Projet de Développement Orienté Objet 2
L'éditeur de corpus parallèles

David Guennec

May 16, 2025

1 Contexte et Objectifs

Dan Mc Gold, PDG de la société UnknownVariable, a récemment mangé avec un excellent ami travaillant sur l'analyse de langues. Ce dernier, Jim, lui a fait par d'un problème auquel il fait face dans son travail : il doit constamment passer des heures à comparer les versions d'un même texte en plusieurs langues et ce sans trop d'outils. Dan, qui n'a jamais fait un git diff de sa vie, décide de lui donner un coup de main et vous contacte pour créer un logiciel d'analyse qui puisse être utile à son ami.

Dan et Jim ont dessiné le prototype sur la nappe de table du restaurant, que voici :

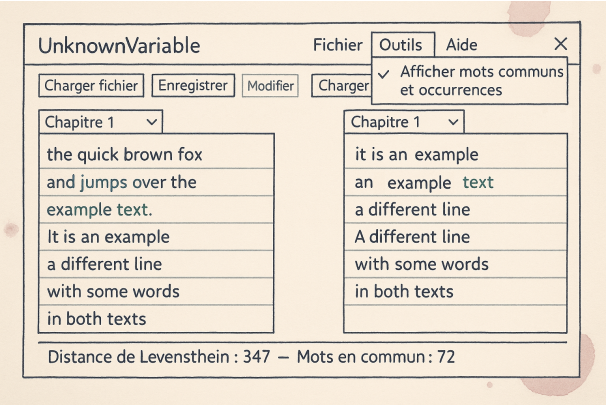

Figure 1: L'interface graphique de l'application demandée. Attention ! Le vrai texte sera beaucoup plus long que ce qui est affiché dans cet exemple.

Le but de ce logiciel est de présenter deux textes parallèles dans une interface découpée en deux grand panneaux, à la manière d'un git diff. Les textes viennent en chapitres identifiés par un chiffre romain (cf. données fournies avec le projet).

Un sélecteur de chapitre, au dessus du texte, permet de décider quel chapitre afficher. Vous avez pour ce faire deux options :

- Soit ne charger dans le panneau de texte que le texte correspondant au chapitre demandé.
- Soit charger tout le texte dans le panneau et défiler vers le chapitre demandée, le reste étant disponible si l'utilisateur fait défiler le texte vers le haut ou vers le bas dans l'interface.

Les textes dans l'interface doivent être pouvoir être sélectionnés (pour, par exemple, copier le contenu) mais ne doivent pas être modifiables par défaut. Pour les modifier, il faut cliquer sur un bouton qui se trouvera au dessus de chacun des 2 panneaux de texte.

L'interface permet bien sûr de charger les fichiers que l'on désire. Le chargement se fait cliquant sur un bouton situé au dessus de chaque panneau texte, puis en sélectionnant un fichier depuis le menu qui apparaît alors. Les fichiers peuvent aussi être sauvegardés via un autre bouton.

En bas de l'écran, un champ texte indique la distance de Levenstein (cf. `https://fr.wikipedia.org/wiki/Distance_de_Levenshtein`) entre les textes affichés dans les deux panneaux ainsi que le nombre de mots en commun. Jim a demandé que si possible, les mots qui apparaissent dans les deux textes soient coloriés dans l'interface. Un choix dans la barre de menu permet d'afficher ces mots et le nombre d'occurrences dans chaque texte.

Note : Pour l'ensemble de ce travail, vous pouvez vous inspirer de ce qui existe, en particulier pour Levenstein. Ne vous embêtez pas à tout réécrire par vous-mêmes. Par contre, pas de plagiat sur vos camarades.

2 Instructions

2.1 Travail demandé

Implémentez l'application décrite ci-dessus en JavaFX. Pour vous aider dans cette tâche, Jim vous fournit un petit corpus de test avec un même texte traduit dans plusieurs langues que vous pourrez utiliser pour vos tests. Assurez-vous du bon fonctionnement de l'application en la testant rigoureusement. Surtout, mettez au point une stratégie de test cohérente avant de commencer et décrivez-la. Il vous sera demandé de la détailler dans votre rendu.

Notez également qu'il est **crucial** de concevoir l'application correctement. Faites en donc le design en UML avant de commencer. L'architecture et sa justification vous seront (bien évidemment) demandées dans le rendu.

