Conception d'Applications Interactives

développement d'IHM en PyQt5

Alexis NEDELEC

Centre Européen de Réalité Virtuelle
Ecole Nationale d'Ingénieurs de Brest

Introduction

Qt (pronounced cute /kju :t/ or cuty /kju :ti :/)

- API orientée objet en C++
- framework pour l'environnement KDE
- toolkit Graphique C++
- Evolution de Qt1 à Qt6 en passant par Qt5 + QtQuick :
  - TrollTech, Qt Software, Nokia, Digia ...
  - `https://www.qt.io/group`
- licences GNU LGPL, commerciale
- multiplateformes : OS classiques et mobiles
- devise : "write once, compile anywhere"

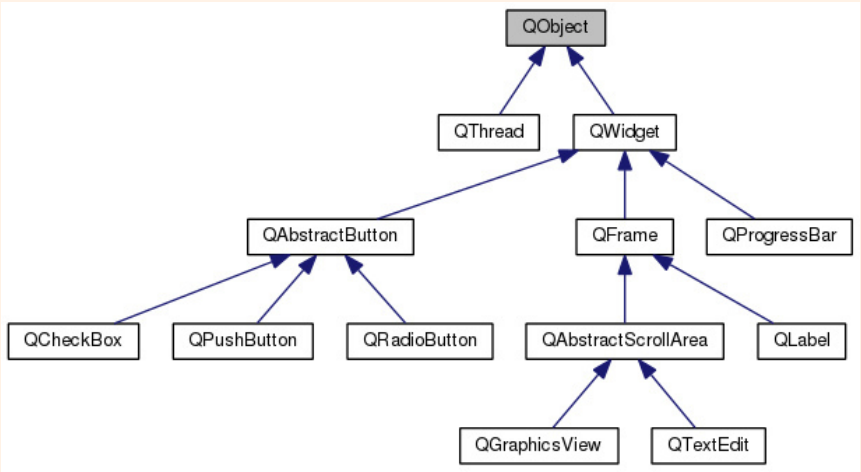
Qt API

Toolkit graphique ... mais pas seulement

- framework pour applications graphiques 2D/3D
- programmation événementielle, signaux/slots (`moc`)
- environnements de développement :
  - `Qt Designer` : générateur d'IHM (fichiers.ui)
  - `Qt Assistant` : documentation complète de Qt hors-ligne
  - `Qt Creator` : IDE Qt pour gestion de projet
- internationalisation (`tr()`, `Qt Linguist`)
- gestion de fichiers, connexion SGBD
- communication inter-processus, réseau
- W3C : XML, SAX, DOM
- multithreading
- ...

Qt API

Héritage de classes



Convention de nommage :

- Nom de classe : `Q` + `CamelCaseName`
  - `QPushButton`,`QGraphicsEllipseItem` ...
- Nom de méthode : `lowerCamelCaseName`
  - `QWidget::setMinimumSize()`

Qt API

Modules Qt Essentials (https://doc.qt.io/qt.html)

- `Qt Core` : classes de base pour tous les modules
- `Qt D-Bus` : communication inter-process
- `Qt GUI` : composants graphiques 2D et 3D (OpenGL)
- `Qt NetWork` : faciliter la programmation réseaux
- `Qt QML` : pour les langages QML et javascript
- `Qt Quick` : création d'applications de manière déclarative
- `Qt Test` : pour faire des test unitaires
- `Qt Widgets` : extension des fonctionnalités GUI pour composants graphiques
- ...

Qt API

Modules Qt Add-Ons (https://doc.qt.io/qt.html)

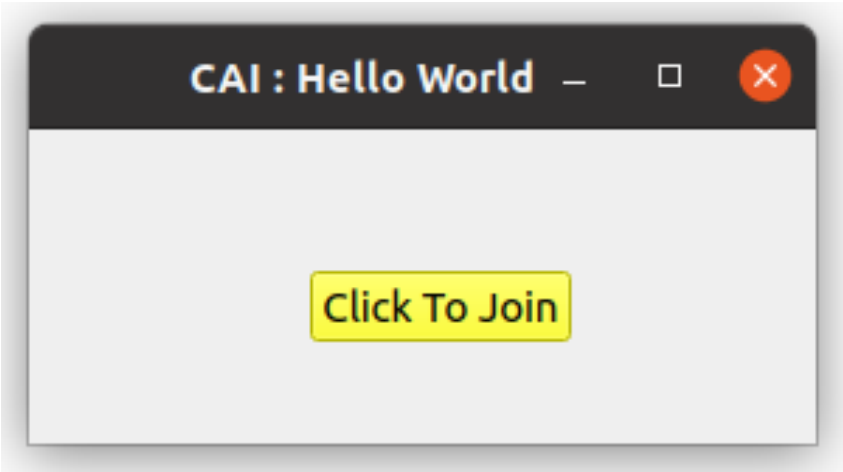
- `Qt SQL` : connexion, manipulation SGBD relationnels
- `Active Qt` : pour applications utilisant ActiveX et COM
- `Qt 3D, Qt OpenGL` : développement d'applications 3D
- `Qt Charts` : visualisation statique,dynamique de données
- `Qt Bluetooth` : Android, iOS, Linux, macOS, WinRT
- `Qt Concurrent` : pour le multi-threading
- `Qt Sensors` : données capteurs, reconnaissance de gestes
- `Qt SVG` : affichage de contenu XML 2D
- `Qt XML` : SAX et DOM sur documents XML
- ...

