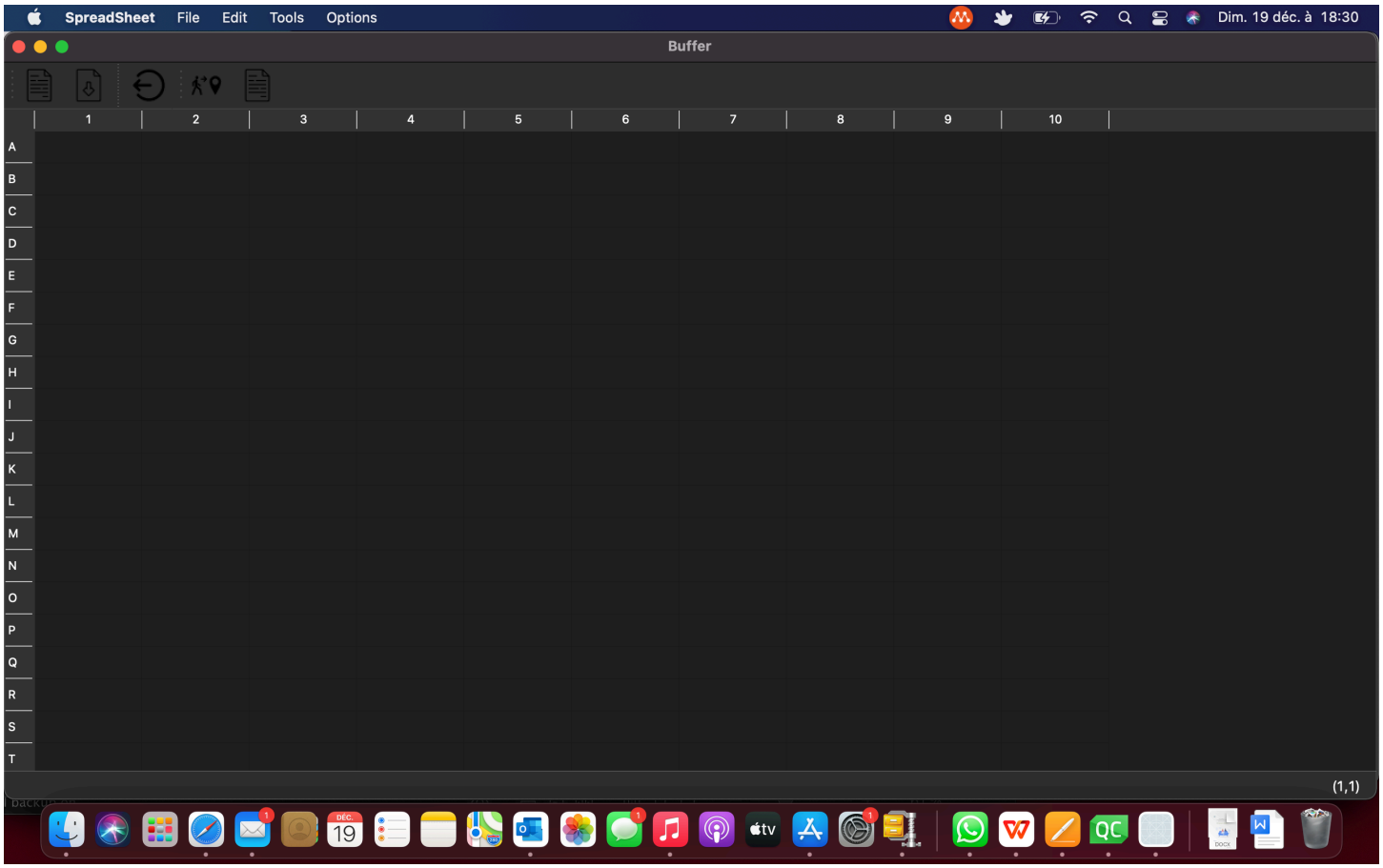
TP3 Spreadsheet

This project aims to implement a spreadsheet application that imitates excel spreadsheet with various features .



