IRIS 2 TP C++ QT: Supervision de Four durée: 8h

Objectifs :
- utiliser QT Creator pour réaliser une supervision et gérer plusieurs fenêtres
- utiliser les méthodes des classes *QString, QFile, QFileDialog*
- mettre en oeuvre les méthodes des classes graphiques *QImage, Qpixmap,QPainter, QPen, QGraphicsScene et QGraphicsView*
- créer des objets et méthodes associées

Mise en situation :
A partir des fichiers joints et notamment de l'exécutable Four pour Windows, on désire développer une autre application en utilisant l'EDI QT Creator afin d'essayer d'obtenir des sources plus ou moins semblables qui seront compilés pour Linux ou pour Windows.

Travail demandé:
1)Travail n°1: Chaînes de caractères, fichiers
1,1) Tests de mise en œuvre de Qstring et composants graphiques

1,1,1) A partir de LineEdit, afficher dans un Text Edit une ligne contenant les 2 informations suivant le format des fichiers joints (.cyc).
Ex: LineEdit1= 30 LineEdit2=50 TextEdit=@30-50$
Dans la suite du TP, lors de la représentation graphique de la courbe de température en fonction du temps, la première coordonnée 30 sera l'abscisse X et la seconde coordonnée 50 sera l'ordonnée Y.

1,1,2) A partir d'un LineEdit contenant les informations X,Y sous par exemple la forme @60-90$, extraire et afficher dans 2 Labels le temps 30 et la température 50.

1,2) Tests unitaires:
Créer et tester une classe Graphe contenant dans un premier temps le constructeur et 3 méthodes:
QString lireX(QString trame) qui extrait et retourne la première coordonnée de trame= " @60-90$ "
QString lireY(QString trame) qui extrait et retourne la seconde coordonnée de trame= " @60-90$ "
QString codeTrame(QString x, QString y) qui crée et retourne la trame à partir de x et y

1,3) Fichiers: classes QFileDialog, QFile ou autres
- Un clic sur un bouton ***Enregistrer*** doit enregistrer une chaîne de caractères dans un fichier texte.
- Modifier afin d'enregistrer la première ligne d'un Text Edit
-Modifier afin d'enregistrer tout le contenu du Text Edit.

2) Travail n°2: représentation graphique
2,1) Tests de mise en œuvre des classes graphiques *QImage, QPixmap, QPen,QGraphicsScene et QGraphicsView*

- Tester le projet joint ***four.pro***
- Ajouter à la classe Graphe :
  - un constructeur qui prend en paramètre un QGraphicsScene
  - deux attributs ***int xCourant et yCourant*** initialisés à 0
  - une méthode trace(int x, int y) qui trace un trait des valeurs courantes xCourant , yCourant vers x,y.

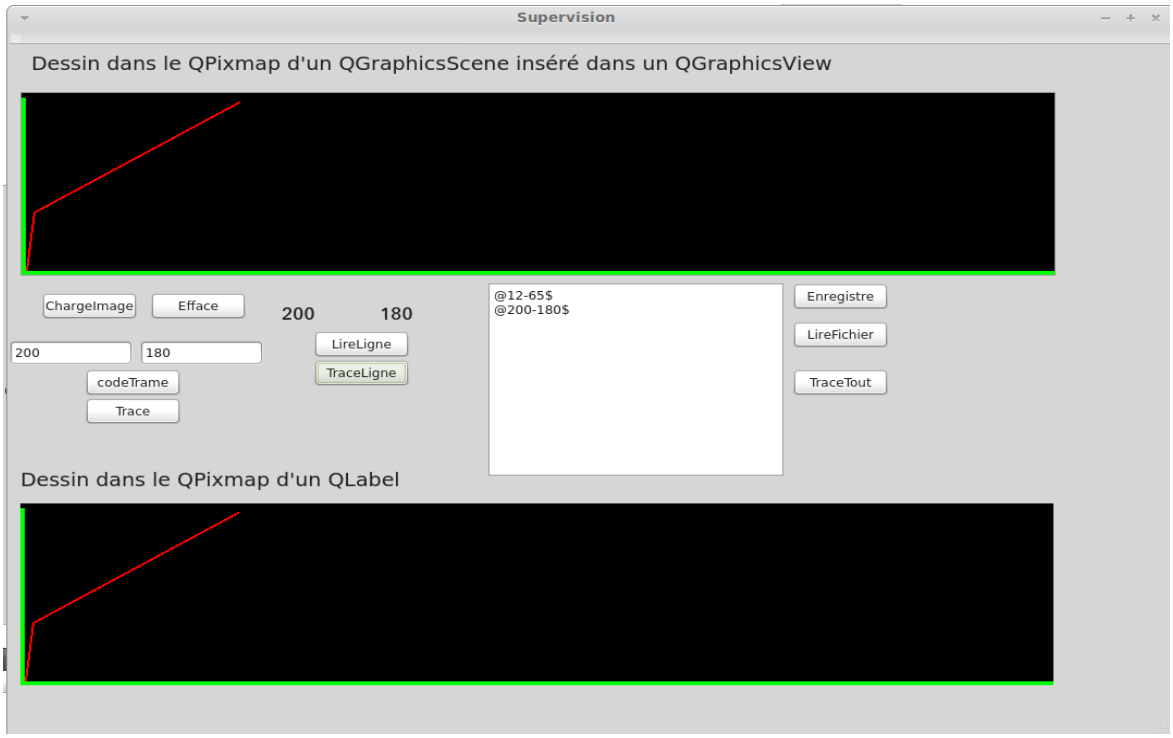
Après le tracé les valeurs x,y doivent devenir les valeurs courantes

