Premiers pas avec Qt

F. Lassabe

Table des matières

1 Présentation rapide de Qt

Qt est un ensemble de bibliothèques C++ développées par l'entreprise finlandaise Trolltech, rachetée depuis par Nokia, puis Digia, avant d'exister à nouveau comme entité indépendante nommée Qt Group PLC. Qt est en premier lieu une API pour créer des interfaces graphiques utilisateur, mais elle propose également d'autres fonctionnalités comme des API réseau (incluant HTTP). Plusieurs outils conseillés sont également fournis par Qt, notamment QtCreator, un IDE qui gère notamment les macros Qt, QtDesigner (intégré à QtCreator, mais pouvant se lancer seul) qui permet de dessiner graphiquement une GUI avant de la lier à du code, et le précompilateur Qt qui convertit la syntaxe spécifique de Qt pour produire des fichiers C++ standards (avec le suffixe _moc) compilables par un compilateur standard (GCC ou Clang par exemple).

L'installation sous Linux est simple (quelques paquets logiciels à installer avec le gestionnaire de paquets de votre distribution). Sous Windows, il vous sera nécessaire de passer par un installateur qui aura le rôle équivalent à votre gestionnaire de paquets (pour Qt seulement).

Qt est un outil avec 2 types de licences : les licences GPL et LGPL permettent le développement d'applications sous les mêmes licences. Le second type de licence est une licence propriétaire, payante, qui permet de produire des logiciels basés sur Qt sous une licence non libre.

Quelques éléments notables sont à connaître avant de se lancer dans un développement avec Qt :

— Classes standard : il est possible d'utiliser les classes du standard C++ dans une application Qt. Pour des raisons de compatibilité, il est souhaitable quand c'est possible de leur substituer les versions Qt. Par exemple, `std::vector` sera remplacé par `QVector`.

— Au niveau des conventions de code, contrairement au standard C++ qui repose sur le *snake case* (tout en minuscules, mots séparés par des tirets bas), la convention dans Qt est l'utilisation du *CamelCase* (mots collés avec première lettre de chaque mot en majuscule).

— La très large majorité des classes Qt sont préfixées par la lettre Q. La documentation indexe donc les classes par leur seconde lettre (`QVector` sera par exemple dans la section **V**).

— Qt gère des événements de l'interface. Ce sujet sera traité dans la séance suivante.

Parmi les logiciels basés sur Qt, on trouve (évidemment) QtCreator et QtDesigner, qbittorrent (le meilleur client torrent de l'univers) et QGIS, un outil de manipulation et de visualisation de données géographiques (utilisant également une autre API intéressante, GDAL, pour Geospatial Data Abstraction Layer), Google Earth, KDE, KeePassXC, VirtualBox, et VLC Media Player. Cette liste n'est bien entendu pas exhaustive.

La production d'un programme basé sur Qt inclut des étapes supplémentaires par rapport à un programme C++ sans Qt. En effet, la boucle d'événements repose sur des signaux (les événements) et des slots (les fonctions à exécuter déclenchées par les événements), qui sont déclarés dans les classes à l'aide de macros. Les classes dépendant de QObject et avec la macro `Q_OBJECT` doivent être converties en C++ standard. Pour cela, Qt a un premier outil, moc, qui produit à partir des classes concernées un nouveau code, portant le nom de la classe préfixé par `_moc`, qui est ensuite compilé par g++, clang ou autre. Par exemple, `ma_fenetre.cpp` produira `moc_ma_fenetre.cpp`. Toute cette étape est automatisée par QtCreator, d'où l'intérêt de l'utiliser pour écrire vos programmes C++ basés sur Qt.

2 Les éléments de l'UI

2.1 QApplication et ses amies

Une application basée sur Qt devra instancier une (et une seule) classe d'application Qt, quelque soit le nombre de fenêtres de l'application. Il en existe 3, dont deux à instancier en fonction du type de l'application qui sera développée :

— QApplication pour les applications graphiques (qui est une classe enfant de QGuiApplication)
— QCoreApplication pour une application sans GUI (appli en ligne de commande). Notez que QCoreApplication est la classe parente de QGuiApplication, et qu'elle est enfant de QObject.

En fonction du type d'application, il faudra écrire la directive correspondante dans le fichier de projet Qt :

— Pour une QApplication : `QT += core gui` (théoriquement, ces deux valeurs sont déjà présentes par défaut, mais l'assistant Qt les ajoute si on sélectionne un projet graphique)
— Pour une QCoreApplication : `QT -= gui`

Cela permettra à qmake d'inclure les bonnes bibliothèques au Makefile généré pour la compilation. Vous pouvez trouver la liste des modules Qt à la page suivante : `https://doc.qt.io/qt-6/qtmodules.html`.

Les éléments d'interface qui sont décrits dans les sections qui suivent sont des enfants (plus ou moins directs) de la classe QWidget. Pour pouvoir les utiliser et compiler le programme, il faut ajouter le module widgets (`QT += widgets`) au fichier de projet.

