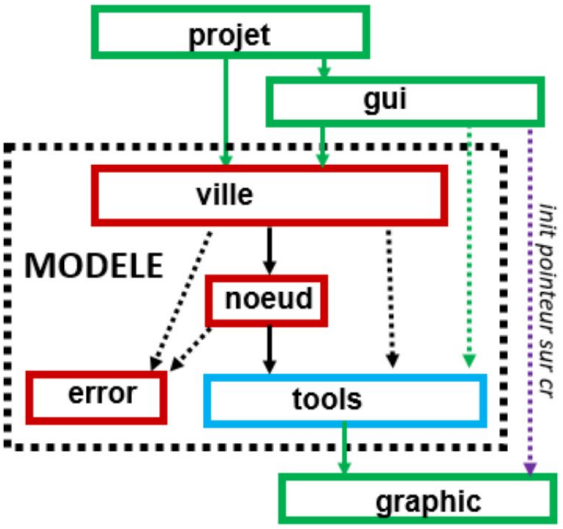
Archipelago – Rapport

Yann Uddhava-Monney (299290), Matthias Kockisch (303000) *24.05.2020*

Architecture



Le module projet contient la fonction main, qui appelle le module ville en transmettant l'argument si un tel est fourni, et ensuite lance gui après terminaison de la lecture du fichier dans ville, si elle a lieu. Projet est, comme requis, complètement indépendant de toute structure de données des autres modules.

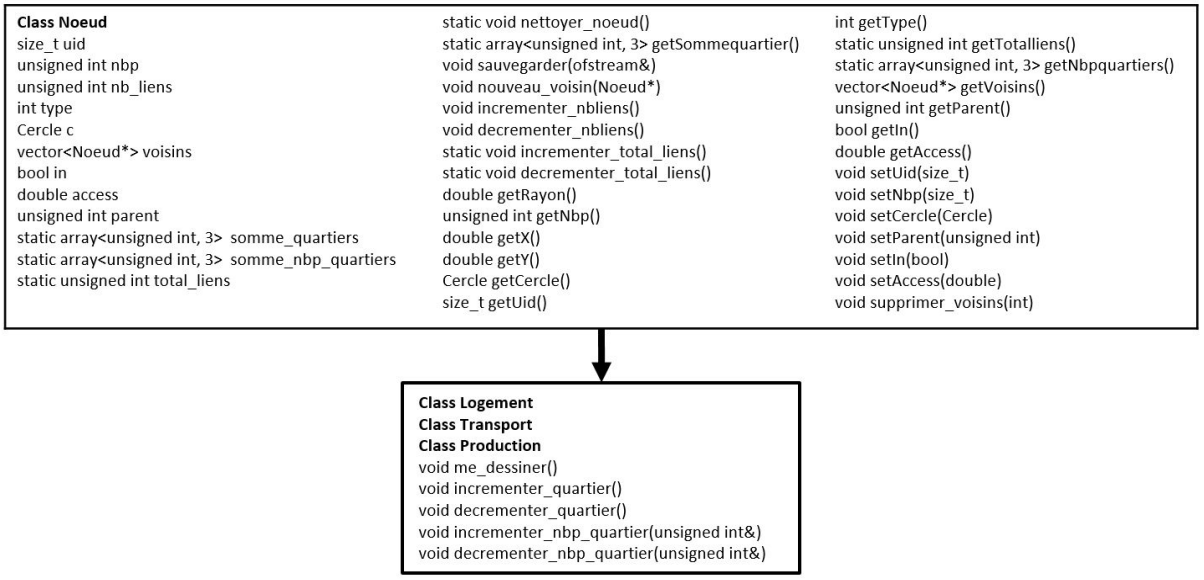
Le module ville contient les fonctions concernant l'ensemble des nœuds ou liens : la lecture et sauvegarde de fichiers, les calculs pour ENJ, CI et MTA, l'algorithme de Dijkstra, les contrôles du dessin de la ville ainsi que les réactions à une modification de la ville via l'interface graphique. Ainsi, toutes les fonctions concernant des modifications ou accès à l'ensemble de la ville se font dans ce module, sans inclure les fonctions spécifiques aux noeuds.

