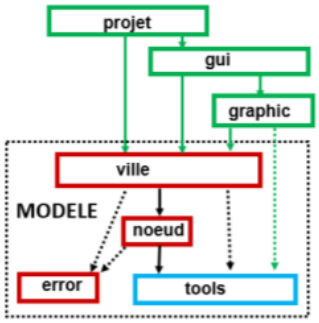
315587 Victor CARTIER-NEGADI
314472 Louis GAVIGNET

Rapport Final

- **Architecture logicielle et description de l'implémentation**

Nous avons choisi l'architecture de la Fig11b2.



Le module de plus bas niveau *tools* contient des structures de base tel que le Point, Cercle, et des fonctions telles que la norme d'un vecteur, le produit scalaire, la distance entre deux points qui sont utilisés à divers endroits du programme. Le module nœud contient la classe quartier, ainsi que les fonctions vérifiant s'il y a des erreurs dans le fichier fourni ou la ville dessinée. Le module ville se charge de lire le fichier fourni, de créer et mémoriser les informations de la ville dans une classe city, il fait appel aux fonctions de vérification du module nœud durant la lecture. Le module graphic contient les fonctions de dessins des nœuds et liens, ces fonctions sont appelées depuis le module gui. Ce dernier module se charge de l'interface graphique, des modifications des valeurs de la ville, et du calcul des critères qui apparaissent dans l'interface. Le calcul des critères est dans ce module car nous ne l'utilisons nulle part ailleurs, ainsi on place les valeurs au plus proche de l'appel de celles-ci.

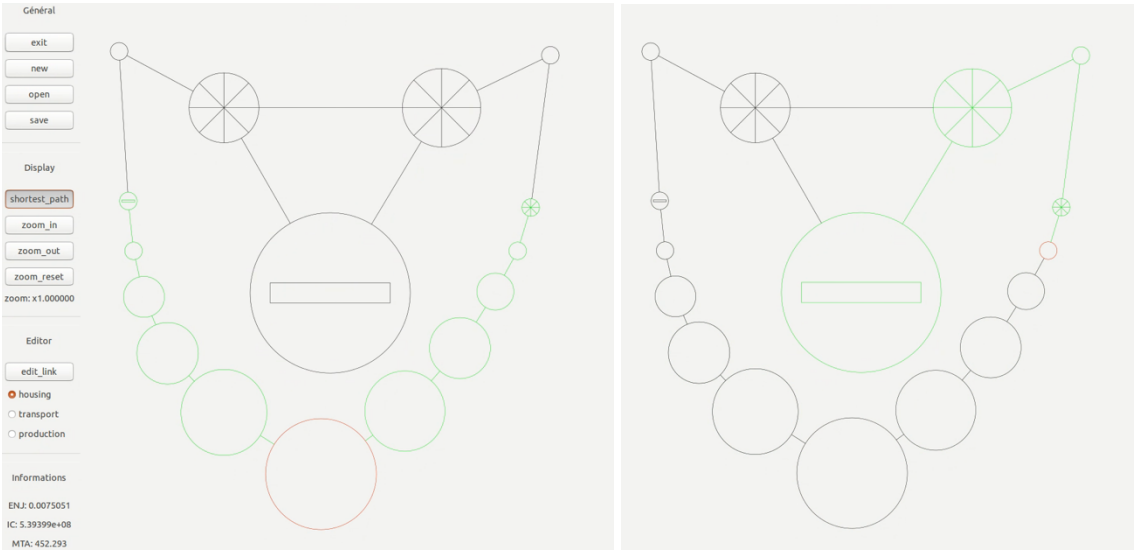
Nous avons une classe City qui mémorise toutes les informations de la ville. Elle contient un tableau contenant les pointeurs des quartiers, un tableau contenant les pointeurs des liens entre les quartiers, et les nombres de quartiers de type logement, transport, et production. Nous avons donc des fonctions qui ont pour rôle de supprimer un quartier et ses liens tout en actualisant le nombre de quartier de ce type, par exemple la fonction delete_q() dans le module gui.

Nous avons adapté le pseudocode fourni de la manière suivante. Nous n'avons pas indexé le tableau TA et ce grâce à l'aide d'une fonction appelée *q_of_uid()* utilisée pour manipuler les données d'un quartier dont nous n'avons que l'uid. Nous avons également créé une fonction *check_liens()* chargée de passer en revue les liens d'un nœud et d'approfondir la recherche si possible. Enfin pour trouver le nœud transport le plus proche d'un logement nous utilisons la même fonction check_liens() à la différence près que la recherche en profondeur est stoppée dès lors qu'elle fait face à un nœud de type trois, c'est-à-dire un nœud de type production dans le code de notre programme.

315587 Victor CARTIER-NEGADI
314472 Louis GAVIGNET

Rapport Final

Dessin du plus court chemin pour deux nœuds d'une même ville :



- **Méthodologie et conclusion**

Nous essayions de travailler ensemble sur les différents modules en nous répartissant les fonctions à coder, ainsi nous avons chacun coder la moitié de chaque module. Nous testions chaque fonction dès qu'elle était écrite avec la méthode de scarfolding, donc nous voyions dans le terminal à l'aide de « cout » si des parties de la fonction étaient défaillantes et donc corriger si nécessaire.

Au début de l'écriture de notre programme nous passions toujours la classe de la ville ou les tableaux qu'elle contient en argument des fonctions. Ainsi apparu une erreur « segmentation fault » due à un problème d'allocation de mémoire, erreur qui est apparue à plusieurs reprises lors de différentes étapes du projet. La solution mis en œuvre fut de transformer ces tableaux de classes ou de valeurs en tableaux de pointeurs.

L'élément qui nous a posé le plus de problème lors de ce projet était la mise en place de l'algorithme de Dijsktra. En effet cet algorithme n'est pas évident à coder donc des petits bugs qui se sont ajoutés au problème d'allocation mémoire d'où la difficulté rencontrée pour écrire cet algorithme.

Un point de difficulté lors de notre projet fut la gestion du temps surtout au début pour le premier rendu. La mise en place d'un environnement de travail où nous pouvons coder en même temps tout en voyant l'évolution du code de notre binôme est un bon point dans ce projet. Nous avons appris à travailler en groupe sur un projet de manière claire et efficace.