Construction d'IHM avec PyQt

Alexis NÉDÉLEC

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

ÉCOLE 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 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"

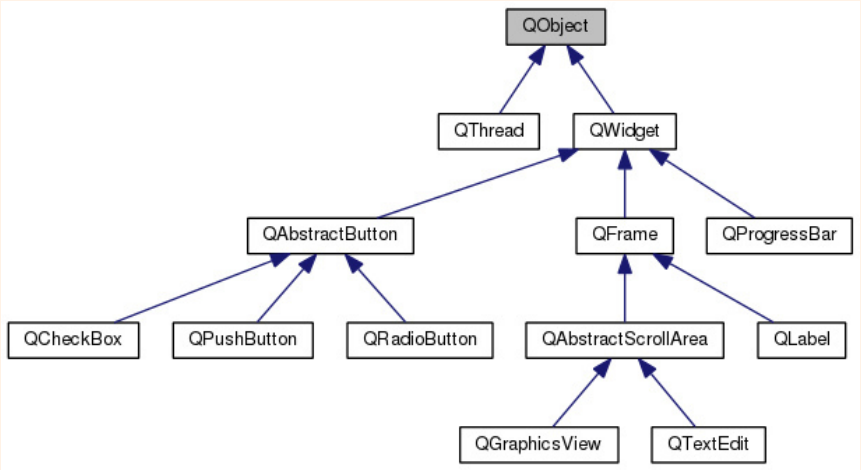
Qt API

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

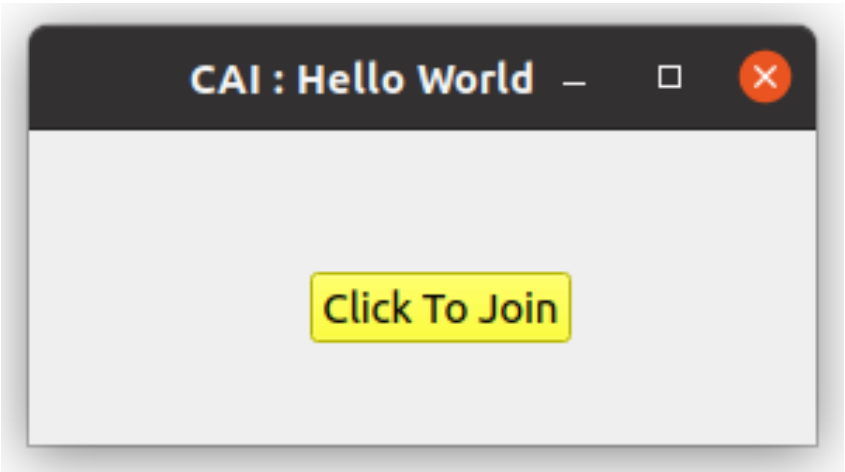
- `Qt 3D` : développement d'applications 3D
- `Qt Android Extras` : API spécifique pour Androïd
- `Qt Bluetooth` : Android, iOS, Linux, macOS, WinRT
- `Qt SCXML` : création de "State Machine" dans des applications
- `Qt Sensors` : données capteurs, reconnaissance de gestes
- `Qt Speech` : pour faire 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");
```

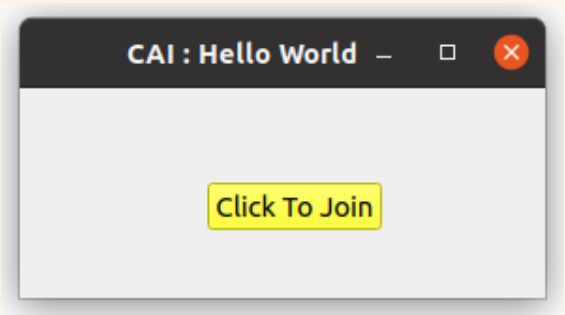


Hello World

Composant graphique (main.cpp)

```
 QPushButton *button= new QPushButton("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

