Outils et méthodes de développement

Compte rendu du TP2

Le 25 octobre 2022

Valentin BESNARD et Corentin DAVID

ESIR2 - IoT, sécurité et ville intelligente

Sommaire:

Introduction

L'objectif de ce TP est de concevoir et d'implémenter un éditeur de texte. Il sera modélisé à l'aide de diagrammes de séquences et diagrammes de classes UML. Ces diagrammes permettent de décrire les multiples scénarios possibles dans l'utilisation de l'éditeur. Pour cela nous procéderons dans un premier temps à une analyse fonctionnelle puis une analyse structurelle. L'implémentation se fera en deux versions : la première comportant les fonctionnalités basiques (*copier*(), *coller*(), *couper*()) et la deuxième étant plus complète avec l'ajout de la fonctionnalité *save*() et *backtrack*(). Enfin nous verrons la partie développement et l'explication de la manière dont l'éditeur de texte a été réalisé.

Analyse fonctionnelle

Notre éditeur de texte doit contenir plusieurs fonctionnalités. Ces dernières sont représentées sur le diagramme de cas d'utilisation. L'analyse fonctionnelle permet de représenter les acteurs qui interagissent avec le système. Ici notre système correspond à l'éditeur de texte tandis qu'il n'y a qu'un seul utilisateur. On définit alors les actions qu'il peut effectuer. Ces actions ont toutes le même niveau d'abstraction car ce sont des commandes intégrées à l'éditeur de texte. Les actions dont la couleur est représentée en vert correspondent à la V2 tandis que les autres sont intégrées dès la V1.

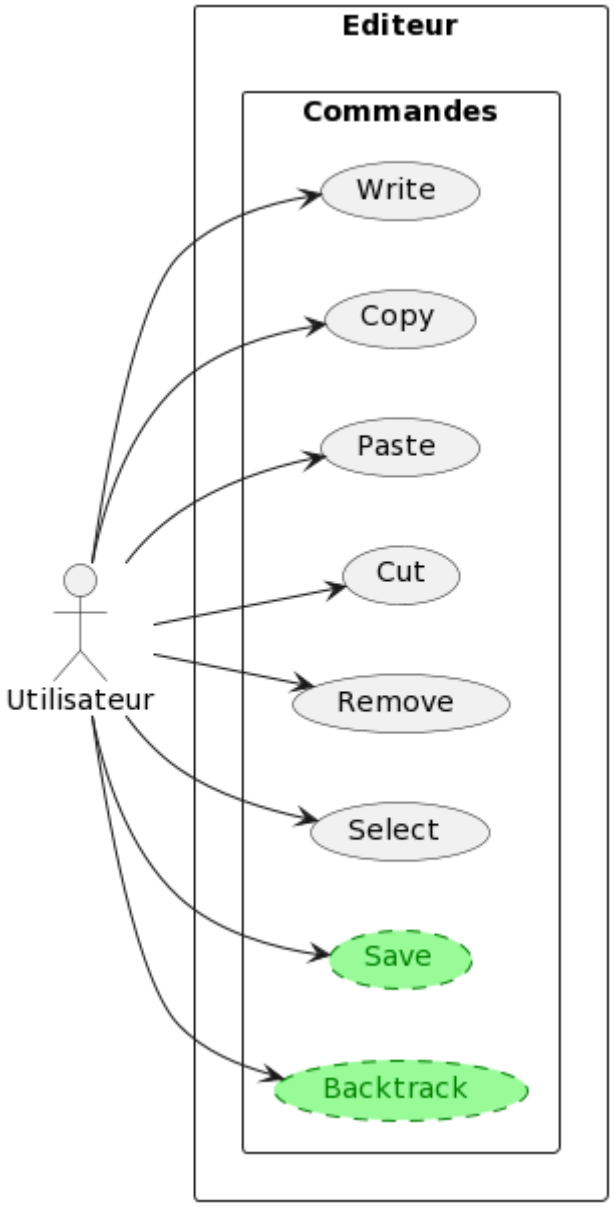

Figure 1 : diagramme des cas d'utilisation de l'éditeur de texte

Nous définissons maintenant les scénarios envisageables dans l'interaction entre l'utilisateur et l'éditeur de texte. Lorsqu'une commande est exécutée l'éditeur exécute diverses opérations en interne qui sont affichées sur le schéma. Le premier correspond à la liste des commandes exécutables dans la V1 alors que le deuxième se focalise sur celles de la V2.