2.2 Bonus

Vous voulez une chance de maximiser votre note en gagnant des points bonus ? Et bien la voici :

En fait, Jim a besoin de comparer plus de 2 version du texte en même temps. Implémentez un mécanisme permettant d'ajouter ou de retirer des panneaux de texte. Chaque panneau devra fournir les boutons "<<" et ">>" pour ajuster le curseur ainsi que ses propres boutons charger, enregistrer et modifier.

3 Contraintes

Quelques précisions supplémentaires :

- **Vous n'êtes pas obligés d'implémenter l'interface en FXML**, vous pouvez le faire nativement en Java. Il n'y a aucune différence de notation entre une interface en Java natif et FXML.
- Vous pouvez modifier l'archive du corpus comme vous le voulez, en ajoutant des textes par exemple. Cependant, l'application doit fonctionner avec les textes fournis sans modification aucune. L'enseignant testera d'ailleurs votre application avec sa propre version des données (et pas la votre).
- Vous devez respecter à la lettre les instructions de rendu. Le correcteur sera sans pitié sur ce point. En particulier, n'oubliez pas que le correcteur ne souhaite pas passer 30 min à installer ou configurer des outils avant de noter chaque projet. Votre rendu devra donc compiler et fonctionner comme demandé lors de la correction.
- Les données du corpus doivent pouvoir être chargées de n'importe où sur le système de fichiers. Vous ne savez pas où le correcteur a mis ses données de test, alors ne présumez de rien (vous ne savez même pas si les mêmes données seront utilisées ou non).
- La documentation que l'on vous demande est très courte (cf. section suivante). Cela signifie que l'on attend de la qualité dans le peu d'informations qui sont demandées : les phrases inutiles (i.e. smalltalk) ne rapporteront aucun point.
- Si vous avez le moindre problème de compréhension ou la moindre incertitude sur le sujet, posez une question à votre encadrant !

4 Rendus

Le projet est à faire **en monome**. Il sera rendu sur moodle au plus tard le vendredi **06/06/2024 à 23h55**. Votre rendu sera constitué d'**une archive zip ou tar.gz à votre nom** et contiendra votre code source (Java, FXML,

CSS, etc.), votre README.md et le diagramme de classe au format PNG.

La documentation sera à rendre dans le répertoire de projet sous la forme d'un fichier README.md ; inutile de faire long, le fichier doit faire l'équivalent d'une page de rapport en police 11 points tout au plus, diagrammes UML exclus. La documentation du projet contiendra les informations suivantes :

1. Votre nom et prénom
2. Une section donnant toutes les instructions d'installation et de lancement nécessaires pour faire tourner votre projet. Attention, n'oubliez pas cette section ! Son absence impliquera **une forte pénalité (jusqu'à 5 points selon la gravité de l'oubli)**. Elle devra impérativement donner des informations précises permettant au correcteur de faire 2 choses :

   Compilation : Les instructions de compilation doivent impérativement être données pour une utilisation en console (i.e. la commande **javac** à utiliser) et sans autre prérequis que le JDK tel que précisé dans le document d'installation sur moodle ("Using java"). Donc pas d'instruction demandant de demande de télécharger IntelliJ et de cliquer sur le gros bouton avec un triangle et pas de Maven. Les commandes expliqueront comment compiler depuis la racine de votre projet ou dans quel répertoire aller, quelles commandes lancer ensuite, etc. Les instructions doivent à minima être données pour une machine Linux.

   Exécution : Les instructions de lancement devront montrer comment l'on fait tourner votre programme ; c'est à dire avec quelle(s) commande(s) et à partir de quel répertoire.
3. Un paragraphe présentant votre modélisation et le diagramme de classe de l'application (6 - 10 lignes). Surtout, si vous avez utilisé un design pattern (conseillé), précisez et expliquez-le !!!
4. Un paragraphe présentant votre stratégie de test, les tests réalisés ainsi que l'état actuel de votre implémentation (6 - 10 lignes).

Le code fourni dans l'archive doit contenir tous les fichiers sources Java dans la bonne arborescence (dans les bons dossiers si vous faites des paquets). Vos éventuelles ressources de test ou corpus étendus peuvent également être envoyés avec votre rendu si vous le souhaitez. Notez qu'un logiciel de détection de plagiat sera utilisé pour vérifier tous les codes sources.