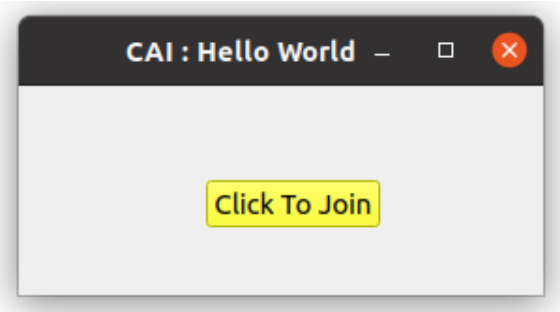
Signaux et slots

```
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 !

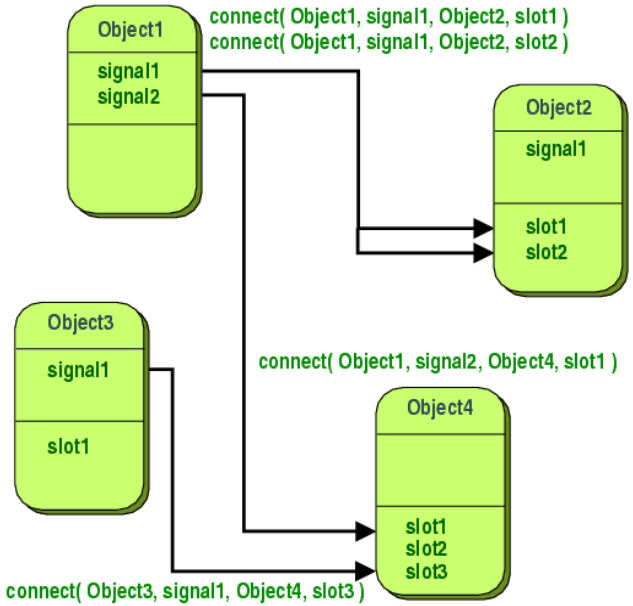
```
if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    widget = QtWidgets.QWidget()
    widget.setGeometry(300,150,300,400)
    widget.setWindowTitle("PyQt5 : Hello 1")
    gui(widget)
    widget.show()
    sys.exit(app.exec_())
```

- instancier un objet d'application (module `QtWidgets`)
- créer une fenêtre principale (composant `QtWidget`)
- système de fenêrage (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

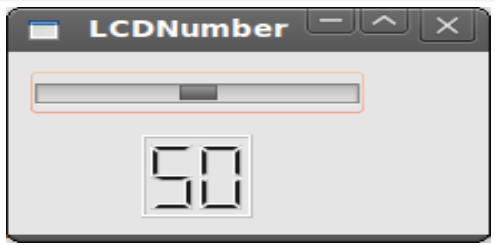
```
class SliderLCD(QtWidgets.QWidget):
  def __init__(self, parent=None):
    QtWidgets.QWidget.__init__(self, parent)
    lcd=QtWidgets.QLCDNumber(self)
    slider=QtWidgets.QSlider(QtCore.Qt.Horizontal,self)
    slider.valueChanged.connect(lcd.display)
```

- 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):
    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()
  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  ?
```

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 !
```

Signaux et slots

Passage d'arguments

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

Comment récupérer les chiffres du pavé numérique ?

Signaux et slots

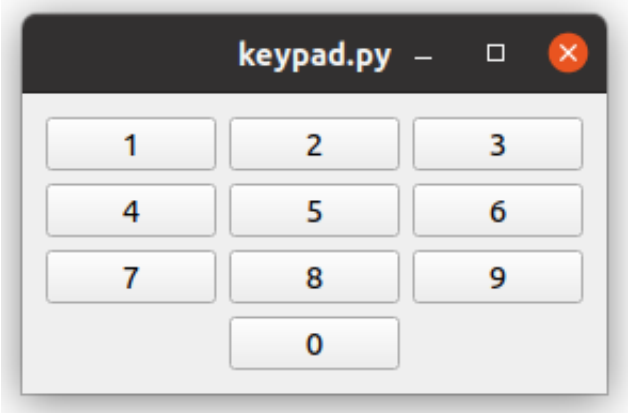
Passage d'arguments : fonctions anonymes (lambda)

```
def create_buttons(self,number) :
    for i in range(1,number) :
      button=QtWidgets.QPushButton(str(i),self)
      self.layout.addWidget(button,i//3,i%3)
      button.clicked.connect(lambda state,x=i : \
                    self.on_selected(state,x) )
      self.buttons.append(button)
      self.layout.addWidget(button)
    button=QtWidgets.QPushButton(str(0),self)
    button.clicked.connect(lambda state,x=0: \
                  self.on_selected(state,x) )
    self.buttons.append(button)
    self.layout.addWidget(button,4,1)
```

Signaux et slots

Passage d'arguments : fonctions anonymes (lambda)

