OMD-S7: TP2

A rendre pour le 28/10/2024

Responsable: FEUILLATRE Hélène

Sommaire

1. Introduction

Ce deuxième travail pratique consiste à concevoir un mini-éditeur de texte. Tout comme pour le premier travail, il débutera par la création des différents diagrammes, mais contrairement à lui, ce projet est plus complet avec deux versions à réaliser et l'implémentation du code. Pour commencer, nous allons analyser différents design patterns et en choisir un qui correspond le mieux à notre problème. Une fois ce design pattern adapté à notre problématique, nous implémenterons la première version. Ensuite, nous ajusterons notre architecture pour intégrer les fonctionnalités supplémentaires de la seconde version, que nous implémenterons à son tour.

2. Version 1

Dans la première version, les fonctionnalités présentes sont :

- La sélection du texte
- L'écriture de texte
- L'effacement du texte
- Le déplacement du curseur
- La copie de la sélection (sauvegardée dans le presse-papier)
- Le coupage de la sélection (sauvegardée dans le presse-papier)
- Le collage de la sélection qui fut au préalable stockée dans le presse-papier.

2.1. Diagramme de cas d'utilisation

Au vu des fonctionnalités que doit proposer la première version de l'éditeur de texte, voici un diagramme de cas d'utilisation envisageable :

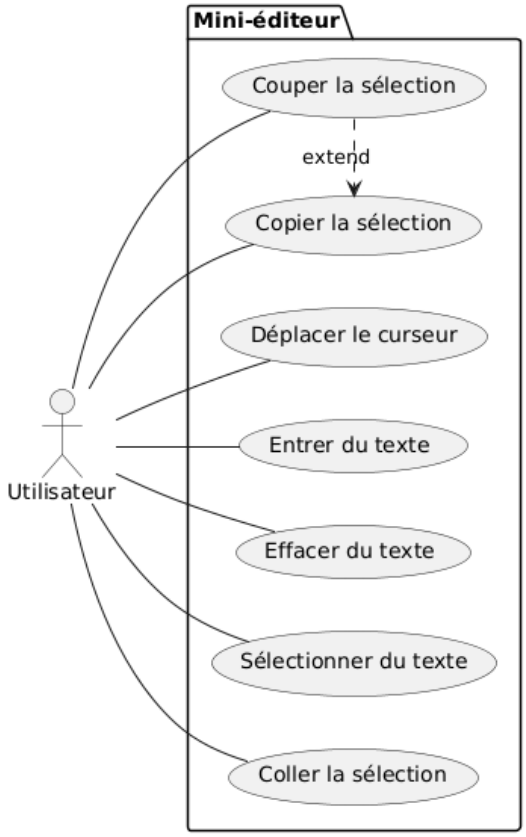

Figure 1: Diagramme de cas d'utilisation pour la première version.

Remarque : Nous avons ajouté une fonctionnalité supplémentaire qui est le zoom/dezoom de la fenêtre pour rendre notre éditeur plus confortable pour l'utilisateur.

2.2. Description des cas d'utilisation

Voici la description de chaque cas d'utilisation. Nous ne détaillerons pas les acteurs puisque nous pensons qu'il n'y a qu'un acteur qui est l'utilisateur de l'application.

Cas d'utilisation	Scénario nominal	Scénario alternatif	Scénario exception
Déplacer le curseur	a) L'utilisateur appuie sur les touches directionnelles. • gauche/droite pour aller à gauche/droite • bas/haut pour aller à début/à la fin b) Le curseur se déplace c) Sa nouvelle position est affichée.		a) La position cible du curseur est invalide (en dehors du texte) : le curseur s'arrête à la limite valide.
Entrer du texte	a) L'utilisateur saisit des caractères. b) Les caractères sont ajoutés à la position du curseur c) La zone de texte est mise à jour.	a) Le texte est ajouté à une sélection existante : remplace la sélection.	a) Caractère non autorisé (caractères spéciaux) : remplacement par dièse
Effacer du texte	a) L'utilisateur appuie sur la touche "Supprimer" ou "Backspace". b) Le caractère ou le texte sélectionné est supprimé c) La zone de texte est mise à jour.	a) Aucun texte à supprimer (curseur au début du texte).	
Sélectionner du texte	a) L'utilisateur utilise Shift + touches directionnelles. (gauche/droite : selectionne à gauche/droite) b) La portion du texte souhaitée est sélectionnée.	a) L'utilisateur peut sélectionner tout le texte avec Ctrl + A	a) Tentative de sélectionner au-delà des limites du texte : sélection seulement jusqu'aux limites autorisées
Copier la sélection	a) L'utilisateur sélectionne du texte. b) L'utilisateur appuie sur Ctrl+C.	a) Aucun texte sélectionné : vide le presse-papier et rien n'est copié.	a) Le presse-papier est plein.
Coller la sélection	a) L'utilisateur positionne le curseur à l'endroit souhaité. b) L'utilisateur appuie sur Ctrl+V. c) Le contenu du presse-papier est inséré à la position du curseur.	a) Le contenu du presse-papier est ajouté à une sélection existante : remplace la sélection.	a) Le presse-papier est vide : rien ne se passe
Couper la sélection	a) L'utilisateur sélectionne du texte. b) L'utilisateur appuie sur Ctrl+X. c) Le texte est supprimé de la zone de texte et copié dans le presse-papier.	a) Aucun texte sélectionné : vide le presse-papier et rien n'est copié.	
Zoomer/dézoomer la fenêtre	a) L'utilisateur sélectionne appuie sur Ctrl+ ou Ctrl-. b) Le zoom de la fenêtre est mis à jour. c) L'affichage de la fenêtre est mis à jour.		