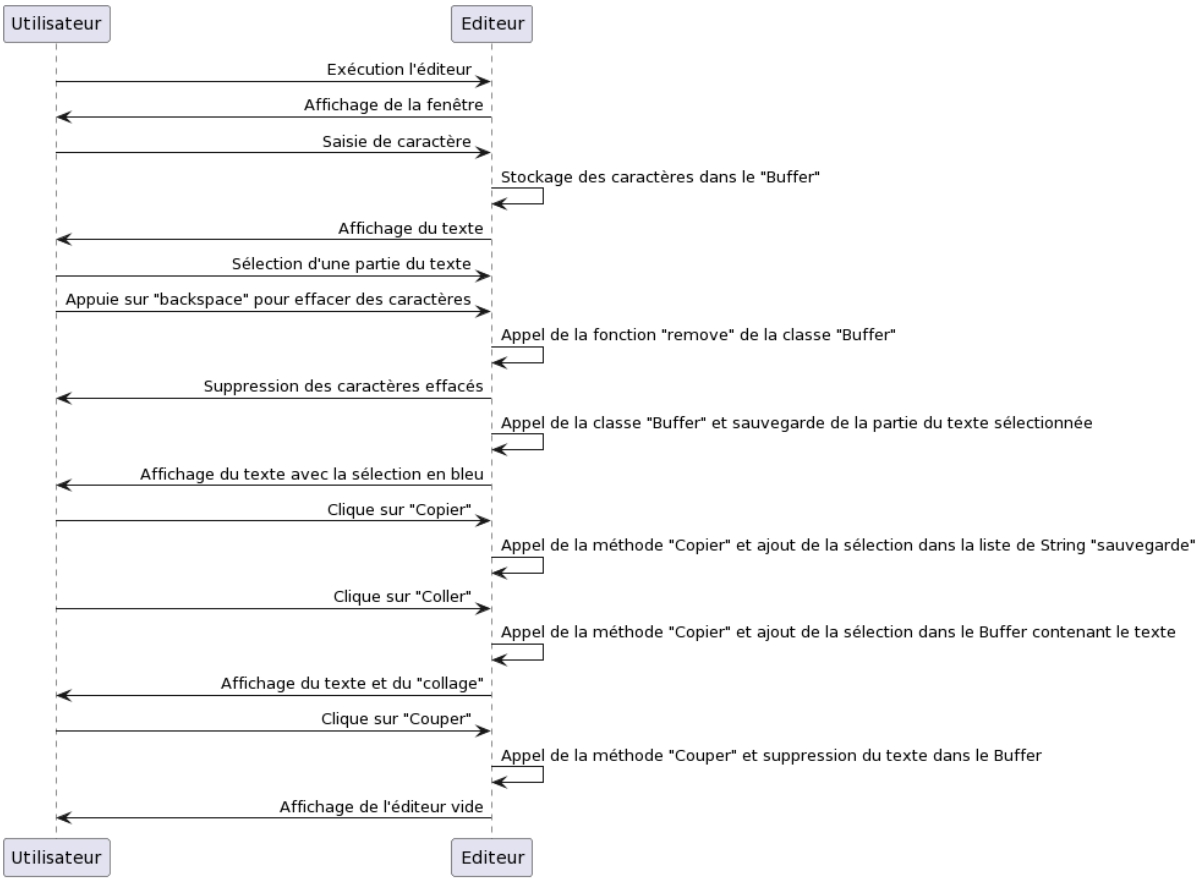

Figure 2 : diagramme de séquences de la V1

On s'intéresse maintenant aux fonctionnalités propres à la V2 qui sont *save*() pour sauvegarder le fichier et *backtrack*() pour faire un retour en arrière.