Hello World

Fenêtre principale (main.cpp)

```
#include <QtWidgets>

int main(int argc, char *argv[]){
 QApplication app(argc,argv);
 QWidget window;
 window.resize(200,100);
 window.setWindowTitle("CAI : Hello World");
```

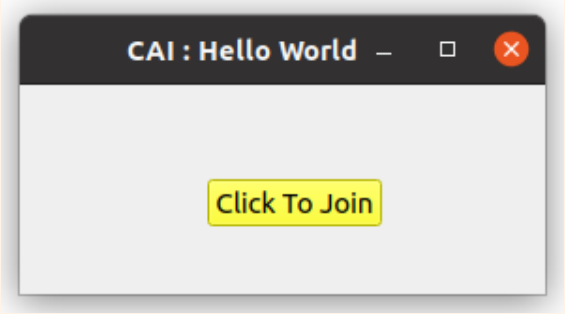


Hello World

Composant graphique (main.cpp)

```
 QPushButton *button= new QPushButton("Click To Join",\
                                      &window);
 button->move(100,50);
 button->setStyleSheet("background-color:yellow;");
 window.show();
 return app.exec();
}
```

{logname@hostname} ./HelloWorld-1



Hello World

Environnement de développement

```
{logname@hostname} tree
HelloWorld-1
|-- HelloWorld-1.pro
|-- main.cpp
```

Configuration de projet (HelloWorld-1.pro)

```
QT += widgets

SOURCES += main.cpp
TARGET = HelloWorld-1
```

Hello World

Génération de Makefile, compilation, exécutable

```
{logname@hostname} qmake -o Makefile HelloWorld-1.pro
{logname@hostname} make
{logname@hostname} tree
HelloWorld-1
|-- HelloWorld-1
|-- HelloWorld-1.pro
|-- main.cpp
|-- main.o
|-- Makefile
0 directories, 5 files
{logname@hostname} ./HelloWorld-1
```

Interaction

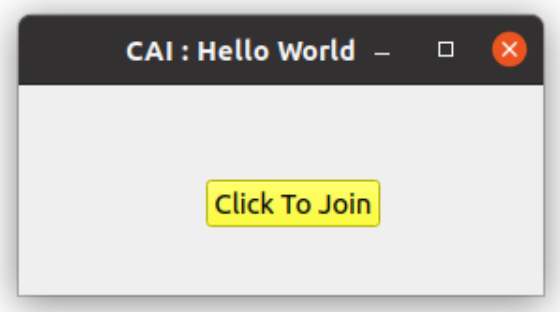
Signaux et slots

```
int main(int argc,char *argv[]){
...
  QPushButton *button= new QPushButton("Click To Join",
                                       &window);
...
  Toplevel* top= new Toplevel(&window);
  QWidget::connect(button,SIGNAL(clicked()),
                   top,SLOT(show()));
  window.show();
  return app.exec();
}
```

Interaction

Fenêtre secondaire (Toplevel)

```
#include <QtWidgets>
class Toplevel : public QDialog
{
  public :
    Toplevel(QWidget* parent);
};
```



Interaction

Fenêtre secondaire (Toplevel)

```
#include <toplevel.h>
Toplevel::Toplevel(QWidget* parent):QDialog(parent){
  this->setWindowTitle("CAI : Dialog Window");
  QVBoxLayout *layout= new QVBoxLayout();
  QLabel *image= new QLabel(this);
  image->setPixmap(QPixmap("pyqt.jpg"));
  QPushButton *button= new QPushButton("Hide me !",
                                       this);
  QWidget::connect(button,SIGNAL(clicked()),
                   this,SLOT(hide()));
  layout->addWidget(image);
  layout->addWidget(button);
  this->setLayout(layout);
}
```

Interaction

Configuration de projet(HelloWorld-2.pro)

```
QT += widgets

DEPENDPATH += . Include Src
INCLUDEPATH += . Include

HEADERS += Include/toplevel.h
SOURCES += Src/main.cpp Src/toplevel.cpp

TARGET = HelloWorld-2
```

Génération de Makefile, compilation, exécutable

```
logname@hostname} qmake -o Makefile HelloWorld-2.pro
logname@hostname} make
```

Interaction

Environnement de développement

```
{logname@hostname} tree
HelloWorld-2
|-- HelloWorld-2
|-- HelloWorld-2.pro
|-- Include
|   |-- toplevel.h
|-- main.o
|-- Makefile
|-- pyqt.jpg
|-- Src
|   |-- main.cpp
|   |-- toplevel.cpp
|-- toplevel.o
2 directories, 9 files
```

PyQt

Bindings pour Python

- `pyQt` : le plus ancien,développé par Riverbank Computing
- `pySide` : lancé par Nokia pour introduire une licence LGPL

PyQt vs Pyside

Hello World !

```
import sys
from PyQt5 import QtWidgets
# from PySide import QtWidgets

def gui(parent):
    button=QtWidgets.QPushButton("Click To Join",parent)
    button.move(100,50)
    button.setStyleSheet("background-color:yellow;")
```

PyQt

Hello World !

