Université de la Rochelle
Licence d'Informatique - IHM (INFO132601)

TP 3 - *Programmation Qt: une application complète*

© B. Besserer, R. Péteri

Année universitaire 2015-2016

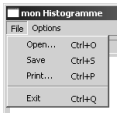
CORRECTION

1 Cahier des charges

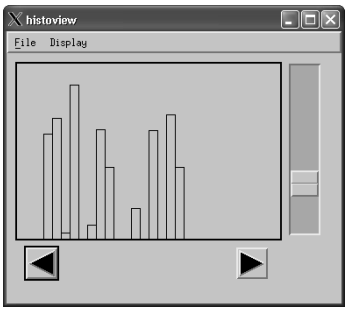
Le but du TP est d'écrire une application complète sous Qt permettant d'afficher graphiquement un histogramme à partir de valeurs contenues dans un fichier texte, que le programme devra ouvrir et lire. Ce TP est prévu pour être réalisé sous Linux avec de préférence un gestionnaire de fenêtres KDE, en utilisant une version de Qt > 4.

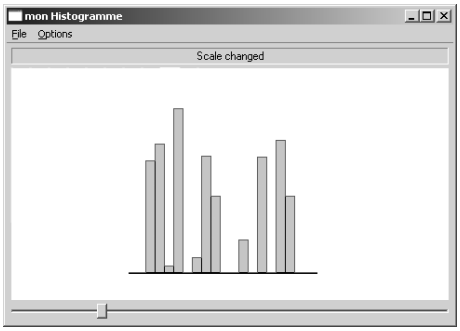
L'application disposera des éléments d'interface suivant :

— Un menu disposant au moins des articles Exit, Open dans un menu File, et Options dans un menu Display,



— Une zone de dessin pour afficher l'histogramme,
— 2 boutons permettant un défilement gauche/droite (panning) de l'histogramme,
— un curseur (slider) permettant de régler l'échelle d'affichage de l'histogramme.





La copie d'écran de gauche montre une version Motif (à remarquer que Motif possède des widgets de type ArrowButton qui sont utilisés ici pour le panning (déplacement gauche/droite de l'histogramme). La copie d'écran de droite montre une version Qt. Le curseur utilisé pour le facteur d'échelle est situé en bas de la fenêtre, les boutons de panning ne sont pas présents. A remarquer que Qt ne possède pas de widget ArrowButton (à moins de mapper l'image adéquate sur le bouton), utilisez des QPushButton en utilisant "«" et "»" comme étiquettes pour ces boutons.

Un code source partiel du projet Qt5 (`TP3_Qt_base.zip`), est à récupérer sous Moodle. Décompressez cette archive dans votre répertoire de travail.

2 Etude du code et classes à concevoir pour l'affichage d'histogramme

• Tout d'abord, compiler le code fourni et vérifier le bon fonctionnement du programme (notamment le dessin d'un carré à l'endroit du clic souris).

• myBox est une classe dérivée de QLabel qui affiche simplement un carré dans une zone de dessin (QPixmap) à l'endroit du clic. La classe myMainWindow construit la fenêtre principale de l'application en créant la barre de menu, en disposant le widget myBox, un widget QPushButton "clear" et en reliant l'appui sur le bouton avec la méthode `clear()` du widget myBox.

```
// Fichier myBox.h

class myBox : public QLabel
{
Q_OBJECT

public:
  myBox();  // constructeur
  ~myBox(); // destructeur
  void clear();
  void draw(); // methode de dessin

/** surchage de la méthode
             de la classe QLabel **/
protected:
  virtual void mousePressEvent(QMouseEvent * event);

  /** methodes pour le glisser-deposer - fin du TP **/
  void myBox::dragEnterEvent(QDragEnterEvent *event);
  void myBox::dropEvent(QDropEvent *event);

private:
  int m_x, m_y; // centre du rectangle
  QPixmap *m_pixmap; // pointeur
                        sur la zone de dessin
  };



// Fichier myBox.cpp

#include <QtGui>
#include "myBox.h"

/** Constructeur (sans le support du drag&drop) **/
myBox::myBox()
  {
  m_x = 150;  // valeur par defaut
  m_y = 150;  // valeur par defaut
  m_pixmap = new QPixmap(300, 300); // creation pixmap
  m_pixmap->fill(Qt::white);  // effacement pixmap
  setPixmap(*m_pixmap);  // association
  }                      // pixmap à la classe de base (QLabel)

/** Destructeur **/
myBox::~myBox()
  {
  delete m_pixmap;
  }

void myBox::mousePressEvent(QMouseEvent * event)
  {
  /** char buf[50]; sprintf(buf,"Mouse loc is %d, %d",
                                event->x(), event->y()); **/
  m_x = event->x();
  m_y = event->y();
  draw();
  }

/** methode d'effacement **/
void myBox::clear()
  {
  m_pixmap->fill(Qt::white);
```

```
  setPixmap(*m_pixmap);
  }

/** Methode de dessin **/
void myBox::draw()
  {
  m_pixmap->fill(Qt::white);
  QPainter p(m_pixmap);
  QPen pen(Qt::black,1);
  p.setPen(pen);
  p.drawRect(m_x-10,m_y-10,20, 20);
  setPixmap(*m_pixmap);
  }
```