Le module nœud contient les fonctions qui décodent les strings envoyés par le module ville et les batteries des tests effectués sur ces données pour trouver d'éventuelles erreurs. Y sont aussi les structures de données des différents types de noeuds et leurs fonctions de dessin. Ces éléments sont fréquemment réutilisés dans le module ville.

Le module tools est limité aux calculs mathématiques effectués pendant les tests et aux fonctions intermédiaires entre le modèle et le dessin. Il inclut donc seulement le module graphic, les interfaces des constantes et du contrôle des couleurs ainsi que la librairie cmath. C'est ainsi une librairie de calculs accessible par les modules du modèle et de regroupements de fonctions de graphic appelés par gui.

Le module gui contient l'interface graphique, les dialogues open et save ainsi que les signal-handlers des boutons, qui appellent des fonctions du module ville. Ce module réagit aux actions effectuées par l'utilisateur, et transmet les actions aux autres modules.

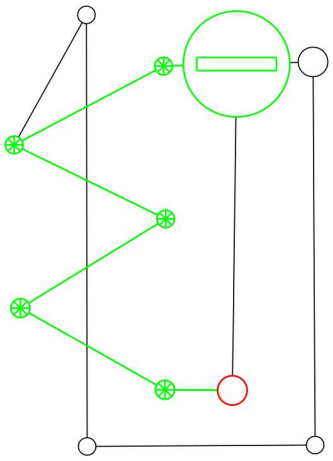
Le module graphic reçoit le pointeur Cairo::Context de la DrawingArea de gui et est le seul à effectuer des modifications sur celui-ci. Ce module est utilisé par les autres via tools et est ainsi du plus bas niveau. Il contient aussi une ébauche d'un dialogue d'erreur, non utilisée et donc mis en commentaires à cause de l'impossibilité d'appliquer set_transient avec notre architecture. Ce dialogue nécessiterait la fonction intermédiaire d'affichage d'erreur dans tools; sinon nous aurions affiché les messages dans le terminal directement depuis le module noeud.



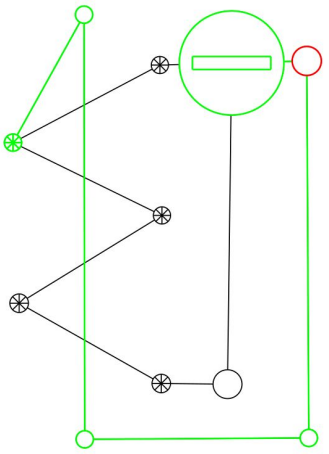
Le module ville contient la classe Ville qui contient un vecteur de pointeurs vers des objets de classe Nœud et un vecteur d'array de deux unsigned int pour les liens. Lors de la lecture, une instance d'une ville est créée, ses attributs static sont réinitialisés et les nœuds et liens y sont ajoutés, sous condition de passer les tests respectifs.

Le module noeud contient la classe Noeud qui contient l'uid, la capacité, le nombre de liens, l'objet Cercle (centre et rayon), les pointeurs vers les voisins et les attributs in, access et parent (utilisé dans l'algorithme de Dijkstra) du noeud, ainsi que des attributs static pour sauvegarder les sommes des capacités par type ainsi que le nombre total de liens. Les sous-classes permettent de différencier entre les fonctions de dessin, mais aussi les compteurs de capacité et du nombre de nœuds par type, qui sont utiles pour le calcul d'ENJ.

L'algorithme de Dijkstra se trouve dans la classe Ville et a été largement inspiré du cours. Toutefois, nous avons dû l'adapter pour prendre en compte les vitesses différentes entre noeuds et le type du noeud recherché, que nous avons choisi de passer en paramètre.



