Optimiseur de coupe – Rapport

IED Paris 8 – Licence d'Informatique L2 – Pierre-Emmanuel PIRNAY 11296315

Introduction

En cherchant à réaliser ce projet, j'avais trois buts en tête : faire quelque chose d'utile, améliorer mes connaissances en C++ et Qt et enfin apprendre la procédure afin de réaliser un programme. Le premier, je pense, a bien été réalisé : l'optimiseur de coupe est un outil pratique pour n'importe quel bricoleur, facilement utilisable. Concernant le C++ et Qt, je suis très fier de ce que j'ai pu apprendre : j'ai une meilleure compréhension de l'intérêt des objets et une meilleure vue de la taille d'un framework tel que Qt qui est extraordinaire tant par la qualité des outils proposés que par la documentation qui est extrêmement bien faite. Pour le dernier but, j'ai plutôt réalisé à quel point il est nécessaire de passer par des étapes précises afin de réaliser quelque chose de correct.

Ce rapport présente ma participation à ce projet ainsi que les difficultés que j'ai été amené à résoudre ainsi que les possibles améliorations à apporter à l'Optimiseur de coupe.

1. Travaux effectués

Choix des outils

Nous avons en premier lieu discuté du programme en général et de son but sans se pencher sur les détails tels que l'algorithme utilisé, l'aspect du programme, etc. Puis, j'ai proposé les outils à utiliser : git pour le versionning et Qt. Ce premier étant très populaire aujourd'hui, il m'a semblé judicieux de nous reposer sur celui-ci en utilisant le site GitHub afin de partager facilement notre code et d'avoir une meilleure vue sur qui a fait quoi, quelles modifications ont été faites, et sur quel fichier, etc. Concernant Qt, ayant déjà un peu appris comment l'utiliser, et vu le projet que l'on avait en tête je me suis dit que ce serait un bon choix afin de réaliser un programme facile à utiliser pour l'utilisateur et je me suis donc proposé pour la création de l'interface graphique.

Décomposition du programme

Afin de répartir les tâches, j'ai proposé que l'on s'inspire du modèle MVC ce qui a donné lieu à deux parties : le moteur de calcul et la GUI. Ainsi chacun peut se concentrer sur sa partie sans se soucier du fonctionnement de l'autre partie.

Réalisation de la maquette

Aidé des documents réalisés par Xavier, j'ai réalisé une maquette avec QtDesigner et on a pu discuter des modifications à faire. La première est l'abandon de l'utilisation d'un champ unique afin d'indiquer les barres disponibles avant la découpe : l'utilisateur devait rentrer une expression tel que (4*3, 10, 5.25*4 afin de désigner 3 barres de 4 cm, une de 10 et 4 de 5,25) et d'utiliser le regex

afin de vérifier les erreurs de syntaxes qui étaient tout à fait probables avec une telle solution. Mais, ne trouvant pas que cette solution serait facile à utiliser par l'utilisateur, j'ai proposé l'utilisation d'ajout dynamique de saisies via des boutons « ajouter/supprimer une saisie » où chaque saisie contiendrait un champ spin-box pour la longueur, un autre pour la quantité et (une fois l'idée de l'unité proposée) un champ combo-box pour les unités. Cette proposition une fois adoptée était devenu beaucoup plus user-friendly. Cette solution permettant d'effectuer aussi un contrôle beaucoup plus simple des entrées de l'utilisateur. Plus besoin du regex, il suffit de définir les valeurs minimales et maximales des champs.

Création de la GUI

On avait donc une maquette de la GUI. Je suis donc passé à sa création avec QtCreator en réfléchissant à la façon dont mon programme devait être organisé, quels objets utilisés, etc. Toujours avec l'aide des documents de Xavier, j'ai donc codé la première version. J'ai ajouté le convertisseur d'unités afin de pouvoir facilement vérifier les saisies de l'utilisateur et ainsi afficher les erreurs nécessaires. Les onglets ont aussi été placés afin de gagner de l'espace et de ne pas faire une trop grande fenêtre. Un petit détail aussi a été ajouté : le numéro des saisies afin que l'utilisateur, s'il souhaite rentrer beaucoup de saisies, puisse se repérer facilement. Plusieurs remarques ont été faites et prises en compte au fur et à mesure par Rebecca et Xavier tel que le texte et les valeurs par défauts à utiliser.