Sur la base des fichiers qui vous sont fournis, on propose de modifier la classe `myMainWindow` et d'écrire une classe `myHisto` qui sera dérivée de `QLabel` (sur le schéma de la classe `myBox`, les nouveaux fichiers `myHisto.cpp` et `myHisto.h` remplaçant les fichiers `myBox.cpp` et `myBox.h` - pensez à modifier votre fichier .pro !).

2.1 La classe `myHisto`

A l'instar de myBox, La classe myHisto sera dérivée de QLabel. Il s'agit du widget qui affichera l'histogramme. La classe disposera de données membres capable de mémoriser les valeurs de l'histogramme (un tableau que l'on appellera `m_data`), d'une donnée membre pour la position horizontale pour le panning (`m_pan`), d'une donnée membre pour l'échelle d'affichage (`m_scale`) ainsi que les méthodes suivantes (liste non exhaustive, à titre d'exemple) :

- `draw()` // méthode de dessin, utilise la valeur des éléments de l'histo et la valeur de m_scale et m_pan
- `setScale()` // changement de la valeur de m_scale ; un réaffichage devra suivre
- `setPan()` // changement de la valeur de m_pan ; un réaffichage devra suivre
- `setData()` // remplissage des valeurs de l'histogramme

Dans un premier temps pour tester son programme, on stockera des valeurs fictives de l'histogramme "en dur" dans la variable `m_data` (l'ouverture à partir d'un fichier se fera à la section 2.3.). On utilisera avantageusement la classe `QVector` permettant de manipuler des tableaux dynamiques dans Qt (voir cours...), ainsi que la classe `QPoint` permettant de stocker les coordonnées d'un point du plan.

Aide pour le tracé de l'histogramme avec gestion du Pan et du Zoom dans la fonction de dessin:
Si `iterator` est un pointeur sur un élément de votre tableau dynamique où vous stockez les valeurs de l'histogramme, on pourra utiliser l'appel suivant (à bien comprendre !) :
`drawRect(iterator->x()* m_scale + m_pan*m_scale,base-iterator->y()* m_scale,10* m_scale,iterator->y()* m_scale);` avec `base` étant l'axe de l'ordonnée où démarre le tracé de l'histogramme.

2.2 La classe `myMainWindow`

Elle sera responsable de la mise en place des widgets (l'objet QSlider, les QPushButton, l'objet myHisto), d'une barre de menu et de l'interaction entre widgets (attacher des actions aux articles de menu, détecter le changement d'échelle et modifier l'objet myHisto en conséquence). Pour un placement correct des objets, consultez de l'aide sur les conteneur disponibles (QVBoxLayout est un exemple d'objet conteneur, voir aussi QGridLayout).

`Faîtes valider par l'enseignant.`  1

2.3 Ouverture du fichier

Le fichier doit pouvoir être ouvert en passant par le menu file, et une boite de dialogue standard de sélection de fichier. La classe de la boite de dialogue pour la sélection du fichier est : QFileDialog. Vous rechercherez les autres informations nécessaires pour l'usage de cette boite de dialogue. Lorsque vous aurez récupéré le chemin du fichier à ouvrir, il faut bien sûr décoder le fichier pour remplir les données membres de la classe myHisto pour l'affichage. Rechercher des informations sur QFile. Vous pouvez aussi regarder le code de la méthode `dropEvent` de myBox.

`Faîtes valider par l'enseignant.`

2.4 Ouverture de document par Drag & Drop

Ajoutez le support du drag & drop. En glissant et en déposant le fichier de données de l'histogramme, le résultat doit être similaire à l'ouverture de même fichier par l'intermédiaire du menu file→open. Le support du drag & drop est fonctionnel pour la classe myBox, et accepte les fichiers portant l'extension .box, ce qui vous donne une base pour que votre classe `myHisto` accepte les fichiers .dat.

Des infos sur le Drag & Drop ont également été donnés à la fin du TP3.