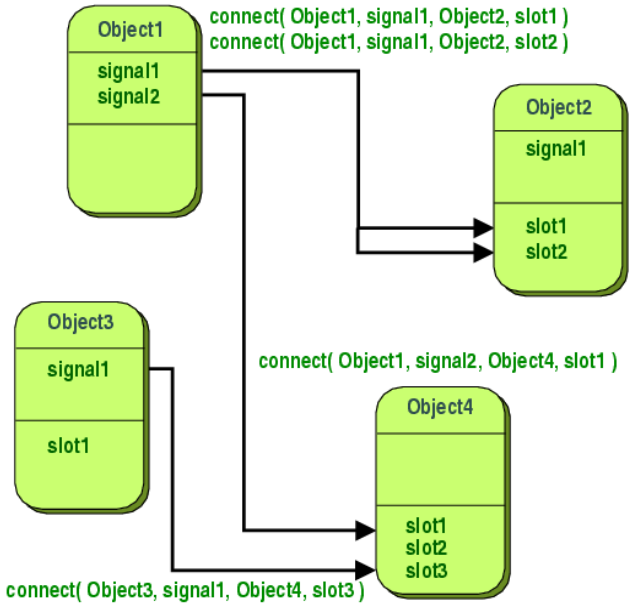
```
if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    widget = QtWidgets.QWidget()
    widget.setGeometry(300,150,300,400)
    widget.setWindowTitle("PyQt5 : Hello 1")
    gui(widget)
    widget.show()
    sys.exit(app.exec_())
```

- instancier un objet d'application (module `QtWidgets`)
- créer une fenêtre principale (composant `QtWidget`)
- système de fenêtrage (origine en haut à gauche)
- création d'arbre de composants graphiques

Signaux et slots

Communication entre composants

- changement d'état d'un objet : émission de signal
- réception de signal par un objet : déclenchement d'un slot
- un slot est un comportement (une méthode) à activer
- programmation par composants (modèle "multi-agents")



Signaux et slots

Héritage QWidget

```
class SliderLCD(QtWidgets.QWidget):
  def __init__(self, parent=None):
    QtWidgets.QWidget.__init__(self, parent)
    lcd=QtWidgets.QLCDNumber(self)
    slider=QtWidgets.QSlider(QtCore.Qt.Horizontal,self)
    slider.valueChanged.connect(lcd.display)
```

- PyQt4 :
  `connect(a,SIGNAL("signal(arg)",b, SLOT("slot(arg)")`
- PyQt5 :
  `a.signal.connect(b.slot)`

Signaux et slots

Héritage QWidget

```
        vbox=QtWidgets.QVBoxLayout()
        vbox.addWidget(slider)
        vbox.addWidget(lcd)
        self.setLayout(vbox)

if __name__ == "__main__" :
    app=QtWidgets.QApplication(sys.argv)
    w=SliderLCD()
    w.show()
    sys.exit(app.exec_())
```

Signaux et slots

Communication entre composants

- un signal, plusieurs slots et réciproquement
- l'émetteur n'a pas à connaître le récepteur et réciproquement
- l'émetteur ne sait pas si le signal est reçu (broadcast)
- un slot peut avoir moins de paramètres qu'un signal
- aspect central de la programmation Qt
- `SLOT`, `SIGNAL` macros : précompilation C++ (`moc`)

Signaux et slots

Héritage QObject

```
from PyQt5 import QtCore
from PyQt5.QtCore import pyqtSignal,pyqtSlot

class SigSlot (QtCore.QObject) :
  value_changed = pyqtSignal(int)

  def __init__(self,name):
    QtCore.QObject.__init__(self)
    self.value=0
    self.name=name
```

Déclaration d'un signal : `pyqtSignal(arg)`

Signaux et slots

Héritage QObject

```
def get_value(self) :
  return self.value
def set_value(self,v) :
  if DEBUG :
    print(type(self).__name__+".set_value()")
  if v!=self.value :
      if (v!=self.value) :
          print(f"set_value({self.name},{v})")
    self.value=v
    self.value_changed.emit(v)
```

Emission d'un signal `emit(arg)`

Signaux et slots

Héritage QObject

```
if __name__ == "__main__" :
  a,b=SigSlot("A"),SigSlot("B")
  a.value_changed.connect(b.set_value)
#  b.value_changed.connect(a.set_value)
  b.set_value(10)
  print(a.get_value()) # 0 or 10 ?
  a.set_value(100)
  print(b.get_value()) # 10 or 100  ?
```

Quel sera l'affichage à l'exécution de ce programme ?

