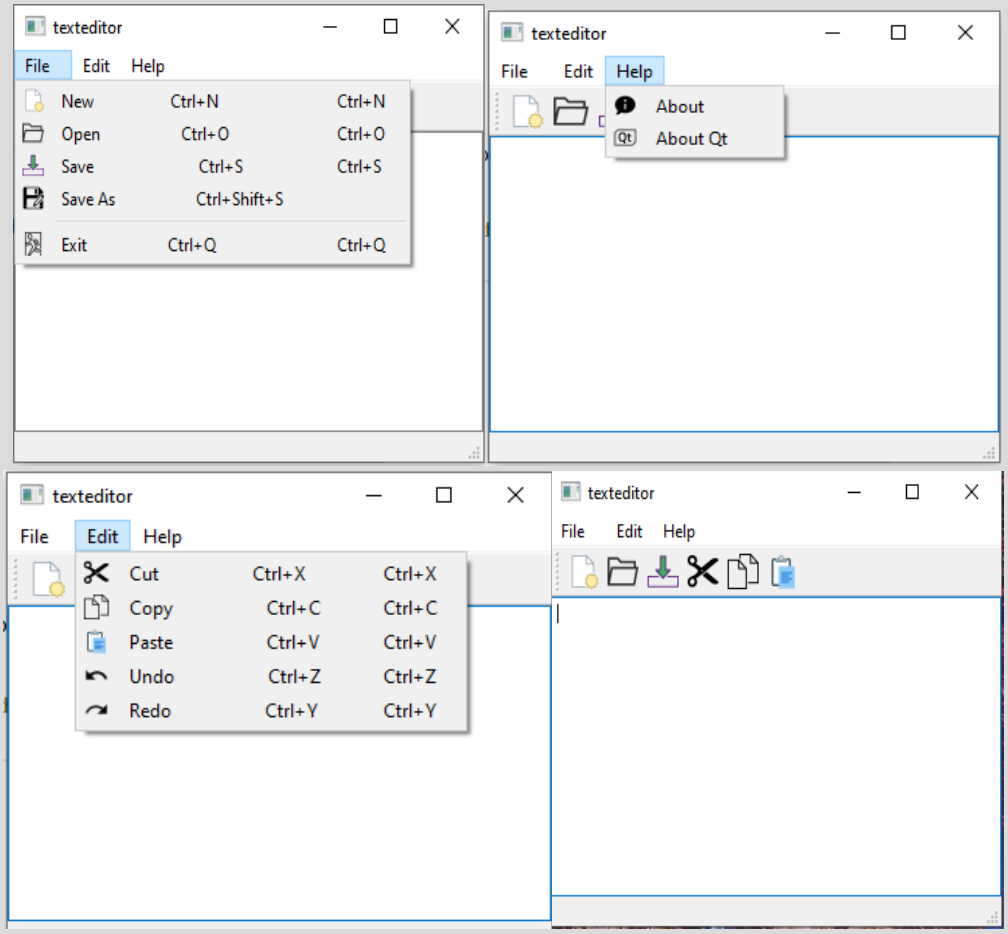
Text Editor

- pour une création d'application rapide ,on va utiliser le Designer . L'application montre comment implémenter une application GUI standard avec des menus, des barres d'outils et une barre d'état. L'exemple lui-même est un simple programme d'édition de texte construit autour de QPlainTextEdit.
- L’application fournit le cadre pour les fenêtres qui ont des menus, des barres d'outils, des fenêtres d'ancrage et une barre d'état. L'application fournit des entrées Fichier, Modifier et Aide dans la barre de menus, avec les menus et les actions contextuels suivants :



