Projet Informatique – Sections Electricité et Microtechnique

Printemps 2020 :Archipelago © R. Boulic & collaborators

Rendu2 (dimanche 3 mai 23h59)

Objectif de ce document : *comme pour le document du rendu1, ce document identifie des ACTIONS à considérer pour réaliser le rendu de manière rigoureuse. Le présent document suit l'architecture minimale du projet (donnée Fig10 et* ***Fig 11 b1****)*[1]*. A nouveau, on s'appuiera sur la série méthodes de développement de projet, qui rappelle en particulier le concept de stub.*

1. Buts du rendu2 (Donnée section 7 rendu2)

Ce rendu et le suivant utilisent toujours GTKmm. Si un nom de fichier est indiqué sur la ligne de commande il doit être ouvert pour initialiser l'interface graphique et le dessin, incluant :

- Affichage de la valeur initiale des 3 critères **ENJ**, **CI** et **MTA**.
- l'algorithme de Dijkstra est inclus dans ce rendu pour le calcul du critère **MTA**

Ce rendu sera testé en effectuant plusieurs lecture/écriture/relecture avec le GUI pour vérifier que l'affichage est bien correct. Avant de commencer la lecture d'un fichier, il faut ré-initialiser les structures de données et libérer la mémoire.

1.1 Syntaxe du lancement de votre exécutable et gestion d'erreur à la lecture :

A partir du rendu2, ces deux syntaxes doivent être opérationnelles :

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

Dans les deux cas on lance le programme avec l'interface graphique GTKmm.

La première syntaxe ouvre immédiatement le fichier fourni sur la ligne de commande en lecture et, **en cas de succès,** affiche l'état initial de la ville.

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), les structures de données sont effacées, l'affichage est un écran blanc, puis le programme attend qu'on utilise l'interface graphique pour ouvrir un autre fichier.

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

2. Architecture du rendu2 (Donnée fig 11 b1) :

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. De même, comme pour le rendu1, Il n'y a pas de structure de donnée de ville dans le module projet car c'est une responsabilité du Modèle.

---

[1] *L'architecture alternative de la Fig 11 b2 est aussi autorisée mais il faut adapter soi-même certaines des ACTIONS pour cette architecture.*

La nouveauté est le lancement 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 visible à la Figure 7 de la donnée.

Les commandes « exit », « new », « open », « save » et l'affichage de la valeur initiale des 3 critères ENJ, CI et MTA devront être opérationnels pour le rendu2 (section 5 de la donnée). Le test des autres boutons se limitera à l'affichage d'*un message* confirmant que le bouton vient d'être utilisé.

Ce module **gui** exploitera l'interface du module **ville** pour déléguer les commandes de lecture et sauvegarde de fichier. Comme pour le rendu précédent, le module **ville** délègue aux autres modules du Modèle les tâches de gestion de leur structure de données (lecture, sauvegarde). Le module **gui** peut aussi exploiter le module **tools** si nécessaire.

Au choix, c'est le module **gui** ou le module **graphic** qui doit effectuer la transformation de coordonnées entre le modèle et la fenêtre GTKmm pour le dessin des éléments représentant l'état de la **ville**. La transformation de coordonnée ne doit pas se trouver dans le **Modèle** (dont le module **tools**) car ce sous-système ignore les informations de la fenêtre graphique et le cadrage choisi en X et en Y.

2.2.1 ACTION : test du module projet avec initialisation de l'interface graphique

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). Chaque bouton confirme son activité en affichant un message dans le terminal dans sa méthode de signal_handler. Les boutons toggle ajoutent l'information de leur état (appuyé, relaché).

2.3 Modules graphic et tools :

Selon l'architecture de la Fig 11 b1 le module **gui** délègue des appels de dessin de bas-niveau au module **graphic** qui lui aussi dépend de GTKmm.

2.3.1 ACTION : fonctions d'affichage du module graphic et tests depuis le module tools

1) Module graphic : s'appuyer sur l'exemple fourni en cours pour initialiser un pointeur vers un context Cairo dans **graphic** et pour s'en servir pour dessiner :
- un segment de droite à partir des 4 paramètres suivants : coordonnées x et y des 2 extrémités.
- un cercle à fond blanc, à partir des 3 paramètres suivants : coordonnées x et y du centre et valeur du rayon

Ce module peut gérer la transformation de coordonnées du Modèle vers l'espace de la fenêtre graphique en mémorisant les paramètres de la fenêtre graphique (largeur, hauteur) et le cadrage selon X et Y.

Ce module offre aussi une fonction pour changer la couleur en fonction de symboles définis avec un enum.
D'abord tester ces fonctions indépendamment du reste du projet.

2) Le module **tools** doit être utilisé par les niveaux supérieurs du Modèle (nœud, ville) pour faire les demandes de dessin puisqu'il est responsable des structures de données de cercle, segment, point2D, etc...

