Option LOA – TP2

Jean-Yves Didier

Objectifs :
- Savoir développer un projet avec le *framework* Qt ;
- Approfondir ses connaissances et expérimenter C++.

Pré-requis :
- Connaître au moins un langage orienté-objets ;
- Savoir écrire une classe C++ ;
- Connaissance des mécanismes de Qt.

Matériel nécessaire :
- Qt $>$ 4.6 ;
- QtCreator préférable mais pas imposé.

Dans le cadre de ce TP, nous allons poursuivre l'implémentation de l'éditeur de texte commencé précédemment.

Préparation de la séance

- Le TP étant noté, il faudra lancer la session appropriée ;
- Vous êtes autorisés à accéder aux documentations suivantes :
  - Documentation Qt : `http://doc.qt.io` ;
  - Documentation C++ :
    - `http://www.cplusplus.com/reference/` ;
    - `http://www.cppreference.com/`.
  - Nomenclature des icônes d'actions prédéfinies : `https://standards.freedesktop.org/icon-theme-spec/`.
- Pour mener à bien votre séance de TP, vous utiliserez une archive contenant les sources départ localisée en `/pub/sujets/qtextpad.zip`. Cette archive contient l'éditeur tel qu'il aurait du être implémenté à la fin de la séance de TP précédente.

1 Régler la police de caractère

La première fonctionnalité à implémenter est relativement simple : vous devrez rajouter à l'application un menu « Préférences »et une option « Régler la police ». Cette dernière lancera une boîte de dialogue particulière permettant de choisir une police de caractères et de modifier la police de la zone d'édition du texte. La police devra également être enregistrée en tant qu'information persistante de manière à ce que le prochain démarrage de l'application utilise la police choisie.

Classes concernées :

- *QFont* : classe qui représente une police de caractères ;
- *QFontDialog* : boîte de dialogue pré-établie permettant de choisir une police de caractères ;
- *QSettings* : classe qui conserve les informations et préférences de l'application de manière persistante.

2 Actualiser la barre d'état

La fenêtre principale de votre application comporte une barre d'état par défaut. Vous devrez faire en sorte que cette barre affiche la position de votre curseur lorsque vous éditez un fichier texte. Fondamentalement, la classe *QTextEdit* qui est utilisée envoie un signal lorsque la position du curseur change. Elle maintient en parallèle un curseur vers son document associé qui exprime sa position en numéro de bloc (en réalité un bloc est une ligne) et en colonne dans le bloc.

Classes concernées :

- *QTextCursor* : curseur texte associé à *QTextEdit* ;
- *QTextDocument* : document texte associé à *QTextEdit* ;
- *QStatusBar* : classe représentant la barre d'état.

3 Aller à une ligne en particulier

Vous ajouterez à votre application une option dans le menu « Éditer ». Cette dernière s'appellera « Se rendre à la ligne ». Lorsqu'elle sera cliquée, elle devra ouvrir une boîte de dialogue demandant un numéro de ligne. Lorsque ce dernier est renseigné, le curseur devra se déplacer à la ligne visée. Bien entendu, vous vous débrouillerez pour que le numéro de ligne entré par l'utilisateur soit contraint et ne dépasse pas le nombre de lignes du document.

Classes concernées :

- Classes précédemment présentées ;
- *QInputDialog* : classe générique pour des boîtes de dialogue ne nécessitant que l'entrée d'une valeur particulière.

4 Mise en place de fonctions de recherche

Vous devrez ajouter une fonctionnalité de type « Rechercher... »et « Rechercher suivant »dans votre menu « Éditer ». La première, lorsqu'elle est activée, vous ouvrira une boîte de dialogue dans laquelle vous pourrez taper le texte à rechercher (dans un premier temps, vous pourrez, pour vos tests, vous contenter d'une boîte de dialogue simple telle que proposée par la classe *QInputDialog*). Le curseur de texte se déplacera à l'occurrence du texte recherché qui suit la position courante de ce curseur. La deuxième fonctionnalité vous permettra d'accéder à l'occurrence suivante (si on arrive

à la fin du document, on repart depuis le début). Si le texte n'est pas trouvé, on ouvrira une fenêtre d'information expliquant que l'expression recherchée ne figure pas dans le document. Votre boîte de dialogue finale pour rechercher une expression doit permettre de faire la distinction entre une expression recherchée sans le respect de la casse et une expression qui doit impérativement respecter la casse (distinction entre majuscules et minuscules).

Classes concernées :
- Classes précédemment présentées ;
- *QDialog* : la classe mère des boîtes de dialogue ;
- *QLabel* : Widget affichant un libellé court ;
- *QLineEdit* : Widget permettant la saisie d'un texte court ;
- *QCheckbox* : Widget de type case à cocher.

Créer une nouvelle boîte de dialogue : Vous pouvez procéder à cette étape en passant par *Qt-Creator*. A gauche, dans l'arborescence de votre projet, vous pouvez faire un clic droit et sélectionner l'option *Add new...* dans le menu contextuel. Cela démarre un assistant. A la première page, à gauche vous sélectionnez *Qt* et à droite *Qt Designer Form Class*. A la page suivante, vous pouvez choisir entre plusieurs modèles de boîte de dialogue (choisissez celui que vous voulez). Ensuite, vous pourrez renseigner le nom de la classe que vous souhaitez générer et enfin quitter l'assistant après un bref récapitulatif. Vous pourrez prototyper graphiquement votre boîte de dialogue.

5 Rechercher et ... remplacer

Mettre en place une fonctionnalité de remplacement qui s'appuiera sur les bases de ce que vous avez développé à la question précédente. La boîte de dialogue associée devra contenir au moins les éléments suivants :
- Une ligne de saisie du texte à remplacer ;
- Une ligne de saisie du texte de substitution ;
- Une case à cocher pour le respect de la casse ou non ;
- Plusieurs boutons :
  - Un pour un remplacement global de toutes les occurrences du texte à remplacer (si cette option est choisie, le boîte de dialogue se ferme et un résumé indiquant le nombre de substitutions effectuées devra être affichée) ;
  - Un pour rechercher la prochaine occurrence ;
  - Un pour remplacer l'occurrence trouvée ;
  - Un pour fermer la boîte de dialogue.

Classes concernées :
- Classes précédemment présentées ;
- *QPushButton* : classe permettant de créer des boutons.

6 Afficher en face du texte les numéros de lignes

Dernière fonctionnalité à réaliser, vous devrez mettre en place dans le menu « préférences », une option « Afficher les numéros de ligne »qui pourra être cochée. Il vous faudra sauvegarder de manière persistante l'état de cette option et mettre en place à côté du widget éditant le texte un widget indiquant les numéros de ligne. Cela peut-être effectué en créant un widget brut et en surdéfinissant ses méthodes *paintEvent()* et *sizeHint()*. On fera attention tout particulièrement au cas où les barres

de défilement apparaissent dans la zone d'édition du texte de manière à faire aussi défiler les numéros de ligne dans le nouveau widget.

Classes concernées :

- Classes précédemment présentées ;
- *QWidget* : classe mère des Widgets Qt ;
- *QFontMetrics* : classe permettant de calculer, en fonction d'une police de caractère, sa taille d'affichage à l'écran.