```
  def on_selected(self,state,index):
    print('state', state)
    print('index', index)
if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    keypad = Keypad()
    keypad.show()
    sys.exit(app.exec_())
```



Événements

Classes d'événements : dérivées de la classe abstraite `QEvent`

- `QKeyEvent, QMouseEvent` : actions clavier,souris
- `QResizeEvent, QPaintEvent` : retailler, redessiner
- `QExposeEvent, QHideEvent` : affichage de fenêtres
- ...
- `QTouchEvent, QGestureEvent` : application Post-WIMP

Signal/Slots vs Evénements

- un événement n'émet pas de signal, est généré de l'"extérieur"
- les signaux/slots permettent la communication inter-classes
- les signaux sont utiles pour la manipulation de widgets
- les événements sont nécessaires pour implémenter des widgets

Événements

Héritage QWidget : surdéfinition de méthodes

```
class Scribble(QtWidgets.QWidget):
    def __init__(self):
        super().__init__()
        ...
    def mousePressEvent(self,event) :
        ...
    def mouseMoveEvent(self,event) :
        ...
        self.update()
    def mouseReleaseEvent(self,event) :
        ...
        self.update()
    def paintEvent(self,event) :
        ...
```

Événements

Héritage QWidget : surdéfinition de méthodes

```
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` : surdéfinition de méthodes

```
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` : surdéfinition de méthodes

```
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

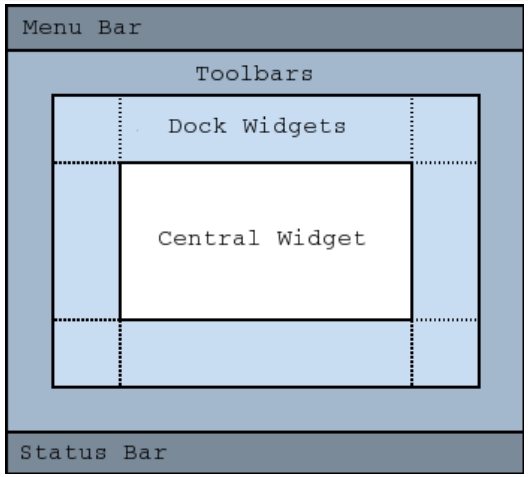
Héritage QWidget : surdéfinition de méthodes

```
  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())
```

Fenêtre principale

QMainWindow

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



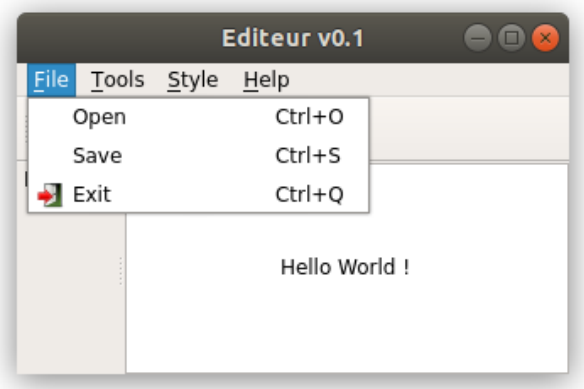
Création d'une fenêtre principale

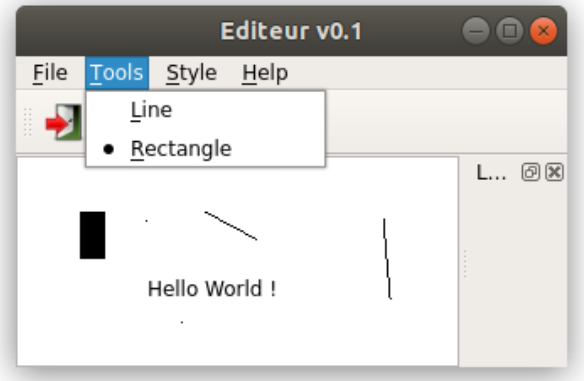
- héritage de `QMainWindow`
- création des zones de travail dans le constructeur

