Théo ALRIC
Maxime BLARET

Rapport TP2 OMD

Introduction

Le projet réalisé est un mini-éditeur graphique de texte. Ce rapport a pour but d'expliquer les choix de conception réalisée, le contenu du dossier rendu, les technologies utilisées et les librairies employées.

Notre éditeur de texte a été développé en deux versions successives. La première permet de saisir du texte et dispose des fonctions de traitement de texte suivantes, qui sont accessibles via des combinaisons de touches du clavier :

- *déplacement du curseur* — flèches droite et gauche
- *sélection de texte* — ctrl + flèche droite : augmenter la taille de sélection
  ctrl + flèche gauche : diminuer la taille de sélection
- *copie de la sélection dans un presse-papier* — ctrl + c
- *copie dans le presse-papier avec effacement de la sélection* — ctrl + x
- *collage du contenu du presse-papier (éventuellement par dessus une sélection)* — ctrl + v
- *suppression d'éléments* — touche supr

La seconde version dispose quant à elle de fonctions supplémentaires :

- *Annuler une action* — ctrl+ z
- *Restaurer des actions annulées* — ctrl + y

Outils utilisés pour l'implémentation

Nous avons opté pour le langage de programmation C++ afin d'obtenir des performances optimales et de pouvoir avoir la main sur tout ce qu'il se passe dans notre système et également par préférence personnelle. Nous avons utilisé la librairie "SDL2"(lien:), qui permet d'utiliser des interfaces graphiques, couplée à la librairie "SDL2_ttf" qui nous a permis de gérer rapidement l'affichage de caractères dans la fenêtre.

Contenu des dossiers :

Le dossier src contient l'ensemble des fichiers de code ainsi que le fichier de police Arial.ttf, allant de pair avec l'utilisation de la librairie SDL2_ttf.(Pour modifier cette dernière, il faut modifier l'initialisation de TTF et télécharger une nouvelle police)

Le dossier "DiagV1" contient l'ensemble des diagrammes UML utilisés pour modéliser la version 1 du système, sous la nomenclature suivante :

Fichier "Use_Case_Diagram" :

Diagramme des cas d'utilisation : il détaille les fonctionnalités de l'éditeur de texte. Le seul acteur est l'utilisateur de l'éditeur dans le cas présent.

Dossier "Séquences" :

Contient les diagrammes de séquences relatifs à chaque cas d'utilisation.

Fichier "Diagramme Classe" :

Le diagramme de classes, qui permet de visualiser les liaisons entre les différentes classes.

Dossier "CU"

Contient l'ensemble des diagrammes de séquences contextuelles. L'utilisation du design pattern "Command" amène les différentes séquences à suivre un fonctionnement relativement similaire.

Le dossier "DiagV2" contient les diagrammes utilisés pour la version 2 sous la même nomenclature.

Choix de conception

Version 1

Pour la première version du mini-éditeur, nous avons créé les classes suivantes:

System :

Cette classe contient l'ensemble des composants du programme. Elle contient la boucle principale du programme et est chargée à chaque itération de gérer les entrées clavier, d'appeler les différentes commandes et d'afficher le document dans la fenêtre graphique.

File :

Cette classe contient l'état des modifications du document : le contenu textuel, la position du curseur, les éléments actuellement sélectionnés ainsi que le contenu du presse-papier. Les attributs sont accessibles uniquement par appels de méthodes. Ainsi chaque modification appelée par un composant extérieur est réalisée par le File, ce qui rend le code plus clair et permet d'éviter les duplications .

Gestion des événements :

Les événements sont stockés par SDL2 dans une pile d'événements. Les événements clavier fournis par SDL2 possèdent un attribut SDL_KeyCode de type enum qui permet de distinguer les différentes touches (c'est le code ascii).

Nous avons utilisé le design pattern "Command" pour l'implémentation des commandes. La classe System contient une liste m_events qui récupère dans la pile de SDL2 les SDL_KeyCode des touches pressées. Ils sont ainsi plus faciles à manipuler.

Command :

Cette classe abstraite décrit le fonctionnement d'une commande générique. Chaque Command lors de son exécution parcourt la liste m_events afin de vérifier que la combinaison de touches attendue y est présente, avant de réellement faire le traitement sur le File dont le pointeur est stocké, en lui indiquant les méthodes qu'il doit utiliser.

La classe Write à l'exception des autres Commands n'attend aucune combinaison particulière, mais doit s'assurer que les touches spécifiques aux autres commandes ne sont pas actives. Ce comportement permet d'éviter de nombreuses lignes de code

Les SDL_KeyCodes présents dans m_events peuvent être supprimés à deux occasions, soit quand la touche correspondante est relâchée, ou à la fin de l'exécution d'une commande, afin d'éviter que celle-ci ne se répète. m_events est donc une variable publique afin d'être simplement modifiable et accessible par les Commands.

Ajouts dans la version 2

Dans la deuxième version, nous devions proposer un moyen de naviguer entre les différentes actions réalisées sur le fichier. Notre implémentation repose uniquement sur les insertions et suppressions qui ont été faites sur le texte, sans se préoccuper des commandes qui ont menées à ces états. Ainsi, nous évitons de stocker des copies du texte et gagnons de la mémoire.

Memento :

Cette classe abstraite stocke une information concernant une insertion/suppression réalisée sur le texte. Elle contient également la position de la modification au moment où elle a eu lieu.

Classe *Insertion* :

Ce Memento est créé suite à une suppression dans le File et contient la chaîne de caractère que l'on voudrait réintroduire dans le texte.

Classe *Deletion* :

Ce Memento est créé suite à une insertion et contient la taille de la chaîne de caractère insérée, ce qui est suffisant pour la supprimer du texte.

History :

Cette classe gère l'ensemble des Mementos existants :

Une liste m_past contient les Mementos liés aux modifications passées effectuées sur le fichier. Lorsque l'on veut revenir en arrière, le dernier Memento est appelé et retiré de la liste. Il annule alors la dernière modification faite et crée un autre Memento qui aura capacité de faire l'action inverse (un Insertion crée un Deletion et inversement). Le Memento créé est récupéré par History qui l'insère dans une liste m_future. Ainsi, on peut faire le cheminement inverse pour naviguer dans les deux sens à travers l'historique des modifications.

Modifications dans la classe Command :

History est accessible par la classe Command en tant que variable statique. Un Memento peut être créé par chaque Command lorsqu'elle modifie le texte du File. History l'ajoute alors à sa liste m_past. Si la liste m_future contient des éléments elle est vidée à ce moment là.

Deux nouvelles Commands, Restore et Cancel, ont été ajoutées pour naviguer dans l'historique.

Installation des librairies nécessaires:

Afin de pouvoir compiler les deux versions de notre éditeur de texte, il faut installer la librairie SDL2 disponible sur ce site :

SDL2

Ainsi que la librairie SDL_TTF(SDL2_ttf-devel-2.20.1-VC sur Windows) disponible ici :

SDL_TTF