Signaux et slots

Passage d'arguments

```
from PyQt5.QtCore import QObject, pyqtSignal, pyqtSlot
class TalkAndListen(QObject):
  signal_talk = pyqtSignal(str)
  def __init__(self,name):
    QObject.__init__(self)
    self.name=name
  def listen_to_me(self,text):
    self.signal_talk.emit(self.name+" who said : "+text)
  @pyqtSlot(str)
  def slot_listen(self,text):
    print(self.name+" listen to "+text)
```

Transmission de données entre composants :

- `pyqtSignal(arg)`,`pyqtSlot(arg)`

Signaux et slots

Passage d'arguments

```
if __name__ == "__main__" :
    talker = TalkAndListen("Dupont")
    listener=TalkAndListen("Durand")

    talker.signal_talk.connect(listener.slot_listen)
    talker.listen_to_me("Did you hear what I say !")

    listener.signal_talk.connect(talker.slot_listen)
    listener.listen_to_me("I'm not deaf !")
```

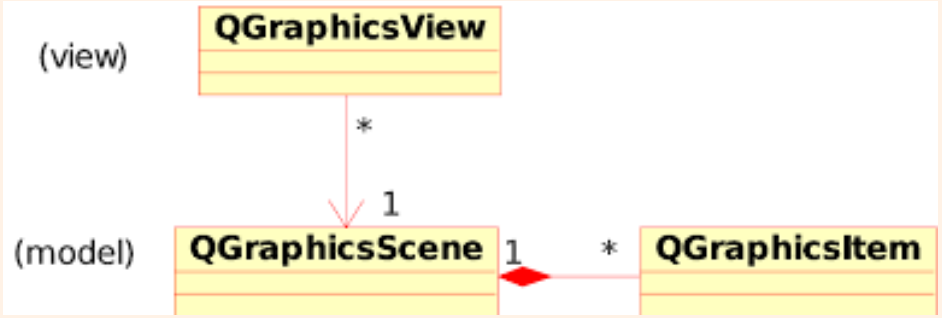
```
{logname@hostname} python talker.py
Durand listen to Dupont who said : Did you hear what I say !
Dupont listen to Durand who said : I'm not deaf !
```

QGraphics Framework

Gestion d'objets (items) graphiques

- `QGraphicsItem` : les items graphiques à créer
- `QGraphicsScene` : le conteneur d'items graphiques
- `QGraphicsView` : la visualisation du conteneur

Basé sur le modèle MVC



QGraphics Framework

Création de scène

```
import sys
from PyQt5 import QtCore,QtGui
from PyQt5.QtWidgets import QApplication,\
QGraphicsScene,QGraphicsView,QGraphicsItem
app=QApplication(sys.argv)
scene=QGraphicsScene()
#------------- scene creation --------------------
rect=scene.addRect(QtCore.QRectF(0,0,100,100))
rect.setFlag(QGraphicsItem.ItemIsMovable)
rect.setFlag(QGraphicsItem.ItemIsSelectable)
#-------------------------------------------------
view=QGraphicsView(scene)
view.show()
sys.exit(app.exec_())
```

QGraphics Framework

Dimension de scène

```
def bounding_rect(scene) :
  top=QtCore.QLineF(scene.sceneRect().topLeft(),
                    scene.sceneRect().topRight())
  left=QtCore.QLineF(scene.sceneRect().topLeft(),
                     scene.sceneRect().bottomLeft())
  right=QtCore.QLineF(scene.sceneRect().topRight(),
                      scene.sceneRect().bottomRight())
  bottom=QtCore.QLineF(scene.sceneRect().bottomLeft(),
                       scene.sceneRect().bottomRight())
  pen = QtGui.QPen(QtCore.Qt.red)
  scene.addLine(top,pen)
  scene.addLine(left,pen)
  scene.addLine(right,pen)
  scene.addLine(bottom,pen)
```

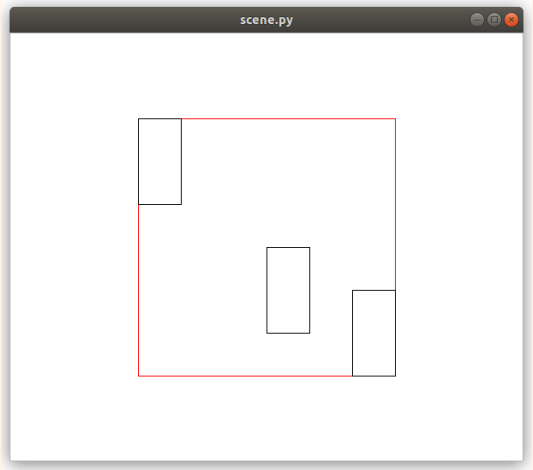
QGraphics Framework

Dimension de scène

```
view = QtWidgets.QGraphicsView()
view.setGeometry(QtCore.QRect(0,0,600,500))
#------------- scene creation --------------------
scene.setSceneRect(-150,-150,300,300)
bounding_rect(scene)
view.setScene(scene)
item=QtWidgets.QGraphicsRectItem(-150,-150,50,100)
scene.addItem(item)
item=QtWidgets.QGraphicsRectItem(0,0,50,100)
scene.addItem(item)
item=QtWidgets.QGraphicsRectItem(100,50,50,100)
scene.addItem(item)
#-------------------------------------------------
```

QGraphics Framework

Dimension de scène



En fixant les mêmes dimensions pour la vue et la scène :

```
view.setGeometry(QtCore.QRect(0,0,600,500))
scene.setSceneRect(0,0,600,500)
```

Quelle serait la visualisation ?