2.3. Diagramme de classe

Au vu du problème que nous avons à modéliser, après avoir consulté le catalogue des design pattern présent sur Refactoring GURU, nous avons décidé de nous baser sur le design pattern nommé Command.

Ce design pattern est de type comportemental, c'est-à-dire qu'ils concernent les algorithmes et l'attribution des responsabilités entre les objets. Plus précisément, le design-pattern Command permet de transformer une requête en un objet qui contient toutes les informations relatives à la requête. Cette transformation est utile car elle encapsule chaque action sous forme d'objet, ce qui permet au client de les transmettre en tant qu'arguments de méthode, de les retarder, de les mettre en file d'attente ou bien de les prendre en charge des opérations annulables. C'est exactement le type de structure que l'on recherche pour notre mini-éditeur.

Voici la structure exposée par ce design pattern :

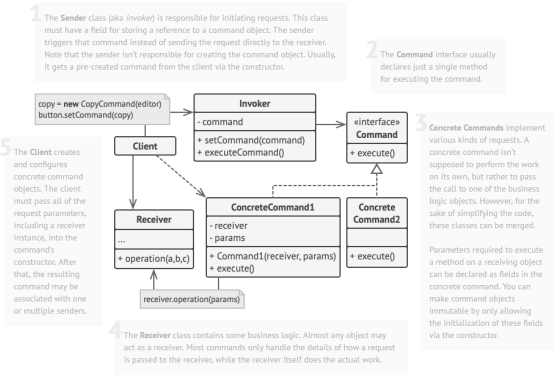

Figure 2: Design Pattern Command

Cette structure est très générale. Nous devons l'adapter à notre problème. Pour ce faire, nous avons du comprendre la structure ci-dessus. Une fois cela fait, nous avons pu la customiser pour produire notre propre diagramme de classes modélisant notre problème. Voici comment nous l'avons adapté :

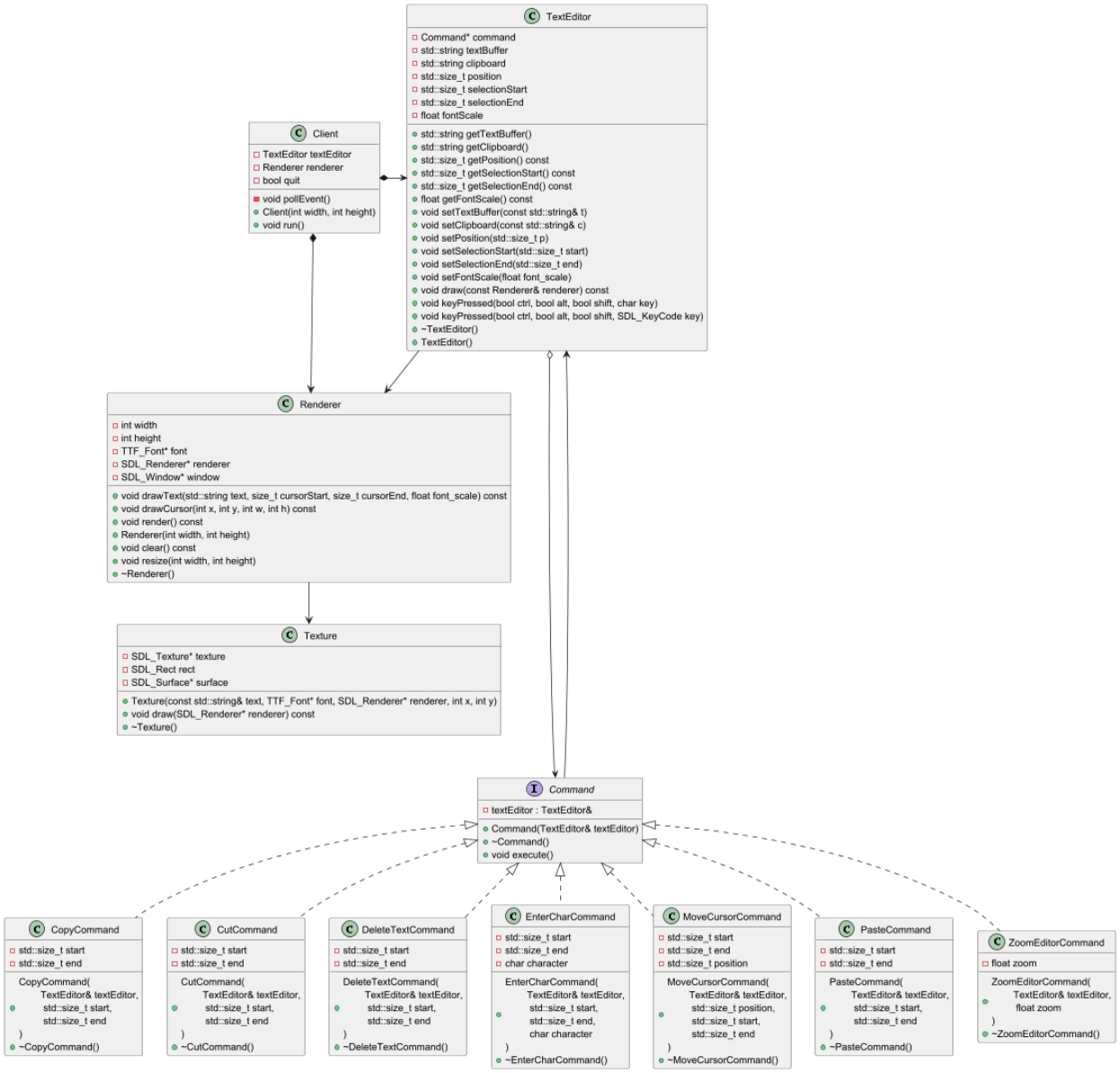

Figure 3: Diagramme de classe basé sur le Design Pattern Command pour la Version 1

Comme vous pouvez le constater, nous avons repris la même structure que le Design Pattern Command, adapté au langage de programmation choisi. Nous avons choisi C++ qui nous semblait être un bon choix étant donné que nous pouvons utiliser la SDL.