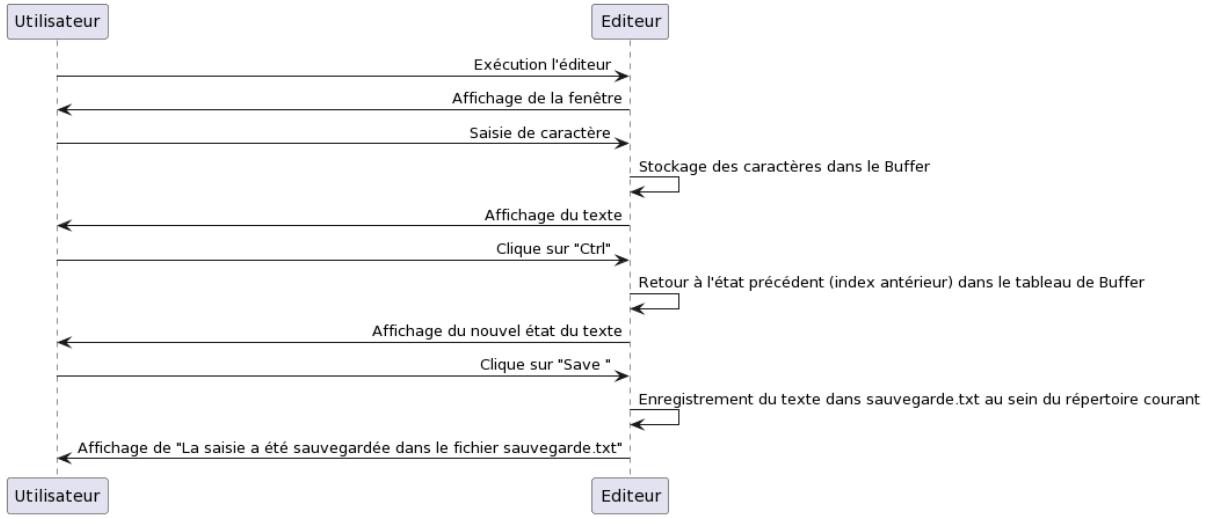

Figure 3 : diagramme de séquences de la V2

Analyse structurelle

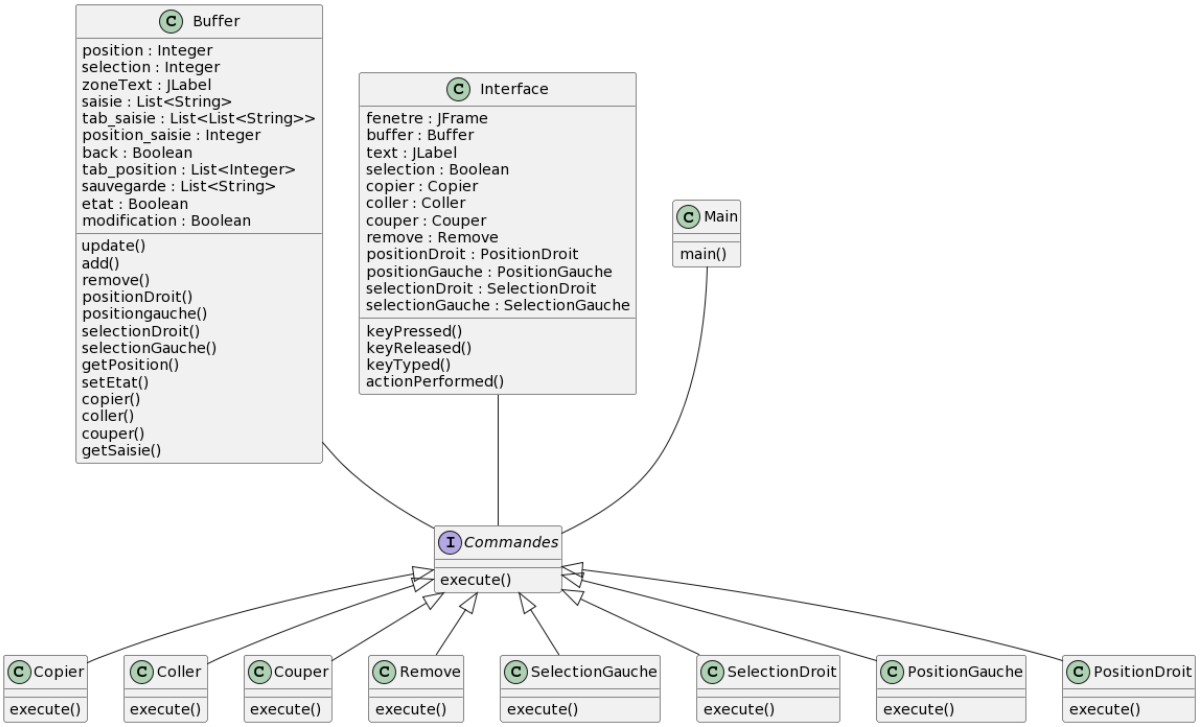

Figure 4 : diagramme de classes de la V1

Notre éditeur de texte comporte deux grandes classes : la classe **Interface** qui permet de gérer les intéractions au clavier de l'utilisateur et la classe **Buffer** qui gère les différentes fonctions qui permettent d'exécuter correctement les commandes. La classe **Main** permet de lancer l'éditeur de texte en créant une nouvelle instance de la classe **Interface**. De plus, chaque commande est déclarée comme une classe qui implémente l'interface **Commandes**. Cette conception permet de rendre plus facilement lisible l'ensemble des fonctionnalités existantes car elles sont toutes reliées via l'interface **Commandes**.