Fenêtre principale

Application

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





Fenêtre principale

MainWindow : création

```
class MainWindow(QtWidgets.QMainWindow):
  def __init__(self):
    QtWidgets.QMainWindow.__init__(self)
    self.resize(500,300)
    self.setWindowTitle("CAI : 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 ...

Fenêtre principale

Scene : Zone cliente

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

- `QGraphicsView` : La vue
- `QGraphicsScene` : modèle associé à la vue
- `self.scene=Scene(self)` : notre modèle de Scène
- `setCentralWidget(view)` : zone cliente

Fenêtre principale

Scene : Zone cliente

```
class Scene (QtWidgets.QGraphicsScene) :
  def __init__(self,parent=None) :
    QtWidgets.QGraphicsScene.__init__(self)
    self.begin=QtCore.QPoint(0,0)
    self.end=QtCore.QPoint(0,0)
    self.pen=QtGui.QPen()
    self.pen.setWidth(3)
    self.pen.setColor(QtCore.Qt.red)
    self.brush=QtGui.QBrush()
    self.brush=QtGui.QBrush(QtCore.Qt.black)
#     self.tool='Line'
#     self.item=None
```

Fenêtre principale

Scene : Zone cliente

```
def mousePressEvent(self, event):
  self.begin=self.end=event.scenePos()
def mouseMoveEvent(self, event):
  self.end = event.scenePos()
def mouseReleaseEvent(self, event):
  self.end = event.scenePos();
  rect=QtWidgets.QGraphicsRectItem(
                    self.begin.x(),self.begin.y(),
                    self.end.x()-self.begin.x(),
                    self.end.y()-self.begin.y()
      )
  rect.setPen(self.pen)
  rect.setBrush(self.brush)
  self.addItem(rect)
```

Fenêtre Principale

MainWindow : création d'actions (QAction)

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

  self.action_pen_color=QtWidgets.QAction(\
                              self.tr("&Color"),self)

  self.action_help_about=QtWidgets.QAction(\
                              self.tr("&Help"),self)
```

Fenêtre Principale

MainWindow : regroupement d'actions (QActionGroup)

```
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)
...
```

Fenêtre Principale

MainWindow : 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)
```

Fenêtre Principale

MainWindow : connexion action/comportement (Signal/Slot)

```
def connect_actions(self) :
  self.action_open.triggered.connect(self.file_open)
  self.action_help_about.triggered.connect(
      self.help_about)
  self.action_pen_color.triggered.connect(
      self.style_pen_color_selection)
  ...
```

Fenêtre Principale

MainWindow : connexion action/comportement (Signal/Slot)

```
self.action_line.triggered.connect(
  lambda checked,tool="Line": \
          self.set_action_tool(checked,tool) )
self.action_rect.triggered.connect(
  lambda checked, tool="Rectangle": \
          self.set_action_tool(checked,tool))
...
```

Fonctions anonymes : transmission de données aux comportements

Fenêtre Principale

MainWindow : définition des comportements (Slot)

```
def file_open(self):
  filename = QtWidgets.QFileDialog.getOpenFileName(self,\
   'Open File', os.getcwd())
  fileopen=QtCore.QFile(filename[0])
  ...
def help_about(self):
  QtWidgets.QMessageBox.information(self,\
                self.tr("About Me"),\
                self.tr("Jean Dupond\n copyright ...")
              )
```

Fenêtre Principale

MainWindow : définition des comportements (Slot)

```
def style_pen_color_selection(self):
  color = QtWidgets.QColorDialog.getColor(\
            QtCore.Qt.yellow,self
          )
  if color.isValid() :
    self.scene.set_pen_color(color.name())

