Construction d'IHM avec PyQt

Alexis NÉDÉLEC

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 sur lequel repose l'environnement KDE
- toolkit Graphique C++
- Evolution de Qt1 à Qt5 + QtQuick :
  - TrollTech, Qt Software, Nokia, Digia ...
  - `http://www.qt.io/developers`
- licences GNU LGPL, commerciale
- multiplateformes : OS classiques et mobiles
- devise : "write once, compile anywhere"

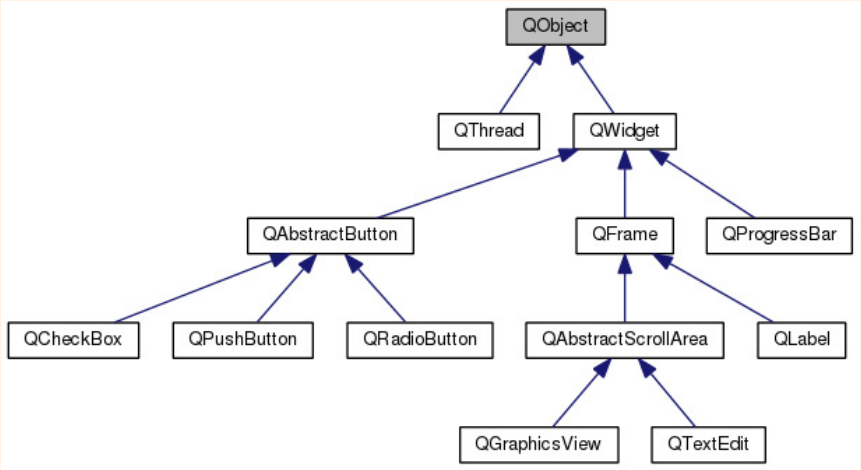
Qt API

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

- `Qt Core` : classes de base pour tous les modules
- `Qt GUI` : composants graphiques 2D et 3D (OpenGL)
- `Qt Multimedia` : audio, video, radio et caméra
- `Qt NetWork` : faciliter la programmation réseaux
- `Qt QML` : pour les langages QML et javascript
- `Qt Quick` : création d'applications de manière déclarative
- `Qt SQL` : connexion, manipulation SGBD relationnels
- `Qt Test` : pour faire des test unitaires
- `Qt Widgets` : extension des fonctionnalités GUI
- ...

Qt API

Modules Qt Add-Ons

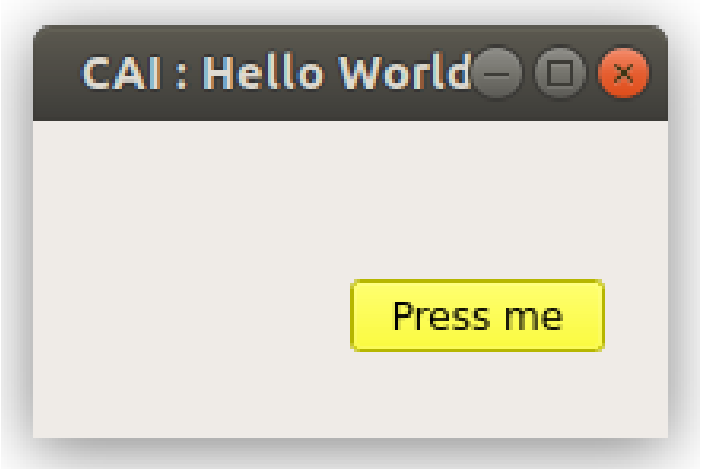
- `Qt 3D` : développement d'applications 3D
- `Qt Android Extras` : API spécifique pour Androïd
- `Qt Bluetooth` : Android, iOS, Linux, macOS, WinRT
- `Qt SCXML` : création de "State Machine" dans des applications
- `Qt Sensors` : données capteurs, reconnaissance de gestes
- `Qt Speech` : pour faire de la synthès vocale (text2speech)
- `Qt SVG` : affichage de contenu XML 2D
- `Qt XML` : SAX et DOM sur documents XML
- `Qt XML Pattern` : XPath, XQuery, XSLT,schemas XML ...
- ...