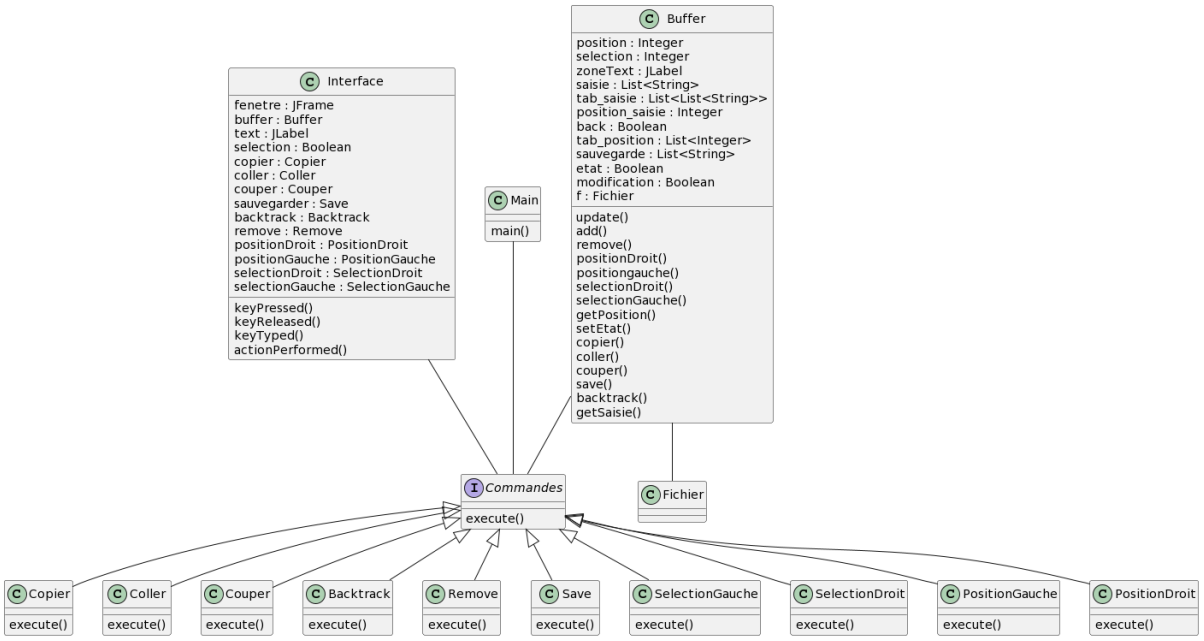

Figure 5 : diagramme de classes de la V2

Notre éditeur de texte comporte deux fonctionnalités supplémentaires dans cette deuxième version. Les classes **Save** et **Backtrack** sont ajoutées pour sauvegarder le texte et faire des “retour en arrière”. Cela implique l’ajout de nouvelles méthodes dans **Buffer** (*save*() et *backtrack*()) ainsi que des attributs de la classe **Interface** (sauvegarder et backtrack). Enfin la sauvegarde du texte implique la création d’un fichier et donc la classe Fichier qui en résulte. La méthode *save*() de la classe **Buffer** enregistre la saisie dans un fichier sauvegarde.txt au sein du répertoire courant.

Développement

Initialisation du projet:

Pour réaliser cet éditeur de texte, nous avons utilisé **Java**. La base du projet prend racine avec la classe **JFrame**; dans notre **interface** nous allons l’utiliser pour initialiser toutes nos composantes, les associer entre elles, tout en récupérant les entrées du clavier; pour assurer une interaction constante avec l’utilisateur. La fonction première de cette classe est d’assurer une **IHM**.