- Text Editor fournit de nombreux Slots qui implémentent des actions à partir des menus Edition, File et Help
  Voici la classe h :

```
#ifndef TEXTEDITOR_H
#define TEXTEDITOR_H
#include <QMainWindow>
#include <QFile>
#include <QFileDialog>
#include <QTextStream>
#include <QMessageBox>
QT_BEGIN_NAMESPACE
namespace Ui { class texteditor; }
QT_END_NAMESPACE

class texteditor : public QMainWindow
{
    Q_OBJECT
public:
    texteditor(QWidget *parent = nullptr);
    ~texteditor();
private slots:
    void on_actionNew_triggered();

    void on_actionOpen_triggered();

    void on_actionSave_triggered();

    void on_actionSave_As_triggered();

    void on_actionExit_triggered();

    void on_actionCut_triggered();

    void on_actionCopy_triggered();

    void on_actionPaste_triggered();

    void on_actionAbout_triggered();

    void on_actionAbout_Qt_triggered();

      void on_actionUndo_Ctrl_Z_triggered();

      void on_actionRedo_Ctrl_Y_triggered();
private:
    Ui::texteditor *ui;
    QString m_filename;
};
#endif // TEXTEDITOR_H
```

- Dans le constructeur, nous commençons par créer un widget QPlainTextEdit en tant qu'enfant de la fenêtre principale (l'objet this). Ensuite, nous appelons QMainWindow::setCentralWidget() pour indiquer que ce sera le widget qui occupera la zone centrale de la fenêtre principale, entre les barres d'outils et la barre d'état.
  Ensuite nous appelons les Shortcut Keys de chaque action dans le constructeur .

```cpp
#include "texteditor.h"
#include "ui_texteditor.h"


texteditor::texteditor(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::texteditor)
{
    ui->setupUi(this);

    this->setCentralWidget(ui->plainTextEdit);

    ui->actionNew->setShortcut(tr("Ctrl+N"));
    ui->actionOpen->setShortcut(tr("Ctrl+O"));
    ui->actionSave->setShortcut(tr("Ctrl+S"));
    ui->actionSave_As->setShortcut(tr("Ctrl+Schift+S"));
    ui->actionExit->setShortcut(tr("Ctrl+Q"));
    ui->actionCopy->setShortcut(tr("Ctrl+C"));
    ui->actionPaste->setShortcut(tr("Ctrl+V"));
    ui->actionCut->setShortcut(tr("Ctrl+X"));
    ui->actionUndo_Ctrl_Z->setShortcut(tr("Ctrl+Z"));
    ui->actionRedo_Ctrl_Y->setShortcut(tr("Ctrl+Y"));
}

texteditor::~texteditor()
{
    delete ui;
}
```

- Nous allons maintenant voir l'implémentation des Slots pour faire la connexion entre les actions et notre classe texteditor :

Le Slot on_actionNew_triggered() : est invoqué lorsque l'utilisateur sélectionne File|New dans le menu

```cpp
void texteditor::on_actionNew_triggered()
{

    ui->plainTextEdit->clear();
    ui->statusbar->showMessage("");
```

Le Slot on_actionOpen_triggered() : est invoqué lorsque l'utilisateur clique sur Fichier|Ouvrir

```
void texteditor::on_actionOpen_triggered()
{

    QString path = QFileDialog :: getOpenFileName(this,"open a file");
    QFile file(path);
    if(!file.open(QIODevice::ReadOnly)){
        QMessageBox::critical(this,"Error",file.errorString());
        return;
    }
    QTextStream stream(&file);
    ui->plainTextEdit->setPlainText(stream.readAll());
    file.close();

    ui->statusbar->showMessage(path);
    m_filename = path;
}
```

Le Slot on_actionSave_triggered() : est invoqué lorsque l'utilisateur clique sur Fichier|Enregistrer.

```
void texteditor::on_actionSave_triggered()
{

    QString path = QFileDialog :: getSaveFileName(this,"Save a file");
    QFile file(path);
    if(!file.open(QIODevice::WriteOnly)){
        QMessageBox::critical(this,"Error",file.errorString());
        return;
    }
    QTextStream stream(&file);
   stream << ui->plainTextEdit->toPlainText();
    file.close();

    ui->statusbar->showMessage(path);
    m_filename = path;


}
```