Traduction du programme

J'ai ajouté un système de traduction du programme via QTranslator qui détecte la langue du système et charge ou non la traduction. Ce système est très pratique, car il suffit d'utiliser la fonction tr(QString str) à la place des chaînes de caractère puis d'utiliser quelques programmes de Qt dont Qt Linguist qui permet facilement de traduire les chaînes contenues dans les fonctions tr. Seule la langue anglaise a été ajoutée, mais il est tout à fait possible d'ajouter d'autres langues.

Proposition d'utilisation du type double

Ayant remarqué que les spin-box de Qt renvoyées un nombre de type double et que Rebecca utilisait le type float, je lui ai proposé d'éviter des casts en utilisant dans le moteur uniquement des nombres de type double, ce qu'elle a acceptée.

Modification de l'organisation du dépôt et ajout de gitignore

On avait chacun beaucoup de fichiers, de code et de la documentation à partager. Trouvant que cela faisant un trop grand désordre, j'ai réorganisé le dépôt en créant un dossier « Doc » pour la documentation et autres fichiers, un dossier « moteur » pour que Rebecca puisse mettre son code et un dernier dossier « gui » pour mon code. Cette organisation était il me semble nécessaire, car la racine du dépôt contenait une quinzaine de fichiers PDF et odt et il devenait difficile de s'y retrouver. J'ai aussi ajouté un fichier gitignore afin d'ignorer les fichiers .o ou .a que Rebecca créait

en compilant le moteur. Ces fichiers étant inutiles pour nous, nous prîmes donc la décision de les ignorer. Les fichiers temporaires (finissant par ~) ont eux aussi été ajoutés à gitignore. Apparemment, Rebecca utilisait un éditeur qui ne supprimait pas les fichiers temporaires.

Tout ceci a été fait dans le but de simplifier un maximum le dépôt et d'améliorer son organisation afin que nous puissions nous concentrer sur l'essentiel.

Proposition d'ajout des conditions DEBUG

Rebecca utilisait la sortie console afin de tester le moteur. Je lui ai proposé d'utiliser des conditions de type #if DEBUG pour les fonctions d'affichage, afin que la gui et la sortie standard ne soient pas combinées lorsqu'on utilise le programme entier (gui et moteur). Cette proposition a été retenue.

Ajout du moteur

Une fois l'interface graphique et le moteur bien avancée, je me suis attelé à la tâche de regrouper les deux. Je parlerai de ce travail plus longuement dans la partie concernant les difficultés rencontrées. Cette étape fut très importante pour nous tous car on commençait à voir quelque chose de concret et cela nous a permis de tester plus facilement le moteur : on s'est rendu compte des limites de l'algorithme utilisé à ce moment par le moteur et de ses bugs. J'ai aussi modifié le texte original des résultats afin qu'il soit user-friendly au maximum.

Ajout des résultats sous forme graphique

Une fois le moteur bien intégré à la GUI, j'ai ajouté les résultats sous forme graphique. Je parlerai aussi de ce travail plus longuement dans la partie concernant les difficultés rencontrées.

2. Difficultés rencontrées

Apprentissage de git et Qt

La première difficulté fut d'apprendre à utiliser git. J'ai donc suivi un cours proposé par OpenClassRooms et lu quelques tutoriels sur le net. Puis j'ai créé un compte sur GitHub et appris comment faire le lien entre mon dépôt git local et GitHub.

Organisation du programme

Je me suis mis à réfléchir à la façon dont devait être organisé le code et plus particulièrement à quels objets créés. J'ai opté pour la solution suivante : une classe pour la fenêtre principale qui gère les onglets, le moteur de calcul et la détection des erreurs. Puis une classe différente pour chaque onglet. Dans le formulaire de saisie, il y a deux systèmes de saisie dynamique : la saisie des barres avant la découpe et la saisie des tronçons désirés. J'ai donc au début créé une classe Saisie avec à l'intérieur deux spin-box pour la longueur et la quantité et un combo-box pour l'unité. Puis avec deux vecteurs de Saisie, 4 boutons « ajouter/supprimer une saisie » et deux fois le système de

création de Saisie ou de suppression de Saisie on obtenait deux systèmes d'affichage dynamique. Mais je trouvais ridicule le fait d'avoir deux fois le même code, dans le même fichier. J'ai donc créé une classe GroupeSaisie qui contient le vecteur de Saisie, un bouton d'ajout et un bouton de suppression et le système de création et de suppression de Saisie. Ainsi il me suffit de créer deux objets instance de la classe GroupeSaisie, un pour les barres avant la découpe et un autre pour les tronçons désirés.