KAMEL Tarek

ABDALLAOUI Mohamed

Table of content

1 - A detailed description on the connexions :

```
void SpreadSheet::makeConnexions()
{

    // --------- Connection for the  select all action ----/
    connect(all, &QAction::triggered,
            spreadsheet, &QTableWidget::selectAll);

    // Connection for the  show grid
    connect(showGrid, &QAction::triggered,
            spreadsheet, &QTableWidget::setShowGrid);

    //Connexion for the exit button
    connect(exit, &QAction::triggered, this, &SpreadSheet::close);


    //connectting the chane of any element in the spreadsheet with the update status bar
    connect(spreadsheet, &QTableWidget::cellClicked, this,  &SpreadSheet::updateStatusBar);

    //Connexion for goDialog
    connect(goCell,&QAction::triggered,this,&SpreadSheet::goCellSlot);

    //connexion for save Action a save Slot
    connect(save,&QAction::triggered, this, &SpreadSheet::saveSlot);
     //connecter le open Action a save Slot
    connect(open, &QAction::triggered,this,&SpreadSheet::openSlot);

    // connect copy action
    connect(copy,&QAction::triggered,this,&SpreadSheet::copySlot);
    //connexion for paste slot
    connect(paste,&QAction::triggered,this,&SpreadSheet::pasteSlot);
    //connexion for load csv action
     connect(loadCsv,&QAction::triggered,this,&SpreadSheet::csvSlot);
     //connexion for 5 recent files

   connect(row,&QAction::triggered, this,&SpreadSheet::selRow);
   connect(Column,&QAction::triggered, this,&SpreadSheet::selCol);
   connect(deleteAction,&QAction::triggered, this,&SpreadSheet::deleteSlot);

}
```

The signals and slots mechanism is fundamental to Qt programming. It enables the application programmer to bind objects together without the objects knowing anything about each other. We have already connected some signals and slots together, declared our own signals and slots, implemented our own slots, and emitted our own signals. Let's take a moment to look at the mechanism more closely.

makeConnexions() method contains all the connection used for making this spreadsheet work properly. The code screenshot comprises comments explaining the role of each connexion.

1- « all » Action : which selects all spreadsheet cells using a QTableWidget method called QTableWidget::selectAll().

2- « showGrid » Action that displays the QTableWidget with a visible grid, we used a predefined slot proper to QTableWidget QTableWidget ::calledsetShowGrid()

3- « close » Action : that closes the spreadsheet using the close() method.

4-updateStatusBar() slot was implemented in order to generate inside the statusBar the exact location of the curent cell or current QTableWidgetItem displayed in this form « (11,3) »

5-« goCell » Action : basically sends the user to a desired cell location, for this purpose we created a new QDialog the prompts the user to enter the cell location and send him to this specific cell location after hitting « OK » QPushButton (explained in depth further in the chapter 4).

6-« save » Action :that saves the spreadsheet content using a slot that we implemented in class saveSlot().

7-« open » Action : that opens a previously saved spreadsheet file using the slot openSlot().

8-« copy »and « paste » Actions :that allow copy/paste the content of a cell using the Qclipboard.

9- loadCsv:

10-« row » , « column » , « deleteAction » Actions : Explained in details in the 3rd chapter

2-The code and details about the Load action.

```
423 }
424 void SpreadSheet::loadContent(QString filename)
425 {
426     //Obtenir un pointeur sur le fichier
427     QFile file(filename);
428 
429     if(file.open(QIODevice::ReadOnly))
430     {
431         //Text Stream
432         QTextStream in (&file);
433         QString line;
434         while(!in.atEnd())
435         {
436             line = in.readLine();
437             auto tokens = line.split(QChar(','));
438             int row = tokens[0].toInt();
439             int col = tokens[1].toInt();
440             spreadsheet->setItem(row,col, new QTableWidgetItem(tokens[2]));
441         }
442     }
443 
444 }
```

The following function loadContent is responsible for loading a file from your desktop or any place in in your computer.
We first implemented a while loop to go through the text stream generated from the loaded file all the way to the end.
Separating the text stream into lines using the readLine() method, then splits those lines into tokens initially separated by commas in CSV (Comma Separated Values) using the split() method, then we create two int values row and col specifying the location of the created cell in order to use them as parameters in setText() method on the actual spreadsheet.

3 - Code for the following actions (Select row & Column and delete cell content

This how we managed to add the functions (add row and add column and delete cell)