Le Slot on_actionSave_As_triggered() :

```
void texteditor::on_actionSave_As_triggered()
{
    QString path = QFileDialog :: getSaveFileName(this,"Save as");
    QFile file(path);
    if(!file.open(QFile::WriteOnly | QFile::Text)){
        QMessageBox::warning(this, "Warning", "Cannot save file :" + file.errorString());
        return;
    }
     m_filename = path;
    setWindowTitle(path);
  QTextStream out(&file);
   QString  text = ui->plainTextEdit->toPlainText();

    out << text;
    file.close();

}
```

Le Slot on_actionExit_triggered() : Lorsque l'utilisateur tente de fermer la fenêtre.

```
void texteditor::on_actionExit_triggered()
{
ui->plainTextEdit->close();
}
```

Le Slot on_actionCut triggered() :

```
void texteditor::on_actionCut_triggered()
{
ui->plainTextEdit->cut();
}
```

Le Slot on_actionCopy triggered() :

```
void texteditor::on_actionCopy_triggered()
{
ui->plainTextEdit->copy();

}
```

Le Slot on_actionPaste triggered() :

```
void texteditor::on_actionPaste_triggered()
{
ui->plainTextEdit->paste();
}
```

Le Slot on_actionAbout triggered() : La zone À propos de l'application est effectuée à l'aide d'une instruction, en utilisant la fonction statique QMessageBox::about() et en s'appuyant sur sa prise en charge d'un sous-ensemble HTML.

```
void texteditor::on_actionAbout_triggered()
{

    QMessageBox::about(this, tr("About Application"),
               tr("The <b>Application</b>  demonstrates how to "
                  "write modern GUI applications using Qt, with a menu bar, "
                  "toolbars, and a status bar."));
}
```

Le Slot on_actionAbout_Qt triggered() :

```
void texteditor::on_actionAbout_Qt_triggered()
{
  QApplication::aboutQt();
}
```

Le Slot on_action Undo_Ctrl_Z triggered() : est utilisé pour annuler la dernière action sur Windows

```
void texteditor::on_actionUndo_Ctrl_Z_triggered()
{
    ui->plainTextEdit->undo();
}
```

Le Slot on_action Redo_Ctrl_Y triggered() : est utilisé Pour annuler votre dernière annulation

```
void texteditor::on_actionRedo_Ctrl_Y_triggered()
{
        ui->plainTextEdit->redo();
}
```

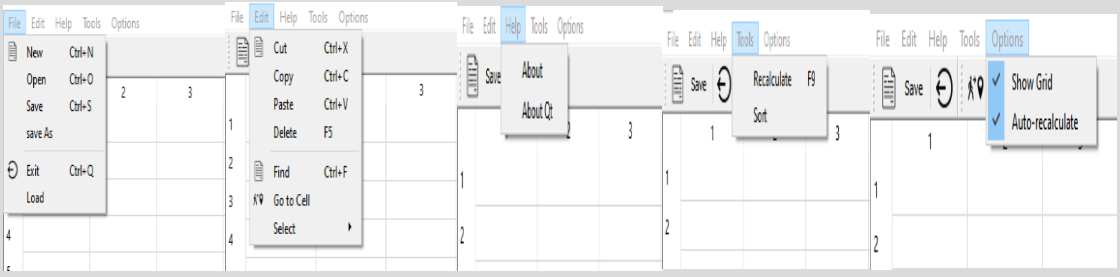
- finalement, on a ajouté un fichier de ressources en ajoutant un ensemble d'icônes prédéfinies

SpreadSheet

- pour créer l'interface utilisateur de l'application Spreadsheet. une sous-classe QTableWidget est utilisée comme widget central .Pour la pleine Fonctionnalité, vous aurez besoin d'une compréhension approfondie sur
  -Implémentation des menus
  -Sous-classer QTableWidget
  -QTableSelectedRange
  -Tri à l'aide de clés.
  -Présentation d'une formule pour chaque cellule

- Un QTableWidget est en fait une grille qui représente un tableau clairsemé à deux dimensions. Il affiche les cellules vers lesquelles l'utilisateur fait défiler, dans les dimensions spécifiées. Lorsque l'utilisateur entre du texte dans une cellule vide, QTableWidget crée automatiquement un QTableWidgetItem pour stocker le texte.