`https://doc.qt.io/qt-5/index.html`

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");
```

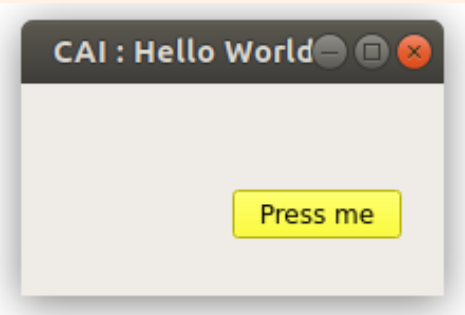


Hello World

Composant graphique (main.cpp)

```
 QPushButton *button= new QPushButton("Press me",\
                                      &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

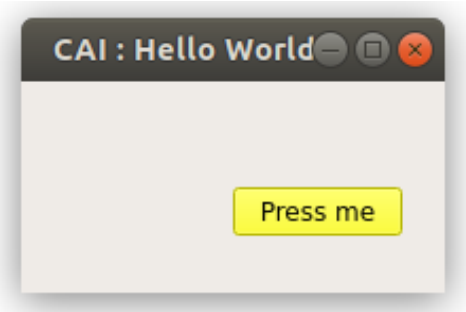
Signaux et slots

```
int main(int argc,char *argv[]){
...
  QPushButton *button= new QPushButton("Press me ", \
                                       &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

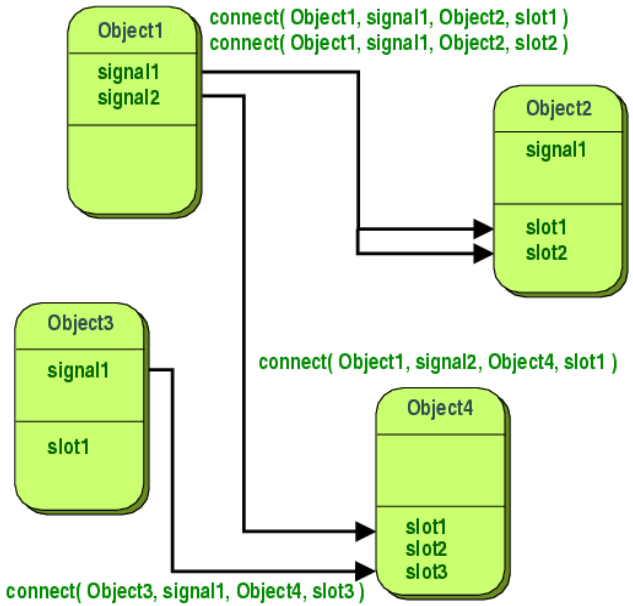
```
{logname@hostname} tree
HelloWorld-2
|-- HelloWorld-2
|-- HelloWorld-2.pro
|-- Include
|   |-- toplevel.h
|-- main.o
|-- Makefile
|-- pyqt.jpg
|-- Src
|   |-- main.cpp
|   |-- toplevel.cpp
|-- toplevel.o
2 directories, 9 files
```

Qt C++

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"



Qt C++

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`)

Héritage QObject

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;
};
```

Héritage QObject

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

Héritage QObject

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 ?
}
```

Héritage QObject

Environnement de développement

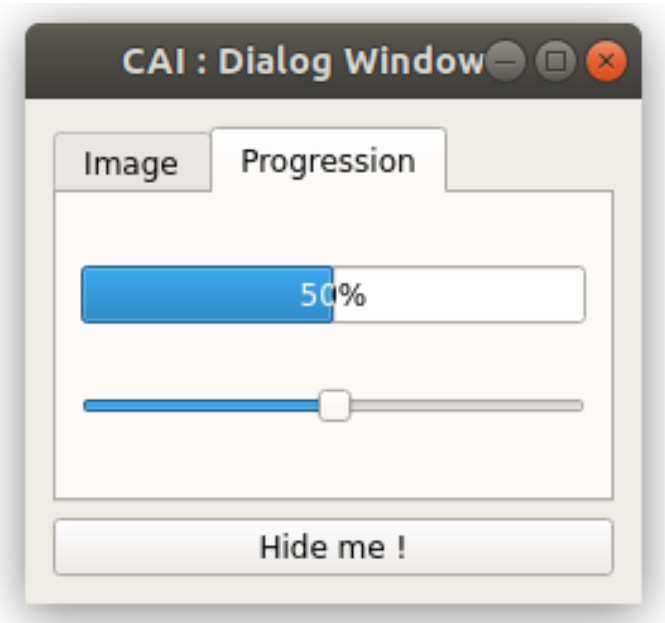
```
SigSlot
|-- Include
|   |-- sigslot.h
|-- main.o
|-- Makefile
|-- moc_sigslot.cpp
|-- moc_sigslot.o
|-- SigSlot
|-- sigslot.o
|-- SigSlot.pro
|-- Src
|   |-- main.cpp
|   |-- sigslot.cpp
2 directories, 10 files
```

Héritage QWidget

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;
};
```