2,2) Test unitaires

Un bouton doit permettre la saisie de x, y placés dans des LineEdit, et un autre l'appel de la méthode trace. Bien sûr l'objet doit être initialisé auparavant.

Critères d'évaluation:
-Degré d'autonomie dans la recherche d'informations
-Maîtrise de l'outil de développement.
-Qualité du programme
- Avancement du travail
- Rédaction du compte-rendu

3) suite Représentation graphique



L'interface présentée n'est pas réalisée dans le but de développer une application.
Son rôle est de réaliser des tests unitaires sur les méthodes de la classe Graphe

```
#include <QPainter>
#include <QImage>
#include <QString>
#include <QStringList>

class Graphe
{
private:
    int xCourant, yCourant;
    int h,w;
    QPainter * monPeintre;
    QImage * monImage;

    QString fichierImage;
public:
    Graphe(QString nomFichierImage,int largeur,int hauteur);
    QString lireX(QString trame);
    QString lireY(QString trame);
    QString codeTrame(QString x, QString y);
    void Trace(int x,int y);
    QImage   getImage();
    void Efface();
    void Trace(QStringList maListe);
};
```

Les méthodes ***Trace*** de cette classe vont dessiner sur un objet QImage.
Afin de visualiser le résultat , la méthode ***getImage*** retourne l'objet QImage qui devra être placé dans un objet QPixmap.
Ce QPixmap peut, par exemple, être le pixmap d'un QGraphicsScene lui même inséré dans un QGraphicsView.
Ce QPixmap peut aussi être le pixmap d'un QLabel.

Exemple d'appel du constructeur ***Graphe*** et de ***getImage*** afin de charger une image de fond

```
void MainWindow::on_chargeImage_clicked()
{
    QPixmap  pixmap;

    scene = new QgraphicsScene(this);      // g et scene sont déclarés dans private de MainWindow.h

    g = new Graphe("axesFondNoir.png",ui->graphicsView->width()-2, ui->graphicsView->height()-2);
    pixmap=pixmap.fromImage(g->getImage());

    //dans le graphicsView
    scene->addPixmap(pixmap);
    ui->graphicsView->setScene(this->scene);
    ui->graphicsView->show();

    //dans un label
    ui->label_3->setPixmap(pixmap);

}
```

Travail demandé

3,1) A partir des informations précédentes, modifier le fichier *header* de la classe ***Graphe***