Graphics Framework

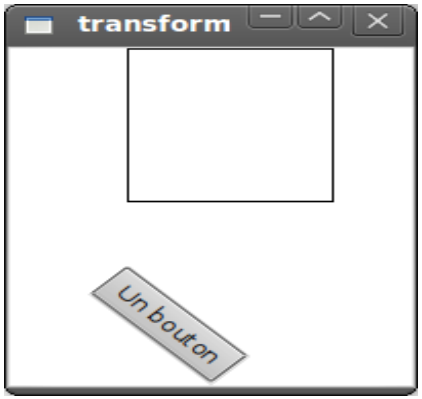
Transformations géométriques

```
#------------- scene creation ------------------
rect=scene.addRect(QtCore.QRectF(0, 0, 100, 100))
rect.setFlag(QGraphicsItem.ItemIsMovable)
button=QPushButton("Un bouton")
proxy=QGraphicsProxyWidget()
proxy.setWidget(button)
scene.addItem(proxy)
scene.setSceneRect(0,0, 300, 300)
matrix=QtGui.QTransform()
matrix.rotate(45)
matrix.translate(100,0)
matrix.scale(1,2)
proxy.setTransform(matrix);
#-----------------------------------------------
```

Graphics Framework

Intégration d'applications

```
#------------- scene creation -------------------
web = QWebView()
web.load(QtCore.QUrl("http://www.enib.fr"))
rect=scene.addRect(QtCore.QRectF(0, 0, 100, 100))
proxy = QGraphicsProxyWidget()
proxy.setWidget(web)
scene.addItem(proxy)
#------------------------------------------------
```



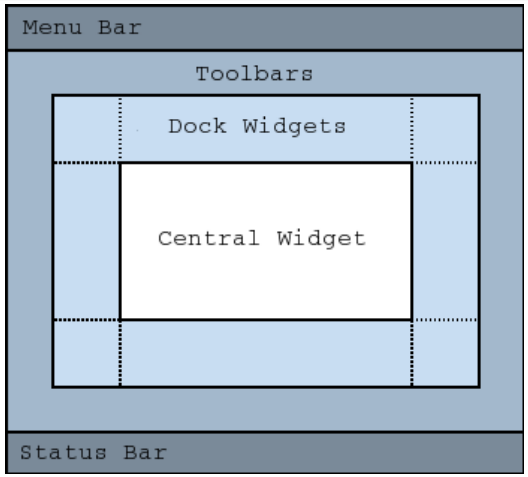
Éditeur Graphique

Fenêtre principale

- barre d'actions avec zone cliente : `QMainWindow`
- zone cliente pour éditer : `QTextEdit`, `QGraphicsView` ...

`QMainWindow`

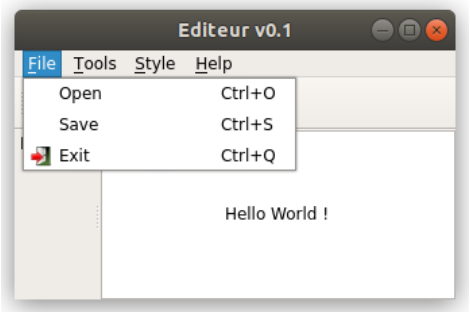
- barres de menu, d'outils, de statut
- zone centrale (cliente)
- autres fonctionnalités

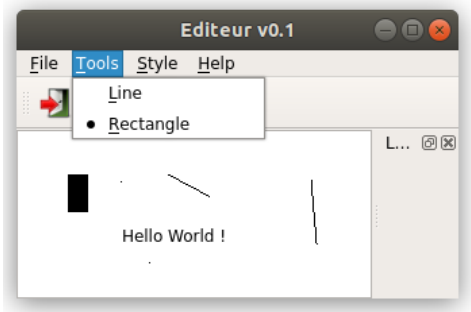


Éditeur Graphique

Programme d'application

```
if __name__ == "__main__" :
    app = QtWidgets.QApplication(sys.argv)
    view=View()       # zone cliente
    position=0,0
    dimension=600,400
    main=MainWindow(view,position,dimension)
    main.show()
    sys.exit(app.exec_())
```





Fenêtre principale

Héritage `MainWindow`

```
class MainWindow(QtWidgets.QMainWindow):
  def __init__(self,view,\
               position=(0,0),dimension=(500,300)):
    QtWidgets.QMainWindow.__init__(self)
    self.setWindowTitle("CAI : Editeur Graphique")
    self.view=view
    scene=QtWidgets.QGraphicsScene()
    self.view.setScene(scene)
    self.setCentralWidget(self.view)
```

A l'initialisation :

- zone cliente : `setCentralWidget(self.view)`
- scene associée : `self.view.setScene(scene)`

Fenêtre principale

Héritage `MainWindow`

```
x,y=position
w,h=dimension
self.view.setGeometry(x,y,w,h)
scene.setSceneRect(x,y,w,h)
self.create_actions()
self.connect_actions()
self.create_menus()
```

A l'initialisation :

- création de toutes les actions (`create_actions()`)
- liaison actions-signal-comportements (`connect_actions()`)
- création des menus de la barre d'actions (`create_menus()`)

Fenêtre Principale

Création des actions

```
def create_actions(self) :
  self.action_file_open=QtWidgets.QAction(
        QtGui.QIcon('Icons/open.png'),"Open",self)
  self.action_file_open.setShortcut("Ctrl+O")
  self.action_file_open.setStatusTip("Open file")
  # to be ctnd (save,save as ...)
```

À une action on peut associer :

- une image (`QIcon`), utile pour la barre d'outils
- un raccourci-clavier, utile pour se dispenser de la souris
- un message, utile pour l'affichage dans la barre d'état
- ...

