture précédente.
  - renvoyer un booléen traduisant le succès (true) ou l'échec (false) de la lecture pour que le programme continue de fonctionner après la lecture avec succès et en cas de détection d'erreur.
  - Supprimer les structures de donnée lues en cas d'échec de la lecture.
- Créer une fonction/ méthode de:
  - **sauvegarde** de la simulation qui va créer un fichier formaté selon les indications de la section 4 de la donnée générale. On lui donnera un nom de fichier en parametre.
  - **execution** d'une seule mise à jour de la simulation des algues (donnée générale pseudocode1 et section 3.1 pour la mise en oeuvre des probabilités en C++)
  - **dessin** de l'état courant de la simulation (section 6 donnée générale)

En vertu du principe d'abstraction le module simulation délègue aux modules de **Lifeform** la partie qui concerne chaque entité pour réaliser les actions qui les concernent pour la sauvegarde, le dessin et la mise à jour d'algue.

2.2.1 ACTION : test d'une suite de plusieurs lectures/sauvegardes (sans le module gui)

Dans cette étape on se concentre sur la partie lecture/écriture de fichier format sans aucun code lié à l'interface graphique (comme pour le rendu1).

L'idée est d'ajouter une nouvelle fonction/méthode de simulation, appelons-la **test**, qui est appelée par projet avec le nom de fichier transmis sur la ligne de commande. Ensuite, dans cette fonction, on demande l'exécution d'une suite d'appels dont le premier la **lecture** du fichier transmis en parametre. On peut enchainer directement avec un appel de **sauvegarde** ou sur une autre **lecture** de fichier (éventuellement avec un nom prédéfini par vous), etc... Si le booléen renvoyé par la **lecture** est true on doit s'attendre à obtenir le même fichier en faisant une **sauvegarde** de la simulation (on n'exige pas d'avoir la même présentation en terme d'espaces et de passages à la ligne). Si le booléen renvoyé par la lecture est false, il faut supprimer les structures de données lues et la sauvegarde doit créer un fichier de simulation "vide" c'est à dire avec la valeur zero pour les 3 nombres d'entités.

2.3 Module gui

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

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

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 dont le nom est fourni à GTKmm.

- Affichage des informations suivantes:
  - nombre de mises à jour : *: ré-initialisé à 0 à chaque lecture d'un fichier*
  - nombre d'entités de type algue
  - nombre d'entités de type corail
  - nombre d'entités de type scavenger

  L'image ci-contre montre l'affichage de la partie gauche du GUI

  Les 2 premières informations devront être actualisées lorsque la simulation de la creation d'algue est lancée avec **start, step** et la checkbox de naissance d'algue. La simulation peut aussi conduire à la disparition d'algues dès qu'elles atteignent leur age maximum.

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).
- « **Naissance d'algue** »: l'action de cette checkbox est d'autoriser la creation d'algue quand elle est validée. Cette checkbox est désactivée par défaut au lancement du programme ; elle conserve son état courant lors de la lecture d'un nouveau fichier.

L'action des boutons **start** et **step** devra produire la mise à jour du compteur de mises à jour affiché à gauche et – si la checkbox est validée - simuler (seulement) l'action de creation d'algue selon la probabilité indiquée dans les sections 2.1.1 et 3.1 de la donnée générale. S'il y a une creation d'algue le dessin et le compteur d'algues doivent être mis à jour.

Les 2 touches de clavier '**s**' et '**1**' (section 6.1 de la donnée générale) devront produire le même comportement que les boutons associés (respectivement, **start** et **step**).

2.3.1.1 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 (ci-dessus) 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 on peut mettre en oeuvre la sauvegarde de fichier.

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

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

Cette étape est conditionnée à l'état activé de la checkbox "**naissance d'algue**".

Dans cette action de mise au point, on veut s'assurer d'obtenir la même séquence de nombres aléatoires chaque fois qu'on ouvre un fichier, c'est à dire au lancement du programme mais aussi à chaque nouvelle lecture de fichier avec l'interface GTKmm.

Pour vérifier cela, il faut ré-initialiser le **default_random_engine** avec un appel à la méthode **seed**() à chaque lecture de fichier (section 3.1.1 donnée générale). Nous recommandons que tous les projets utilises la même valeur entière de **1** 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.

L'exécution après lecture d'un nouveau fichier doit donner la même suite de 0 et de 1.

2.3.2 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 cours 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 générale*. 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 paramètres de style de dessin comme 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.