`Faîtes valider par l'enseignant.`

myHisto.cpp:

```
#include "myHisto.h"
#include <QtGui>

myHisto::myHisto()
{

    m_scale=1;// valeur par defaut
    m_pan=0;// valeur par defaut
    m_pixmap = new QPixmap(300, 300); // creation pixmap
    m_pixmap->fill(Qt::white);  // effacement pixmap
    setPixmap(*m_pixmap);  // association pixmap à la classe de base (QLabel)
   // setAcceptDrops(true);

    //(*toto)[0]=QPoint(3,6);
    m_data=new QVector <QPoint>();
//    m_data=new QVector <QPoint>(3);
    m_data->append(QPoint(10,85));
    m_data->append(QPoint(20,98));
    m_data->append(QPoint(40,5));
    m_data->append(QPoint(50,36));
    draw();
    setAcceptDrops(true);

}


/** methode d'effacement **/
void myHisto::clear()
{
        m_pixmap->fill(Qt::white);
        setPixmap(*m_pixmap);
        m_data->clear();
}
```

```
/** Methode de dessin **/
void myHisto::draw()
{
                m_pixmap->fill(Qt::white);
                int base = m_pixmap->height()-20;

                QPainter p(m_pixmap);
                QPen pen(Qt::black,1);

                p.setPen(pen);
                p.setBrush(QBrush(Qt::red)); //fill rectangle with red

                QVector<QPoint>::iterator iterator;
                int i=1;
                for(iterator = m_data->begin(); iterator != m_data->end(); iterator++)
                {

                    p.drawRect(iterator->x()* m_scale + m_pan*m_scale,base-iterator->y()* m_
i++;
                }

                setPixmap(*m_pixmap);
}


/** methode apell lors du "glisser" **/
void myHisto::dragEnterEvent(QDragEnterEvent *event)
{
         qApp->beep();  // indication acoustique
        if(event->mimeData()->hasText())
        {
                printf("The dragged data is seen as text");
                event->acceptProposedAction();
        }
        else if (event->mimeData()->hasUrls())
        {
                printf("The dragged data is seen as URLs");
                event->acceptProposedAction();
        }
}


/** methode du "deposer" **/
void myHisto::dropEvent(QDropEvent *event)
{
 int m_x, m_y; // temporary variables...
    /** le code convient pour le modele URL seulement **/
        const QMimeData *mimeData = event->mimeData();
        if ( mimeData->hasText())
        {
                QUrl u = mimeData->text();
                QString filename = QFileInfo( u.toString() ).fileName();

                // Attention... le fichier doit tre dans le repertoire de l'application

                // Test sur l'extension du fichier
                if(QFileInfo( u.toString() ).completeSuffix() == "box")
                {
                        QFile file(filename);
                        if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) // ouverture
                                return;

                        QByteArray line;
                        while (!file.atEnd()) // lecture ligne par ligne jusqu'a la fin du f
                        {
```

```
                                line = file.readLine();
                                sscanf(line,"%d %d", &m_x, &m_y);

                                draw();
                        }
                        event->acceptProposedAction();
                }
        }
        else if ( mimeData->hasUrls())
        {
                QString filename;
                QUrl u;
                foreach ( u, mimeData->urls() )
                        filename = QFileInfo( u.toString() ).fileName();

                // Attention... le fichier doit tre dans le repertoire de l'application

                // Test sur l'extension du fichier
                if((QFileInfo( u.toString() ).completeSuffix() == "box")||(QFileInfo( u.toStrin
                {
                        QFile file(filename);
                        if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) // ouverture en
                                return;

                        QByteArray line;
                        m_data->clear(); // empty array...

                        while (!file.atEnd()) // lecture ligne par ligne jusqu'a la fin du fich
                        {
                                line = file.readLine();
                                sscanf(line,"%d %d", &m_x, &m_y);
                                m_data->append(QPoint(m_x,m_y));

                        }
                        draw();
                        event->acceptProposedAction();
                }
        }
}
```