Fenêtre Principale

Création des actions

```
self.action_tools=QtWidgets.QActionGroup(self)
self.action_tools_line=QtWidgets.QAction(
self.tr("&Line"),self)
self.action_tools_line.setCheckable(True)
self.action_tools_line.setChecked(True)
self.action_tools.addAction(self.action_tools_line)
# to be ctnd (rectangle, ellipse ...)
```

On peut également regrouper les actions (`QActionGroup()`) :

- item graphiques, alignement de texte, ...
- en rendant visible la sélection courante (checking)
- mis à jour automatique an cas d'action utilisateur
- ...

Fenêtre Principale

Barres d'actions, outils, statut

```
def create_menus(self) :
  menubar = self.menuBar()
  menu_file = menubar.addMenu("&File")
  menu_file.addAction(self.action_file_open)
  menu_file = menubar.addMenu("&Tools")
  menu_file.addAction(self.action_tools_line)
  # to be ctnd (add actions in menus)
 toolbar=self.addToolBar("Tools")
 toolbar.addAction(self.action_tools_line)
 statusbar=self.statusBar()
```

- `QMenuBar` : toutes les actions possibles
- `QToolBar` : actions les plus fréquentes
- `QStatusBar` : information sur l'action courante

Fenêtre Principale

Connexion action/comportement (Signal/Slot)

```
def connect_actions(self) :
  self.action_file_open.triggered.connect(
            self.file_open
  )
  self.action_tools_line.triggered.connect(
    lambda checked,tool="line": \
            self.tools_selection(checked,tool)
  )
  self.action_tools_rect.triggered.connect(
    lambda checked, tool="rectangle": \
            self.tools_selection(checked,tool))
  ...
```

`action.signal.connect(lambda arg1,arg2=value,...: slot(arg1,arg2,...))`

Fenêtre Principale

Définition des comportements (Slot)

```
def file_open(self):
  filename = QtWidgets.QFileDialog.getOpenFileName(self,
   '"pen File",os.getcwd())
  fileopen=QtCore.QFile(filename[0])
  ...
def tools_selection(self,checked,tool) :
  print("checked : ",checked)
  print("tool : ",tool)
  self.view.select_tool(tool)
```

Zone Cliente

Héritage QGraphicsView

```
class View (QtWidgets.QGraphicsView) :
  def __init__(self,position=(0,0),dimension=(600,400)):
    QtWidgets.QGraphicsView.__init__(self)
    x,y=position
    w,h=dimension
    self.setGeometry(x,y,w,h)
    self.begin=QtCore.QPoint(0,0)
    self.end=QtCore.QPoint(0,0)
```

Propriétés de la zone cliente (QGraphicsView)

- position,dimension de la fenêtre
- coordonnées de début,fin de traçé

Zone Cliente

Héritage QGraphicsView

```
self.item,self.offset=None,QtCore.QPoint(0,0)
self.tool="line"
self.pen,self.brush=None,None
self.create_style()
```

Propriétés de la zone cliente (QGraphicsView)

- item sélectionné pour déplacement (self.item)
- offset coordonnées item-souris (self.offset)
- type d'item graphique à dessiner (self.tool)
- création du style de dessin (QPen,QBrush)

Zone Cliente

Propriétés de dessin

```
def create_style(self) :
  self.create_pen()
  self.create_brush()
def create_pen(self) :
  self.pen=QtGui.QPen()
  self.pen.setColor(QtCore.Qt.red)
def create_brush(self) :
  self.brush=QtGui.QBrush()
  self.brush.setColor(QtCore.Qt.blue)
  self.brush.setStyle(QtCore.Qt.CrossPattern)
```

Style de dessin à l'initialisation

- `QPen` : crayon rouge
- `QBrush` : remplissage en bleu avec motif en croix

Zone Cliente

Propriétés de dessin

```
def select_tool(self,tool) :
  self.tool=tool
def select_pen_color(self,color) :
  self.pen.setColor(color)
def select_brush_color(self,color) :
  self.brush.setColor(color)
```

Modification des sélections

- d'item graphique à dessiner
- de couleur de crayon
- de couleur de remplissage
- ...

Zone Cliente

Gestion des événements

```
def mousePressEvent(self, event):
  self.begin=self.end=event.pos()
  if self.scene() :
    self.item=self.scene().itemAt(
                    self.begin,QtGui.QTransform())
    if self.item :
      self.offset =self.begin-self.item.pos()
    # ...
```

Début de dessin :

- initialisation coordonnées début-fin de tracé
- si une scène est associée à la vue
  - détection de collision avec un item graphique
  - si collision : calcul de l'offset coordonnées item-souris
  - sinon ...