Ajout du moteur

L'une des parties les plus difficiles pour moi fut l'ajout du moteur au programme. En effet, n'ayant pas codé cette partie, j'étais au début dérouté sur la façon dont je devais m'y prendre. D'un côté je ne voulais pas trop rentrer dans la complexité du code, mais il fallait tout de même que j'intègre cette partie à mon programme. Trois choses m'ont aidés à résoudre ce problème : les commentaires de Rebecca dans son code, les indications qu'elle m'a envoyé, et surtout les fonctions d'affichage de résultats. En effet, pour tester son moteur, Rebecca utilisait des fonctions toutes nommées affichage() qui affiche le résultat sur la sortie standard. Je me suis donc inspiré de ces fonctions d'affichage afin de créer l'onglet des résultats. J'ai compris l'importance de bien commenter son code, car il se peut que quelqu'un d'autre ai à l'utiliser mais surtout à le documenter. Cette difficulté m'a fait comprendre que si j'ai à coder quelques classes ou fonctions, qui doivent être utilisées par d'autres personnes, je dois créer une mini documentation indiquant juste la façon dont il faut utiliser ces classes ou ces fonctions sans en détailler le fonctionnement. C'est capital selon moi pour un bon partage de code sans que l'autre n'ai à se plonger dans mon travail et vice-versa.

Création du graphique des résultats

La première difficulté rencontrée avec cette partie fut d'apprendre à utiliser QPainter, la classe permettant de dessiner avec Qt. En effet, elle est un peu particulière à utiliser et pour cela quelques vidéos sur YouTube m'ont beaucoup aidés.

Mais la plus grosse difficulté est que afin de savoir ce que je devais dessiner, j'envoyais à l'objet instance de la classe WigetGraphique gérant l'onglet du graphique des résultats une std::list de Combinaison, une classe créée par Rebecca contenant toutes les informations nécessaires sur la découpe des barres à effectuer. Pour chaque barre à découper, un nouveau objet instance de la classe BarreGraphique est créé. Le constructeur de cet objet reçoit un objet instance de la classe Combinaison puis une fonction spéciale de Qt : `void paintEvent (QPaintEvent *event)` se charge de dessiner la barre en accédant à l'objet envoyé reçu par le constructeur de BarreGraphique. Le problème est que lors de l'accès de l'un des attributs de ce dernier (de type std::list) via une méthode d'accès, j'obtenais une erreur de segmentation. J'ai utilisé le Debugger de QtCreator mais cela ne m'a pas beaucoup avancé. Après plusieurs heures de tests et de recherches, je me suis rendu compte que c'est l'accès à cet attribut à l'intérieur de la fonction paintEvent qui causait le problème mais à partir d'une autre fonction de la classe, tel que le constructeur aucun problème. La raison à cela m'est toujours inconnue. J'ai donc créé des attributs

à la classe BarreGraphique et via le constructeur ces attributs contiennent maintenant les attributs de l'objet d'instance de la classe BarreGraphique. Ainsi la fonction paintEvent peut dessiner la barre via les attributs de sa propre classe.

3. Possibilités d'amélioration

Je pense qu'il serait possible d'améliorer le programme de la façon suivante :

- Ajout d'une fonction d'export en HTML des résultats
- Traduction dans d'autres langues que l'anglais
- Optimisation du code via l'utilisation de plus de pointeurs
- Amélioration du design du logiciel (surtout du graphique)
- Amélioration ou création du packaging pour les systèmes GNU/Linux et Windows
- Amélioration de l'algorithme du moteur de calcul qui reste pour l'instant limité

Conclusion

Bien qu'ayant appris beaucoup de technique, je pense que la chose apprise la plus intéressante fut la conception du logiciel avant sa création et l'organisation du développement en groupe via git ou la répartition du travail (et ainsi, l'utilisation d'un code utilisé par une autre personne). Je suis très satisfait du travail que nous avons effectué et des résultats obtenus et sera utile pour nos développements futurs.