Héritage QWidget



Communication entre widgets

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

Héritage QWidget

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");
  ...
}
```

Héritage QWidget

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;
}
```

Héritage QWidget

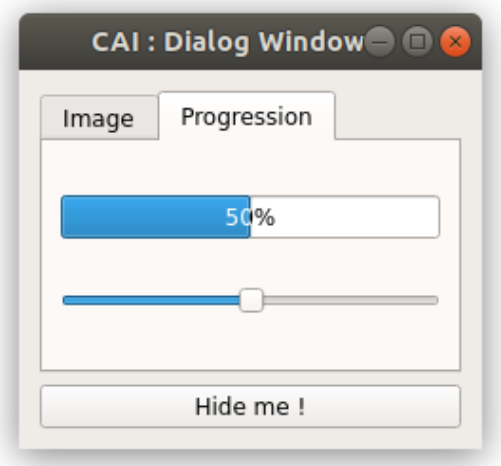
Création d'onglets : communication QSlider/QProgressBar

```
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)) );
```

Héritage QWidget

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;
}
```



Événements

Héritage QWidget

```
class Scribble : public QWidget {
 public:
  Scribble(QWidget *parent=0);
 protected:
   ...
   void mouseReleaseEvent(QMouseEvent *event) override;
   ...
 private:
     QPoint _start;
     QPoint _end;
     int _released;
     ...
};
```

Événements

surdéfinition de méthodes

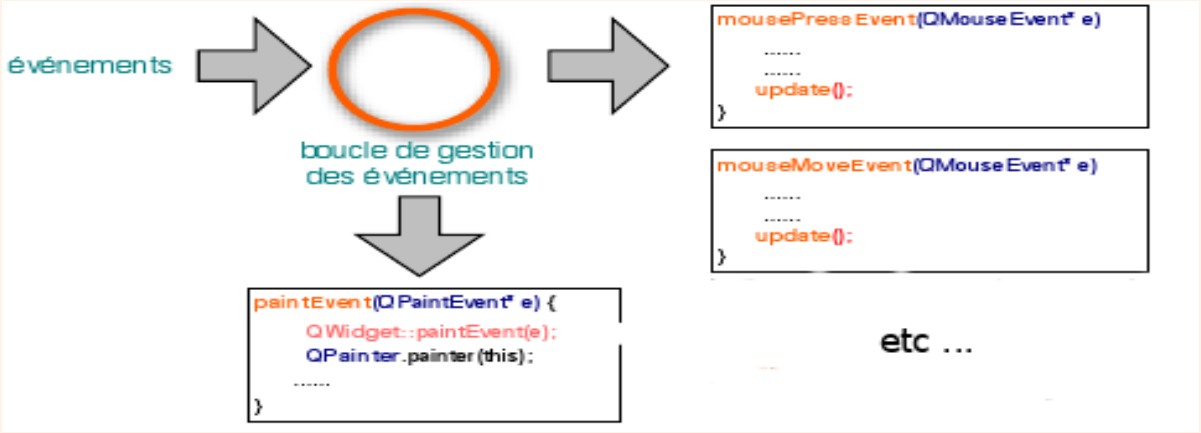
- `void mousePressEvent(QMouseEvent* event);`
- `void mouseMoveEvent(QMouseEvent* event);`
- `void mouseReleaseEvent(QMouseEvent* event);`
- `void mouseDoubleClickEvent(QMouseEvent* event);`
- ...

gestion de l'affichage

- demande d'affichage, de rafraîchissement : `update()`
- modification d'affichage : `paintEvent()`

Événements

boucle dévénements



demande de rafraîchissement

```
void Scribble::mouseReleaseEvent(QMouseEvent *event) {
  if (event->button()==Qt::LeftButton) {
    // TODO when left button is released
    _released=true
  }
  update();
}
```

Événements

affichage dans le widget (painter(this))

```
void Scribble::paintEvent(QPaintEvent *event)
  if (_released==true) {
    QPainter painter(this);
    painter.setPen(QPen(_color,_penWidth,\
                        Qt::SolidLine,\
                        Qt::RoundCap,\
                        Qt::RoundJoin\
                       )
                   );
    painter.drawLine(_start,_end);
  }
}
```

Événements

Affichage : classes de base

- `QPainter` : outil pour dessiner
- `QPaintDevice` : surface pour dessiner
- `QPaintEngine` : moteur de rendu entre l'outil et la surface

Affichage : héritage de `QpaintDevice`

- `QWidget`
- `QPixmap, QImage, QPicture`
- `QPrinter`
- ...

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 !