L'utilisation de QApplication est simple :

```
int main(int argc, char *argv[]) {
  QApplication app{argc, argv};
  // Définir une fenêtre principale et l'afficher
  return app.exec();
}
```

2.2 La fenêtre principale

La fenêtre principale est généralement la fenêtre qui s'affiche pour permettre à l'utilisateur d'interagir avec votre application. La fenêtre principale peut contenir divers éléments (voir la documentation de la classe sur qt.io).

En général, on crée une classe enfant de QMainWindow, à laquelle on associe des éléments d'interface. La fenêtre principale n'a pas de parent. Elle peut être le parent direct ou indirect, de l'ensemble des éléments qui la constitue (le constructeur de tous les widgets a un paramètre parent, qui permet notamment la propagation des événements. Exemple de fenêtre principale :

```
class my_window: public QMainWindow {
  Q_OBJECT

public:
  my_window(QWidget *parent);
  ~my_window() {}
};

int mainint argc, char *argv[]) {
  QApplication app{argc, argv};
  my_window w{};
  w.show();
```

```
  return app.exec();
}

my_window::my_window(QWidget *parent) {
  setWindowTitle(QString{"Hello window!"})
}
```

On remarque l'utilisation de la macro `Q_OBJECT` qui permet l'utilisation des événements par la classe. Cette macro est indispensable pour permettre au programme de fonctionner, voire de compiler (dès qu'on utilise des événements).

2.3 Les fenêtres

Il existe une classe QWindow permettant de créer d'autres fenêtres, en plus de la principale. Ceci peut être utile pour ouvrir des fenêtres pour les paramètres, ou pour une fonction qui en aurait besoin.

2.4 La barre de menu

La barre de menu est la barre qui permet d'avoir des menus et qui s'affiche généralement en haut de la fenêtre.

2.5 Les entrées des menus

Les entrées du menu sont des éléments cliquables permettant d'accéder à des fonctionnalités offertes par le menu (quitter, aide, etc.). Il s'agit d'une structure arborescente (un élément peut directement être cliquable, ou dérouler une autre liste de menus).

2.6 Les boutons

Les boutons sont des zones cliquables qui déclenchent des événements de l'interface graphique, par exemple, soumettre les données d'un formulaire, etc.

2.7 Les zones de texte

Les zones de texte permettent d'afficher du texte, ou d'en saisir.

2.8 Les listes

Les listes permettent d'afficher des lignes de texte, sélectionnables.

2.9 Les zones dessinables

Les zones dessinables sont des zones pour dessiner librement en 2D ou 3D. Dessiner se fait par des primitives pour tracer des points, lignes, etc. ou en affichant des textures (i.e. des images).

2.10 Exemple

Plutôt que de faire un exemple limité par widget, voici un exemple qui illustre l'ensemble des éléments listés ci-dessus. Vous y trouverez également l'utilisation d'une disposition pour placer les éléments. L'exemple se décompose en 4 fichiers :

— le main, qui instancie et affiche une application et une fenêtre.
— le mainwindow.h avec la déclaration de la fenêtre principale et de ses éléments.
— le mainwindow.cpp qui définit les déclarations de son header.
— le fichier de projet.

Le main :

```
#include "mainwindow.h"

#include <QApplication>
```

```
int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    MainWindow w{};
    w.show();
    return app.exec();
}
```

La déclaration de la fenêtre principale :

```
#include <QMainWindow>
#include <QScopedPointer>

class QMenu;
class QGroupBox;
class QPushButton;
class QLineEdit;
class QListWidget;

class MainWindow : public QMainWindow
{
    Q_OBJECT

    // All pointers managed by the main window
    QMenu *_file_menu;
    QMenu *_help_menu;
    QScopedPointer<QGroupBox> _main_widget;
    QScopedPointer<QPushButton> _button;
    QScopedPointer<QLineEdit> _text_edit;
    QScopedPointer<QListWidget> _list;

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
};
```

Les définitions de la fenêtre principale :

```
#include <QWidget>
#include <QLineEdit>
#include <QPushButton>
#include <QListWidget>
#include <QStringList>
#include <QVBoxLayout>

using namespace std;

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent) {
    setWindowTitle(QString{"Coin²"});

    // Create menus
    _file_menu = menuBar()->addMenu(QString{tr("&File")});
    _help_menu = menuBar()->addMenu(QString{tr("&Help")});

    // Populate menus (menu items)
    _file_menu->addAction(QString{"Preferences"});
    _file_menu->addAction(QString{"Quit"});
    _help_menu->addAction(QString{"Manual"});
    _help_menu->addAction(QString{"About"});
```