Zone Cliente

Gestion des événements

```
def mouseMoveEvent(self, event):
  self.end=event.pos()
  if self.scene() :
    if self.item :
      self.item.setPos(event.pos() - self.offset)
    else :
      print("draw bounding box !")
```

Traçé en cours :

- modification de coordonnées de fin de traçé
- si une scène est associée à la vue
  - si un item est en cours de déplacement, le déplacer
  - sinon afficher le contour de l'objet à dessiner
  - ...

Zone Cliente

Gestion des événements

```
def mouseReleaseEvent(self, event):
  self.end=event.pos()
  if self.scene() :
    if self.item :
      self.item.setPos(event.pos() - self.offset)
      self.item=None
    elif self.tool=="line" :
```

Fin de traçé :

- modification de coordonnées de fin de traçé
- si une scène est associée à la vue
  - si un item était sélectionné, le déselectionner
  - sinon suivant l'item à créer

Zone Cliente

Gestion des événements

```
        elif self.tool=="line" :
          line=QtWidgets.QGraphicsLineItem(
                              self.begin.x(),self.begin.y(),
                              self.end.x(),self.end.y())
          line.setPen(self.pen)
          self.scene().addItem(line)
        elif self.tool=="rectangle" :
          # ...
```

Fin de traçé :

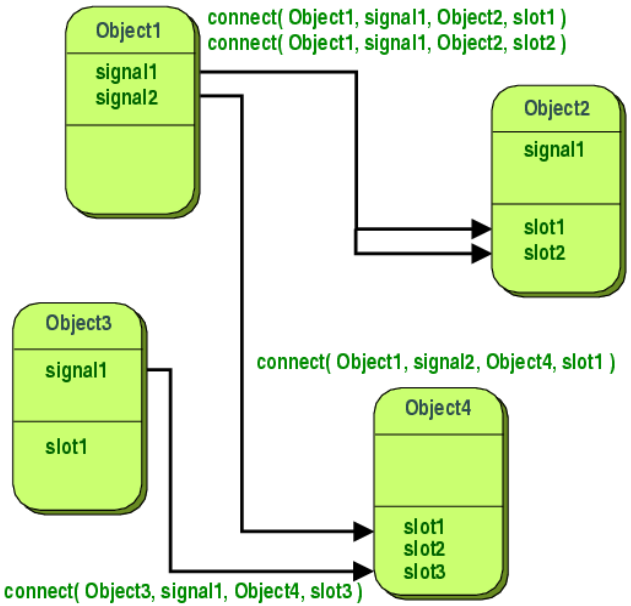
- créer l'item graphique (`QGraphics...Item()`) sélectionné
- lui associer les styles(`QPen,QBrush`) sélectionnés
- ajouter l'item dans la scène
- ...

Annexe : Signaux et slots

Signaux et slots

Communication entre composants

- changement d'état d'un objet : émission de signal
- réception de signal par un objet : déclenchement d'un slot
- un slot est un comportement (une méthode) à activer
- programmation par composants, modèle "multi-agents"



Annexe : Signaux et slots

Signaux et slots

- modulaire, flexible
  - un signal, plusieurs slots et réciproquement
  - l'émetteur n'a pas à connaître le récepteur et réciproquement
  - l'émetteur ne sait pas si le signal est reçu (broadcast)
- transmission de données
  - typage fort : les types de données doivent être les mêmes
  - un slot peut avoir moins de paramètres
- remarques
  - différent des mécanismes de *callbacks, listeners*
  - aspect central de la programmation Qt
  - `SLOT, SIGNAL` sont des macros : précompilation (`moc`)

Annexe : Signaux et slots

Mots-clés Qt : Q_OBJECT,slots,signals

```
#include <QObject>
class SigSlot : public QObject {
Q_OBJECT
  public:
    SigSlot():_value(0) {}
    int  getValue() const {return _value;}
  public slots:
    void setValue(int);
  signals:
    void valueChanged(int);
  private:
    int  _value;
};
```

Annexe : Signaux et slots

classe SigSlot : implémentation

```
#include "sigslot.h"
void SigSlot::setValue(int v) {
   if (v!=_value) {
    _value=v;
    emit valueChanged(v);
  }
}
```

Emission de signal : emit

- valueChanged(v) : avec la nouvelle valeur v
- v != _value : si cette dernière a changé

Annexe : Signaux et slots

Connexion : QObject::connect()

```
#include <QDebug>
#include <QPushButton>
#include "sigslot.h"
int main(int argc,char* argv[]) {
SigSlot a,b;
QObject::connect(&a,SIGNAL(valueChanged(int)),\
                 &b,SLOT(setValue(int)));
// QObject::connect(&b,SIGNAL(valueChanged(int)),\
//                  &a,SLOT(setValue(int)));
b.setValue(10);
qDebug() << a.getValue(); // 0 or 10 ?
a.setValue(100);
qDebug() << b.getValue(); // 10 or 100 ?
}
```