```
from PyQt5 import QtGui
# from PySide import QtGui
import sys

app = QtGui.QApplication(sys.argv)
hello = QtGui.QPushButton("Hello World!", None)
hello.show()
app.exec_()
```

PyQt

importation de modules Qt

```
import sys
from PyQt5 import QtCore,QtGui,QtWidgets
// TODO  : class Scribble

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    mw = Scribble()
    mw.resize(300,200)
    mw.show()
    app.exec_()
```

Événements

Héritage QWidget

```
class Scribble(QtWidgets.QWidget):
  def __init__(self):
    super().__init__()
    self.start=QtCore.QPoint(0,0)
    self.end=QtCore.QPoint(0,0)
    self.pen_color=QtCore.Qt.blue;
    self.pen_width=3;
```

Événements

Héritage QWidget

```
def mousePressEvent(self,event) :
  if event.button() == QtCore.Qt.LeftButton :
    self.start = self.end = event.pos();
def mouseMoveEvent(self,event) :
  if event.buttons() & QtCore.Qt.LeftButton :
    self.end = event.pos();
  self.update()
def mouseReleaseEvent(self,event) :
  if  event.button()== QtCore.Qt.LeftButton :
    self.update()
```

Événements

Héritage QWidget

```
  def paintEvent(self,event) :
    painter=QtGui.QPainter(self)
    painter.setPen(QtGui.QPen(
                                self.pen_color,\
                                self.pen_width,\
                                QtCore.Qt.SolidLine,\
                                QtCore.Qt.RoundCap,\
                                QtCore.Qt.RoundJoin\
                               )
                   )
    painter.drawLine(self.start,self.end);
  def resizeEvent(self,event) :
    print(self.width(),self.height())
```

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

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

Signaux et slots

Héritage QObject

```
def get_value(self) :
  return self.value

def set_value(self,v) :
 if (v!=self.value) :
  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(),SigSlot()
  QtCore.QObject.connect(a,\
            QtCore.SIGNAL('value_changed(int)'),\
                        b,\
            QtCore.SLOT('set_value(int)') )
  b.set_value(10)
  print(a.get_value()) # 0 or 10 ?
  a.set_value(100)
  print(b.get_value()) # 10 or 100  ?
```

Connexion signal/slot en version PyQt4 :

- `connect(a,SIGNAL("signal(arg)",b, SLOT("slot(arg)")`

Signaux et slots

Héritage QObject

```
if __name__ == "__main__" :
  a,b=SigSlot(),SigSlot()
  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  ?
```

Connexion signal/slot en version PyQt5 :

- `a.signal.connect(b.slot)`

Signaux et slots

Connexion signal/slot

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

Signaux et slots

Connexion signal/slot

```
    slider.valueChanged.connect(lcd.display)
    # self.connect(slider,\
    #                 QtCore.SIGNAL('valueChanged(int)'),\
    #                lcd,\
    #                 QtCore.SLOT('display(int)') )
    # self.resize(250, 150)
if __name__ == "__main__" :
  app = QtWidgets.QApplication(sys.argv)
  qb = SliderLCD()
  qb.show()
  sys.exit(app.exec_())
```

Signaux et slots

Connexion signal/slot

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

class TalkAndListen(QObject):
  signal_talk = pyqtSignal(str)
  def __init__(self):
    QObject.__init__(self)
  def listen_to_me(self,text):
    self.signal_talk.emit(text)
  @pyqtSlot(str)
  def slot_listen(self,text):
    print("You say : " + text)
```

Création de signaux et de slots :

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

Signaux et slots

Connexion signal/slot

```
if __name__ == "__main__" :
  talker = TalkAndListen()
  listener=TalkAndListen()
  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
You say : Did you hear what I say !
You say : I'm not deaf !
```