Chaque action spécifique de l'éditeur sera modélisée sous forme de classes dérivées de la classe abstraite Command. Cette abstraction nous permettra de nous assurer que toutes les commandes disposent d'une méthode execute() qui est appelée par l'éditeur lorsque l'utilisateur effectue une action.

Chacune de ces classes dérivées encapsulera des détails spécifiques à une commande particulière. Par exemple, la classe CopyCommand contiendra des attributs pour indiquer la portion de texte à copier. Elles seront créées avec un lien vers l'éditeur de texte (TextEditor&), pour pouvoir manipuler directement le contenu du buffer de texte, de la position du curseur, ou du presse-papier.

La classe TextEditor sera le centre de l'application et maintiendra l'état du texte et de gérer les interactions de l'utilisateur. Elle contiendra :

- Un buffer de texte (textBuffer)
- Un presse-papier (clipboard)
- La position du curseur (position)
- La sélection de texte (selectionStart, selectionEnd)

La classe Client sera le point d'entrée pour l'application car elle gèrera la boucle principale de l'éditeur, surveillera les événements, et interagira avec l'éditeur de texte pour lui demander d'exécuter des actions spécifiques.

Le Renderer s'occupera de l'affichage visuel (texte, curseur et redimensionnement de la fenêtre) grâce à la bibliothèque SDL. Il utilisera la classe Texture pour gérer les surfaces et textures SDL.

2.4. Diagrammes de séquence

Pour bien comprendre commment vont se produire les interactions avec les classes lors de l'utilisation de l'application, nous avons produit deux diagrammes de séquences.

2.4.1. Couper la sélection

Le cas d'utilisation "Couper la sélection" a été décrit ainsi dans la partie précédente :

- Scénario nominal :
  - 1. L'utilisateur sélectionne du texte.
  - 2. L'utilisateur appuie sur Ctrl+X.
  - 3. Le texte est supprimé de la zone de texte et copié dans le presse-papier.
- Scénarios alternatifs :
  - 1. Aucun texte sélectionné : vide le presse-papier et rien n'est copié.

Un diagramme de séquence correspondant pourrait être illustré comme suit :