- L'application fournit les menus File,Edit, Help,Tools, and Options avec les actions contextuels suivants :



-La déclaration des actions et des menus dans la classe h

```
// --------------- Actions       --------------//
QAction * newFile;
QAction * open;
QAction * save;
QAction * saveAs;
QAction * exit;
QAction *cut;
QAction *copy;
QAction *paste;
QAction *deleteAction;
QAction *find;
QAction *row;
QAction *Column;
QAction *all;
QAction *goCell;
QAction *recalculate;
QAction *sort;
QAction *showGrid;
QAction *auto_recalculate;
QAction *about;
QAction *aboutQt;
QAction *load;


// ---------- Menus ----------
QMenu *FileMenu;
QMenu *editMenu;
QMenu *toolsMenu;
QMenu *optionsMenu;
QMenu *helpMenu;
```

- l'application spreadsheet fournit un Status bar : il affiche une description de l'élément de menu ou le boutant toolbar sous le curseur .

-dans le constructeur on a créer les label pour le status bar

```
//Creating the labels for the status bar (should be in its proper function)
cellLocation = new QLabel("(1, 1)");
cellFormula = new QLabel("");
statusBar()->addPermanentWidget(cellLocation);
statusBar()->addPermanentWidget(cellFormula);
```

```
void mySpreadSheet::updateStatusBar(int row, int col)
{
    QString cell{"(%0, %1)"};
   cellLocation->setText(cell.arg(row+1).arg(col+1));

}

void mySpreadSheet::close()
{

    auto reply = QMessageBox::question(this, "Exit",
                                       "Do you really want to quit?");
    if(reply == QMessageBox::Yes)
        qApp->exit();
}
```

Et un tool bar :

```
  //creating tool bar
 createToolBars();
```

```
void mySpreadSheet::createToolBars()
{

    //Crer une bare d'outils
    auto toolbar1 = addToolBar("File");


    //Ajouter des actions acette bar
    toolbar1->addAction(newFile);
    toolbar1->addAction(save);
    toolbar1->addSeparator();
    toolbar1->addAction(exit);


    //Creer une autre tool bar
    auto toolbar2  = addToolBar("ToolS");
    toolbar2->addAction(goCell);

}
```

- Dans la section des Slots privés, nous déclarons des emplacements qui correspondent aux entrées de menu,et dans la section protected de la classe, nous avons divers membres qui seront expliqués en temps voulu.

```
private slots:
    void close();
    void updateStatusBar(int, int); //Respond for the call changed
    void goCellSlot();
    void findCell();
    void saveSlot(); //pour répondre a l'appell de sauvegarde
    void saveAsSlot();
    void loadSlot();
    void newfileSlot();
    void aboutQtSlot();
    void aboutSlot();
    void openSlot();
    void cutSlot();

     void copySlot();
     void documentWasModified();
```

```
protected:
    void setupMainWidget();
    void createActions();
    void createMenus();
    void createToolBars();
    void makeConnexions();
    void saveContent(QString filename) const;
    void loadContent(QString filename) const;
    void setCurrentFile(const QString &fileName);
```

Ensuite, on a fait l'implémentation de la fonction createMenues ()