Signaux et slots

Passage d'arguments : fonctions anonymes (lambda)

```
class Keypad(QtWidgets.QWidget):
  def __init__(self,nbuttons=10,parent=None) :
    super(Keypad, self).__init__()
    self.layout = QtWidgets.QVBoxLayout(self)
    self.buttons=[]
    self.create_buttons(nbuttons)
   ...
```

Signaux et slots

Passage d'arguments : fonctions anonymes (lambda)

```
def create_buttons(self,number) :
  for i in range(number) :
    button=QtWidgets.QPushButton(str(i),self)
    button.clicked.connect(lambda state,x=i:\
                           self.on_selected(state,x))
    self.buttons.append(button)
    self.layout.addWidget(button)

def on_selected(self,state,index):
  print('state', state)
  print('index', index)
```

Signaux et slots

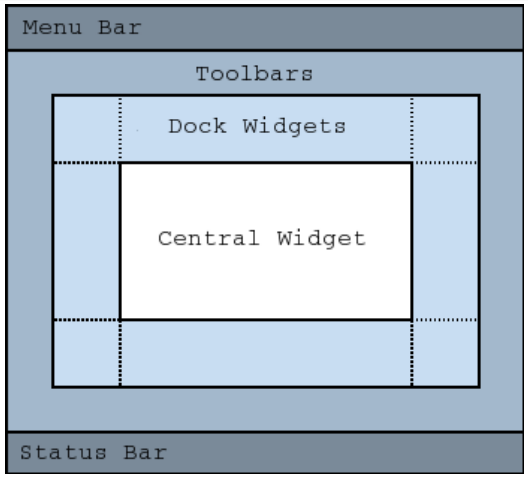
Passage d'arguments : fonctions anonymes (lambda)

```
if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    keypad = Keypad(5)
    keypad.show()
    sys.exit(app.exec_())
```

Fenêtre principale

QMainWindow

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



Création d'une fenêtre principale

- héritage de `QMainWindow`
- création des zones de travail dans le constructeur

QMainWindow

Héritage

```
class MainWindow(QtWidgets.QMainWindow):
  def __init__(self):
    QtWidgets.QMainWindow.__init__(self)
    self.resize(500,300)
    self.setWindowTitle("Editeur v0.1")
    self.create_scene()
    self.create_actions()
    self.create_menus()
    self.connect_actions()
```

Zone cliente : self.create_scene()

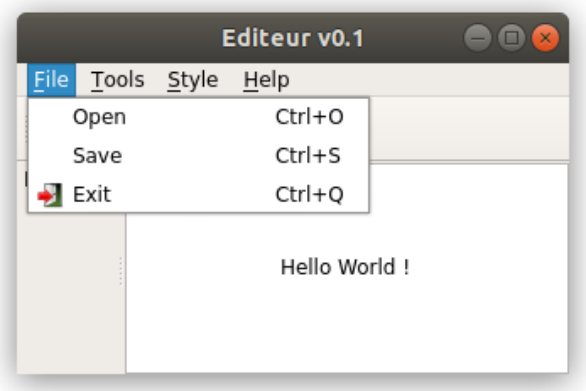
- QMainWindow.setCentralWidget(widget)
- Editeur : QTextEdit, QGraphicsView ...

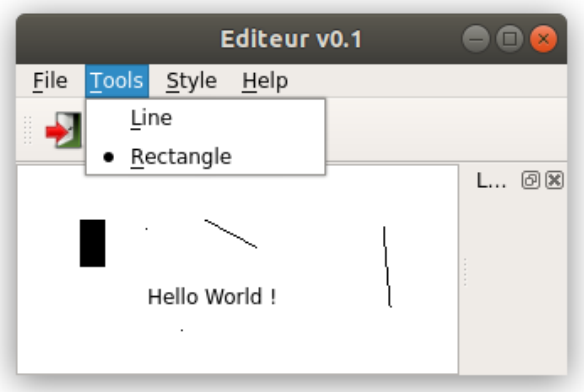
QMainWindow

Création de la scène

```
def create_scene(self) :
  view=QtWidgets.QGraphicsView()
  self.area=Scene(self)
  text= self.area.addText("Hello World !")
  text.setPos(0,0)
  text.setVisible(True)
  view.setScene(self.area)
```

