Historique d’actions dans un éditeur de texte(Undo/Redo)

Introduction

Dans le monde du développement la création d'un **éditeur de texte** est une excellente opportunité pour comprendre la gestion des entrées utilisateur, le traitement des données et l'implémentation des fonctionnalités avancées comme **l'historique des actions**(undo/redo).

L'objectif de ce projet est de concevoir un **éditeur de texte interactif,** permettant aux utilisateurs d'éditer, enregistrer et restaurer leurs modifications de manière fluide, tout en utilisant Qt pour assurer une **interface graphique efficace et moderne**.

Les technologies utilisées

Pour réaliser ce projet, plusieurs outils et bibliothèques ont été choisis:

💻**Qt Framework** (C++):

- Permet la création d'interfaces graphiques performantes.
- Utilisation de **Qt widgets** pour gérer l'éditeur de texte.

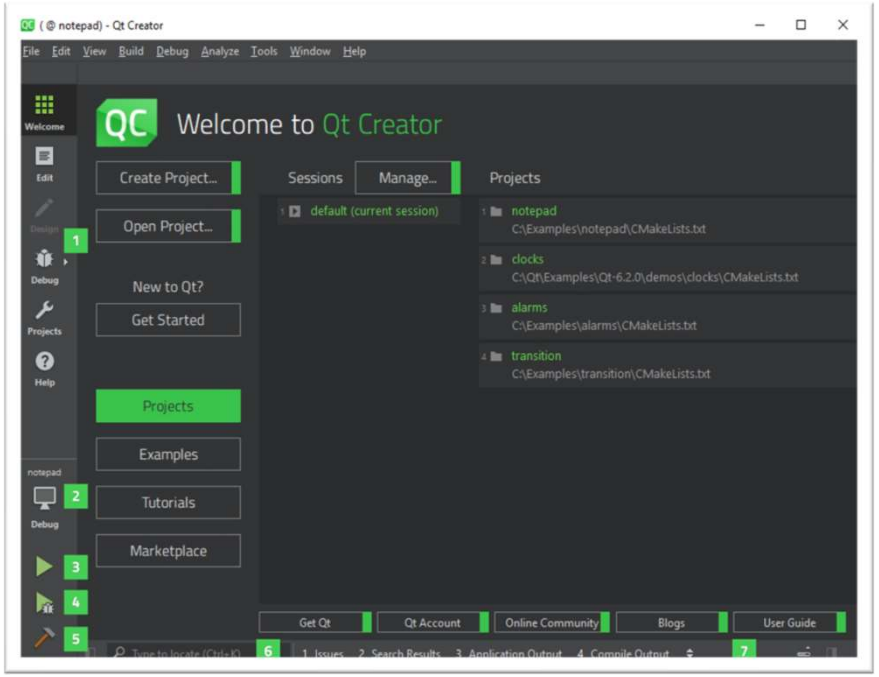

Figure 1: Interface Qt creator

Les technologies utilisées

Composants clés:

- QTextEdit : composant permettant l'édition de texte.
- QUndoStack et QUndoCommand : structures dédiées a l'historique des modifications.
- QFileDialog : gestion des fichiers a ouvrir et enregistrer.
- Inno setup : outil gratuit permettant de créer des installateurs pour les logiciels Windows.
- Flacticon.com : plateforme proposant une vaste collection d’icones et stickers vectoriels gratuits et premium.
- App.logomaker.com : plateforme de conception du logo de l’exécutable.

Architecture du projet

L'éditeur de texte repose sur une architecture modulaire organisée autour de plusieurs composants:

📌**organisation globale**:

- **Interface utilisateur** : gère l'affichage et les interactions (écriture, annulation/restauration, sauvegarde).
- **Moteur de gestion des actions** : permet de conserver et d'appliquer les modifications.
- **Système de stockage** : assure la sauvegarde et l'ouverture de fichiers.

Gestion de l'historique des actions

La **gestion de l'historique** est essentielle pour améliorer l'expérience utilisateur. Voici comment elle est implémentée :

Undo et Redo avec QUndoStack et QUndoCommand :

QUndoStack conserve les modifications sous formes de commandes.

QUndoCommand permet d'annuler et restaurer chaque action individuellement.