Et createActions () : elle est utilisée pour définir toutes les métadonnées supplémentaires nécessaires à la gestion de l'action.

```
void mySpreadSheet::createActions()
{
    //New File
   QPixmap newIcon(":/new_file.png");
   newFile = new QAction(newIcon, "&New", this);
   newFile->setShortcut(tr("Ctrl+N"));


    // open file
   open = new QAction("&Open", this);
   open->setShortcut(tr("Ctrl+O"));


    //open file
   save = new QAction("&Save", this);
   save->setShortcut(tr("Ctrl+S"));

    // open file
   saveAs = new QAction("save &As", this);


    //open file
   QPixmap cutIcon(":/cut_icon.png");
   cut = new QAction(newIcon, "Cu&t", this);
   cut->setShortcut(tr("Ctrl+X"));

   // Cut menu
   copy = new QAction( "&Copy", this);
   copy->setShortcut(tr("Ctrl+C"));


   paste = new QAction( "&Paste", this);
   paste->setShortcut(tr("Ctrl+V"));
```

```
deleteAction = new QAction( "&Delete", this);
deleteAction->setShortcut(tr("Del"));
row  = new QAction("&Row", this);
// all->setShortcut(tr("Ctrl+A"));
Column = new QAction("&Column", this);
//all->setShortcut(tr("Ctrl+A"));
all = new QAction("&All", this);
all->setShortcut(tr("Ctrl+A"));

QPixmap findIcon(":/search_icon.png");
find= new QAction(newIcon, "&Find", this);
find->setShortcut(tr("Ctrl+F"));

QPixmap goCellIcon(":/go_to_icon.png");
goCell = new QAction( goCellIcon, "&Go to Cell", this);
deleteAction->setShortcut(tr("f5"));

about =  new QAction("&About");
aboutQt = new QAction("About &Qt");

recalculate = new QAction("&Recalculate",this);
recalculate->setShortcut(tr("F9"));

sort = new QAction("&Sort");

load =new QAction("&Load");
showGrid = new QAction("&Show Grid");
showGrid->setCheckable(true);
showGrid->setChecked(spreadsheet->showGrid());
auto_recalculate = new QAction("&Auto-recalculate");
auto_recalculate->setCheckable(true);
auto_recalculate->setChecked(true);
```

```
// --------- exit -------------------
    QPixmap exitIcon(":/quit_icon.png");
    exit = new QAction(exitIcon,"E&xit", this);
    exit->setShortcut(tr("Ctrl+Q"));}
```

createMenues () :

```
void mySpreadSheet::createMenus(){
    //filemenu
    FileMenu = menuBar()->addMenu("&File");
    FileMenu->addAction(newFile);
    FileMenu->addAction(open);
    FileMenu->addAction(save);
    FileMenu->addAction(saveAs);
    FileMenu->addSeparator();
    FileMenu->addAction(exit);
    FileMenu->addAction(load);
    //edit menu
    editMenu = menuBar()->addMenu("&Edit");
    editMenu->addAction(cut);
    editMenu->addAction(copy);
    editMenu->addAction(paste);
    editMenu->addAction(deleteAction);
    editMenu->addSeparator();
    editMenu->addAction(find);
    editMenu->addAction(goCell);
    auto select = editMenu->addMenu("&Select");
    select->addAction(row);
    select->addAction(Column);
    select->addAction(all);
    //help menu
    helpMenu = menuBar()->addMenu("&Help");
    helpMenu->addAction(about);
    helpMenu->addAction(aboutQt);
    //Tools menu
    toolsMenu = menuBar()->addMenu("&Tools");
    toolsMenu->addAction(recalculate);
    toolsMenu->addAction(sort);
    //Optins menus
    optionsMenu = menuBar()->addMenu("&Options");
    optionsMenu->addAction(showGrid);
    optionsMenu->addAction(auto_recalculate);
}
```

- Dans la fonction makeconnection () ,Nous établissons une connexion signal-slot entre l'action et notre slot my spreatsheet .
- Connexion for the select all action

```
connect(all, &QAction::triggered,
        spreadsheet, &QTableWidget::selectAll);
```

. Connection for the show grid

```
connect(showGrid, &QAction::triggered,
        spreadsheet, &QTableWidget::setShowGrid);
```

. Connection for the exit button

```
connect(exit, &QAction::triggered, this, &mySpreadSheet::close);
```

connectting the chane of any element in the spreadsheet with the update status bar

```
connect(spreadsheet, &QTableWidget::cellClicked, this,  &mySpreadSheet::updateStatusBar);
```

connection pour le goDiolog

```
connect(goCell ,&QAction::triggered,this,&mySpreadSheet::goCellSlot);

connect(find,&QAction::triggered,this,&mySpreadSheet::findCell);
```

connecter le save action

```
connect(save,&QAction::triggered,this,&mySpreadSheet::saveSlot);
```

connecter le saveas action

```
//connecter le saveas action
connect(saveAs,&QAction::triggered,this,&mySpreadSheet::saveAsSlot);
```