QMainWindow





Zone client, palette d'outils

```
self.setCentralWidget(view)
self.dock=QtWidgets.QDockWidget("Left Right Dock",\
                                self)
self.dock.setAllowedAreas(\
            QtCore.Qt.LeftDockWidgetArea\
          | QtCore.Qt.RightDockWidgetArea )
self.addDockWidget(\
            QtCore.Qt.LeftDockWidgetArea,\
            self.dock )
```

QMainWindow

Création d'actions

```
def create_actions(self) :
  self.action_save = QtWidgets.QAction(\
        QtGui.QIcon('icons/save.png'),"Save",self)
  self.action_save.setShortcut("Ctrl+S")
  self.action_save.setStatusTip("Save to file")
  self.action_open = QtWidgets.QAction(\
        QtGui.QIcon('icons/open.png'),"Open",self)
  self.action_open.setShortcut("Ctrl+O")
  self.action_open.setStatusTip("Open file")
  ...
```

QMainWindow

Création de groupe d'actions

```
self.group_action_tools=QtWidgets.QActionGroup(self)
self.action_line=QtWidgets.QAction(\
        self.tr("&Line"),self)
self.action_line.setCheckable(True)
self.action_line.setChecked(True)
self.action_rect=QtWidgets.QAction(\
        self.tr("&Rectangle"),self)
...
self.group_action_tools.addAction(self.action_line)
self.group_action_tools.addAction(self.action_rect)
...
```

QMainWindow

Barre d'Actions (QMenuBar)

```
def create_menus(self) :
    menubar = self.menuBar()
    menu_file = menubar.addMenu("&File")
    menu_file.addAction(self.action_open)
    ...
    menu_tools = menubar.addMenu("&Tools")
    menu_tools.addAction(self.action_line)
    ...
```

Barre d'outils (QToolBar)

```
toolbar=self.addToolBar("Exit")
toolbar.addAction(self.action_exit)
```

QMainWindow

Comportements liés aux actions

```
def connect_actions(self) :
  self.action_open.triggered.connect(self.file_open)
  self.action_save.triggered.connect(self.file_save)
  self.action_exit.triggered.connect(self.file_exit)
  ...
  self.action_about.triggered.connect(self.help_about)
  self.action_pen_color.triggered.connect(
    self.pen_color_selection )
  self.action_brush_color.triggered.connect(
    self.brush_color_selection )
```

QMainWindow

Comportements liés aux actions

```
self.action_line.triggered.connect(
  lambda checked,tool="line":
          self.set_action_tool(tool) )
self.action_rect.triggered.connect(
  lambda checked, tool="rectangle":
          self.set_action_tool(tool))
...
```

Transmission de données aux comportements

QMainWindow

Comportements liés aux actions

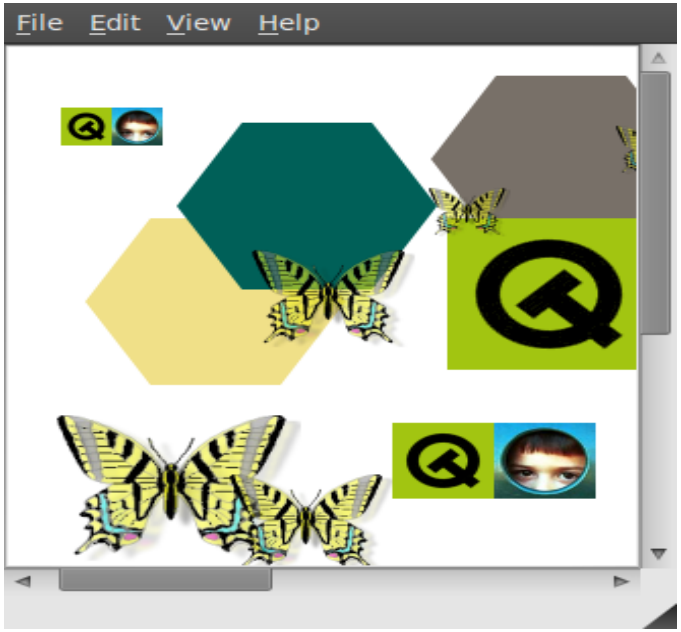
```
def file_exit(self):
  exit(0)
def help__about(self):
  QtWidgets.QMessageBox.information(self,\
            self.tr("About Me"),\
            self.tr("Jean Dupond\n copyright 2019") )
def pen_color_selection(self):
  color = QtWidgets.QColorDialog.getColor(\
                                QtCore.Qt.yellow,self )
  if color.isValid() :
    print("Color Choosen : ",color.name())
    self.area.set_pen_color(color)
def set_action_tool(self,tool) :
  self.area.set_tool(tool)
```