def set_action_tool(self,checked,tool) :
  self.scene.set_tool(tool)
```

Fenêtre Principale

MainWindow : interaction sur la zone client

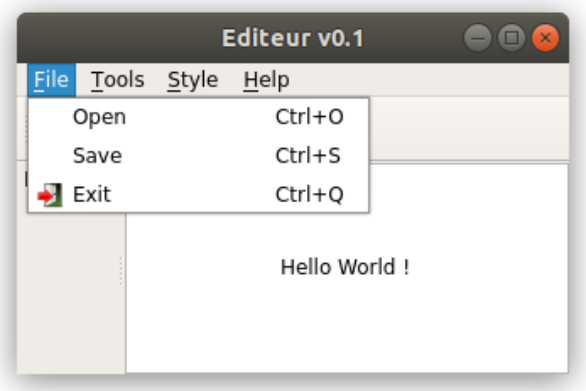
```
class Scene (QtWidgets.QGraphicsScene) :
  def __init__(self,parent=None) :
    ...
    self.tool='Line'
    self.item=None
  def set_tool(self,tool) :
    self.tool=tool
  def set_pen_color(self,color) :
    self.pen.setColor(QtGui.QColor(color))
```

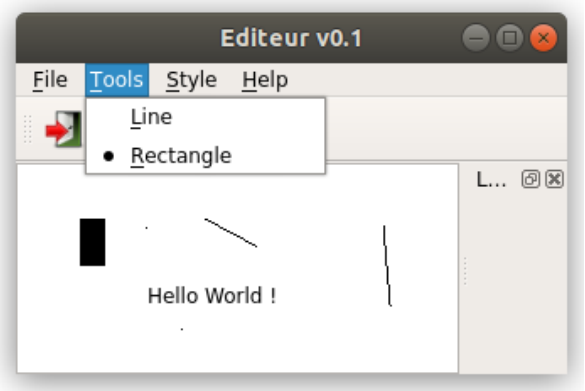
Fenêtre Principale

Interaction sur la zone client

```
def mouseReleaseEvent(self, event):
  self.end =event.scenePos()
  item=None
  if self.tool=="Line" :
    item=QtWidgets.QGraphicsLineItem(
          self.begin.x(),self.begin.y(),
          self.end.x(),self.end.y()
        )
    item.setPen(self.pen)
  elif self.tool=="Rectangle" :
    ...
```

Fenêtre Principale





Palette d'outils

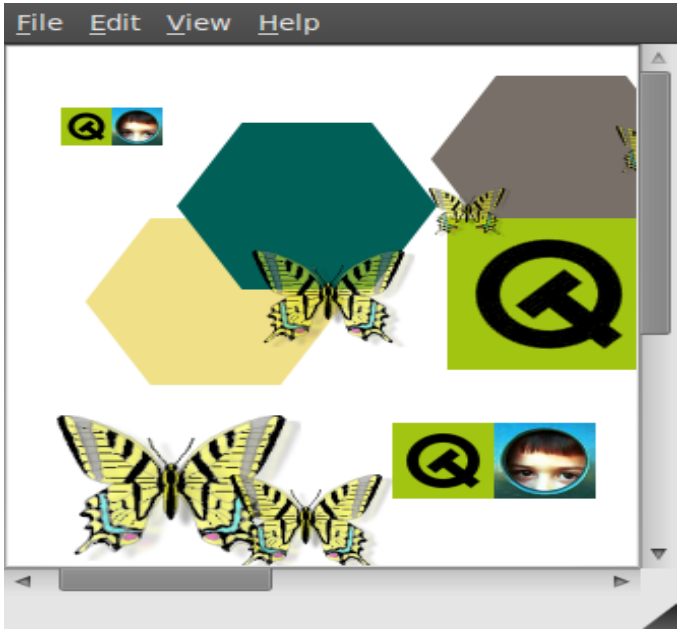
```
self.dock=QtWidgets.QDockWidget("Left Right Dock",\
                                self)
self.dock.setAllowedAreas(\
            QtCore.Qt.LeftDockWidgetArea\
          | QtCore.Qt.RightDockWidgetArea )
self.addDockWidget(\
            QtCore.Qt.LeftDockWidgetArea,\
            self.dock )

...
```

Graphics 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 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 Framework

QGraphicsView

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

QGraphicsItem

Eléments standards :

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

Graphics 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 Framework

Repérage de scène

```
view = QtWidgets.QGraphicsView()
view.setGeometry(QtCore.QRect(0,0,600,500))
#------------- scene creation --------------------
scene.setSceneRect(-150,-150,300,300)
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())
view.setScene(scene)
```

Graphics Framework

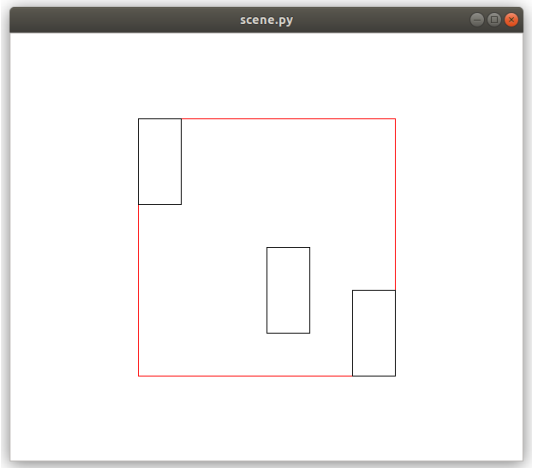
Repérage de scène