- Go to a Cell slot connect the action to its proper slot in the makeConnexions function :

```
void mySpreadSheet::goCellSlot(){
    //creer le dialog
    Dialog D;
    //executer le dialog
     auto reply =  D.exec();    //
    //verifier si le dialog a été accepté
     if(reply == Dialog::Accepted)
        {
         //b32
         //extraire le text
         QString cell =D.getCell();

         //extraire la ligne
         int row = cell[0].toLatin1() -'A';

         //extraire la colonne
         cell =cell.remove(0,1);
         int col= cell.toInt()-1;

         //changer de cellule
         statusBar()->showMessage("changing the current cell",2000);
         spreadsheet->setCurrentCell(row,col);

     }
}
```

- La fonction findCell() est utilisé pour trouver la chaîne de texte stockée dans n'importe quelle cellule.

```
void mySpreadSheet::findCell(){

    findDialog f;

     auto reply =  f.exec();     Value stored to '

    //extraire le text
    QString cell =f.getCell();
    int rows = spreadsheet->rowCount();
    int cols = spreadsheet->columnCount();

    for (int i=0;i<rows ;i++){
       for (int j=0;j<cols ;j++){
          auto location = spreadsheet->item(i,j);
          if(location && location->text()==cell){
              spreadsheet->setCurrentCell(i, j);
              return;
       }
    }
   }
}
```

- Le slot saveslot() est invoqué lorsque l'utilisateur clique sur File|Save.

```
void mySpreadSheet::saveSlot(){
    if(!currentFile){
        //Creer Factory design
        QFileDialog D; //factory
        auto filename = D.getSaveFileName();
         currentFile = new QString(filename);
        setWindowTitle(*currentFile);
    }
    //sauvegarder le content
    saveContent(*currentFile);
}
```

- SaveAsSlot() :

```
void mySpreadSheet::saveAsSlot(){
    if(!currentFile){
        //Creer Factory design
        QFileDialog D; //factory
        auto filename = D.getSaveFileName();
         currentFile = new QString(filename);
        setWindowTitle(*currentFile);
    }
```

- LoadSlot() :

```
void mySpreadSheet::loadSlot(){

    QFileDialog D; //factory
    auto filename = D.getOpenFileName();

    if (filename != ""){
       currentFile = new QString(filename);
       setWindowTitle(filename);
       loadContent(*currentFile);
    }

;


};
```

- Le slot newFile() est invoqué lorsque l'utilisateur sélectionne File|New dans le menu.

```
void mySpreadSheet::newfileSlot(){

            spreadsheet->clear();

}
```

- AboutSlot() :

```
void mySpreadSheet::aboutSlot(){
    QMessageBox::about(this, tr("About Application"),
             tr("The <b>Application</b>  demonstrates how to "
                "write modern GUI applications using Qt, with a menu bar, "
                "toolbars, and a status bar."));

}
```

- About_QtSlot() :

```
void mySpreadSheet::aboutQtSlot(){

            QApplication::aboutQt();
}
```

- Le slot openSlot() est invoqué lorsque l'utilisateur clique sur File|Open

```
void mySpreadSheet::openSlot(){

    QString fileName = QFileDialog::getOpenFileName(this);
           if (!fileName.isEmpty())
               loadContent(fileName);
}
```

- Finalement dans le main on a ajouté une fonction read pour lire un fichier csv

```cpp
#include<iostream>
#include<fstream>
#include <iostream>
#include <string>

#include <sstream>


#include "mySpreadSheet.h"

#include <QApplication>
#include "mySpreadSheet.h"


int main(int argc, char *argv[])
{

    std::vector<double> matrix;
     //readfile
     std::fstream file;
     file.open("test.csv");
     std::string line;
     while (getline( file, line,'\n'))
       {
         std::istringstream templine(line);
        std:: string data;
         while (getline( templine, data,','))
         {
           matrix.push_back(atof(data.c_str()));
         }
       }
     file.close();
```

Fin.

Réalisé par :

EDDOUKS OUMAYMA

EDDOUKS SALIMA