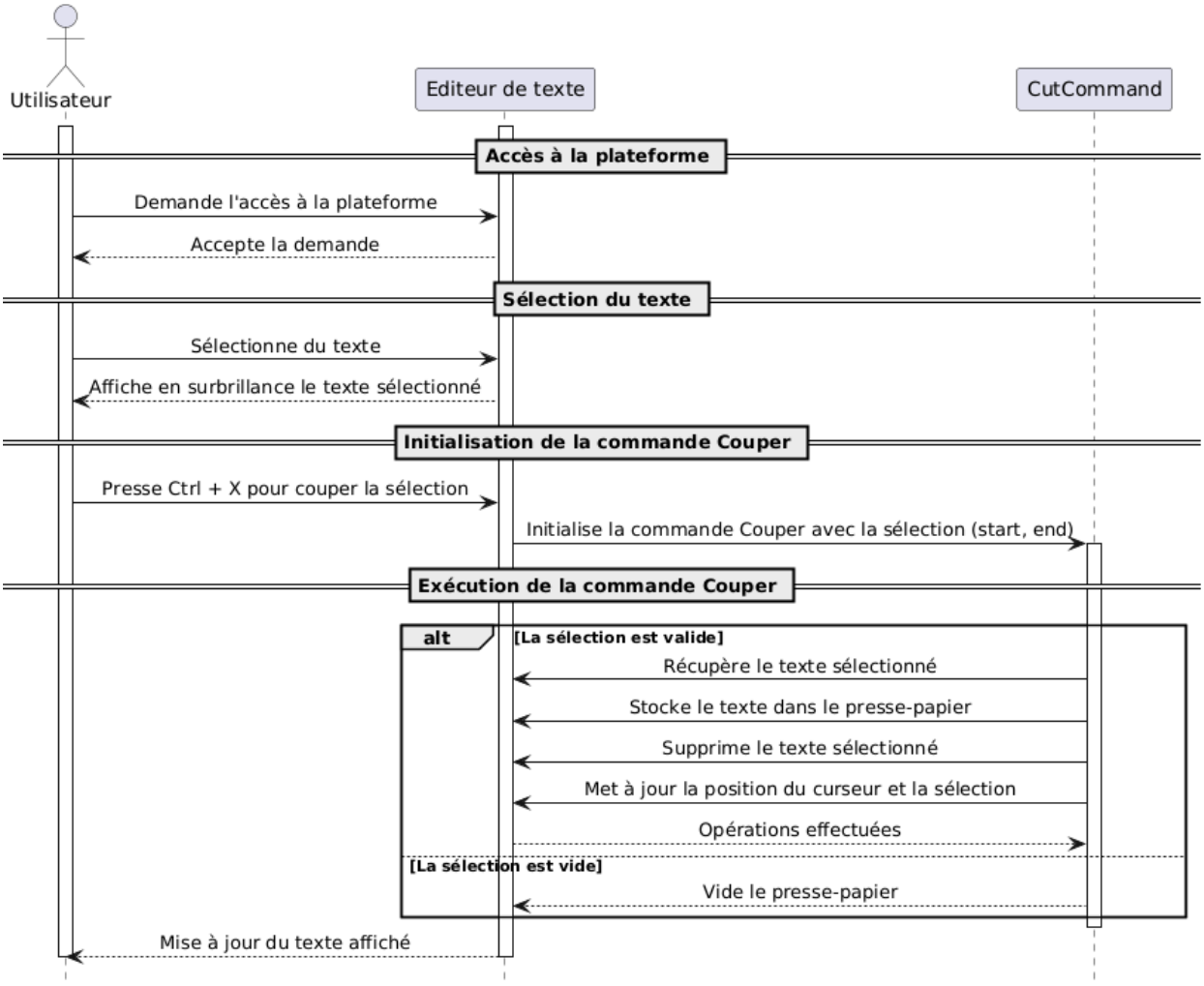

Figure 4: Diagramme de séquence pour couper la sélection

2.4.2. Entrer du texte

Le cas d'utilisation "Entrer du texte" a été décrit ainsi dans la partie précédente :

- Scénario nominal :
  - 1. L'utilisateur saisit des caractères.
  - 2. Les caractères sont ajoutés à la position du curseur
  - 3. La zone de texte est mise à jour.
- Scénarios alternatifs :
  - 1. Caractère non autorisé (caractères spéciaux) : remplacement par dièse
- Scénarios d'exception :
  - 1. Le texte est ajouté à une sélection existante : remplace la sélection.

Un diagramme de séquence correspondant pourrait être illustré comme suit :

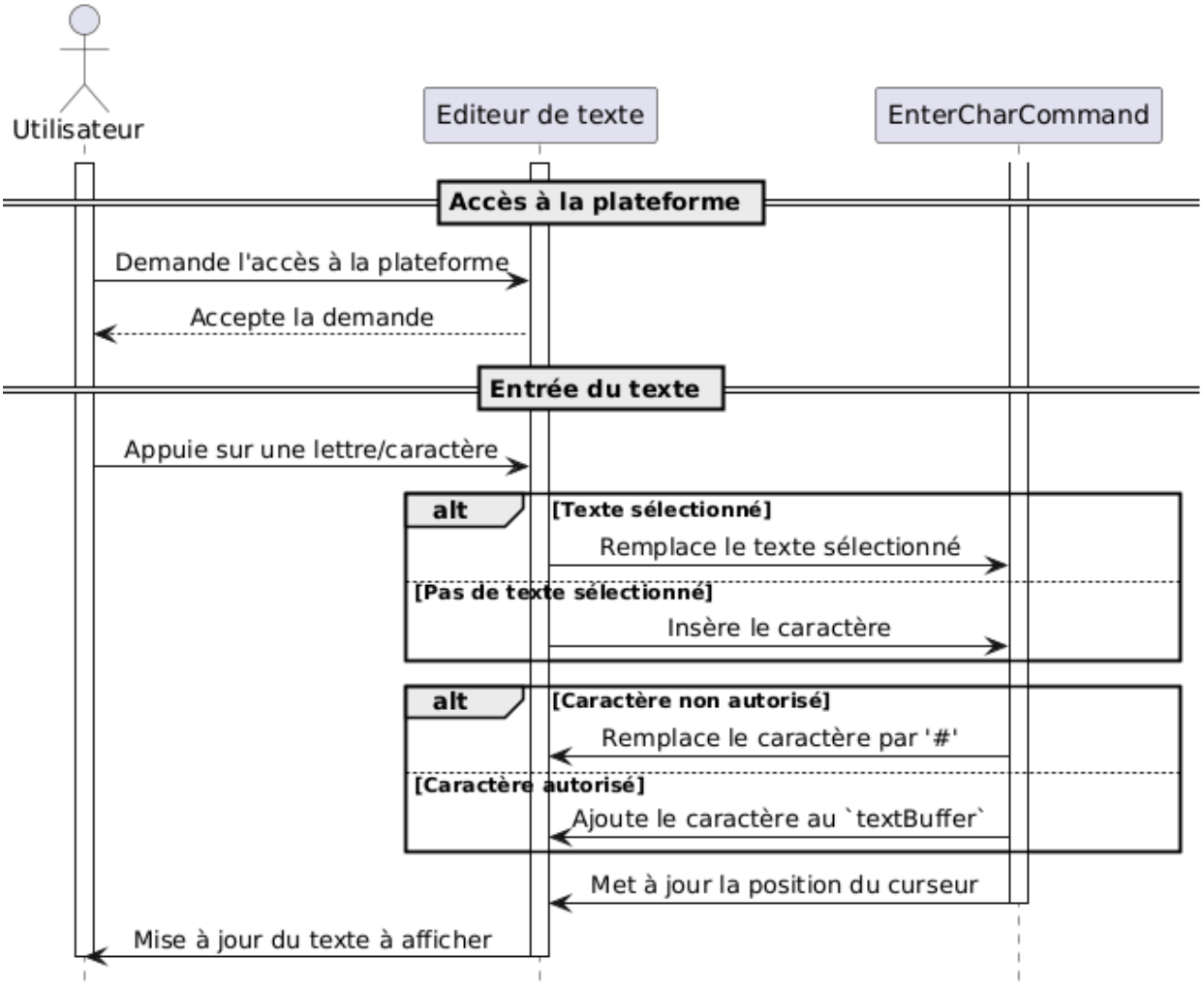

Figure 5: Diagramme de séquence pour entrer du texte

2.5. Implémentation du code

Conformément aux consignes et aux diagrammes que nous avons pu établir, nous avons réalisé la première version de l'éditeur de texte. Vous pourrez la trouver dans la branche nommée "V1" du dépôt Gitlab. Nous avons également réalisé des tests pour s'assurer du bon fonctionnement de l'éditeur de texte.