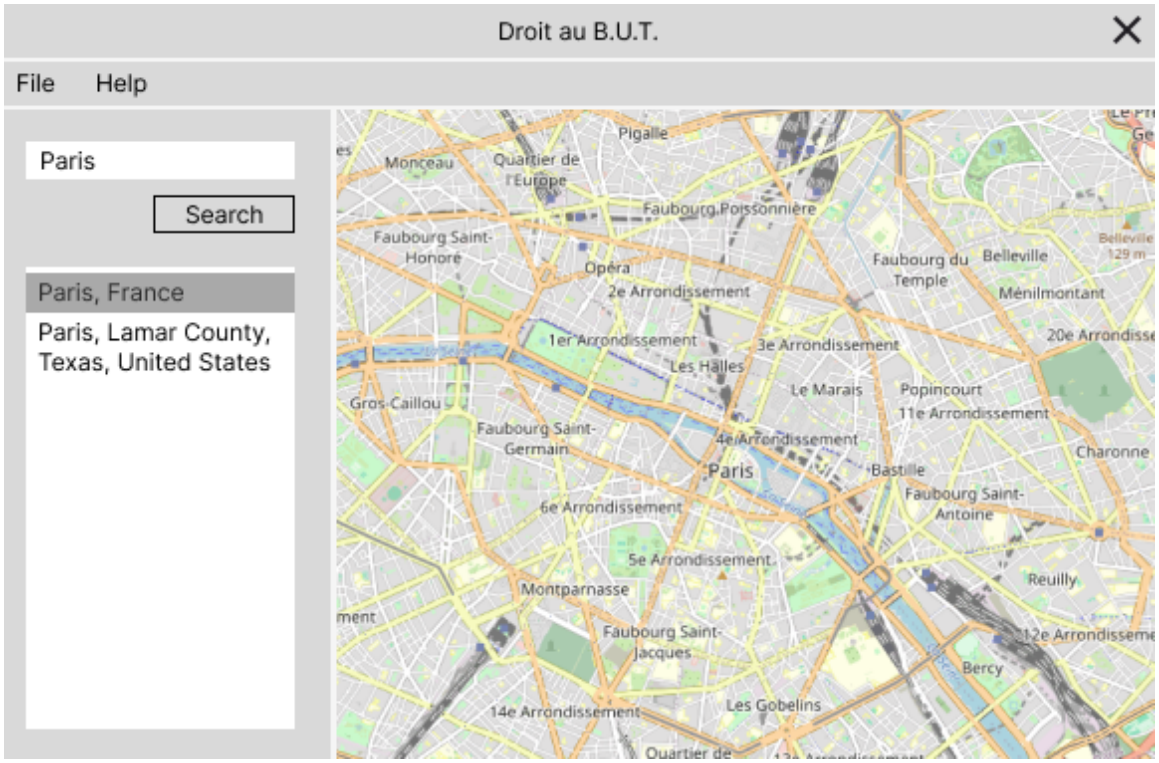
```
    // Container for window
    _main_widget.reset(new QGroupBox{this});
    setCentralWidget(_main_widget.get());
    // Layout in container
    QVBoxLayout *layout = new QVBoxLayout{};
    // Button
    _button.reset(new QPushButton{QString{"Press"}, _main_widget.get()});
    layout->addWidget(_button.get());
    // Text area
    _text_edit.reset(new QLineEdit{_main_widget.get()});
    layout->addWidget(_text_edit.get());
    // List
    QStringList places = {"New York", "Paris", "Beijing"};
    _list.reset(new QListWidget{_main_widget.get()});
    _list->addItems(places);
    layout->addWidget(_list.get());

    // Put layout into central widget
    _main_widget->setLayout(layout);
}

MainWindow::~MainWindow() {}
```

Le fichier projet :

```
QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++20

SOURCES += \
    main.cpp \
    mainwindow.cpp

HEADERS += \
    mainwindow.h
```

3 Objectif de la séance

L'objectif de la séance est de créer l'UI suivant le schéma ci-dessous. Les interactions avec les éléments de l'UI seront implémentées à la séance suivante. Vous pouvez faire appel à des outils externes, notamment des IA génératives, pour vous assister dans cette tâche. Toutefois, gardez à l'esprit que les IA commettent des erreurs (parfois bénignes, parfois bien plus problématiques) et que vous devez donc être capables d'analyser et de qualifier la qualité des solutions que vous obtenez par ce biais.


Droit au B.U.T.
File
Help
Paris
Search
Paris, France
Paris, Lamar County, Texas, United States
Pigalle
Monceau
Quartier de l'Europe
Faubourg Poissonnière
Faubourg Saint-Honoré
Opéra
2e Arrondissement
Faubourg du Temple
Belleville
Ménilmontant
1er Arrondissement
3e Arrondissement
Les Halles
Le Marais
Popincourt
11e Arrondissement
Gros-Caillou
Faubourg Saint-Germain
4e Arrondissement
Paris
Bastille
Charonne
Faubourg Saint-Antoine
6e Arrondissement
5e Arrondissement
Montparnasse
Reuilly
Faubourg Saint-Jacques
Bercy
14e Arrondissement
Les Gobelins