Rendu du projet d’affichage SVG avec interaction réseau :

Ladislas WALCAK, Simon DRIEUX, Baptiste JOFFROY, Luka MERCIER

Licence 3 - Ingénierie Informatique - Université d’Orléans 2020

Introduction :

Ce projet a pour but le développement d'un projet en C++ qui réunit deux principales compétences vues en Licence cette année : Réseau et Programmation Orienté Objet.

Le but de cette application est de pouvoir modifier un fichier SVG, situé sur un serveur ayant un socket UDP ouvert. Ainsi, nous proposons une version client qui permet d'entrer et envoyer des valeurs au serveur qui se chargera alors de la modification du SVG et du rafraîchissement de l'image.

Structure du code :

Le projet a été structuré en plusieurs répertoires, ainsi qu'un fichier Makefile adapté contant plusieurs commandes utiles. Nous avons utilisé CMake pour le développement du projet.

La structure du projet est :

- Un dossier *include* contenant tous les fichiers en-têtes (.hpp)
- Un dossier *src* contenant tous les fichiers sources (.cpp)
- Un dossier *obj* créé lors de la compilation contenant tous les fichier objets (.o)
- Un dossier *bin* créé lors de la compilation contenant les fichiers exécutables
- Un dossier *Image Samples* contenant le fichier SVG utilisé par le serveur pour la modification.
- Un fichier *Makefile* contenant les commandes liées au projet

De plus, ce projet contient un nombre de fichiers supplémentaires lié au projet :

- Un fichier *LICENCE.md* contenant la licence open-source.
- Un fichier *README.md* contenant des instructions de lancements
- Un fichier *.gitignore* lié au répertoire Git du projet
- Un fichier *Sujet.pdf* contenant le sujet détaillé du projet

Le dossier *src* est constitué de 4 couples de fichiers en-têtes / sources, ainsi que de deux fichier source supplémentaire pour différencier le serveur et le client.

- Le fichier *main.cpp* est le fichier faisant la liaison entre tous les fichiers lié au serveur. Il créer une instance de *server* et lance l'écoute.
- Les fichiers *server* contiennent la classe *Server*. Cette classe est chargé de lancer un socket UDP écoutant sur un port donné. De plus, cette classe contient en attribut l'instance de *XMLController* afin de lui passer l'information lorsqu'une donnée est reçu.
- Les fichiers *XMLController* contiennent la classe *XMLController*, lié à l'utilisation de la bibliothèque *librsvg* qui permet la modification du XML d'un fichier SVG. De plus, il contient en attribut l'instance de *ServerUI* afin de mettre à jour l'affichage lorsque l'image est modifié.
- Les fichiers *serverUI* contiennent la classe *ServerUI*, lié à l'affichage de l'image avec la librairie *GTK2* ainsi que du rafraîchissement d'image à chaque fois que l'image est modifié.
- Les fichiers *client* contiennent un client UDP sans interface utilisateur, qui est préparé pour envoyer des requêtes à un port spécifié.
- Le fichier *clientUI.cpp* contiennent l'interface IHM étroitement lié à la classe *client* pour faciliter l'envoi de données au serveur (Pas finalisé et donc pas utilisé dans le projet final).

Fonctionnement

Le programme est divisé de manière à s'exécuter "fichier par fichier".

Dans un premier temps, le fichier main va instancier une classe Server qui s'occupera de générer un serveur UDP qui va écouter sur un port spécifique. Il va ensuite ajouter une nouvelle classe en attribut, XMLController pour faire transiter les changements de données. Cette classe XMLController va elle se charger de modifier l'image SVG grâce à l'utilisation de balise XML dites "driven". De plus, cette même classe va également ajouter à ses attributs une instance de la classe ServerUI. Cette instance va être chargé de créer l'affichage graphique de l'image, en créer une fenêtre gtk dans un thread à part.

Pour le client, lors du lancement du programme, le client va ouvrir un socket UDP avec l'adresse et le port spécifié. Il demandera ensuite à l'utilisateur de rentrer deux valeurs numériques sous la forme "x y" (séparé d'un seul espace). Dans le cas où l'utilisateur ne respecterait pas ce format, le programme demandera de nouveau deux valeurs. Lorsque deux valeurs correctes seront entrées, le client enverra ces deux valeurs au serveur, puis se fermera.