3. Version 2

Dans la seconde version, les fonctionnalités ajoutées par rapport à la première version sont les suivantes :

- enregistrement des actions de l'utilisateur (ctrl s)
- annulation de la dernière action de l'utilisateur (ctrl z)
- réajout de la dernière annulée action de l'utilisateur (ctrl y)
- écoute/arrêt de l'écoute des actions de l'utilisateur (ctrl m)
- rééxécution de la combinaison d'actions de l'utilisateur (alt m)

3.1. Diagramme des cas d'utilisation

Au vu des fonctionnalités supplémentaires que doit proposer la seconde version de l'éditeur de texte, voici un diagramme de cas d'utilisation envisageable. A noter que le diagramme d'utilisation de la première version vient s'y ajouter. Nous avons ajouté les nouveaux cas d'utilisation en bleu :

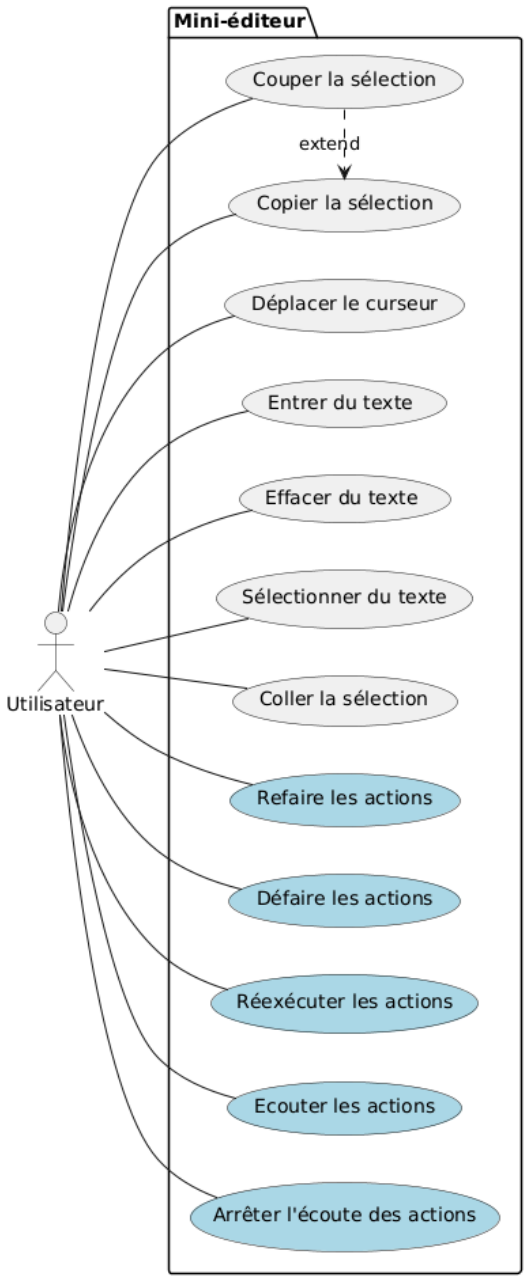

Figure 6: Diagramme de cas d'utilisation pour la seconde version.

3.2. Description des cas d'utilisation

Voici la description de chaque cas d'utilisation. A noter que la description des cas d'utilisation de la première version vient s'y ajouter. Nous avons ajouté les nouvelles description de cas d'utilisation en bleu :