Les trois composantes principales de notre éditeur sont, la fenêtre engendrée par **JFrame**, la zone de texte défini par le buffer de la classe **Buffer** et le texte représenté par un **JLabel**. Leur implémentation se fait en cascade, le texte est placé dans le buffer qui lui même est mis dans la fenêtre. Pour le design nous nous sommes basé sur l’éditeur des fichiers txt classique:

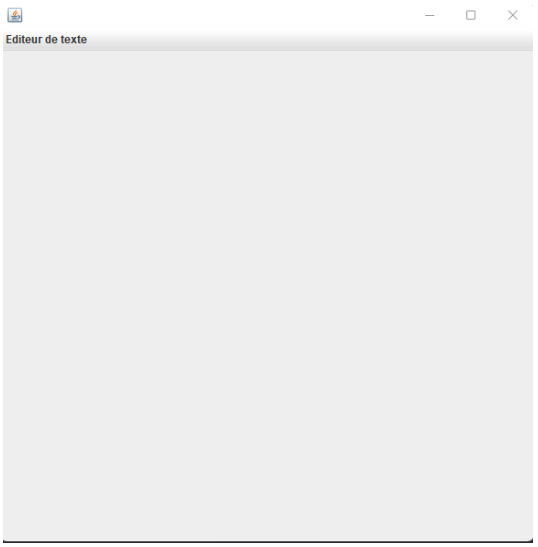

Figure 6 : capture d'écran de l'éditeur de texte

Maintenant nous allons voir comment nous avons ajouté la possibilité de saisir du texte dans notre fenêtre. Pour cela nous allons nous concentrer sur le buffer, car c'est là que la majeure partie du travail se fait. Celui-ci va contenir plusieurs attributs: la **zone de texte** dans laquelle nous allons définir le texte à afficher dans la fenêtre, un tableau de chaîne de caractères représentant tous les caractères de la **saisie** de l'utilisateur, un entier représentant la **position** du curseur courant dans la saisie (utile pour savoir où nous allons saisir les prochains caractères), un entier **sélection** pour délimiter la zone du texte que nous avons placé dans notre sélection, un autre tableau de chaîne de caractères qui va prendre tous les caractères qui vont être **sauvegardés** (utile pour la commande copier et couper) et pour finir un booléen qui va nous dire si l'éditeur est dans un **état** de sélection.

Ajout du texte:

Pour éditer le texte nous utilisons le clavier, grâce à la classe **KeyListener**, qui va récupérer les informations sur la touche activée, si elle ne correspond pas à une touche utilisée pour une de nos commandes (on verra en détail plus tard), alors nous l'ajoutons dans la saisie du buffer grâce à la fonction *add()*. Celle-ci va récupérer la chaîne de caractère saisie et le pointeur courant de notre buffer, puis va ajouter cette chaîne dans la liste à la position du curseur. Nous avons aussi mis en place un saut de ligne manuel (comme dans un éditeur classique ou le saut automatique n'existe pas), si on appuie sur la touche de retour chariot nous ajoutons la balise HTML **
** dans notre saisie.

Pour maintenir à jour constamment notre texte, nous avons créé une fonction update, qui va maintenir à jour les informations. Dans celle-ci on va découper la saisie en trois parties: le début du texte jusqu'au pointeur, la zone de sélection (qui va du pointeur à la sélection) et le reste du texte. Nous opérons ainsi pour isoler la partie sélectionnée et la mettre en avant en changeant la couleur en bleu. Après ça on va set le texte dans le JLabel

en y ajoutant du HTML, pour ajouter les différentes couleurs; ce qui nous donne le résultat suivant:

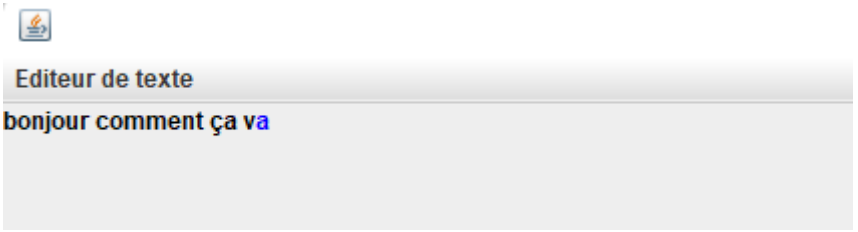

Figure 7: capture d'écran de l'éditeur de texte après la saisie d'une phrase

Pour déplacer le curseur et donc pouvoir éditer n'importe ou dans la saisie, il suffit juste d'utiliser les flèches directionnels droite et gauche. Cette action va modifier la valeur du pointeur courant.