Graphics View Framework

Basé sur le modèle MVC

Dessiner des objets, remplace la classe `QCanvas` de Qt3

- `QGraphicsScene` : la scène
- `QGraphicsView` : les vues
- `QGraphicsItem` : les objets



Graphics View Framework

QGraphicsScene

Conteneur d'objets (items) graphiques

- gérer un grand nombres d'éléments graphiques
- propager les évènements aux objets graphiques
- gérer les états des éléments (sélection, focus ...)
- fonctionnalités de rendu
- ...

Graphics View Framework

QGraphicsView

- widget de visualisation de la scène
- associer plusieurs vues à une scène
- ...

QGraphicsItem

Eléments standards :

- rectangle : `QGraphicsRectItem`
- ellipse : `QGraphicsEllipseItem`
- texte : `QGraphicsTextItem`
- ...

Graphics View Framework

Création de scène

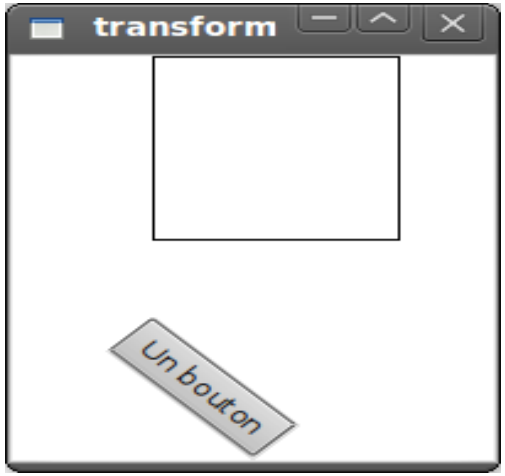
```
import sys
from PyQt5 import QtCore,QtGui
from PyQt5.QtWidgets import QApplication,QWidget,\
QGraphicsScene,QGraphicsView,QGraphicsItem

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

Graphics View Framework

Transformations géométriques d'items

```
#------------- 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);
#-----------------------------------------------
```

Framework Graphics View



Intégration de primitives évoluées

```
#------------- 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)
#----------------------------------------------
```

Graphics View Framework

Interaction avec les items

```
class WebProxy(QGraphicsProxyWidget) :
  def __init__(self):
    super().__init__()
    self.angle=0.0
  def get_rotate(self) :
    return self.angle
  def set_rotate(self,angle) :
    self.angle=angle
```

Graphics View Framework

Interaction avec les items

```
class Scene(QGraphicsScene) :
  def __init__(self):
    super().__init__()
    #------------- The Scene -------------------------
    self.setSceneRect(0,0,1000,800)
    web=QWebView()
    web.load(QtCore.QUrl("http://www.developpez.com"))
    self.proxy=WebProxy()
    self.proxy.setWidget(web)
    self.addItem(self.proxy)
    #-------------------------------------------------
```

Graphics View Framework

Interaction avec les items

```
def mouseMoveEvent(self,event) :
  if (event.buttons() & QtCore.Qt.LeftButton) :
    delta=QtCore.QPointF(event.scenePos() \
                         - event.lastScenePos())
    rotation=delta.x()
    self.proxy.set_rotate(rotation \
                          + self.proxy.get_rotate())
  matrix=QtGui.QTransform()
  matrix.translate(self.proxy.widget().width()/2.0,
                   self.proxy.widget().height()/2.0)
  matrix.rotate(self.proxy.get_rotate(),
                QtCore.Qt.YAxis)
  self.proxy.setTransform(matrix)
```

Graphics View Framework

Interaction avec les items

```
if __name__ == "__main__" :
  app=QApplication(sys.argv)
  button = QPushButton("Quit application")
  button.move(100,100)
  button.clicked.connect(qApp.quit)
  proxy = QGraphicsProxyWidget()
  proxy.setWidget(button)
  scene=Scene()
  scene.addItem(proxy)
  view=QGraphicsView(scene)
  view.setWindowTitle("Ma scene WEB")
  view.show()
```

Graphics View Framework

Work in Progress ...

. . .

Editeur Graphique Qt4.8

Programme principal (main.cpp)