Cas d'utilisation	Scénario nominal	Scénario alternatif	Scénario exception
Déplacer le curseur	a) L'utilisateur appuie sur les touches directionnelles. • gauche/droite pour aller à gauche/droite • bas/haut pour aller à début/à la fin b) Le curseur se déplace c) Sa nouvelle position est affichée.		a) La position cible du curseur est invalide (en dehors du texte) : le curseur s'arrête à la limite valide.
Entrer du texte	a) L'utilisateur saisit des caractères. b) Les caractères sont ajoutés à la position du curseur c) La zone de texte est mise à jour.	a) Le texte est ajouté à une sélection existante : remplace la sélection.	a) Caractère non autorisé (caractères spéciaux) : remplacement par dièse
Effacer du texte	a) L'utilisateur appuie sur la touche "Supprimer" ou "Backspace". b) Le caractère ou le texte sélectionné est supprimé c) a zone de texte est mise à jour.	a) Aucun texte à supprimer (curseur au début du texte).	
Sélectionner du texte	a) L'utilisateur utilise Shift + touches directionnelles. (gauche/droite : selectionne à gauche/droite) b) La portion du texte souhaitée est sélectionnée.	a) L'utilisateur peut sélectionner tout le texte avec Ctrl + A	a) Tentative de sélectionner au-delà des limites du texte : sélection seulement jusqu'aux limites autorisées
Copier la sélection	a) L'utilisateur sélectionne du texte. b) L'utilisateur appuie sur Ctrl+C.	a) Aucun texte sélectionné : vide le presse-papier et rien n'est copié.	a) Le presse-papier est plein.
Coller la sélection	a) L'utilisateur positionne le curseur à l'endroit souhaité. b) L'utilisateur appuie sur Ctrl+V. c) Le contenu du presse-papier est inséré à la position du curseur.	a) Le contenu du presse-papier est ajouté à une sélection existante : remplace la sélection.	a) Le presse-papier est vide : rien ne se passe
Couper la sélection	a) L'utilisateur sélectionne du texte. b) L'utilisateur appuie sur Ctrl+X. c) Le texte est supprimé de la zone de texte et copié dans le presse-papier.	a) Aucun texte sélectionné : vide le presse-papier et rien n'est copié.	
Zoomer/dézoomer la fenêtre	a) L'utilisateur sélectionne appuie sur Ctrl+ ou Ctrl-. b) Le zoom de la fenêtre est mis à jour. c) L'affichage de la fenêtre est mis à jour.		
Refaire les actions	a) L'utilisateur active la fonction de réexécution (Ctrl+Y). b) Le système réapplique l'action précédemment annulée.		a) Aucune action à refaire : ne rien faire, nous sommes revenus à l'état final.
Défaire les actions	a) L'utilisateur active la fonction d'annulation (Ctrl+Z). b) Le système revient à l'état précédent l'action la plus récente. c) Le fichier de script est mis à jour (ajout de la fonction d'annulation)		a) Il n'y a plus d'actions à annuler : ne rien faire, nous sommes rendus à l'état initial.
Réexécuter les actions	a) L'utilisateur appuie sur Alt+M. b) Le système exécute de manière séquentielle les actions enregistrées dans l'historique de la macro.		a) Aucune action dans l'historique : aucune action n'est exécutée.
Ecouter les actions	a) L'utilisateur appuie sur Ctrl+M pour démarrer l'écoute. b) Toutes les actions de l'utilisateur sont enregistrées dans l'historique de la macro jusqu'à ce que l'écoute soit arrêtée.	a) L'écoute démarre automatiquement en vidant l'historique précédent.	
Arrêter l'écoute des actions	a) L'utilisateur appuie de nouveau sur Ctrl+M. b) L'écoute s'arrête et les actions de l'utilisateur ne sont plus enregistrées.		

Remarque: A noter que si l'utilisateur défait (ctrl z) puis écrit du texte, alors la file des actions à refaire (ctrl y) est vidée.

3.3. Diagramme de classes

Dans cette seconde version du mini-éditeur de texte, nous avons repris la structure de la première version tout en l'enrichissant pour intégrer les nouvelles fonctionnalités demandées.

Voici les principales nouveautés :

- Nous avons introduit les classes `UndoCommand` et `RedoCommand`, qui permettent de revenir en arrière ou d'avancer dans l'historique des actions. Elles utilisent des snapshots (classe `Snapshot`) pour sauvegarder les états du texte et de la sélection pour chaque action effectuée par l'utilisateur.
- Pour gérer l'état d'enregistrement des actions de l'utilisateur, nous avons introduit un champ `macroRecord` dans la classe `TextEditor`, ainsi qu'un vecteur `macroHistory` qui stocke l'historique des macros est stocké pour pouvoir rejouer les macros ultérieurement. De plus, elle possède une référence à la commande en cours (pointeur de Command) pour déléguer les actions à une commande via la méthode executeCommand().

Pour ce faire, nous avons conservé la structure de base de la première version, c'est-à-dire le design pattern Command. Nous l'avons simplement étendu pour intégrer les commandes `Undo` et `Redo`. Cependant, nous y avons également incorporé le design pattern Memento pour stocker l'état du texte et des sélections pour permettre de pouvoir défaire/refaire comme demandé. Ce dernier est représenté par la classe `Snapshot`.

Voici le diagramme de classes en conséquence.

