Optimiseur de Coupe – Réseau

IED Paris 8 – Licence d'Informatique L2

Il a été convenu que Rebecca Charbit s'occupait du serveur et Pierre-Emmanuel Pirnay du client. Ce document décrit le travail de chacun.

1. Travail réalisé par Rebecca Charbit

Afin de pouvoir réalisé un serveur permettant de répondre à plusieurs clients en même temps, nous avons choisi Pierre-Emmanuel et moi d'utiliser des threads. Après avoir lu la documentation de Qt qui propose plusieurs méthodes pour gérer les threads, mon choix s'est porté sur QThreadPool et Qrunnable.

J'ai donc créé une classe Serveur qui gère le serveur TCP de Qt (QTcpServer) et s'occupe de demander à l'utilisateur. À chaque fois qu'un client se connecte, une instance de la classe Client est créé. C'est cette instance qui gère la socket permettant de communiquer entre le client et le serveur. À chaque fois que cette socket reçoit un message, un thread est créé via un objet de type Task. Le thread exécute alors tout le contenu de la méthode Task::run qui s'occupe de lancer le moteur et de renvoyer le résultat pour que la socket renvoie le résultat. Pour cela le thread utilise la classe ProtocoleODC_serveur (basé sur la classe ProtocoleODC_client) qui permet d'analyser le message du client et de générer le résultat à envoyer.

Le moteur a du être légèrement modifié afin que les fonctions List::toStr et Combinaison::toStr soient adaptées au protocole de l'Optimiseur de Coupe.

2. Travail réalisé par Pierre-Emmanuel Pirnay

Afin de réaliser l'Optimiseur de Coupe avec l'utilisation du réseau, j'ai en premier lieu créé le protocole à utiliser afin que le serveur et les clients puissent facilement communiquer entre eux. Les détails concernant ce protocole se trouve dans le fichier « **Protocole ODC.md** ».

Puis j'ai réalisé le client du programme. Pour cela, j'ai réalisé les opérations suivantes à partir du code de l'Optimiseur de Coupe original :

- suppression du moteur qui doit dorénavant faire partie du serveur
- création de la classe ProtocoleODC_client qui permet :
  - de générer le message à envoyer au serveur lors de l'envoi d'un nouveau formulaire

    - d’exploiter la réponse du serveur en vérifiant qu’elle soit bien valide et en la convertissant en données exploitables
- création de la classe DemandeIpPort qui gère la fenêtre à afficher lors du lancement du client demandant l’ip et le port du serveur
- création de la classe Client qui gère la socket permettant de communiquer avec le serveur. Cette classe envoie les messages au serveur et récupère les réponses du serveur
- Modification de la classe FenetrePrincipale qui dorénavant n’utilise plus le moteur directement mais une instance de la classe Client et de ProtocoleODC_client afin d’afficher les résultats demandés.
- tests du client et du serveur une fois terminés

Une grosse majorité du travail réalisé se trouve dans le dossier **reseau** du code source.