myMainWindow.cpp:

```
    #include <QtGui>

#include "myMainWindow.h"
#include "myHisto.h"

        /** constructeur **/
    myMainWindow::myMainWindow()
    {
            QWidget *w = new QWidget;
        setCentralWidget(w);

                /** creation des wigets **/
                /** zone d'affichage, peut servir pour message de debug **/
                m_infoLabel = new QLabel(tr("Zone d'affichage, peut toujours servir"));
        m_infoLabel->setFrameStyle(QFrame::StyledPanel | QFrame::Sunken);
        m_infoLabel->setAlignment(Qt::AlignCenter);

                /** mon widget **/
                m_myHisto = new myHisto();
```

```
                /* boutons */
                        m_clear = new QPushButton(tr("clear")) ;
                        backward = new QPushButton(tr("<<<")) ;
                        forward = new QPushButton(tr(">>>")) ;

                        /* slider */
                        m_scale = new QSlider(Qt::Horizontal) ;
                        m_scale->setRange(-2,20) ;

                        /* contenur */
                        QGridLayout *gbox = new QGridLayout() ;
                        gbox->addWidget(m_infoLabel,0,0,1,2,Qt::AlignHCenter | Qt::AlignVCen
                        gbox->addWidget(m_myHisto,1,0,1,2,Qt::AlignHCenter | Qt::AlignVCente
                        gbox->addWidget(m_scale,2,0,1,2,Qt::AlignVCenter) ;
                        gbox->addWidget(backward,3,0,Qt::AlignHCenter | Qt::AlignVCenter) ;
                        gbox->addWidget(forward,3,1,Qt::AlignHCenter | Qt::AlignVCenter) ;
                        gbox->addWidget(m_clear,4,0,1,2,Qt::AlignHCenter | Qt::AlignVCenter)

                        /** w est le widget parent, qui affiche tous les autres **/
                        w->setLayout(gbox) ;

// Connections
                        connect(this->forward,SIGNAL(clicked()),this,SLOT(pan_forward()));
                        connect(this->backward,SIGNAL(clicked()),this,SLOT(pan_backward()));
                        connect(this->m_scale,SIGNAL(valueChanged(int)),this,SLOT(set_scale(


                /** association slot -> methodes, voir methode **/
        createActions();
                /** creation barre de menu, voir methode **/
        createMenus();

                setWindowTitle(tr("TracÈ d'histogramme - RP"));
        setMinimumSize(310, 450);
        resize(310, 450);
    }

        /** destructeur **/
    myMainWindow::~myMainWindow()
        {
                delete m_infoLabel;
                delete m_myHisto;
                delete m_clear;
        }

    void myMainWindow::open()
    {
                m_infoLabel->setText(tr("Invoked <b>File|Open</b>"));
                m_myHisto->draw();
    }

    void myMainWindow::save()
    {
        m_infoLabel->setText(tr("Invoked <b>File|Save</b>"));
    }

        void myMainWindow::exit()
    {
                m_infoLabel->setText(tr("Invoked <b>Edit|Format|Set Paragraph Spacing</b>"))
                ::exit(0);
    }

    void myMainWindow::reset()
```

```
    {
        m_infoLabel->setText(tr("Invoked <b>Display|Reset to defaults</b>"));
    }

        void myMainWindow::clear()
    {
                m_infoLabel->setText(tr("Button <b>clear</b> pressed"));
                m_myHisto->clear();
    }

    void myMainWindow::createActions()
    {
                /** mise en relation article de menu -> slots **/
        openAct = new QAction(tr("&Open..."), this);
        openAct->setShortcut(tr("Ctrl+O"));
        connect(openAct, SIGNAL(triggered()), this, SLOT(open()));

        saveAct = new QAction(tr("&Save"), this);
        saveAct->setShortcut(tr("Ctrl+S"));
        connect(saveAct, SIGNAL(triggered()), this, SLOT(save()));

        exitAct = new QAction(tr("E&xit"), this);
        exitAct->setShortcut(tr("Ctrl+Q"));
        connect(exitAct, SIGNAL(triggered()), this, SLOT(close()));

        resetAct = new QAction(tr("Reset to defaults..."), this);
        connect(resetAct, SIGNAL(triggered()), this, SLOT(reset()));

                /** mise en relation boutons -> slots **/
                connect(m_clear, SIGNAL(clicked()), this, SLOT(clear()));
    }

    void myMainWindow::createMenus()
    {
        fileMenu = menuBar()->addMenu(tr("&File"));
        fileMenu->addAction(openAct);
        fileMenu->addAction(saveAct);
        fileMenu->addSeparator();
        fileMenu->addAction(exitAct);

        optionMenu = menuBar()->addMenu(tr("&Display"));
        optionMenu->addAction(resetAct);
    }

    void myMainWindow::pan_forward()
    {
this->m_myHisto->m_pan++;
this->m_myHisto->draw();
    }

    void myMainWindow::pan_backward()
    {
this->m_myHisto->m_pan--;
this->m_myHisto->draw();
    }

    void myMainWindow::set_scale(int scale)
    {
this->m_myHisto->m_scale=scale;
this->m_myHisto->draw();
    }
```