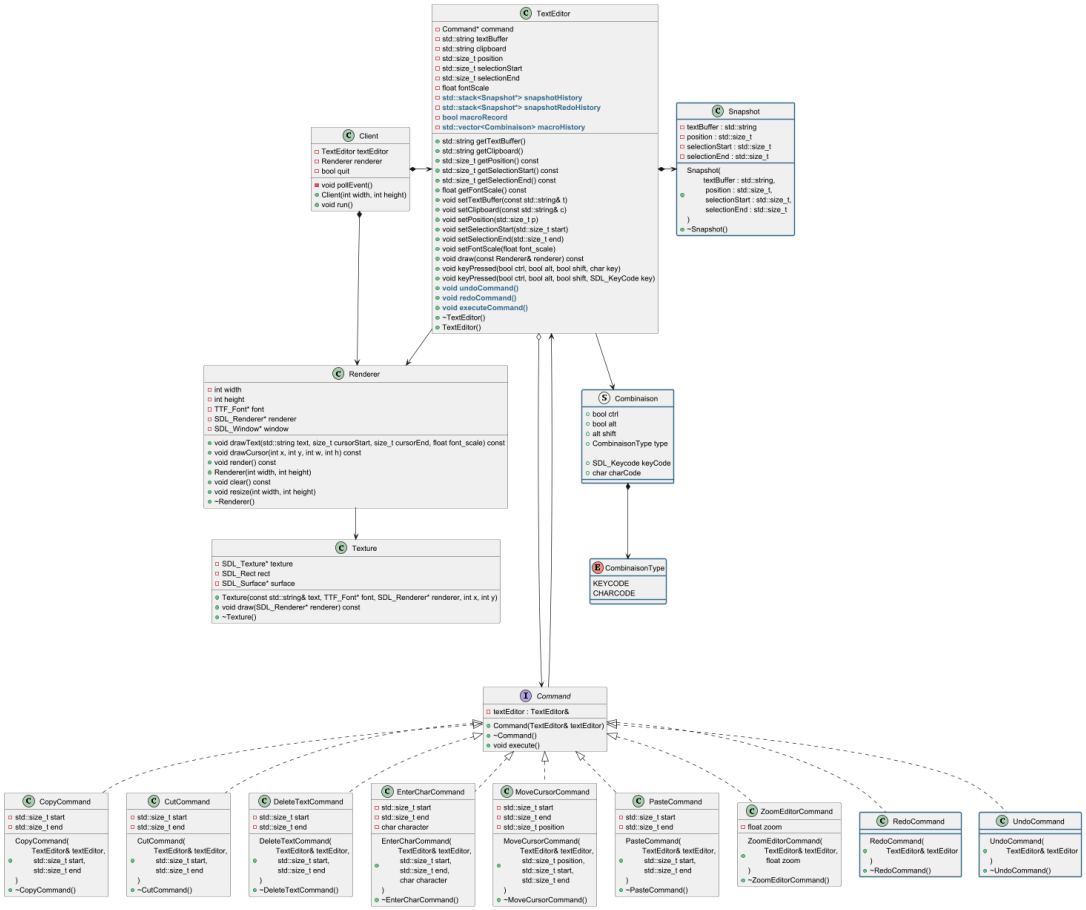

Figure 7: Diagramme de classe basé sur le Design Pattern Command pour la Version 2

3.4. Diagramme de séquence

Pour bien comprendre commment vont se produire les interactions avec les classes lors de l'utilisation de l'application, nous avons produit trois diagrammes de séquences.

3.4.1. Faire/défaire

Voici un scénario que nous avons inventé pour modéliser les actions de faire/défaire :

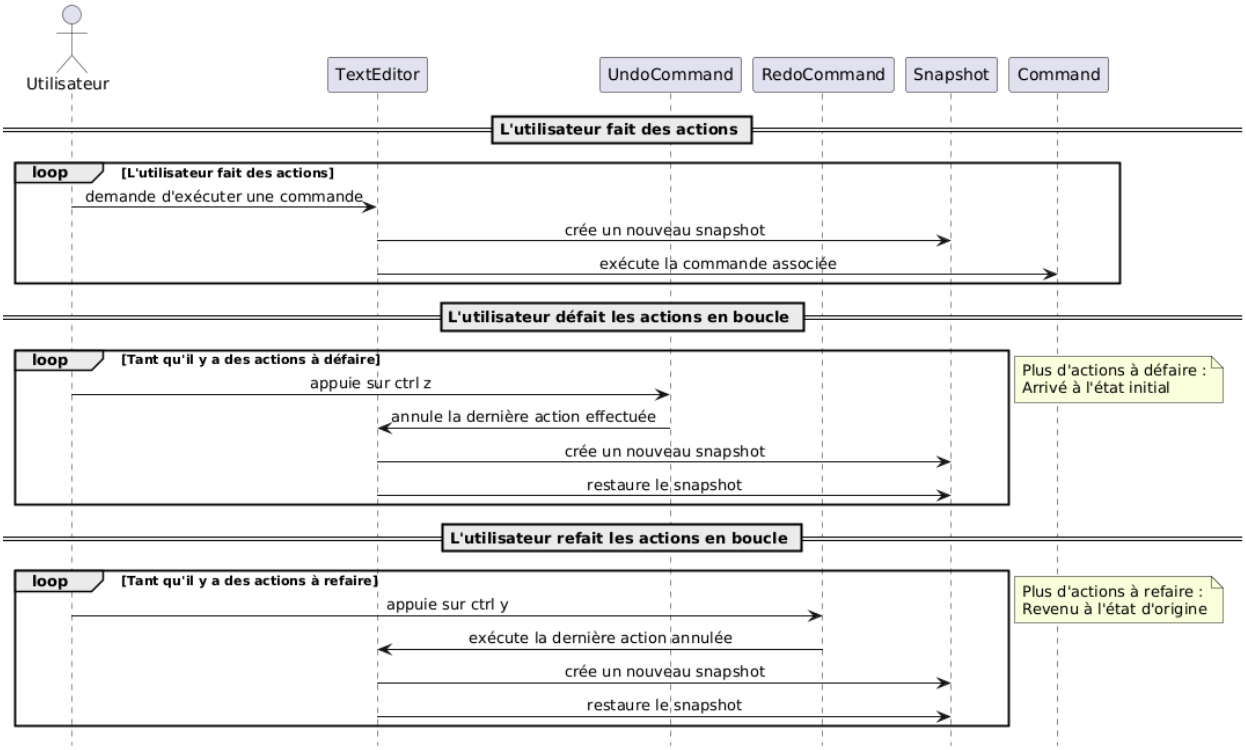

Figure 8: Diagramme de séquence pour faire et défaire les actions

3.4.2. Réxectuter les actions

Voici un scénario que nous avons inventé pour modéliser l'action de réxecution des actions :