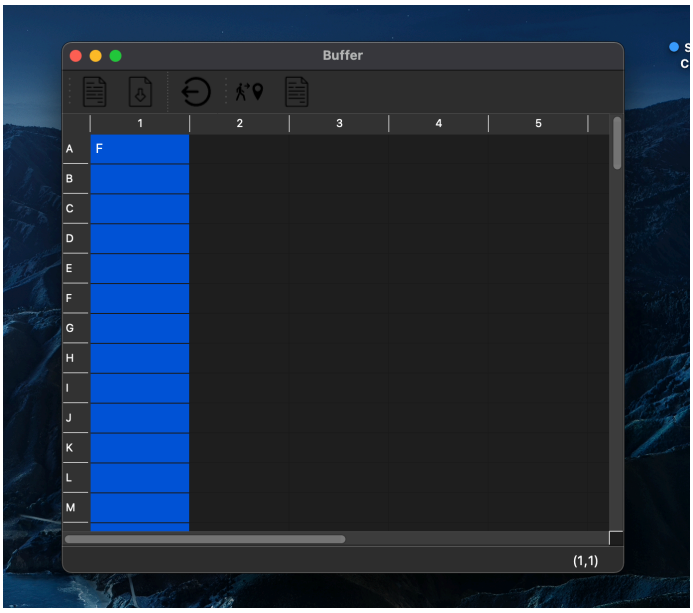
After looking through the documentation of QT we've added three new slots to : select the curent Row, select the current Column and the last one is to delete the content of a given cell in the spread sheet

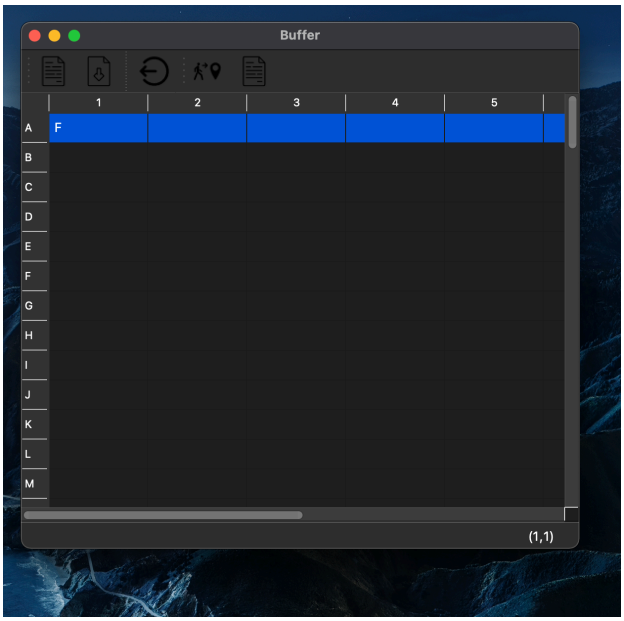
Then we've connected these three slots in the makeconnection() method respectably for row, Column, & delete action QActions. Results bellow :

```
}
void SpreadSheet::selRow(){
    spreadsheet->selectRow(spreadsheet->currentRow());
}
void SpreadSheet::selCol(){
    spreadsheet->selectColumn(spreadsheet->currentColumn());
}
void SpreadSheet::deleteSlot(){
    spreadsheet->currentItem()->setText(NULL);
}
```

```
    connect(row,&QAction::triggered, this,&SpreadSheet::selRow);
    connect(Column,&QAction::triggered, this,&SpreadSheet::selCol);
    connect(deleteAction,&QAction::triggered, this,&SpreadSheet::deleteSlot);

}
```