3,2) Implémenter et tester le constructeur ***Graphe*** et la méthode ***getImage***.
Rôle du constructeur: initialise les attributs
`Graphe::Graphe( QString nomFichierImage,int largeur,int hauteur)`
Le constructeur mémorise le nom du fichier Image et la largeur et hauteur du conteneur qui contiendra l'image.
Les attributs xCourant et Ycourant seront respectivement initialisés avec 0 et *hauteur*.
L'attribut ***monImage*** sert à instancier un objet QImage avec paramètre le nom du fichier Image.
L'attribut ***monPeintre*** sert à instancier un objet Qpainter avec paramètre l'attribut ***monImage.***
Le crayon de l'objet ***monPeintre*** sera initialisé avec une largeur de 2 et une couleur rouge.
Un objet ***QPen*** sera donc instancié et initialisé avant l'appel de ***setPen*** de l'objet ***monPeintre.***

La méthode ***getImage*** retourne l'objet ***monImage***
Un bouton " chargeImage " doit permettre **le test unitaire** partiel de ces 2 méthodes.

Note: * monImage=monImage->scaled(largeur,hauteur,Qt::IgnoreAspectRatio); //remplit le conteneur
Qt::red // rouge

3,3)Implémentation et test de la méthode Trace(int x, int y)
Cette m éthode utilise drawLine de l'objet monPeintre afin de dessiner une ligne partant du point xCourant,yCourant et arrivant au point x,y.
Attention à l'axe des ordonnées qui doit être croissant vers le haut: (donc hauteur de l'image-y)
Exemple d'appel de la méthode ***Trace***

```
void MainWindow::on_Trace_clicked()
{
    QPixmap  pixmap;
    bool b;
    int x=ui->lineEdit->text().toInt(&b,10);
    int y=ui->lineEdit_2->text().toInt(&b,10);
    g->Trace(x,y);
    pixmap=pixmap.fromImage(g->getImage());

    //dans le graphicsView
    scene->addPixmap(pixmap);
    ui->graphicsView->show();

    // dans un label
    ui->label_3->setPixmap(pixmap);
}
```

3,4) Implémenter et tester la méthode ***Efface***
Cette méthode efface les tracés effectués en rechargeant l'image.
Etapes: - détruire l'objet *monPeintre*
- recharger le fichier image avec la méthode ***load*** de la classe *monImage*
- instancier *monPeintre* avec *monImage*
- réinitialiser *le crayon , xCourant et yCourant*

Exemple d'appel de la méthode *Efface*

```
void MainWindow::on_Efface_clicked()
{    g->Efface();
    QPixmap    pixmap=pixmap.fromImage(g->getImage());

    //dans le graphicsView
    scene->addPixmap(pixmap);
    ui->graphicsView->show();

    //dans un label
    ui->label_3->setPixmap(pixmap);
}
```

3,5) Implémenter et tester la méthode Trace(QStringList maListe)
Cette méthode lit chaque ligne et appelle la méthode Trace (int x, int y) en ayant au préalable extrait et convertit les coordonnées x et y.

Exemple d'appel de cette méthode

```
void MainWindow::on_TraceTout_clicked()
{
    QStringList maListe=ui->textEdit->document()->toPlainText().split("\n");
    g->Trace(maListe);

    QPixmap pixmap=pixmap.fromImage(g->getImage());

    //dans le graphicsView
    scene->addPixmap(pixmap);
    ui->graphicsView->show();

    //dans un label
    ui->label_3->setPixmap(pixmap);
}
```

4) Retour au QString. Test avec le contenu du QtextEdit

4,1) Ajouter et tester la méthode ***QStringList Graphe::Formate(QStringList maListe)*** qui formate chaque ligne de façon à ce que les coordonnées x et y soient représentées systématiquement sur 4 chiffres.
@15-87$ devient @0015-0087.

Ceci permettra d'effectuer plus facilement un tri de la liste. On suppose que l'on ne dépassera pas 9999 secondes pour x.

4,2) Ajouter et tester la méthode ***QStringList Graphe::TriParX(QStringList maListe)*** qui trie la liste selon les valeurs de x

4,3) Ajouter et tester la méthode ***QString Graphe::xMax(QStringList maListe)*** qui retourne la valeur maximale de x contenue dans la liste.

4,4) Ajouter et tester la méthode ***QString Graphe::yMax(QStringList maListe)*** qui retourne la valeur maximale de y contenue dans la liste.

Critères d'évaluation: -Degré d'autonomie dans la recherche d'informations
-Maîtrise de l'outil de développement. - Avancement du travail
-Qualité du programme - Rédaction du compte-rendu