Bien que le lien direct entre le noeud sélectionné et le noeud Production soit plus court, le chemin le plus rapide pour relier les deux passe par les noeuds Transport



Le noeud sélectionné ne peut pas rejoindre le noeud Transport derrière le noeud Production vu qu'il est interdit de le traverser

Méthodologie

Nous avons programmé le rendu1 sur Visual Studio via Live Share, donc nous avons pu travailler à deux sur le même module en même temps et ne les avons pas concrètement partagé entre nous. Néanmoins, Yann s'est plutôt occupé de la structure des données dans le module ville tandis que Matthias a conçu les tests dans noeud. C'est en oubliant "cout << " avant l'appel du message de succès que nous avons perdu des points sur ce rendu.

Nous avons travaillé sur Linux et Geany pour le rendu2, à cause de difficultés lors de l'installation de gtkmm pour Visual Studio sous Windows. Matthias a donc pris la responsabilité du module gui tandis que Yann a ajouté les fonctions dessin dans le modèle et créé le module graphic. Après toutes les fonctionnalités installés, nous avons synchronisé nos travaux via des partages d'écran. Pendant cette période, le module tools a été complété selon les besoins de chacun et la majorité du module noeud a été retravaillée, tandis que le module projet, bien que petit et simple en sa version finale, est passé par de nombreuses versions.

Finalement, pour le rendu3, nous sommes retournés sur Visual Studio Code en utilisant WSL (Windows Subsystem for Linux), qui permet d'installer et exécuter des packages linux dans une shell sous Windows, et donc de compiler avec gtkmm. Suite à la répartition des tâches pour le second rendu, Matthias à finalisé le module gui en ajoutant les signal handlers nécessaires (et d'autres) et Yann a terminé graphic en ajoutant le zoom et les headers additionnels. Le travail sur les modules du modèle s'est fait de nouveau par Live Share, nous avons donc écrit à deux les fonctions de modification et manipulation de la ville et celles qui en découlent dans les autres modules.

Nous n'avons pas utilisé de Debugger ni sous Windows, ni sous Linux; les tests du programme se sont donc faits par exécution "normale" du programme, et les erreurs ont été détectées par ajout de sorties de contrôles à différentes étapes dans le programme. Cette approche fut peu efficace lors de l'apparition de segmentation faults, mais nous avons remarqué une accélération de la résolution de ces problèmes avec l'avancée du projet. Avec ce gain d'expériences, le manque de Debugger ne nous a plus gêné à la fin du projet, bien qu'un tel serait à considérer pour des travaux plus complexes.

La faute la plus fréquente lors de nos tests fut probablement un mauvais calcul du critère MTA. Ceci était lié à une condition manquante dans l'algorithme de Dijkstra (renvoi de l'indice du noeud non seulement si son type est correct, mais aussi si son temps d'accès n'est pas infini), mais aussi à un second appel trop rapide de l'algorithme (l'indice de la destination est récupéré, mais les temps d'accès et indices parents sont écrasés par le second appel avant leur accès), un oubli de réinitialisation du critère entre chargement de fichiers et un maintien de la valeur infinite_time après l'ouverture d'un nouveau fichier si le MTA de celui-ci n'est pas égal à 0, ce qui nous a coûté un point sur le second rendu.

Conclusion

Nous sommes finalement satisfaits par notre performance sur ce projet, bien que nous ayons perdu quelques points stupidement sur les deux premier rendus. Le travail sur Visual Studio et la collaboration directe par vidéo-chat étaient des grands avantages pour ce travail, et rendirent la conception et écriture de code très intéressantes et souvent amusantes. Nous avons hâte d'avoir une nouvelle occasion de créer un tel programme, bien que la recherche, correction et certitude sur l'absence de bugs furent parfois frustrantes.