```
#include <QApplication>
#include "mainWindow.h"

int main(int argc, char* argv[]) {
     QApplication app(argc, argv);
/*
     QGraphicsScene scene;
     scene.addEllipse(-10, -10, 20, 20);
     QGraphicsView view(&scene);
     view.show();
*/
     MainWindow mw;
     mw.show();
     return app.exec();
```

Editeur Graphique Qt4.8

Fenêtre Principale (mainwindow.h)

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QWidget>
#include <QMainWindow>
#include <QGraphicsView>
#include <QGraphicsScene>
#include "paintArea.h"

class MainWindow : public QMainWindow {
  public:
   MainWindow(QWidget *parent = 0);
};
#endif
```

Editeur Graphique Qt4.8

Fenêtre Principale (mainwindow.cpp)

```
#include "mainWindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent) {
  QGraphicsView *view = new QGraphicsView();
  Scene *scene = new Scene (0,0,600,800,this);
  QGraphicsTextItem * text =
            scene->addText("Je peux me deplacer !");
  text->setPos(100,100);
  text->setVisible(true);
  view->setScene(scene);
  setCentralWidget(view);
}
```

Editeur Graphique Qt4.8

Zone cliente (scene.h)

```
class Scene : public QGraphicsScene {
  Q_OBJECT
  public:
   Scene(qreal x,qreal y,
         qreal width,qreal height,QObject* parent);
  protected :
    void mousePressEvent(QGraphicsSceneMouseEvent* );
    void mouseMoveEvent(QGraphicsSceneMouseEvent* );
    void mouseReleaseEvent(QGraphicsSceneMouseEvent* );
  private :
   QPointF  _startPoint,_endPoint, _offset;
   QGraphicsItem * _item;
};
```

Editeur Graphique Qt4.8

Zone cliente (scene.cpp)

```
Scene::Scene(qreal x,qreal y,
             qreal width,qreal height,
             QObject * parent = 0) :
                    QGraphicsScene(x,y,
                                   width,height,
                                   parent) {
  _startPoint = _endPoint = _offset = QPointF(0,0);
  addRect(QRectF(100,100,200,300));
  _item=NULL;
}
```

Editeur Graphique Qt4.8

Zone cliente (scene.cpp)

```
void Scene::mousePressEvent(
                    QGraphicsSceneMouseEvent* evt) {
  _startPoint = _endPoint = evt->scenePos();
  _item=itemAt(_startPoint);
  if (_item) {
    QList<QGraphicsItem *> items=collidingItems(_item);
    for (int i =0; i< items.size();i++) {
     items.value(i)->hide();
    }
    _offset =  _startPoint - _item->pos();
//    _item->setPos(_startPoint - _offset );
    _item->grabMouse();
}
```

Editeur Graphique Qt4.8

Zone cliente (scene.cpp)

```
void Scene::mouseMoveEvent(
                    QGraphicsSceneMouseEvent* evt) {
  if (_item) {
    _item->setPos(evt->scenePos() - _offset);
  }
  _endPoint = evt->scenePos();
}
```

Editeur Graphique

Zone cliente (scene.cpp)

```
void Scene::mouseReleaseEvent(
                    QGraphicsSceneMouseEvent* evt)
{
  if (_item) {
    _item->setPos(evt->scenePos() - _offset);
    _item->ungrabMouse();
    _item=NULL;
  }
```

Editeur Graphique Qt4.8

Zone cliente (scene.cpp)

```
  else if (_startPoint != _endPoint) {
    QGraphicsRectItem *rect =
        addRect(_startPoint.x(),
                _startPoint.y(),
                _endPoint.x() - _startPoint.x(),
                _endPoint.y() - _startPoint.y());
    rect->setFlag(QGraphicsItem::ItemIsMovable);
 }
 _endPoint = evt->scenePos();
  qDebug() << items().size();
}
```

Bibliographie

Adresses "au Net"

- les livres : `http://qt.developpez.com/livres`
- documentation officielle : `http://doc.qt.io`
- la communauté française : `http://www.qtfr.org`
- club des pro. de l'info. : `http://qt-devnet.developpez.com`
- Eric Lecolinet : `http://www.infres.enst.fr/~elc/graph`
- Pierre Puiseux : `http://web.univ-pau.fr/~puiseux/enseignement`
- Thierry Vaira : `http://tvaira.free.fr`