```
pen=QtGui.QPen(QtCore.Qt.red)
scene.addLine(top,pen)
scene.addLine(left,pen)
scene.addLine(right,pen)
scene.addLine(bottom,pen)
```

Graphics Framework

Repérage de scène

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



Graphics Framework

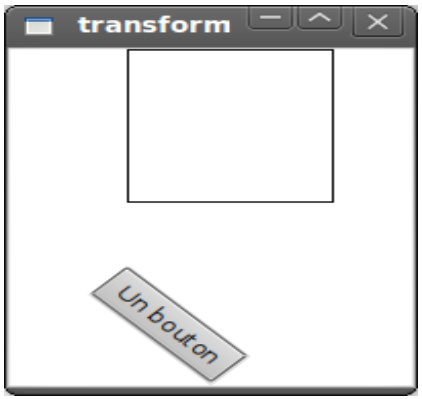
Transformations géométriques

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



Graphics Framework

Interaction avec les items graphiques

```
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 Framework

Interaction avec les items graphiques

```
class Scene(QGraphicsScene) :
  def __init__(self):
    super().__init__()
    #------------- scene creation ------------------
    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 Framework

Interaction avec les items graphiques

```
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 Framework

Interaction avec les items graphiques

```
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("CAI : Scene WEB")
  view.show()
```

Editeur Graphique

Fenêtre Principale : initialisation

```
import sys
from PyQt5 import QtCore,QtGui,QtWidgets
from scene import Scene
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()
```

Editeur Graphique

Fenêtre Principale : création de scène

```
def create_scene(self) :
    view=QtWidgets.QGraphicsView()
    self.scene=Scene(self)
    text=self.scene.addText("Hello World !")
    text.setPos(100,200)
    view.setScene(self.scene)
    self.setCentralWidget(view)
```

Editeur Graphique

Fenêtre Principale : création des actions

```
def create_actions(self) :
  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.group_action_tools.addAction(self.action_line)
```

Editeur Graphique

Fenêtre Principale : création des menus

```
def create_menus(self) :
  menubar = self.menuBar()
  menu_file = menubar.addMenu('&Tools')
  menu_file.addAction(self.action_line)
```

Fenêtre Principale : Signal/Slot

```
def connect_actions(self) :
  self.action_line.triggered.connect(
          lambda checked,tool="line": \
          self.set_action_tool(checked,tool))
```

Editeur Graphique

Fenêtre Principale : définition des comportements (slots)

```
def set_action_tool(self,checked,tool) :
  print("checked : ",checked)
  print("tool : ",tool)
  self.scene.set_tool(tool)
```

Editeur Graphique

Fenêtre Principale : Application

```
if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    mw=MainWindow()
    mw.show()
    sys.exit(app.exec_())
```

Editeur Graphique

La scène : initialisation

```
class Scene(QtWidgets.QGraphicsScene) :
  def __init__(self,x=0,y=0,\
                width=500,height=400,parent=None) :
    QtWidgets.QGraphicsScene.__init__(self)
    self.start=self.end=QtCore.QPointF(0,0)
    self.offset=QtCore.QPointF(0,0)
    self.item=None
    self.pen.setColor(QtCore.Qt.red)
    self.pen.setWidth(3)
    self.brush=QtGui.QBrush(QtCore.Qt.green)
```

Editeur Graphique

La scène : définition des comportements

```
def set_tool(self,tool) :
  print("set_tool(self,tool)",tool)
  self.tool=tool
def set_pen_color(self,color) :
  self.pen.setColor(color)
def set_brush_color(self,color) :
  print("set_brush_color(self,color)",color)
  self.color_brush=color
```