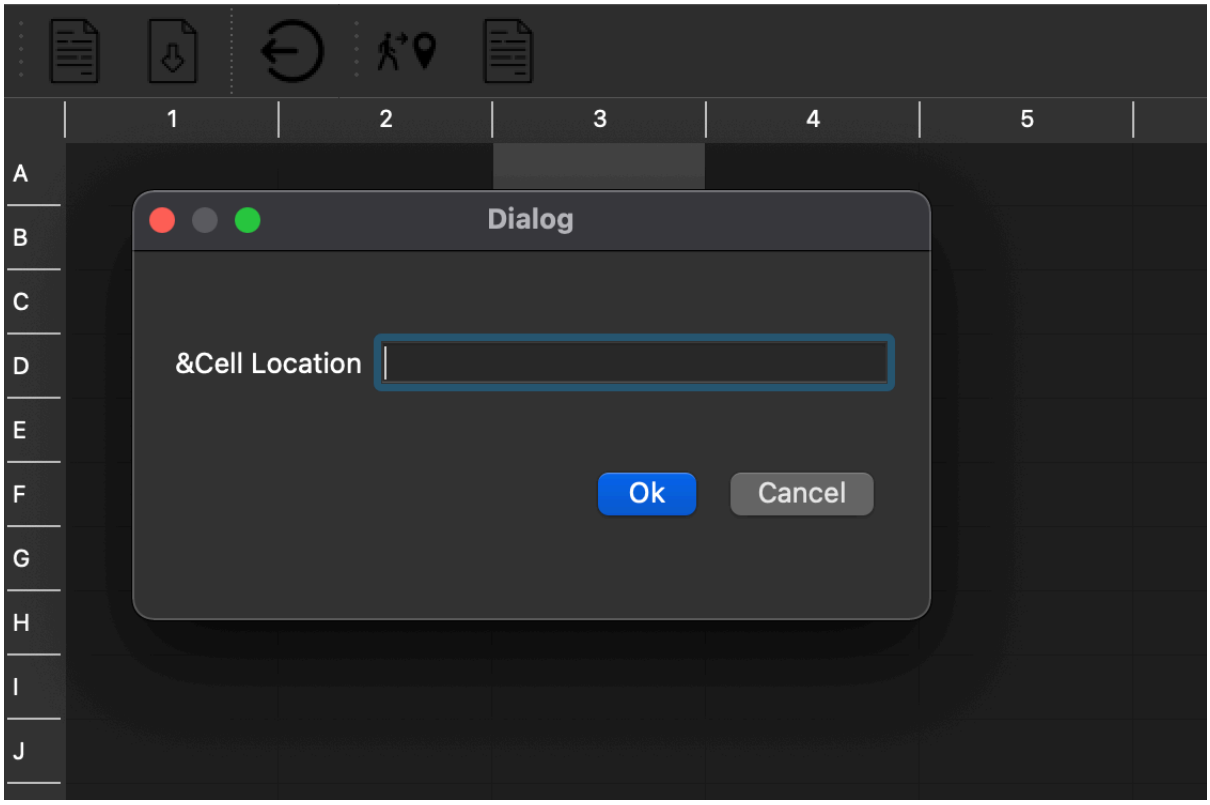




4 - Creating the second application using the designer:

In this chapter we will share our experience in creating dialog boxes using Qt. Dialog boxes present users with options and choices, and allow them to set the options to their preferred values and to make their choices. They are called dialog boxes, or simply "dialogs", because they provide a means by which users and applications can "talk to" each other.

GoDialog :



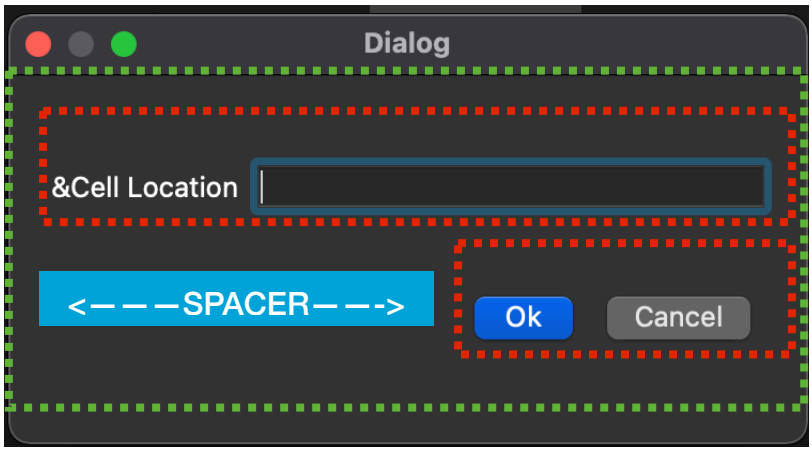
This is probably the most amusing part of Qt programming, the designer is a user-friendly environment which is lot faster than hand-coding and makes it easy to test different designs and to change designs later.

For the « goCell » dialog we added :

-a QLabel containing « Cell Location » label that we made buddy to a QLineEdit that will prompt the user the the cell location, the user response should be respecting a regular expression that we defined using QRegExp
QRegExp exp{« [A-Z][1-9][0,9]{0,2}"}; allowing the user to enter only a capital letters and to numbers
Then we used a validator to validate the user's response

- An « OK » QPushButton that executes the goCell action if triggered.
- -A « Cancel » QPushButton that cancels the operation and closes the goDialog.

LAYOUTS :



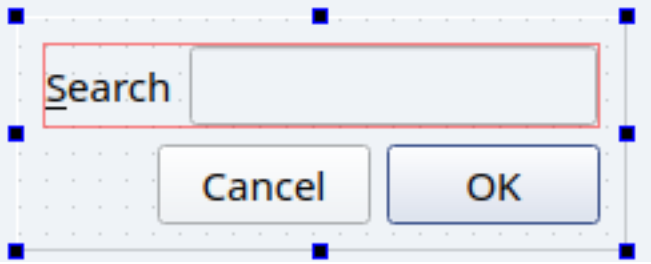
QHBoxLayout

QVBoxLayout

Find Dialog

We will move now for the **Find** dialog. This dialog prompts the user for an input and seek a cell that contains the entered text.

1. Create a **Form Class** with the following *ui*:



Find Dialog ui form.