Chaque modification(ajout, suppression de texte) est encapsulée dans une commande spécifique.

Gestion de l'historique des actions

📌 **Principe de fonctionnement** (exemple) :

1. **L'utilisateur saisit du texte** enregistre dans QTextEdit.
2. **Chaque modification est stockée** sous forme de commande via QUndoStack.
3. **Annulation ou rétablissement d'actions** avec QUndoCommand.
4. **Sauvegarde et récupération des fichiers** avec QFileDialog.

Interface utilisateur

Une bonne **interface utilisateur** améliore l'efficacité du logiciel.

📌**Eléments clés du design** :

Fenêtre principale avec une zone de texte (QTextEdit).

Boutons undo/redo accessibles pour gérer l'historique.

Menu pour ouvrir/enregistrer un fichier(QFileDialog).

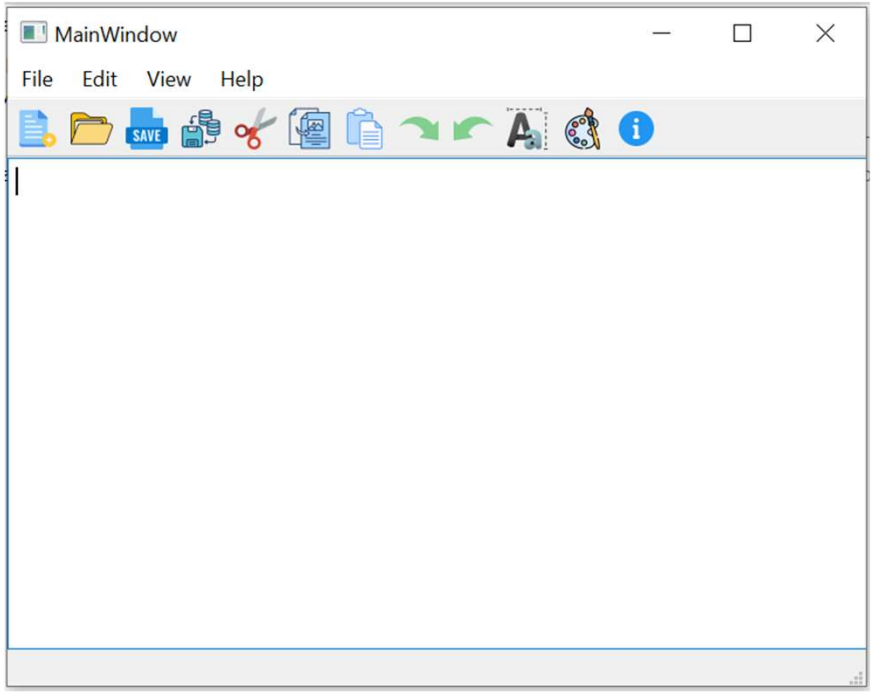

Figure 2: interface utilisateur

Difficultés rencontrées et solutions

Les difficultés :

- Consommation significative de la mémoire en particulier avec de gros fichiers.
- Synchronisation entre l'interface et l'historique.
- Maintien de la mise en forme du texte après annulation / rétablissement.

Les solutions :

- Limiter la taille de l'historique en supprimant les anciennes actions après un certain seuil.
- Mettre à jour l'état du texte après chaque modification et gérer les évènements avec QTextCursor
- Utiliser QTextDocument et QTextBlock pour stocker les styles et les appliquer correctement.

Démo et résultats

Dans cette section, nous allons donner les différents avantages de cette approche et montrer le fonctionnement de notre application à travers quelques images

🎯**Avantages de cette approche** :

✓Gestion flexible des actions.

✓Reduction des erreurs de manipulation.

✓Expérience fluide et intuitive.

Perspectives d’amélioration

- Intégration du format **markdown(**langage de balisage léger utilisé pour formater du texte de manière simple et lisible**)** ou de l’export **PDF**.
- Ajout d’une **fonction de recherche et remplacement dans le texte**.
- Possibilité de personnalisation avancée de l’interface.

Conclusion

Ce projet démontre la **puissance de Qt** pour créer des applications interactives, tout en offrant un système **d'historique des actions efficace.**