Annexe : Signaux et slots

Environnement de développement

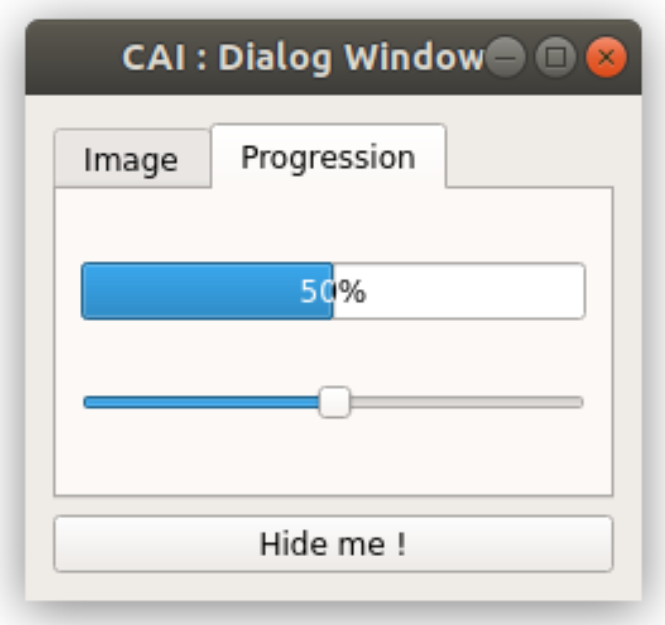
```
SigSlot
|-- Include
|   |-- sigslot.h
|-- main.o
|-- Makefile
|-- moc_sigslot.cpp
|-- moc_sigslot.o
|-- SigSlot
|-- sigslot.o
|-- SigSlot.pro
|-- Src
|   |-- main.cpp
|   |-- sigslot.cpp
2 directories, 10 files
```

Annexe : Toplevel

Fenêtre secondaire (Toplevel)

```
//class Toplevel:public QDialog{
class Toplevel:public QWidget{
Q_OBJECT
  public :
   Toplevel(QWidget* parent);
  protected :
   QWidget* imageTab(void);
   QWidget* progressTab(void);
  private :
   QTabWidget* _tabs;
};
```

Annexe : Toplevel



Communication entre widgets

- `QSlider` : signal `sliderMoved(int)`
- `QProgressBar` : slot `setValue(int)`

Annexe : Toplevel

Fenêtre secondaire : héritage QDialog ou QWidget

```
// QDialog is a toplevel widget :
// Toplevel::Toplevel(QWidget* parent):QDialog(parent)
// QWidget with parent is not a toplevel widget :
// Toplevel::Toplevel(QWidget* parent):QWidget(parent)
Toplevel::Toplevel(QWidget* parent)
{
  this->setWindowTitle("CAI : Dialog Window");
  _tabs= new QTabWidget(this);
  _tabs->addTab(this->imageTab(),"Image");
  _tabs->addTab(this->progressTab(),"Progression");
  ...
}
```

Annexe : Toplevel

Création d'onglets : affichage de QPixmap dans un QLabel

```
QWidget *Toplevel::imageTab(void) {
  float width=200;
  float height=100;
  QWidget* onglet=new QWidget();
  QVBoxLayout *vbox=new QVBoxLayout();
  QLabel* image=new QLabel();
  QPixmap pixmap("pyqt.jpg");
  image->setPixmap(pixmap.scaled(width,height));
  vbox->addWidget(image);
  onglet->setLayout(vbox);
  onglet->setStyleSheet("background-color:black;");
  return onglet;
}
```

Annexe : Toplevel

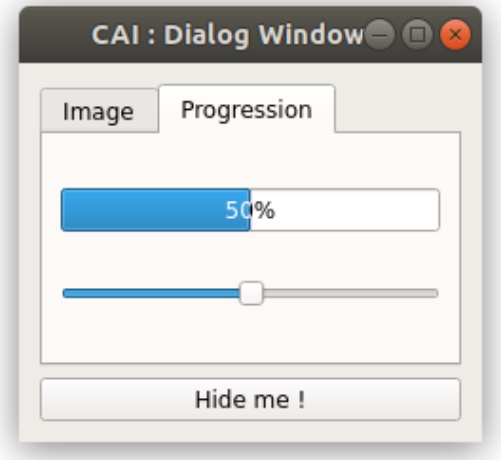
Création d'onglets : communication QSlider/QProgressBar

```
QWidget* Toplevel::progressTab(void) {
  float value=50;
  QProgressBar* progress = new QProgressBar();
  progress->setValue(value);
  QSlider* slider = new QSlider(Qt::Horizontal);
  slider->setValue(value);
  QObject::connect(slider, SIGNAL(sliderMoved(int)),\
                   progress, SLOT(setValue(int)) );
```

Annexe : Toplevel

Création d'onglets : communication Slider/ProgressBar

```
  QWidget* onglet = new QWidget();
  QVBoxLayout *vbox = new QVBoxLayout();
  vbox->addWidget(progress);
  vbox->addWidget(slider);
  onglet->setLayout(vbox);
  return onglet;
}
```



Bibliographie

Adresses "au Net"

- `Qt` : documentation officielle
- `pyQt` : Riverbank Computing
- `pySide` : Qt for Python
- Livres : Developpez.com : Les meilleurs livres Qt
- Articles : Qt Developer Network (traduction)
- Cours : Thierry Vaira
- Cours : Eric Lecolinet