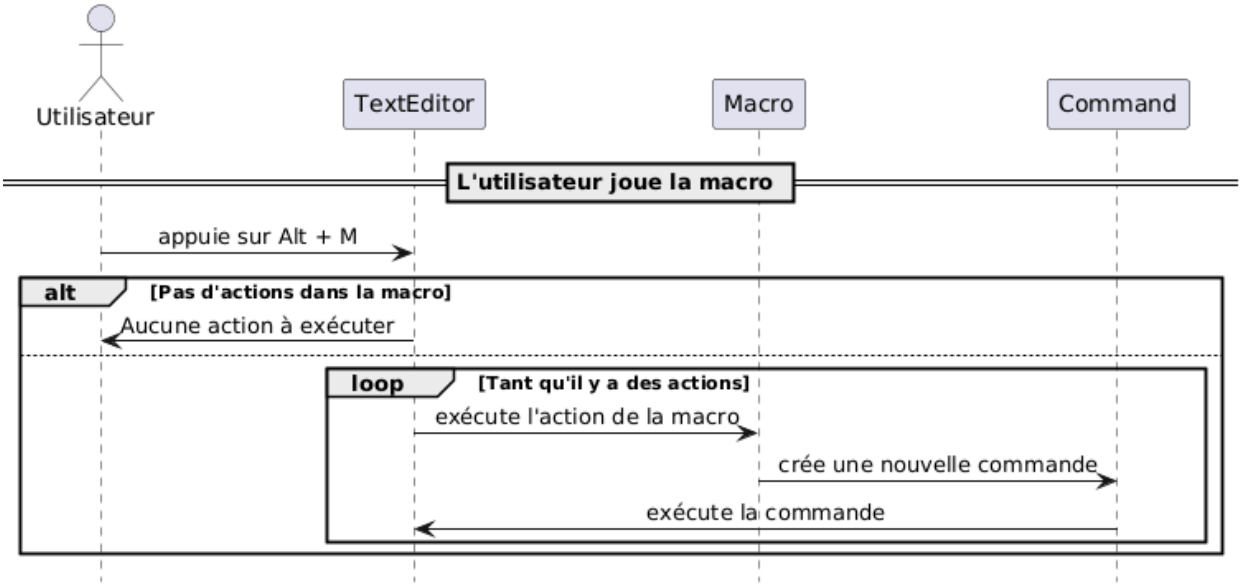

Figure 9: Diagramme de séquence pour réxécuter les actions

3.4.3. Ecouter/Arrêter l'écoute

Voici un scénario que nous avons inventé pour modéliser les actions d'écoute/arrêt de l'écoute :

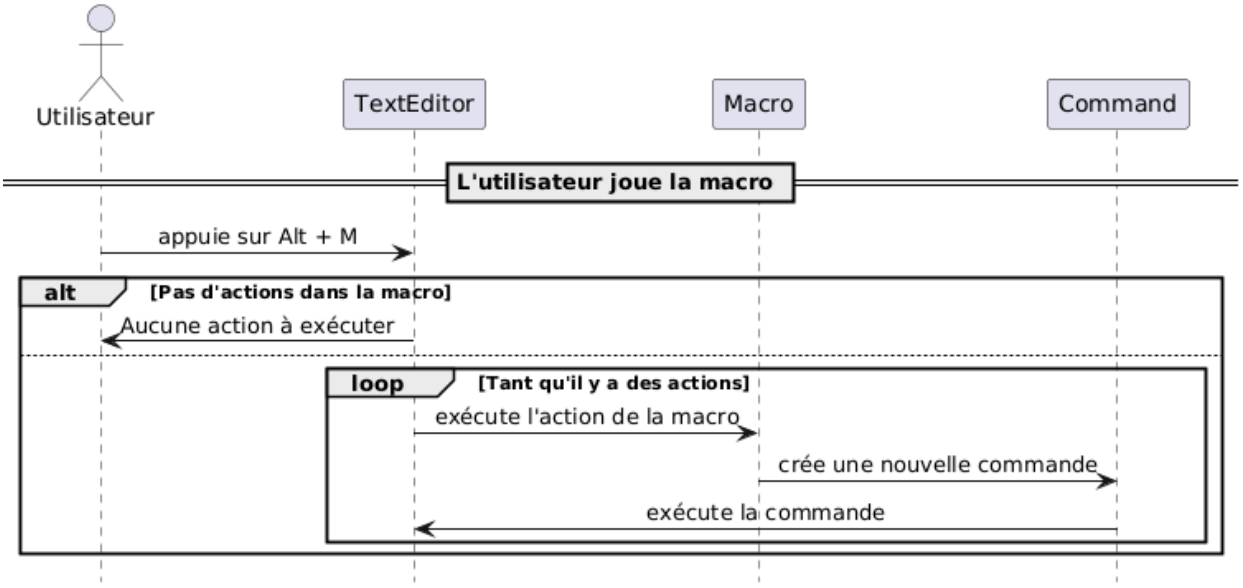

Figure 10: Diagramme de séquence pour écouter/arrêter l'écoute des actions

3.5. Implémentation du code

Conformément aux consignes et aux diagrammes que nous avons pu établir, nous avons réalisé la seconde version de l'éditeur de texte. Vous pourrez la trouver dans la branche nommée "V2" du dépôt Gitlab.

4. Conclusion

Pour conclure, ce projet d'éditeur de texte nous a permis de comprendre l'importance de concevoir un code dès la version initiale en prévision des évolutions futures, afin de garantir qu'il soit maintenable.

Grâce à cette approche en spirale, nous avons avoir un code factorisé et optimisé dès le début, mais facilitant tout de même les ajustements. Finalement, elle permet l'amélioration continue tout en conservant une version finale plus stable et robuste.