La sélection:

Nous allons nous pencher sur la manière pour sélectionner une zone de texte dans l'éditeur. Pour activer cette fonctionnalité, il faut presser une fois la touche MAJ (si on presse une nouvelle fois elle est désactivée). La console indique si nous l'avons activée, si oui nous pouvons gérer sa taille à l'aide des flèches directionnels droite et gauche (en sachant que la flèche de droite permet d'étendre la sélection et celle de gauche de la réduire). La zone choisie sera donc surlignée en bleu:

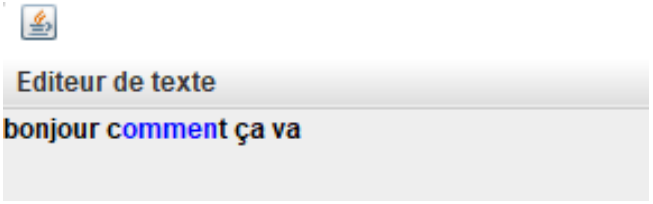

Figure 8 : capture d'écran de l'éditeur de texte en mode sélection

Les Commandes:

Nous allons voir maintenant les différentes commandes que nous pouvons utiliser dans cet éditeur de texte. Dans un premier temps nous avons créé interface **Commandes**, qui va servir pour l'héritage de toutes les commandes, avec pour seule composante une fonction *execute()* qui sera implémentée dans les classes filles. Chacune des classes filles aura comme seul attribut le buffer courant, et sa fonction execute appellera la méthode liée à la classe dans la classe buffer. C'est donc dans la classe buffer qu'on va implémenter les fonctionnalitées des commandes:

Remove: cette commande va permettre de supprimer une zone de texte de la saisie (ça peut être le caractère situé au niveau du curseur, ou toute la zone sélectionnée). Pour la réaliser on crée une nouvelle liste qui va prendre tous les caractères de la saisie sauf ceux qui sont compris dans la zone de sélection, ensuite on remplace la saisie par cette nouvelle liste.

PositionDroit: c'est la commande qui gère le déplacement du curseur vers la droite, en incrémentant position, si il ne dépasse pas la taille de la saisie.

PositionGauche: c'est la commande qui gère le déplacement du curseur vers la gauche, en décrémentant position, si il n'est pas null.

SelectionDroit: uniquement en mode sélection, elle permet d'agrandir la zone de sélection, en incrémentant sélection, si il ne dépasse pas la taille de la saisie.

SelectionGauche: uniquement en mode sélection, elle permet de rétrécir la zone de sélection, en décrémentant sélection, si il n'est pas plus petit que position.

Copier: cette commande va sauvegarder la zone de sélection dans la liste sauvegarde, grâce à une boucle sur une zone de la saisie délimitée par position et sélection.

Coller: c'est la commande qui va ajouter le contenu de la liste sauvegarde dans la saisie, au niveau du curseur; pour ça on crée une chaîne de caractères contenant toute la sauvegarde puis on l'ajoute à l'aide de la fonction *add()*.

Couper: cette commande va sauvegarder la zone de sélection dans la sauvegarde à l'image de copier, sauf que cette fois on va supprimer de la saisie la portion de texte que nous copions grâce à la fonction *remove()*.

v2_Save: dans la deuxième version on va pouvoir sauvegarder le texte dans un fichier externe dans la racine du projet. Pour cela on a créé une nouvelle classe **Fichier** qui va utiliser la bibliothèque **file** de java; cette classe aura comme seul attribut le contenu du fichier et dans le constructeur nous allons créer le fichier **sauvegarde.txt**, puis on va y ajouter le contenu. Dans la fonction du buffer on va juste initialiser un nouveau fichier avec comme contenu la saisie convertit en une chaîne de caractères.

v2_Backtrack: la deuxième commande cette version ressemble au CTRL+Z qu'on peut trouver dans un éditeur de texte classique. Pour la réaliser nous allons initialiser deux nouvelles listes représentant toutes les saisies et les pointeurs enregistrées depuis le lancement du programme. De plus, nous allons utiliser un **position_saisie** sur ces listes, ce qui va nous permettre de savoir où nous nous situons dans le retour en arrière. Donc si on active cette commande on remplace la saisie et la position par leur version précédente situées à l'index moins un de la position_saisie sur la liste des saisies et des pointeurs, à la fin on oublie pas de décrémenter position_saisie.