Améliorations possibles

- Amélioration de l'interface client pour afficher le SVG et voir ses changements afin d'offrir une meilleur expérience utilisateur.
- Coté serveur, une possible gestion des clients avec blocage de réception de données par exemple.
- Meilleur gestion de la mémoire.
- Permettre le choix de la taille de l'image initiale.
- Meilleure gestion du client en permettant l'envoi de plusieurs valeurs successives sans avoir à relancer le programme.
- Meilleur gestion des erreurs au niveau de la lecture des valeurs données par le client.
- Le serveur étant une boucle infini, le seul moyen d'arrêter le programme est l'utilisation de CTRL + C, et donc la mémoire n'est jamais libéré.
- Meilleur documentation.

Difficultés rencontrées

- 1ere utilisation des libraires open-source non intégrées dans le langage C++, donc difficultées à exécuter les libraires, notamment avec le Makefile et le CMake.
- La documentation des libraires était pauvre, difficile à comprendre, pas assez d'explications sur le fonctionnement des fonctionnalités et des paramètres utilisés.
- Utilisation d'un mélange de C et de C++ dû à l'utilisation d'un socket UDP écrit en C.

Répartition du travail :

Ladislas WALCAK :

Tâches principales : Gestion de la communication UDP client/serveur en utilisant la libcbor et Multi-Threading avec Simon.

Dans un premier temps, je me suis attelé à la réalisation d'un client UDP pouvant envoyer des valeurs numériques au format Cbor. J'ai pour cela repris le client de base donné, que j'ai modifié afin qu'il utilise le format de données Cbor, mais également que la gestion de la mémoire soit correcte. Dû à la faible quantité de documentation de libcbor, la tâche à été assez difficile. J'ai donc par la suite logiquement continué en développant la partie serveur, permettant la réception des données envoyées. Bien qu'ayant développé l'envoie de données, le manque de documentation a encore une fois posé de gros problèmes, nous obligeant à tester petit à petit, en essayant de deviner les fonctions à utiliser en fonction de leurs noms. Pour finir, après que la partie SVG ait été faite, je me suis joint à Simon, afin de trouver un moyen de lancer deux boucles infinies en simultanées en lançant la seconde dans un thread séparé. Pour finir, j'ai réalisé quelque corrections mineurs, puis nous avons rédigé le rapport.

Simon DRIEUX :

Tâches principales : Affichage du SVG, rafraîchissement de l'image après modification, Contrôleur du SVG Handle avec Baptiste, Multi-Threading avec Ladislas, Gestion du projet (Drive, Trello).

En première période, je me suis chargé de gérer les outils de gestion pour le projet, c'est à dire, un Trello, un Drive avec des schéma pour visualiser le fonctionnement de l'application, un recensement des informations utiles pour les libraires utilisées et le programme lui-même. En phase de développement, je me suis chargé de se focaliser sur la technologie SVG et ses librairies à l'aide de Baptiste JOFFROY. Puis j'ai commencé la programmation orientée objet du fichier exemple de M. Becker afin de séparer l'affichage et la manipulation du SVG. J'ai donc développé un contrôleur qui utilise seulement la librairie RSVGHandle qui permet de modifier un fichier SVG et une interface graphique GTK qui s'occupe de récupérer le RSVGHandle et l'afficher sur un template Cairo. Mon travail s'est porté aussi sur le rafraîchissement de l'image automatique, qui n'est pas une chose aisée quand on connait vaguement la librairie GTK. Puis en phase d'intégration des fonctionnalités avec Ladislas, nous avons dû utiliser du multi-threading pour que deux boucles infinies puissent s'exécuter, une fonctionnalité du C++ que j'ai peu utilisé.

Baptiste JOFFROY :

Tâches principales : Mise en place du changement dynamique de l'image pour lier XML/SVG

Lorsque mes camarades ont finalisé le serveur UDP et le GTK Client, j'ai mis le XMLController qui récupère le document XML pour, par la suite, modifier la position du soleil au gré de l'utilisateur. L'image, mise à jour, est ensuite envoyée au serveur UI pour pouvoir l'afficher. J'ai été aidé pour cela de Ladislas WALCAK et de Simon DRIEUX qui m'ont indiqué comment mettre en place le dispositif.

Luka MERCIER :

Tâches principales: création du serveur UDP et recherches sur le threading en cpp et tinyXML pour l'utilisation de balises specifiques.

J'ai dans un premier temps aidé à la création de l'exécutable client, en travaillant en pair avec Ladislas lors de sa réalisation. Par la suite, j'ai été amené à travailler avec toutes les personnes du groupe, individuellement, en apportant mon aide, notamment dans la résolution des bugs, par de la recherche dans les différentes documentations disponibles, bien que très rares.