Editeur Graphique

La scène : gestion d'événements

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

Editeur Graphique

La scène : gestion d'événements

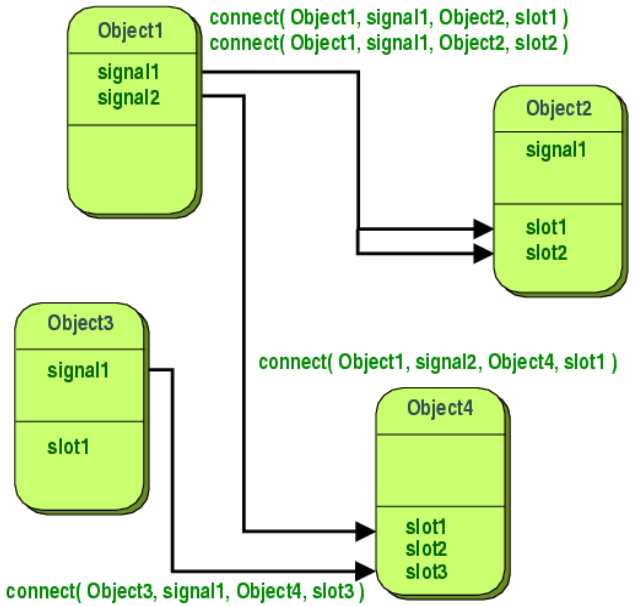
```
def mouseReleaseEvent(self, event):
  self.end=event.scenePos()
  if self.item :
    self.item.setPos(event.scenePos()-self.offset)
    self.item=None
  elif self.tool=='line' :
    self.addLine(self.begin.x(),self.begin.y(),
                 self.end.x(),self.end.y(),self.pen)
  else :
    print("no item selected and nothing to draw !")
```

Qt C++ : 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"



Qt C++ : 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`)

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

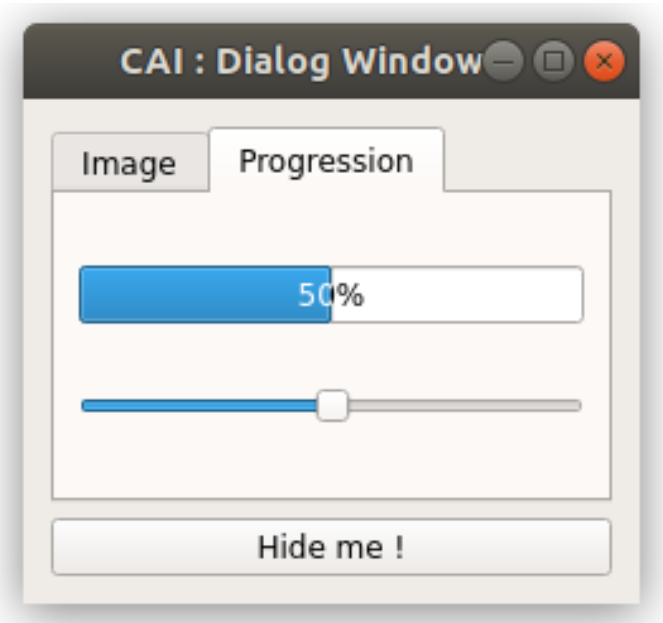
```
SigSlot
|-- Include
|   |-- sigslot.h
|-- main.o
|-- Makefile
|-- moc_sigslot.cpp
|-- moc_sigslot.o
|-- SigSlot
|-- sigslot.o
|-- SigSlot.pro
|-- Src
|   |-- main.cpp
|   |-- sigslot.cpp
2 directories, 10 files
```

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

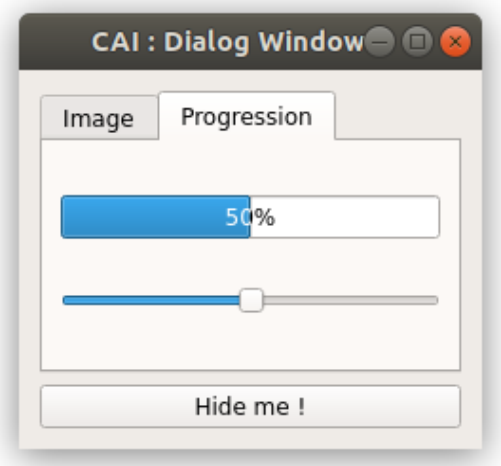
Création d'onglets : communication QSlider/QProgressBar

```
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;
}
```



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`