Il joue le rôle de module intermédiaire ; d'un coté il offre une ou plusieurs fonctions de dessin qui reçoit en paramètre un ou des éléments dont il est responsable (cercle, segment, point etc..) et de l'autre il appelle les fonctions mises à disposition par l'interface du module **graphic**. Il doit aussi offrir une fonction de changement de couleur à l'aide des symboles fournis par l'interface de **graphic**.

Dans un second temps, tester les fonctions de **tools** qui utilisent l'interface de **graphic**, indépendamment du reste du projet.

2.4 Module noeud

Le module **nœud** (ou la hiérarchie de classe s'il y en a une) est celui qui a accès aux informations d'uid, de centre et de population. Il est responsable de la sauvegarde de ces informations dans un fichier[2].

Le module **noeud** est aussi responsable de se dessiner car il peut calculer le rayon du quartier et déterminer sa représentation visuelle en fonction de son type et de son rayon.

2.4.1 ACTION : test de nœud + tools + graphic

Tester indépendamment les méthodes de dessin des trois types de nœud. Chaque méthode de nœud effectue les appels nécessaires aux fonctions de **tools** pour obtenir le dessin désiré.

Ensuite tester le dessin de deux nœuds reliés par un lien: d'abord afficher le lien qui relie les centres des 2 nœuds puis les représentations visuelles des nœuds. Si vous avez bien programmé le dessin de cercle plein à fond blanc dans **graphic** cela devrait masquer la portion du lien sous les cercles, comme dans la démo.

2.5 Module ville

Ce module est l'unique point de contact du Modèle vis-à-vis du module **gui**. Il fournit au moins des fonctions/méthodes de lecture, sauvegarde, dessin et de calcul des valeurs des critères dans son interface.

2.5.1 ACTION : tests de ville + nœud + tools + graphic

Concernant l'affichage de la Ville, le module **ville** doit d'abord faire afficher les liens (via **tools**) car les liens doivent être masqués par la représentation des nœuds. Après l'affichage des liens, la demande d'affichage des nœuds est déléguée au module **nœud**.

2.5.2 ACTION : intégration avec tests de open-save-new en tirant parti du dessin

L'action 2.2.1 a validé l'initialisation de l'interface. Commencer par connecter le bouton « *open* » à votre approche pour lire un fichier. Le moyen le plus intuitif et efficace pour valider cette action est de dessiner l'état de la Ville après l'opération de lecture du fichier.

Tester ensuite le bouton « *new* » qui doit effacer toutes les structures de données et les remettre dans leur état initial d'avant la lecture. Ici aussi on aura confirmation que ça s'est bien passé en faisant dessiner l'état résultant de la Ville. Ce même comportement doit être obtenu quand on détecte une erreur à la lecture d'un fichier.

Le test de « *save* » intervient après une lecture de fichier. Il suffit ensuite de comparer le texte du fichier écrit avec le texte original. La mise en page peut être différente.

Ce qui est important est que la lecture du fichier sauvegardé doit donner le même résultat que la lecture au lancement du programme. Pour cela, il suffit d'effectuer la suite : lecture, sauvegarde, bouton new, puis lecture du fichier sauvegardé. On a confirmation du bon fonctionnement grâce à l'affichage du dessin.

2.5.3 ACTION : intégration avec tests du calcul des critères en tirant parti du dessin

Cette action bénéficie aussi du dessin comme moyen de vérification visuelle de la Ville pour laquelle on veut calculer les 3 critères. Comme dans toute approche de tests, on commencera par les fichiers de test les plus simples possibles pour lesquels on peut calculer séparément les valeurs attendues. La valeur choisie pour **infinite_time** permet de faire des calculs approchés qui sont aussi utiles pour cette étape.

Le gros morceau est le calcul du critère MTA qui utilise l'algorithme de Dijkstra. Commencer par les cas particuliers (Ville vide, des nœuds sans liens de voisinage, nœud Production qui ne permet pas le passage, etc). La donnée fournit un pseudocode qui est commenté en cours. Vous pouvez faire des choix différents concernant l'implémentation de cet algorithme.

---

[2] Par contre la sauvegarde de l'information des liens est une responsabilité du niveau supérieur car il ne faut pas sauvegarder deux fois le même lien.

3. Evaluation du rendu2 :

Nous effectuerons une évaluation manuelle de votre programme comme suit :

- Lancement avec les 2 syntaxes possibles.

- types de séquence de test SANS relancer le programme :
  - 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 du bouton **new**, 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 écran blanc et les valeurs 0 pour les critères.

- un Clic sur les autres boutons doit faire afficher un message dans le terminal indiquant quel bouton est cliqué avec le signal handler des boutons ; il n'y a pas d'autre action à effectuer pour le rendu2.

- **Affichage** :
Les proportions, formes et couleurs des éléments de la ville doivent être respectées.
Le changement de taille de la fenêtre graphique sera testé dans le rendu3.

4. 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`
Exemple pour le groupe Wagnières/Zimmer: `301438_301900.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[3].

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)

*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 du cours (c'est la même qu'au premier semestre).

[3] L'unique groupe de 3 personnes rajoute le 3ieme SCIPER sur une troisième ligne dans mysciper.txt