En cas de lecture de fichier avec erreur (ou de lancement du programme sans fournir de fichier de configuration pour le rendu3), la simulation est ré-initialisée à « vide ». La méthode **on-draw**() du module **gui** se charge seulement de peindre le fond du dessin en blanc puis fait afficher la bordure du Monde montrant un carré par l'intermédiaire de fonctions de **graphic.** Le méthode de dessin de simulation ne dessinera rien.

2.3.2.1 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 fenêtre.

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 :

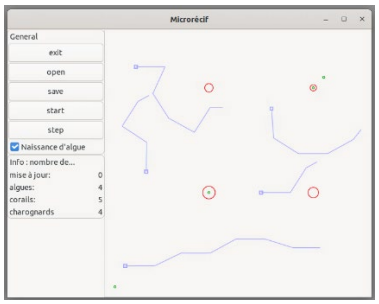
Le barème détaillé est visible à la suite de la donnée générale. Nous effectuerons une évaluation manuelle de votre programme SANS LE RELANCER :

- 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…

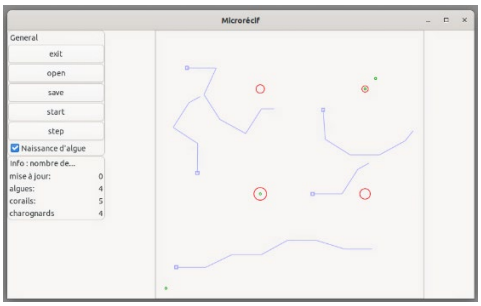
Dans le cas d'une lecture avec échec les structures de données doivent être détruites pour ne pas perturber la lecture suivante. L'affichage doit alors montrer un Monde vide avec seulement le carré indiquant le domaine. Le fonctionnement attendu des boutons est décrit en section 2.3.1.

- **Affichage** :

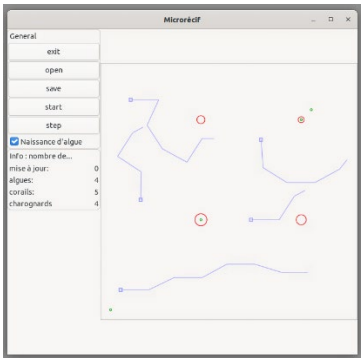
Les proportions, formes et couleurs des éléments de la simulation doivent être respectées. L'absence de distorsion est à mettre en oeuvre dès ce rendu2 ; voici un exemple avec le fichier **fig1.txt** pour lequel l'espace Monde est centré dans l'espace libre pour le dessin. On peut choisir un autre choix de cadrage du moment qu'on n'introduit pas de distorsion:



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.

- *Seulement deux* méthodes/fonctions au-dessus de 40 lignes (max 80 lignes) sont autorisées pour l'ensemble du code 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**[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** après décompression du fichier **zip** en lançant la commande* **make** *dans un terminal de la VM (sans utiliser VSCode).*

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 lorsqu'elle est lancée dans un *terminal de la VM* (sans utiliser VSCode) et que celui-ci fonctionne correctement.

Exécution sur la VM: votre projet sera évalué sur la VM à distance (compilation avec l'option -std=c++17).

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

Debugging : Visual Studio Code offre un outil intéressant pour la recherche de bug (VSCode tuto).

5. Travail en groupe

Les éléments déjà fournis dans la section 5 du rendu1 concernant la **responsabilité de travailler en groupe** et la **gestion du code** restent valables. Nous insistons sur l'implication des 2 membres du groupe dans le travail fourni. Comme indiqué dans la section précédente, à partir du rendu2, nous demandons de préciser (en gros) le pourcentage du code écrit par chaque membre du groupe dans l'en-tête au début des **fichiers .cc** . Si vous travaillez ensemble il suffit d'indiquer 50% pour les 2 personnes du groupe.

Gestion de désaccord :
Comme indiqué en section 1 du guide de conception d'algorithmes du premier semestre, la « mythe de la solution unique » rappelle qu'il existe un *espace de solutions* plutôt qu'une solution unique. Il est donc possible que les deux membres d'un groupe soient en désaccord sur l'approche retenue.

Dans ce cas il convient d'apporter les arguments (*qualitatif* => facilité de mise au point, ou *quantitatif* => performance de la simulation) en faveur de son approche et surtout de ***savoir trouver un compromis*** pour fournir un rendu. Cette partie du travail à deux est une compétence transversale qu'il est fondamental de maitriser pour réussir l'EPFL (au-delà des connaissances sur la matière elle-même).

Rappel et complément concernant les groupes: nous n'acceptons pas de groupe « solo ». Si deux groupes sont d'accord pour échanger leur composition, ils doivent informer au plus tôt l'enseignant pour qu'il valide ce changement.

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