Activation des commandes:

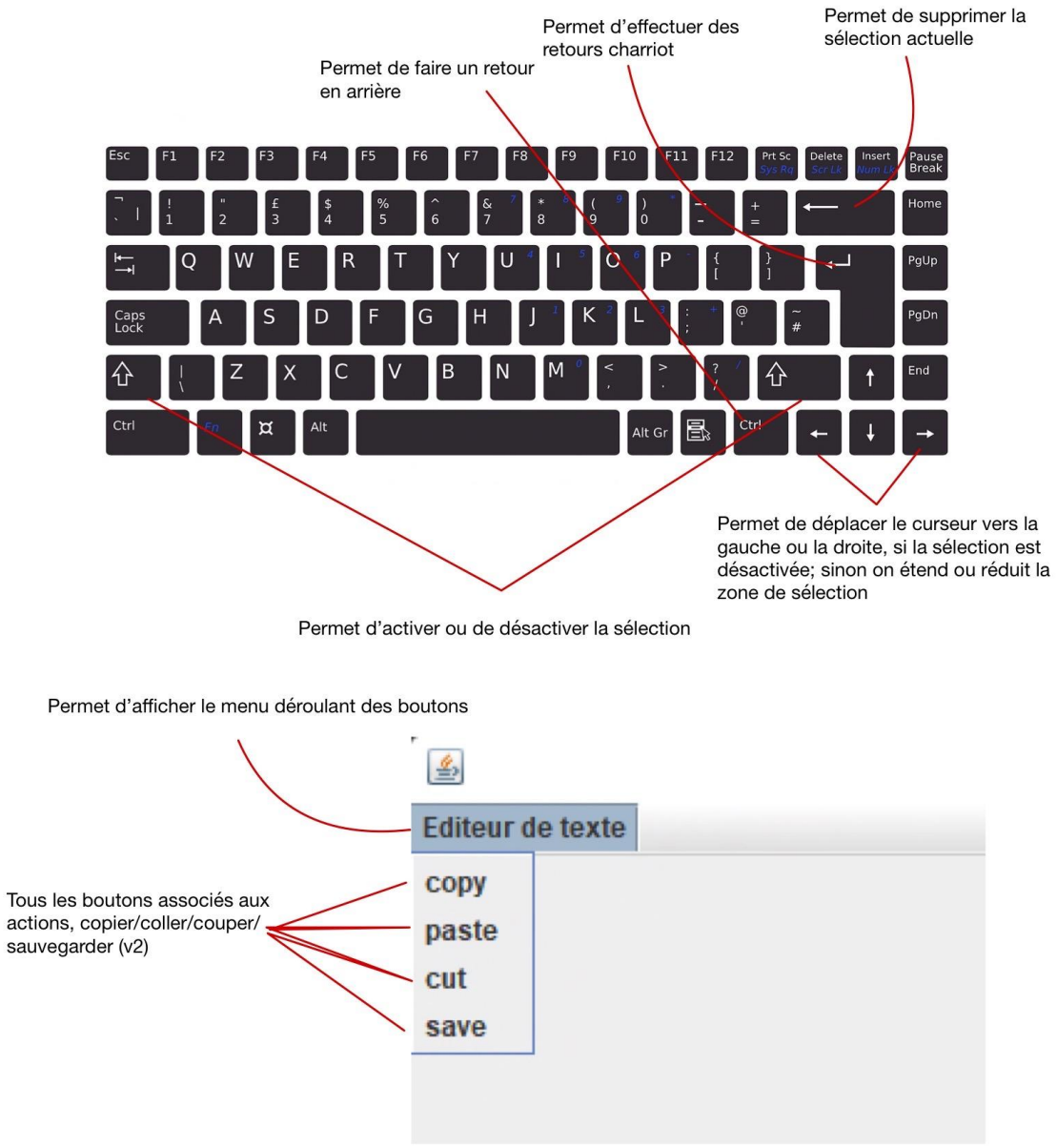

Figure 9 : Schéma des moyens d'activation des commandes

Conclusion:

Dans ce deuxième TP, nous devions réaliser un éditeur de texte par nous même, de la conception à la réalisation. Nous avons donc dans un premier temps réalisé les différentes analyses pour mener à bien ce projet. Et pour finir nous avons implémenté notre solution de manière structurée.